Views, Stored Procedures and User Defined Functions

- Views
  - Create
  - Use
  - Alter
  - View Definition
- Stored Procedures
  - Create
  - Use
  - Alter
  - View Definition
- Scalar Functions
  - Create
  - Use
  - Alter
  - View Definition
- Drop objects
  - View
  - Procedures
  - Functions
- The GO command

Views

- A View is a virtual table based on the result set of an SQL query
- You assign a name to a view and reference it the same way you would a table
- The code in a view can include multiple tables and columns as well as a WHERE clause
- Views are stored in the database and make complex code reusable

```
CREATE VIEW [dbo].[ArtistAlbum_v] AS
SELECT
    A.Name AS ArtistName
    ,AL.Title AS AlbumTitle
    ,A.ArtistId
    ,AL.AlbumId
FROM Artist A
JOIN Album AL
    ON A.ArtistId = AL.ArtistId
```

```
SELECT *
FROM ArtistAlbum_v
```

Results | Messages

	ArtistName	AlbumTitle	ArtistId	AlbumId
1	AC/DC	Let There Be Rock	1	4
2	AC/DC	For Those About To Rock We Salute You	1	1
3	Accept	Balls to the Wall	2	2
4	Accept	Restless and Wild	2	3
5	Aerosmith	Big Ones	3	5
6	Alanis Morissette	Jagged Little Pill	4	6
7	Alice In Chains	Facelift	5	7
8	Antônio Carlos Jobim	Warner 25 Anos	6	8

Create View

- Use the CREATE VIEW keywords to create a view
- Follow the keywords with the view name and the AS keyword
  - CREATE VIEW [Name of View] AS
- A select statement follows the AS keyword
- Only a single statement is allowed in the CREATE VIEW

```
CREATE VIEW ArtistAlbum_v AS
SELECT
    A.Name AS ArtistName
    ,AL.Title AS AlbumTitle
    ,A.ArtistId
    ,AL.AlbumId
FROM Artist A
JOIN Album AL
    ON A.ArtistId = AL.ArtistId
```

Using a View

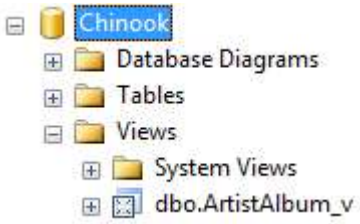
- Treat a view like you would any other table
- The columns you used in the view are available to you when writing new statements

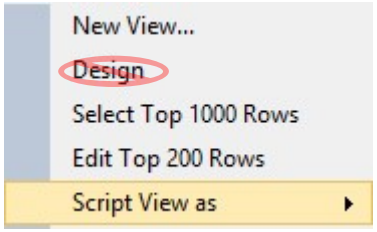
```
SELECT
    AA.*
    ,T.Name AS TrackName
FROM ArtistAlbum_v AA
JOIN Track T
    ON T.AlbumId = AA.AlbumId
WHERE AA.ArtistName = 'Alice In Chains'
```

	ArtistName	AlbumTitle	ArtistId	AlbumId	TrackName
1	Alice In Chains	Facelift	5	7	We Die Young
2	Alice In Chains	Facelift	5	7	Man In The Box
3	Alice In Chains	Facelift	5	7	Sea Of Sorrow
4	Alice In Chains	Facelift	5	7	Bleed The Freak
5	Alice In Chains	Facelift	5	7	I Can't Remember
6	Alice In Chains	Facelift	5	7	Love, Hate, Love
7	Alice In Chains	Facelift	5	7	It Ain't Like That
8	Alice In Chains	Facelift	5	7	Sunshine
9	Alice In Chains	Facelift	5	7	Put You Down
10	Alice In Chains	Facelift	5	7	Confusion
11	Alice In Chains	Facelift	5	7	I Know Somethin (Bout You)
12	Alice In Chains	Facelift	5	7	Real Thing

View Definition

- Information on a view is located in the Object Explorer
- Right-click the view name and select “Script View as” to see view code in a query window
- Alternately use sp_helptext to display view script
- I do NOT recommend using the Design option. It is a GUI that does not work well with complex queries.






```
EXEC sp_helptext ArtistAlbum_v
```

