**VIEW**

**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://msdn.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://msdn.microsoft.com/en-us/library/ms177634.aspx) and [SELECT (Transact-SQL)](http://msdn.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://msdn.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://msdn.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://msdn.microsoft.com/en-us/library/ms190334.aspx), [sys.columns](http://msdn.microsoft.com/en-us/library/ms176106.aspx), and [sys.sql\_expression\_dependencies](http://msdn.microsoft.com/en-us/library/bb677315.aspx). The text of the CREATE VIEW statement is stored in the [sys.sql\_modules](http://msdn.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://msdn.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://msdn.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://msdn.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://msdn.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:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl53other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl53other');)

--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:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl54other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl54other');)

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:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl55other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl55other');)

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:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl56other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl56other');)

{ [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://msdn.microsoft.com/en-us/library/ms179856.aspx) and [OPENROWSET (Transact-SQL)](http://msdn.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://msdn.microsoft.com/en-us/library/ms174335.aspx) statement. For more information, see [Bulk Exporting Data from or Bulk Importing Data to a View](http://msdn.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://msdn.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.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl72other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl72other');)

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.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl73other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl73other');)

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.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl74other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl74other');)

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.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl75other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl75other');)

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.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl76other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl76other');)

--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://msdn.microsoft.com/en-us/library/ms187956.aspx).

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

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

}

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 view to change.

*column*

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

|  |
| --- |
| **ms173846.note(en-us,SQL.100).gifImportant:** |
| Column permissions are maintained only when columns have the same name before and after ALTER VIEW is performed. |
| **ms173846.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 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://msdn.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://msdn.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*.

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

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

|  |
| --- |
| **ms173846.note(en-us,SQL.100).gifNote:** |
| 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.

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

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

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

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl30other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl30other');)

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.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl31other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl31other');)

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

**DELETE (Transact-SQL)**

Removes rows from a table or view.

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

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

[ WITH <common\_table\_expression> [ ,...n ] ]

DELETE

    [ TOP ( expression ) [ PERCENT ] ]

    [ FROM ]

    { <object> | rowset\_function\_limited

      [ WITH ( <table\_hint\_limited> [ ...n ] ) ]

    }

    [ <OUTPUT Clause> ]

    [ FROM <table\_source> [ ,...n ] ]

    [ WHERE { <search\_condition>

            | { [ CURRENT OF

                   { { [ GLOBAL ] cursor\_name }

                       | cursor\_variable\_name

                   }

                ]

              }

            }

    ]

    [ OPTION ( <Query Hint> [ ,...n ] ) ]

[; ]

<object> ::=

{

    [ server\_name.database\_name.schema\_name.

      | database\_name. [ schema\_name ] .

      | schema\_name.

    ]

        table\_or\_view\_name

}

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

WITH <common\_table\_expression>

Specifies the temporary named result set, also known as common table expression, defined within the scope of the DELETE statement. The result set is derived from a SELECT statement.

Common table expressions can also be used with the SELECT, INSERT, UPDATE, and CREATE VIEW statements. For more information, see [WITH common\_table\_expression (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms175972.aspx).

TOP **(** *expression* **)** [ PERCENT ]

Specifies the number or percent of random rows that will be deleted. *expression* can be either a number or a percent of the rows. The rows referenced in the TOP expression used with INSERT, UPDATE, or DELETE are not arranged in any order.

Parentheses delimiting *expression* in TOP are required in INSERT, UPDATE, and DELETE statements. For more information, see [TOP (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms189463.aspx).

FROM

Is an optional keyword that can be used between the DELETE keyword and the target *table\_or\_view\_name*, or *rowset\_function\_limited*.

*server\_name*

Is the name of the server (using a linked server name or the [OPENDATASOURCE](http://msdn.microsoft.com/en-us/library/ms179856.aspx) function as the server name) on which the table or view is located. If *server\_name* is specified, *database\_name* and *schema\_name* are required.

*database\_name*

Is the name of the database.

*schema\_name*

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

*table\_or view\_name*

Is the name of the table or view from which the rows are to be removed.

A **table** variable, within its scope, also can be used as a table source in a DELETE statement.

The view referenced by *table\_or\_view\_name* must be updatable and reference exactly one base table in the FROM clause of the view. For more information about updatable views, see [CREATE VIEW (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187956.aspx).

*rowset\_function\_limited*

Is either the [OPENQUERY](http://msdn.microsoft.com/en-us/library/ms188427.aspx) or [OPENROWSET](http://msdn.microsoft.com/en-us/library/ms190312.aspx) function, subject to provider capabilities. For more information about the capabilities required by the provider, see [UPDATE and DELETE Requirements for OLE DB Providers](http://msdn.microsoft.com/en-us/library/ms188702.aspx).

WITH **(** <table\_hint\_limited> [... *n*] **)**

Specifies one or more table hints that are allowed for a target table. The WITH keyword and the parentheses are required. NOLOCK and READUNCOMMITTED are not allowed. For more information about table hints, see [Table Hints (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187373.aspx).

