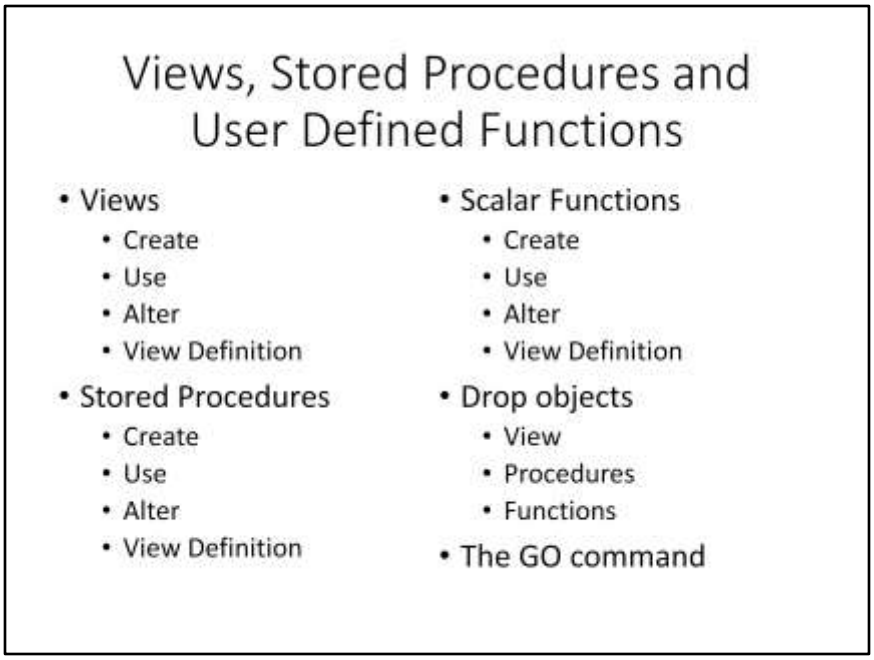


This presentation will cover views, stored procedures and user defined scalar functions. I will talk about how these objects are created, as well as how they are used with in a database.

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

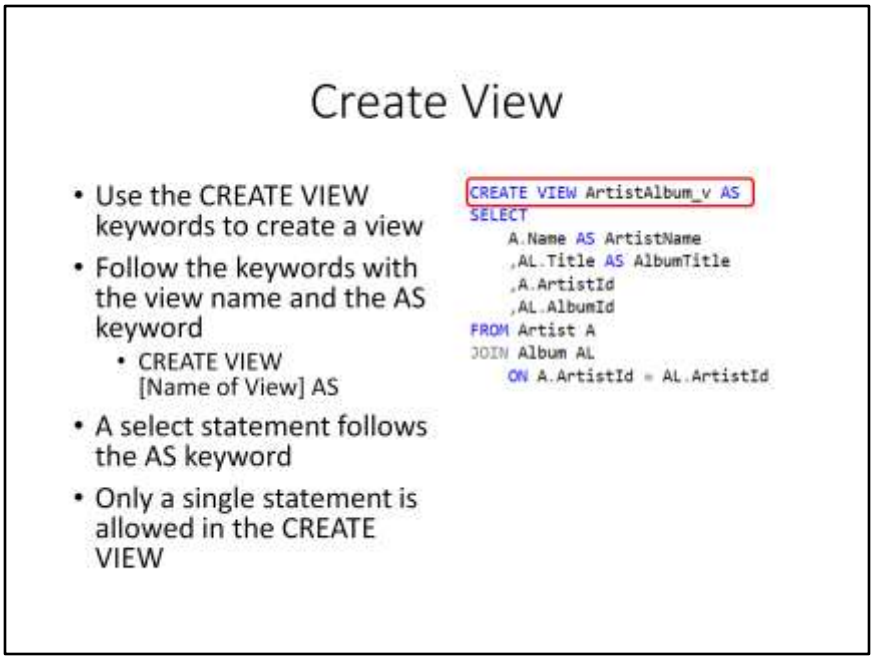
```
CREATE VIEW [dbo].[ArtistAlbum_v] AS
SELECT
    A.Name AS ArtistName
    ,AL.Title AS AlbumTitle
    ,A.ArtistId
    ,AL.AlbumId
FROM Artist A
JOIN Album AL
    ON A.ArtistId = AL.ArtistId
```

```
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

A view is a virtual table based on a SQL statement you provide it. You can insert multiple tables, WHERE clauses, and complex column expressions into a view. Views are beneficial because you only need to write your select statement once and assign your view a name. Afterwards you can reference the view in the same way you would a regular table.



The syntax for creating a view is relatively simple. You write the CREATE VIEW keywords, the name you want to assign the view, and the AS keyword on a single line. Your SELECT statement follows. You are only allowed to have a single SELECT statement in a view. Other data manipulation language statements such as INSERT and UPDATE are also not allowed.

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

Once your view is created, you can use it like a normal table in your select statements. In the example the Artist Album view is joined to the Track table. When the select statement is executed all the columns in the view are displayed as well as the TrackName column of the Track table.

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.

You can manage your views by using the Object Explorer. Expand the views folder and right-click on the view you wish you edit. Then select Script View As, ALTER To, then New Query Editor Window. This will open a new tab with the script of the view ready to be altered. Another option when you right click on the view is the Design option. I do not recommend using Design. It displays a GUI version of your view script, similar to what you would see in an Access database. Design does not work well with complex SELECT statements, and you're not writing SQL when you use it. If you want to see your view's script using SQL, you can execute the sp_helptext stored procedure followed by the view name. You may want to change your result output to text view.

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
```