Results

```
Text
----------------------------------
CREATE VIEW ArtistAlbum_v AS
SELECT
    A.Name AS ArtistName
    ,AL.Title AS AlbumTitle
FROM Artist A
JOIN Album AL
    ON A.ArtistId = AL.ArtistId
```

```
USE [Chinook]
GO

/****** Object:  View [dbo].[ArtistAlbum_v]
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

ALTER VIEW [dbo].[ArtistAlbum_v] AS
SELECT
    A.Name AS ArtistName
    ,AL.Title AS AlbumTitle
FROM Artist A
JOIN Album AL
    ON A.ArtistId = AL.ArtistId

GO
```

ALTER VIEW

- Use the ALTER VIEW keywords to alter an existing view
- Follow the keywords with the view name and the AS keyword
  - ALTER VIEW [Name of View] AS
- The statement that follows the AS keyword will overwrite the previous view statement

```
ALTER VIEW ArtistAlbum_v AS
SELECT
     A.Name AS ArtistName
     ,AL.Title AS AlbumTitle
     ,T.Name AS TrackName
FROM Artist A
JOIN Album AL
     ON A.ArtistId = AL.ArtistId
JOIN Track T
     ON T.AlbumId = AL.AlbumId
```

```
SELECT *
FROM ArtistAlbum_v
WHERE ArtistName = 'AC/DC'
ORDER BY TrackName
```

	ArtistName	AlbumTitle	TrackName
1	AC/DC	Let There Be Rock	Bad Boy Boogie
2	AC/DC	For Those About To Rock We Salute You	Breaking The Rules
3	AC/DC	For Those About To Rock We Salute You	C.O.D.
4	AC/DC	Let There Be Rock	Dog Eat Dog
5	AC/DC	For Those About To Rock We Salute You	Evil Walks

ORDER BY Restriction

- You cannot use ORDER BY in a view unless you include the TOP keyword with your select statement

```
ALTER VIEW ArtistAlbum_v AS
SELECT
    A.Name AS ArtistName
    ,AL.Title AS AlbumTitle
FROM Artist A
JOIN Album AL
    ON A.ArtistId = AL.ArtistId
ORDER BY ArtistName
```

Messages

```
Msg 1033, Level 15, State 1, Procedure ArtistAlbum_v, Line 9
The ORDER BY clause is invalid in views, inline functions, de
```

```
ALTER VIEW ArtistAlbum_v AS
SELECT TOP 100 PERCENT
    A.Name AS ArtistName
    ,AL.Title AS AlbumTitle
FROM Artist A
JOIN Album AL
    ON A.ArtistId = AL.ArtistId
ORDER BY ArtistName
```

Messages

```
Command(s) completed successfully.
```

Stored Procedures

- A set of SQL statements stored under an assigned name
- Multiple statements can be assigned to a single stored procedure
- Both DDL (create, alter, drop) and DML (select, insert, update, delete) statements can be entered into a procedure
- Stored procedures are run using the EXECUTE or EXEC keyword followed by the procedure name

```
CREATE PROC CustomerAndEmployee_p AS
SELECT TOP 5
    CustomerId, FirstName, LastName
FROM Customer
SELECT
    EmployeeId, FirstName, LastName
FROM Employee
```

```
EXEC CustomerAndEmployee_p
```

Results | Messages

	CustomerId	FirstName	LastName
1	1	Luís	Gonçalves
2	2	Leonie	Köhler
3	3	François	Tremblay
4	4	Bjørn	Hansen
5	5	František	Wichterlová

	EmployeeId	FirstName	LastName
1	1	Andrew	Adams
2	2	Nancy	Edwards
3	3	Jane	Peacock
4	4	Margaret	Park