<OUTPUT\_Clause>

Returns deleted rows, or expressions based on them, as part of the DELETE operation. The OUTPUT clause is not supported in any DML statements targeting views or remote tables. For more information, see [OUTPUT Clause (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms177564.aspx).

FROM <table\_source>

Specifies an additional FROM clause. This Transact-SQL extension to DELETE allows specifying data from <table\_source> and deleting the corresponding rows from the table in the first FROM clause.

This extension, specifying a join, can be used instead of a subquery in the WHERE clause to identify rows to be removed.

For more information, see [FROM (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms177634.aspx).

WHERE

Specifies the conditions used to limit the number of rows that are deleted. If a WHERE clause is not supplied, DELETE removes all the rows from the table.

There are two forms of delete operations based on what is specified in the WHERE clause:

* Searched deletes specify a search condition to qualify the rows to delete. For example, WHERE *column\_name* = *value*.
* Positioned deletes use the CURRENT OF clause to specify a cursor. The delete operation occurs at the current position of the cursor. This can be more accurate than a searched DELETE statement that uses a WHERE *search\_condition* clause to qualify the rows to be deleted. A searched DELETE statement deletes multiple rows if the search condition does not uniquely identify a single row.

<search\_condition>

Specifies the restricting conditions for the rows to be deleted. There is no limit to the number of predicates that can be included in a search condition. For more information, see [Search Condition (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms173545.aspx).

CURRENT OF

Specifies that the DELETE is performed at the current position of the specified cursor.

GLOBAL

Specifies that *cursor\_name* refers to a global cursor.

*cursor\_name*

Is the name of the open cursor from which the fetch is made. If both a global and a local cursor with the name *cursor\_name* exist, this argument refers to the global cursor if GLOBAL is specified; otherwise, it refers to the local cursor. The cursor must allow updates.

*cursor\_variable\_name*

Is the name of a cursor variable. The cursor variable must reference a cursor that allows updates.

OPTION **(** <query\_hint> [ **,**... *n*] **)**

Are keywords that indicate that optimizer hints are used to customize the way the Database Engine processes the statement. For more information, see [Query Hints (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms181714.aspx).

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

DELETE can be used in the body of a user-defined function if the object modified is a **table** variable.

The DELETE statement may fail if it violates a trigger or tries to remove a row referenced by data in another table with a FOREIGN KEY constraint. If the DELETE removes multiple rows, and any one of the removed rows violates a trigger or constraint, the statement is canceled, an error is returned, and no rows are removed.

When a DELETE statement encounters an arithmetic error (overflow, divide by zero, or a domain error) occurring during expression evaluation, the Database Engine handles these errors as if SET ARITHABORT is set ON. The rest of the batch is canceled, and an error message is returned.

The setting of the SET ROWCOUNT option is ignored for DELETE statements against remote tables and local and remote partitioned views.

If you want to delete all the rows in a table, use the DELETE statement without specifying a WHERE clause, or use [TRUNCATE TABLE](http://msdn.microsoft.com/en-us/library/ms177570.aspx). TRUNCATE TABLE is faster than DELETE and uses fewer system and transaction log resources.

**Deleting Rows from a Heap**

When rows are deleted from a heap the Database Engine may use row or page locking for the operation. As a result, the pages made empty by the delete operation remain allocated to the heap. When empty pages are not deallocated, the associated space cannot be reused by other objects in the database.

To delete rows in a heap and deallocate pages, use one of the following methods.

* Specify the TABLOCK hint in the DELETE statement. Using the TABLOCK hint causes the delete operation to take a shared lock on the table instead of a row or page lock. This allows the pages to be deallocated. For more information about the TABLOCK hint, see [Table Hints (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187373.aspx).
* Use TRUNCATE TABLE if all rows are to be deleted from the table.
* Create a clustered index on the heap before deleting the rows. You can drop the clustered index after the rows are deleted. This method is more time consuming than the previous methods and uses more temporary resources.

For more information about locking, see [Locking in the Database Engine](http://msdn.microsoft.com/en-us/library/ms190615.aspx).

**Using an INSTEAD OF Trigger on DELETE Actions**

When an INSTEAD OF trigger is defined on DELETE actions against a table or view, the trigger executes instead of the DELETE statement. Earlier versions of SQL Server only support AFTER triggers on DELETE and other data modification statements. The FROM clause cannot be specified in a DELETE statement that references, either directly or indirectly, a view with an INSTEAD OF trigger defined on it. For more information about INSTEAD OF triggers, see [CREATE TRIGGER (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms189799.aspx).

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

DELETE permissions are required on the target table. SELECT permissions are also required if the statement contains a WHERE clause.

DELETE permissions default to members of the **sysadmin** fixed server role, the **db\_owner** and **db\_datawriter** fixed database roles, and the table owner. Members of the **sysadmin**, **db\_owner**, and the **db\_securityadmin** roles, and the table owner can transfer permissions to other users.

