Create and Alter Tables

Section Objective:

The objective of this section may include but is not limited to: computed and persisted columns; schemas; scripts to deploy changes to multiple environments, for example, dev, test, production; manage permissions (GRANT, DENY, REVOKE)

Table Definition:

In relational databases and flat file databases, a table is a set of data elements (values) that is organized using a model of vertical columns (which are identified by their name) and horizontal rows. A table has a specified number of columns, but can have any number of rows. Each row is identified by the values appearing in a particular column subset which has been identified as a candidate key.

Table is another term for relations; although there is the difference in that a table is usually a multi-set (bag) of rows whereas a relation is a set and does not allow duplicates. Besides the actual data rows, tables generally have associated with them some meta-information, such as constraints on the table or on the values within particular columns.

The data in a table does not have to be physically stored in the database. Views are also relational tables, but their data are calculated at query time. Another example are nicknames, which represent a pointer to a table in another database.

In terms of the relational model of databases, a table can be considered a convenient representation of a relation, but the two are not strictly equivalent. For instance, an SQL table can potentially contain duplicate rows, whereas a true relation cannot contain duplicate tuples. Similarly, representation as a table implies a particular ordering to the rows and columns, whereas a relation is explicitly unordered. However, the database system does not guarantee any ordering of the rows unless an ORDER BY clause is specified in the SELECT statement that queries the table.

An equally valid representations of a relation is as an n-dimensional chart, where n is the number of attributes (a table's columns). For example, a relation with two attributes and three values can be represented as a table with two columns and three rows, or as a two-dimensional graph with three points. The table and graph representations are only equivalent if the ordering of rows is not significant, and the table has no duplicate rows.

Links to Microsoft Information

- Create Table Definition: http://msdn.microsoft.com/en-us/library/ms174979.aspx
- Alter Table Definition: http://msdn.microsoft.com/en-us/library/ms190273.aspx

There are different type(s) of table objects that can exist in SQL Server.

Example(s):

Table Type	Description	Example
Traditional	Exist always as long as user has security access to them. These tables are stored in the target database that the user specifies or the database that the user is defaulted to use.	dbo.MyTable
Temporary - Local	Follow the same rules as “traditional” tables, except are visible only in the current session and cannot be partitioned. You will also notice that the local temporary table starts with a single ‘#’ sign.	#MyTable
Temporary - Global	Follow the same rules as “traditional” tables, except are visible to all sessions and cannot be partitioned. You will also notice that the global temporary table starts with a double ‘##’ sign.	##MyTable
Variable	A table variable is scoped to the stored procedure, batch, or user-defined function just like any local variable you create with a DECLARE statement. The variable will no longer exist after the procedure exits - there will be no table to clean up with a DROP statement. Although you cannot use a table variable as an input or output parameter, you can return a table variable from a user-defined function – we will see an example later in this article. However, because you can’t pass a table variable to another stored procedure as input – there still are scenarios where you’ll be required to use a temporary table when using calling stored procedures from inside other stored procedures and sharing table results. There is no statistics stored for this type of table. Can not use the “Alter” table command.	@MyTable

Create Table Syntax

```
-- ======================================================
-- Traditional Table
-- ======================================================
-- Syntax:
IF EXISTS (SELECT *
             FROM sys.objects
            WHERE object_id = OBJECT_ID(N'<schema>.<MyTables>')
              AND type in (N'U'))
  DROP TABLE <schema>.<MyTables>
GO
```

```
CREATE TABLE <schema>.<MyTables>
(
    <MyTableID>       INT IDENTITY(1, 1)  NOT NULL
  , <Column1>         INT                     NULL
  , <Column2>         VARCHAR(10)             NULL
  , <Column3>         CHAR(2)                 NULL
  , <Column4>         DATETIME                NULL
  , <Column5>         AS ISNULL(Column3, '') + ISNULL(Column2, '')
  , CreatedOn         DATETIME                NULL DEFAULT(GETDATE())
  , CreatedBy         VARCHAR(5)              NULL
)
GO
```

Example(s) : Create Table

Table Type	Example

Traditional	`USE AdventureWorks`
	`GO`
	
	`IF EXISTS (SELECT *`
	`FROM sys.objects`
	`WHERE object_id = OBJECT_ID(N'dbo.MyTables')`
	`AND type in (N'U'))`
	`DROP TABLE dbo.MyTables`
	`GO`
	
	`CREATE TABLE dbo.MyTables`
	`(`
	`MyTableID INT IDENTITY(1, 1) NOT NULL`
	`, Column1 INT NULL`
	`, Column2 VARCHAR(10) NULL`
	`, Column3 CHAR(2) NULL`
	`, Column4 DATETIME NULL`
	`, CreatedOn DATETIME NULL DEFAULT(GETDATE())`
	`, CreatedBy VARCHAR(5) NULL`
	`)`
	`GO`

Local Temporary	`IF EXISTS (SELECT *`
	`FROM sys.objects`
	`WHERE object_id = OBJECT_ID(N'#MyTables')`
	`AND type in (N'U'))`
	`DROP TABLE #MyTables`
	`GO`
	
	`CREATE TABLE #MyTables`
	`(`
	`MyTableID INT IDENTITY(1, 1) NOT NULL`
	`, Column1 INT NULL`
	`, Column2 VARCHAR(10) NULL`
	`, Column3 CHAR(2) NULL`
	`, Column4 DATETIME NULL`
	`, CreatedOn DATETIME NULL DEFAULT(GETDATE())`
	`, CreatedBy VARCHAR(5) NULL`
	`)`
	`GO`

Global Temporary	`IF EXISTS (SELECT *`
	`FROM sys.objects`
	`WHERE object_id = OBJECT_ID(N'##MyTables')`
	`AND type in (N'U'))`