Benefits of Stored Procedures

- Encapsulated: Only need to write the code once
- Optimized: SQL Server optimizes the execution strategy after the first run so subsequent runs will be more efficient
- Security: You can provide access only to the procedure instead of the underlying tables. Users can be denied permission to see the underlying code in a procedure.
- Usability: Coding languages like Java and C# can use procedures to read data from and push data to a SQL Server
- Flexibility: Procedures can accept parameters as input allowing you to use the same procedure for different purposes

```
CREATE PROC Customer_p AS
SELECT
    FirstName
    ,LastName
    ,Country
FROM Customer
WHERE Country = 'Canada'

GO

EXEC Customer_p
```

	FirstName	LastName	Country
1	François	Tremblay	Canada
2	Mark	Philips	Canada
3	Jennifer	Peterson	Canada
4	Robert	Brown	Canada
5	Edward	Francis	Canada

Create Stored Procedure

- Syntax is similar to that for creating a view
- Uses the CREATE PROC (or PROCEDURE) keywords and the AS keyword
- Additional input for parameters is optional

```
CREATE PROC Customer_p AS
SELECT
    FirstName
    ,LastName
    ,Country
FROM Customer
WHERE Country = 'Canada'

GO

EXEC Customer_p
```

	FirstName	LastName	Country
1	François	Tremblay	Canada
2	Mark	Philips	Canada
3	Jennifer	Peterson	Canada
4	Robert	Brown	Canada
5	Edward	Francis	Canada

Stored Procedure Parameters

- You can include parameters in a procedure
- Declare the parameter after the procedure name and include a datatype for the parameter
- All parameter names must start with an ampersand (@)
- The value of the parameter will be applied wherever the parameter exists in the code
- Parameters cannot be used to replace database objects such as columns or tables

```
CREATE PROC Customer_p @Country varchar(50) AS
SELECT
    FirstName
    ,LastName
    ,Country
FROM Customer
WHERE Country = @Country

GO

EXEC Customer_p 'Germany'
```

	FirstName	LastName	Country
1	Leonie	Köhler	Germany
2	Hannah	Schneider	Germany
3	Fynn	Zimmermann	Germany
4	Niklas	Schröder	Germany

Parameter Options

- You can have multiple parameters if you separate them with a comma
- Parameters are required unless you include a default option
- Parameters need to be declared in order unless you include the parameter name

```
CREATE PROC ArtistGenre_p
    @ArtistName varchar(50)
    ,@GenreName varchar(50) = 'Rock' AS
SELECT
    A.Name AS ArtistName
    ,T.Name AS TrackName
    ,G.Name AS GenreName
FROM Artist A
JOIN Album AL ON AL.ArtistId = A.ArtistId
JOIN Track T ON T.AlbumId = AL.AlbumId
JOIN Genre G ON G.GenreId = T.GenreId
WHERE A.Name = @ArtistName
    AND G.Name = @GenreName
```

```
EXEC ArtistGenre_p 'U2'
```

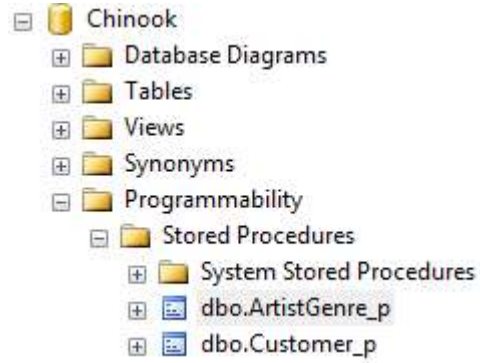
	ArtistName	TrackName	GenreName
1	U2	Zoo Station	Rock
2	U2	Even Better Than The Real Thing	Rock

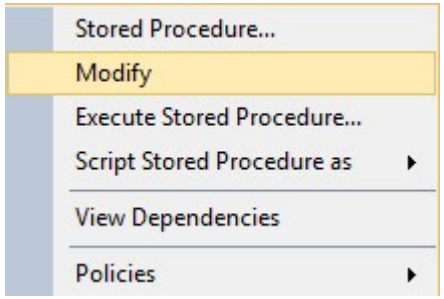
```
EXEC ArtistGenre_p @ArtistName='U2', @GenreName='pop'
```