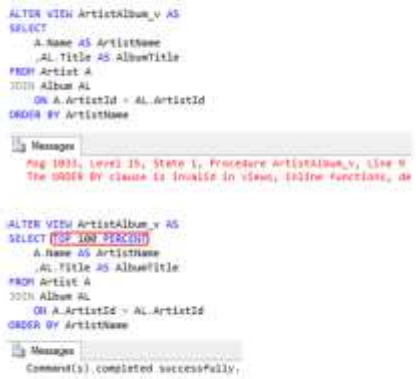
```
SELECT *
FROM ArtistAlbum_v
WHERE ArtistName = 'AC/DC'
ORDER BY TrackName
```

	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

Altering a view is pretty simple. You start with the ALTER VIEW keywords followed by the View name and the AS keyword. The select statement you write after the AS keyword will replace the previous statement stored in the database. The ALTER VIEW syntax is set up for you automatically when you use the Object Explorer to alter a view.

ORDER BY Restriction

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



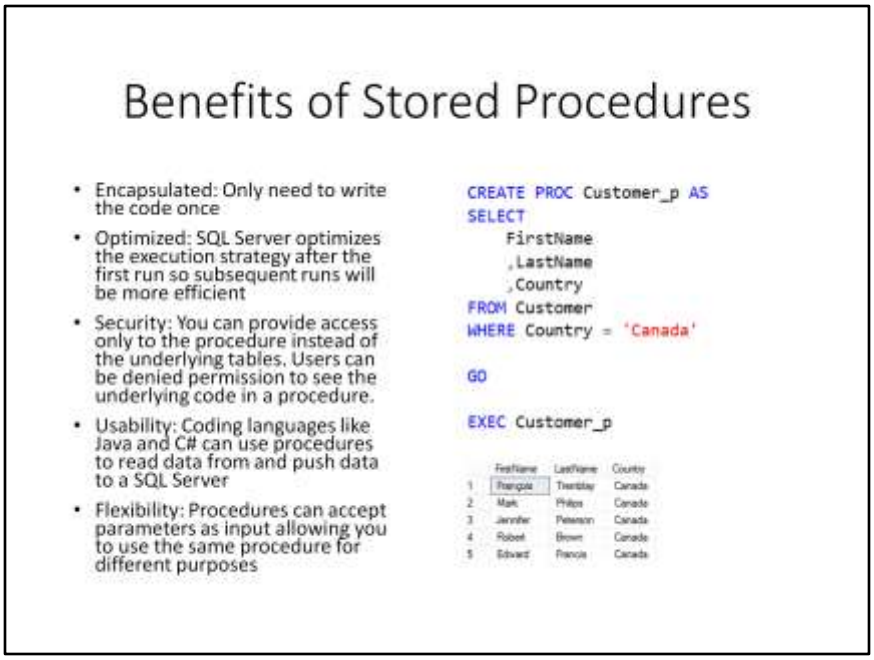
One of the restrictions with a View is you cannot insert an ORDER BY clause into it without a little extra work. I suppose the thinking here was that you can always add an ORDER BY to the SELECT statement that is running the view. In any event you are allowed you place an ORDER BY clause in a view as long as you include the TOP keyword at the beginning of the SELECT clause. In the example I used SELECT TOP 100 PERCENT which will return all the rows and still allow for an ORDER BY clause.

