Create and Alter Views

Section Objective:

This objective may include but is not limited to: WITH ENCRYPTION; WITH SCHEMABINDING; WITH CHECK OPTION; manage permissions (GRANT, DENY, REVOKE)

View Definition:

In database theory, a view consists of a stored query accessible as a virtual table composed of the result set of a query. Unlike ordinary tables (base tables) in a relational database, a view does not form part of the physical schema: it is a dynamic, virtual table computed or collated from data in the database. Changing the data in a table alters the data shown in subsequent invocations of the view.

Views can provide advantages over tables:

- Views can represent a subset of the data contained in a table
- Views can join and simplify multiple tables into a single virtual table
- Views can act as aggregated tables, where the database engine aggregates data (sum, average etc) and presents the calculated results as part of the data
- Views can hide the complexity of data; for example a view could appear as Sales2000 or Sales2001, transparently partitioning the actual underlying table
- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data it presents
- Depending on the SQL engine used, views can provide extra security
- Views can limit the degree of exposure of a table or tables to the outer world.

Links to Microsoft Information

- Create View Definition: http://msdn.microsoft.com/en-us/library/ms187956.aspx

Different Type(s) of Views

View Type	Description
Standard	This type of view is based on one or more base tables. This view can include joins, filters (where clause) and row count restriction (Top & Order By). Note: “ORDER BY” can not be used without the “TOP”
Updateable	This type of view is based on one table and can be updated directly. You can excute the following DML statements: INSERT, UPDATE, DELETE, and MERGE. This view also allows for the developer to be able to define the “INSTEAD OF INSERT/UPDATE/DELETE” triggers.
Indexed	This allow for the user the ability to create one or more indexes on a view, so

	that the queries can be optimized. The “SCHEMABINDING” option must be used in association with this type of view.
Partitioned	A partitioned view joins data that is spread across a table partitioned horizontally

Create View 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 ]     }
```

Example(s) : Create View (Basic)

Table Type	Example
Single Table (Updateable View)	`USE AdventureWorks` `GO` `IF EXISTS (SELECT *` `FROM sys.views` `WHERE object_id = OBJECT_ID(N'dbo.vPersonEmailAndPhone'))` `DROP VIEW dbo.vPersonEmailAndPhone` `GO` `CREATE VIEW dbo.vPersonEmailAndPhone` `(` `ID` `, LastFirstName` `, Email` `, ContactPhone` `)` `AS` `SELECT ContactID` `, LastName + ', ' + FirstName` `, EmailAddress` `, Phone` `FROM Person.Contact` `GO` `SELECT *` `FROM dbo.vPersonEmailAndPhone`
Multiple Tables (Standard View)	`USE AdventureWorks` `GO` `IF EXISTS (SELECT *` `FROM sys.views` `WHERE object_id = OBJECT_ID(N'dbo.vPersonAddress'))` `DROP VIEW dbo.vPersonAddress` `GO`

	`` CREATE VIEW dbo.vPersonAddress (ID , AddressLine1 , AddressLine2 , City , StateProvince , CountryName , ZipCode) AS SELECT A.AddressID , A.AddressLine1 , ISNULL(A.AddressLine2, '') , A.City , SP.StateProvinceCode , CR.Name , A.PostalCode FROM Person.Address AS A LEFT OUTER JOIN Person.StateProvince AS SP ON A.StateProvinceID = SP.StateProvinceID LEFT OUTER JOIN Person.CountryRegion AS CR ON SP.CountryRegionCode = CR.CountryRegionCode GO SELECT * FROM dbo.vPersonAddress ```

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 View

Links to Microsoft Information

- Alter View Definition: http://msdn.microsoft.com/en-us/library/ms173846.aspx

The "ALTER VIEW" command modifies the previously created view object. This can include an indexed view. By altering the view object with this DDL statement, the dependant objects (Stored Procedures, Triggers) or permissions are effected.

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;

WITH ENCRYPTION

This flag will encrypt the entries in the sys.syscomments object that contains the text of the create/alter view statement. By using this flag, the database knows to prevent the viewfrom being published as part of the SQL Server replication process.

Example:

```
USE AdventureWorks
GO

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

Area	Without Encryption	With Encryption
SQL	USE AdventureWorks GO IF EXISTS (SELECT * FROM sys.views WHERE object_id = OBJECT_ID(N'dbo.vPersonEmailAndPhone')) DROP VIEW dbo.vPersonEmailAndPhone GO CREATE VIEW dbo.vPersonEmailAndPhone (ID , LastFirstName , Email , ContactPhone) AS SELECT ContactID , LastName + ', ' + FirstName , EmailAddress , Phone FROM Person.Contact GO SELECT sc.text FROM syscomments sc JOIN sysobjects so ON sc.id = so.id WHERE so.name = 'vPersonEmailAndPhone' GO	USE AdventureWorks GO IF EXISTS (SELECT * FROM sys.views WHERE object_id = OBJECT_ID(N'dbo.vPersonEmailAndPhone')) DROP VIEW dbo.vPersonEmailAndPhone GO CREATE VIEW dbo.vPersonEmailAndPhone (ID , LastFirstName , Email , ContactPhone) WITH ENCRYPTION AS SELECT ContactID , LastName + ', ' + FirstName , EmailAddress , Phone FROM Person.Contact GO SELECT sc.text FROM syscomments sc JOIN sysobjects so ON sc.id = so.id WHERE so.name = 'vPersonEmailAndPhone' GO
Results	text ---------- CREATE VIEW dbo.vPersonEmailAndPhone (ID	text ---------- NULL

	, LastFirstName , Email , ContactPhone) AS SELECT ContactID , LastName + ', ' + FirstName , EmailAddress , Phone FROM Person.Contact (1 row(s) affected)	

WITH SCHEMABINDING

This flag binds the view to the schema of the underlying table(s). This enforces that the base table(s) cannot be modified in a way that would affect the view object definition.

Limitation(s):

- The view definition must be modified or dropped to remove the dependencies on the underlying table that is to be modified.
- The "SELECT" statement must include the two-part naming structure (schema.object) of the tables, views or UDFs that are being referenced
- Cannot be used if view contains "alias" date type columns

Example(s):

Create View

```
USE AdventureWorks
GO

IF EXISTS (SELECT *
             FROM sys.views
            WHERE object_id =
OBJECT_ID(N'dbo.vPersonEmailAndPhone'))
  DROP VIEW dbo.vPersonEmailAndPhone
GO

CREATE VIEW dbo.vPersonEmailAndPhone
(
    ID
  , LastFirstName
  , Email
  , ContactPhone
)
WITH SCHEMABINDING
AS
SELECT ContactID
     , LastName + ', ' + FirstName
     , EmailAddress
     , Phone
  FROM Person.Contact
GO
```

Alter View

```
USE AdventureWorks
GO

ALTER VIEW dbo.vPersonEmailAndPhone
(
    ID
  , LastFirstName
  , Email
  , ContactPhone
)
WITH SCHEMABINDING
AS
SELECT ContactID
     , LastName + ', ' + FirstName
     , EmailAddress
     , Phone
  FROM Person.Contact
GO
```

WITH CHECK OPTION;

This flag forces all DML statements that are executed against the view object to follow the criteria set within the select statement. This is to make sure that an insert or update statement would not cause the rows to no longer appear in the view when the view is called to produce data.

Example:

```
USE tempdb
GO

SET NOCOUNT ON

IF EXISTS (SELECT *
             FROM sys.views
            WHERE object_id = OBJECT_ID(N'vw_Customers_Tacoma'))
  DROP VIEW vw_Customers_Tacoma
GO

IF EXISTS (SELECT *
             FROM sys.tables
            WHERE object_id = OBJECT_ID(N'Customers'))
  DROP TABLE Customers
GO

CREATE TABLE Customers
(
    ID      INT IDENTITY(1,1) NOT NULL
  , name    VARCHAR(100)      NOT NULL
  , City    VARCHAR(50)       NOT NULL
  , status  CHAR(1)               NULL
)
GO

INSERT Customers( name, city ) VALUES( 'Name 1', 'Tacoma' )
INSERT Customers( name, city ) VALUES( 'Name 2', 'Salt Lake City' )
INSERT Customers( name, city ) VALUES( 'Name 3', 'Tacoma' )
INSERT Customers( name, city ) VALUES( 'Name 4', 'Salt Lake City' )
INSERT Customers( name, city ) VALUES( 'Name 5', 'Tacoma' )
go

CREATE VIEW vw_Customers_Tacoma
WITH schemabinding
AS
SELECT id
     , NAME
     , STATUS
     , City
  FROM dbo.Customers
 WHERE city = 'Tacoma'
  WITH CHECK OPTION
GO

--Will be successfull
update vw_Customers_Tacoma set
       status = 'M'
GO

--Will error out
update vw_Customers_Tacoma set
```

```
       city = 'Salt Lake City'
GO

--Will be successfull
update Customers set
       city = 'Salt Lake City'
 where city = 'Tacoma'
GO

SELECT *
  FROM vw_Customers_Tacoma

SELECT *
  FROM Customers
```

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> ]
```