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

**A. Using DELETE with no WHERE clause**

The following example deletes all rows from the SalesPersonQuotaHistory table because a WHERE clause is not used to limit the number of rows deleted.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl55other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl55other');)

USE AdventureWorks;

GO

DELETE FROM Sales.SalesPersonQuotaHistory;

GO

**B. Using DELETE on a set of rows**

The following example deletes all rows from the ProductCostHistory table in which the value in the StandardCost column is more than 1000.00.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl56other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl56other');)

USE AdventureWorks;

GO

DELETE FROM Production.ProductCostHistory

WHERE StandardCost > 1000.00;

GO

**C. Using DELETE on the current row of a cursor**

The following example deletes a single row from the EmployeePayHistory table using a cursor named complex\_cursor. The delete operation affects only the single row currently fetched from the cursor.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl57other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl57other');)

USE AdventureWorks;

GO

DECLARE complex\_cursor CURSOR FOR

SELECT a.EmployeeID

FROM HumanResources.EmployeePayHistory AS a

WHERE RateChangeDate <>

(SELECT MAX(RateChangeDate)

FROM HumanResources.EmployeePayHistory AS b

WHERE a.EmployeeID = b.EmployeeID) ;

OPEN complex\_cursor;

FETCH FROM complex\_cursor;

DELETE FROM HumanResources.EmployeePayHistory

WHERE CURRENT OF complex\_cursor;

CLOSE complex\_cursor;

DEALLOCATE complex\_cursor;

GO

**D. Using DELETE based on a subquery and using the Transact-SQL extension**

The following example shows the Transact-SQL extension used to delete records from a base table that is based on a join or correlated subquery. The first DELETE statement shows the ISO-compatible subquery solution, and the second DELETE statement shows the Transact-SQL extension. Both queries remove rows from the SalesPersonQuotaHistory table based on the year-to-date sales stored in the SalesPerson table.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl58other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl58other');)

-- SQL-2003 Standard subquery

USE AdventureWorks;

GO

DELETE FROM Sales.SalesPersonQuotaHistory

WHERE SalesPersonID IN

(SELECT SalesPersonID

FROM Sales.SalesPerson

WHERE SalesYTD > 2500000.00);

GO

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl59other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl59other');)

-- Transact-SQL extension

USE AdventureWorks;

GO

DELETE FROM Sales.SalesPersonQuotaHistory

FROM Sales.SalesPersonQuotaHistory AS spqh

INNER JOIN Sales.SalesPerson AS sp

ON spqh.SalesPersonID = sp.SalesPersonID

WHERE sp.SalesYTD > 2500000.00;

GO

**E. Using DELETE with the TOP clause**

The following example deletes 2.5 percent of the rows (27 rows) in the ProductInventory table.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl60other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl60other');)

USE AdventureWorks;

GO

DELETE TOP (2.5) PERCENT

FROM Production.ProductInventory;

GO

**F. Using DELETE with the OUTPUT clause**

The following example shows how to save the results of a DELETE statement into a table variable.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl61other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl61other');)

USE AdventureWorks;

GO

DELETE Sales.ShoppingCartItem

OUTPUT DELETED.\* ;

--Verify all rows in the table have been deleted.

SELECT COUNT(\*) AS [Rows in Table] FROM Sales.ShoppingCartItem;

GO

**G. Using OUTPUT with from\_table\_name in a DELETE statement**

The following example deletes rows in the ProductProductPhoto table based on search criteria defined in the FROM clause of the DELETE statement. The OUTPUT clause returns columns from the table being deleted, DELETED.ProductID, DELETED.ProductPhotoID, and columns from the Product table. This is used in the FROM clause to specify the rows to delete.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl62other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl62other');)

USE AdventureWorks

GO

DECLARE @MyTableVar table (

ProductID int NOT NULL,

ProductName nvarchar(50)NOT NULL,

ProductModelID int NOT NULL,

PhotoID int NOT NULL);

DELETE Production.ProductProductPhoto

OUTPUT DELETED.ProductID,

p.Name,

p.ProductModelID,

DELETED.ProductPhotoID

INTO @MyTableVar

FROM Production.ProductProductPhoto AS ph

JOIN Production.Product as p

ON ph.ProductID = p.ProductID

WHERE p.ProductModelID BETWEEN 120 and 130;

--Display the results of the table variable.

SELECT ProductID, ProductName, ProductModelID, PhotoID

FROM @MyTableVar

ORDER BY ProductModelID;

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://msdn.microsoft.com/en-us/library/ms177563.aspx)

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

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

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 to remove.

http://i.msdn.microsoft.com/Global/Images/clear.gif 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://msdn.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://msdn.microsoft.com/en-us/library/ms174402.aspx).

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

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

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

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

The following example removes the view Reorder.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl22other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl22other');)

USE AdventureWorks ;

GO

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

DROP VIEW dbo.Reorder ;

GO