	DROP TABLE ##MyTables GO CREATE TABLE ##MyTables (MyTableID INT IDENTITY(1, 1) NOT NULL , Column1 INT NULL , Column2 VARCHAR(10) NULL , Column3 CHAR(2) NULL , Column4 DATETIME NULL , CreatedOn DATETIME NULL DEFAULT(GETDATE()) , CreatedBy VARCHAR(5) NULL) GO
Variable	DECLARE @MyTables TABLE (MyTableID INT IDENTITY(1, 1) NOT NULL , Column1 INT NULL , Column2 VARCHAR(10) NULL , Column3 CHAR(2) NULL , Column4 DATETIME NULL , CreatedOn DATETIME NULL DEFAULT(GETDATE()) , CreatedBy VARCHAR(5) NULL) GO

The **DEFAULT** key word is used to make sure data is entered into the field upon insert if the field is NULL. The default value has to be a value that will fit the data type.

Alter Table

Modifies a table definition by altering, adding, or dropping columns and constraints, reassigning partitions, or disabling or enabling constraints and triggers.

Action	Example
Add one column	USE AdventureWorks GO ALTER TABLE dbo.MyTables ADD Column7 VARCHAR(20) NULL; GO
Add two columns	USE AdventureWorks GO ALTER TABLE dbo.MyTables ADD Column7 VARCHAR(20) NULL , Column8 VARCHAR(10) NULL; GO
Delete a column	USE AdventureWorks GO ALTER TABLE dbo.MyTables DROP COLUMN Column7;

Constraints on Alter Table Statement:

1. It is important to note that the column that you are adding has to be constrained to "NULL".

2. You can not use “Alter Table” process on a “Variable Table”. For a “Variable Table” all columns need to be created when the table is declared.

Computed Field

A computed column is computed from an expression that can use other columns in the same table. The expression can be a noncomputed column name, constant, function, and any combination of these connected by one or more operators. The expression cannot be a subquery. Unless otherwise specified, computed columns are virtual columns that are not physically stored in the table. Their values are recalculated every time they are referenced in a query.

Example:

```
USE AdventureWorks
GO

ALTER TABLE dbo.MyTables
  ADD Column5 AS ISNULL(Column3, '') + ISNULL(Column2, '')
```

Some Limitations

- You can not reference columns from other tables for a computed column expression directly.
- You can not apply insert or update statements on computed columns.
- If you are combining operators of two different data types in your expression then operator of lower precedence will be converted to that of higher precedence. If implicit conversion is not possible then error will be generated.
- A subquery can not be used as an expression for creating a computed column.
- Computed columns can be used in SELECT lists, WHERE or ORDER BY clauses and as regular expressions , but to use a computed column as CHECK, FOREIGN KEY or NOT NULL constraints you have to set it to Persisted.
- To use a computed column as Primary or Unique Key constraint it should be defined by a deterministic expression and data type of computed column expression should be indexable.

Persisted Field

The Database Engine uses the PERSISTED keyword in the CREATE TABLE and ALTER TABLE statements to physically store computed columns in the table. Their values are updated when any columns that are part of their calculation change. By marking a computed column as PERSISTED, you can create an index on a computed column that is deterministic but not precise.

Example:

```
USE AdventureWorks
GO
```

```
ALTER TABLE dbo.MyTables
  ADD Column5 AS ISNULL(Column3, '') + ISNULL(Column2, '') PERSISTED
```

Here are a few rules:

- If Persisted property is off then calculated column will be just a virtual column. No data for this column will be stored on disk and values will be calculated every time when referenced in a script. If this property is set active then data of computed column will be stored on disk.
- Any update in referenced column will be synchronized automatically in computed column if it is Persisted.
- Along with some other conditions Persisted is required to create an index on the computed column.

Manage Permissions

GRANT Object Permissions

Grants statement allows a user to be assigned permissions on a table, view, table-valued function, stored procedure, extended stored procedure, scalar function, aggregate function, service queue, or synonym.

Links to Microsoft Information

- Grant Permission: http://msdn.microsoft.com/en-us/library/ms188371.aspx

Grant Database Permission Syntax

```
GRANT <permission> [ ,...n ] ON
    [ OBJECT :: ][ schema_name ]. object_name [ ( column [ ,...n ] ) ]
    TO <database_principal> [ ,...n ]
    [ WITH GRANT OPTION ]
    [ AS <database_principal> ]
```

DENY Object Permissions

Deny statement allows a user to be Denies permissions on a member of the OBJECT class of securables. These are the members of the OBJECT class: tables, views, table-valued functions, stored procedures, extended stored procedures, scalar functions, aggregate functions, service queues, and synonyms.

Links to Microsoft Information

- Deny Permission: http://msdn.microsoft.com/en-us/library/ms173724.aspx

Deny Database Permission Syntax

```
DENY <permission> [ ,...n ] ON
    [ OBJECT :: ][ schema_name ]. object_name [ ( column [ ,...n ] ) ]
        TO <database_principal> [ ,...n ]
    [ CASCADE ]
        [ AS <database_principal> ]
```

REVOKE Object Permissions

Revoke statement allows a user to be have permissions removed from them on a table, view, table-valued function, stored procedure, extended stored procedure, scalar function, aggregate function, service queue, or synonym.

Links to Microsoft Information

- Revoke Permissions: http://msdn.microsoft.com/en-us/library/ms187719.aspx

Revoke Database Permission Syntax

```
REVOKE [ GRANT OPTION FOR ] <permission> [ ,...n ] ON
    [ OBJECT :: ][ schema_name ]. object_name [ ( column [ ,...n ] ) ]
        { FROM | TO } <database_principal> [ ,...n ]
    [ CASCADE ]
    [ AS <database_principal> ]
```