	ArtistName	TrackName	GenreName
1	U2	Instant Karma	Pop
2	U2	#9 Dream	Pop
3	U2	Mother	Pop

Procedure Definition

- Information on a stored procedure is located in the Object Explorer
- Right-click the procedure name and select “Modify” to open the code in an alterable state
- Alternately use sp_helptext to display the procedure script






```
EXEC sp_helptext 'ArtistGenre_p'
```

Results

```
Text
----------------------------------------
CREATE PROC ArtistGenre_p
    @ArtistName varchar(50)
    ,@GenreName varchar(50) = 'Rock' AS
SELECT
    A.Name AS ArtistName
    ,T.Name AS TrackName
    ,G.Name AS GenreName
FROM Artist A
JOIN Album AL ON AL.ArtistId = A.ArtistId
JOIN Track T ON T.AlbumId = AL.AlbumId
JOIN Genre G ON G.GenreId = T.GenreId
WHERE A.Name = @ArtistName
    AND G.Name = @GenreName
```

```
USE [Chinook]
GO
/****** Object:  StoredProcedure [dbo].[Art
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[ArtistGenre_p]
    @ArtistName varchar(50)
    ,@GenreName varchar(50) = 'Rock' AS
SELECT
    A.Name AS ArtistName
    ,T.Name AS TrackName
    ,G.Name AS GenreName
FROM Artist A
JOIN Album AL ON AL.ArtistId = A.ArtistId
JOIN Track T ON T.AlbumId = AL.AlbumId
JOIN Genre G ON G.GenreId = T.GenreId
WHERE A.Name = @ArtistName
    AND G.Name = @GenreName
```

ALTER Stored Procedure

- Use the ALTER PROC (or ALTER PROCEDURE) keywords to alter an existing stored procedure
- After the keywords include the procedure name, any parameters you wish to include and the AS keyword
  - ALTER PROC [Name of Procedure] AS
- The statement that follows the AS keyword will overwrite the previous procedure statement

```
ALTER PROC [dbo].[ArtistGenre_p]
    @ArtistName varchar(50)
    ,@GenreName varchar(50) = 'Rock' AS
SELECT
    A.Name AS ArtistName
    ,T.Name AS TrackName
    ,G.Name AS GenreName
FROM Artist A
JOIN Album AL ON AL.ArtistId = A.ArtistId
JOIN Track T ON T.AlbumId = AL.AlbumId
JOIN Genre G ON G.GenreId = T.GenreId
WHERE A.Name = @ArtistName
    AND G.Name = @GenreName
```

User Defined Scalar Functions

- Functions are objects that receive input, perform an internal action with that input, and return a result
- User Defined Functions are built by the user as opposed to built-in functions which are part of the SQL Server engine
- Scalar functions are a type of function that only return a single value

```
CREATE FUNCTION DayOfBirth_fn (@date date)
RETURNS varchar(10)
AS
BEGIN

RETURN
    DATENAME(WEEKDAY,@date)
END
```

```
SELECT
    BirthDate
    ,dbo.DayOfBirth_fn(BirthDate) AS DayOfBirth
FROM Employee
```

	BirthDate	DayOfBirth
1	1962-02-18 00:00:00.000	Sunday
2	1958-12-08 00:00:00.000	Monday
3	1973-08-29 00:00:00.000	Wednesday
4	1947-09-19 00:00:00.000	Friday
5	1965-03-03 00:00:00.000	Wednesday
6	1973-07-01 00:00:00.000	Sunday
7	1970-05-29 00:00:00.000	Friday
8	1968-01-09 00:00:00.000	Tuesday

Create User Defined Scalar Function

- Start with CREATE FUNCTION keywords and the function name
- Include the parameter(s) name and datatype
- The returned value datatype must be defined using the RETURNS keyword
- Include the AS Keyword
- The function code must be enclosed with the BEGIN and END keywords
- You define value returned using the RETURN keyword
- Only a single value can be returned by a scalar function

```
CREATE FUNCTION DayOfBirth_fn (@date date)
RETURNS varchar(10)
AS
BEGIN
RETURN DATENAME(WEEKDAY,@date)
END
```

Using a User Defined Scalar Function

- Use it the same way you would a built in scalar function
- Must include schema name before the function name
- Schema name is dbo by default

```
SELECT
    BirthDate
    ,dbo.DayOfBirth_fn(BirthDate)
        AS DayOfBirth
FROM Employee
```

Results | Messages

	BirthDate	DayOfBirth
1	1962-02-18 00:00:00.000	Sunday
2	1958-12-08 00:00:00.000	Monday
3	1973-08-29 00:00:00.000	Wednesday
4	1947-09-19 00:00:00.000	Friday
5	1965-03-03 00:00:00.000	Wednesday
6	1973-07-01 00:00:00.000	Sunday
7	1970-05-29 00:00:00.000	Friday
8	1968-01-09 00:00:00.000	Tuesday

User Defined Scalar Function with a SELECT Statement and a Separate Return Variable

- Scalar Functions support SELECT statements
- Only a single value can be returned from the function
- Declare a variable in the function to capture the value
- Set that variable as what is returned from the function

```
CREATE FUNCTION Supervisor_fn (@ReportsTo int)
RETURNS varchar(50)
AS
BEGIN
DECLARE @Supervisor varchar(50)
SELECT
    @Supervisor = CONCAT(FirstName,' ',LastName)
FROM Employee
WHERE EmployeeId = @ReportsTo
RETURN
    @Supervisor
END
```

Using the 2nd UDF Example

- The dbo.Supervisor_fn scalar function returns the name of the employee's supervisor
- The function saves you the trouble of having to write the self-join and concatenation code

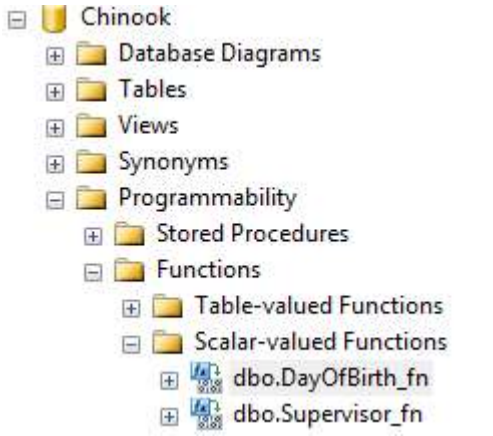
```
SELECT
    EmployeeId
    ,FirstName
    ,LastName
    ,ReportsTo
    ,dbo.Supervisor_fn(ReportsTo)
       AS Supervisor
FROM Employee
```

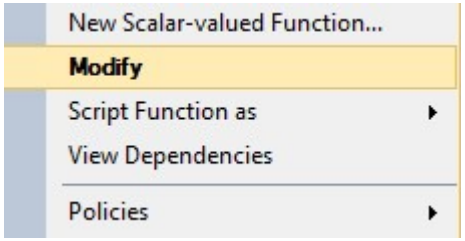
Results | Messages

	EmployeeId	FirstName	LastName	ReportsTo	Supervisor
1	1	Andrew	Adams	NULL	NULL
2	2	Nancy	Edwards	1	Andrew Adams
3	3	Jane	Peacock	2	Nancy Edwards
4	4	Margaret	Park	2	Nancy Edwards
5	5	Steve	Johnson	2	Nancy Edwards
6	6	Michael	Mitchell	1	Andrew Adams
7	7	Robert	King	6	Michael Mitchell
8	8	Laura	Callahan	6	Michael Mitchell

Function Definition

- Information on a scalar function is located in the Object Explorer
- Right-click the function name and select “Modify” to open the code in an alterable state
- Alternately use sp_helptext to display the function script






```
EXEC sp_helptext 'dbo.DayOfBirth_fn'
```

Results

```
Text
----------------------------------

CREATE FUNCTION DayOfBirth_fn
    (@date date)
RETURNS varchar(10)
AS
BEGIN
RETURN DATENAME(WEEKDAY,@date)
END
```

```
USE [Chinook]
GO
/****** Object:  UserDefinedFunction
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER FUNCTION [dbo].[DayOfBirth_fn]
    (@date date)
RETURNS varchar(10)
AS
BEGIN
RETURN DATENAME(WEEKDAY,@date)
END
```

ALTER Scalar Function

- Use the ALTER FUNCTION keywords to alter an existing function
- Other than replacing CREATE with ALTER, the ALTER FUNCTION syntax is identical to the CREATE FUNCTION syntax
- Any previous code in the function name will be overwritten with the new code

```
ALTER FUNCTION [dbo].[Supervisor_fn]
    (@ReportsTo int)
RETURNS varchar(50)
AS
BEGIN
DECLARE @Supervisor varchar(50)
SELECT
    @Supervisor =
    CONCAT(FirstName,' ',LastName)
FROM Employee
WHERE EmployeeId = @ReportsTo
RETURN
    @Supervisor
END
```

Dropping Objects

- Views, Stored Procedures and Functions are all removed using the same DROP keyword
- After the Drop keyword enter the object type keyword then the object name

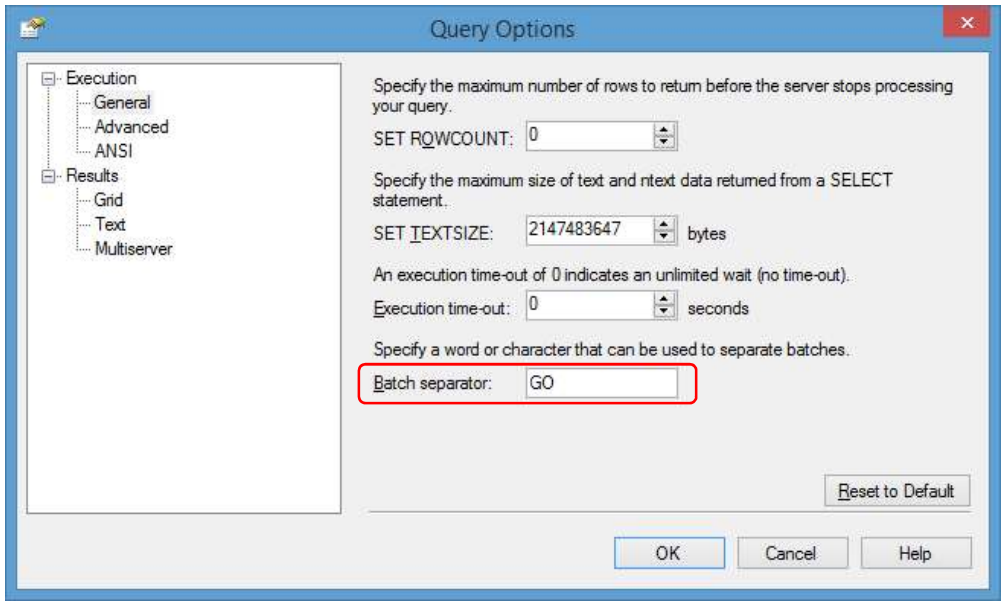
```
DROP VIEW ArtistAlbum_v

DROP PROCEDURE ArtistGenre_p

DROP FUNCTION dbo.Supervisor_fn
```

The GO Command

- The GO command is used to separate SQL batches that are sent to the server
- There are some SQL commands that cannot be run in the same batch (e.g. DROP and CREATE)
- The GO command isn't a SQL statement, but a command recognized by several MS utilities including SQL Server Management Studio
- You have the option to change the command keyword if you wish




```
DROP FUNCTION DayOfBirth_fn

GO

CREATE FUNCTION DayOfBirth_fn (@date date)
RETURNS varchar(10)
```

Summary

- Views
  - Create
  - Use
  - Alter
  - View Definition
- Stored Procedures
  - Create
  - Use
  - Alter
  - View Definition
- Scalar Functions
  - Create
  - Use
  - Alter
  - View Definition
- Drop objects
  - View
  - Procedures
  - Functions
- The GO command