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
```

A stored procedure is a set of SQL statements that are stored under an assigned name. It differs from a view in that it cannot be inserted in the FROM clause of a SELECT statement. Instead you run a stored procedure by typing the EXECUTE command followed by the procedure name. Stored procedures are more flexible than views. You can add DML statements such as INSERT, UPDATE and DELETE, and you can also add DDL statements like CREATE, ALTER and DROP. A stored procedure can house multiple statements.



Stored procedures are one of the most useful objects in SQL Server. For starters it allows you to encapsulate complex code into a single procedure statement. When run for the first time SQL Server will remember the optimal execution strategy for subsequent runs. With a stored procedure you can lock out users from messing with the underlying script. This is particularly relevant with data-driven websites. By encapsulating your code, you also make it more usable for those who may not be familiar with SQL. Executing a stored procedure can be easier to learn than writing the code. Finally stored procedures can be built to accept one or more parameters.

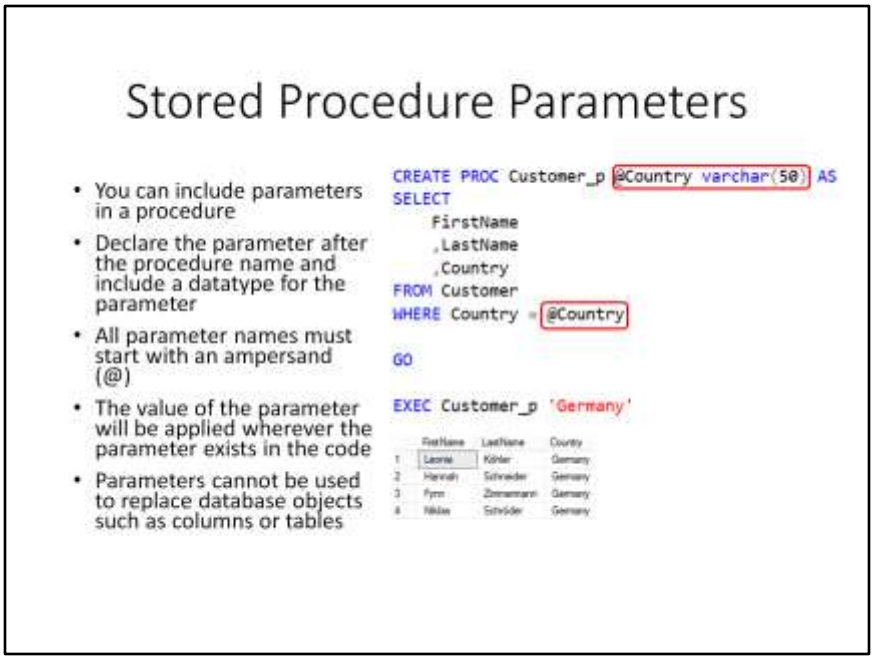
Create Stored Procedure

- 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

The syntax for creating a stored procedure is similar to that for creating a view. You start with the CREATE PROCEDURE keywords followed by the procedure name and the AS keyword. Anything that follows the AS keyword will be part of the stored procedure. One of the differences between view and procedure creation is you have the option to add parameters when creating a procedure.



You add parameters by entering the parameter name and its datatype after the procedure name in the CREATE statement. The names for all parameters in SQL must start with an ampersand. After declaring your parameters you can add them to your SQL statement. The value you enter for a parameter will be the one used in the SQL statement. In the example the customer_p stored procedure has a parameter named @Country. When "Germany" is entered as the parameter value, then the WHERE clause within the procedure will filter on "Germany". Note that you are not allowed to substitute table and column names with parameters.

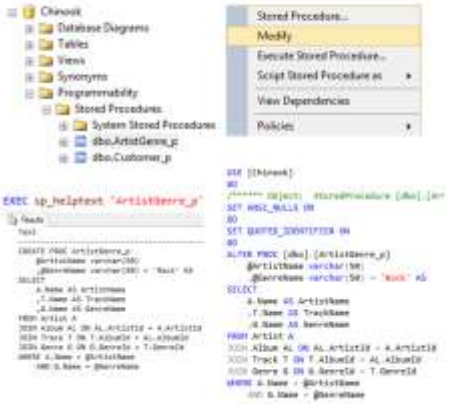
Parameter Options

- 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

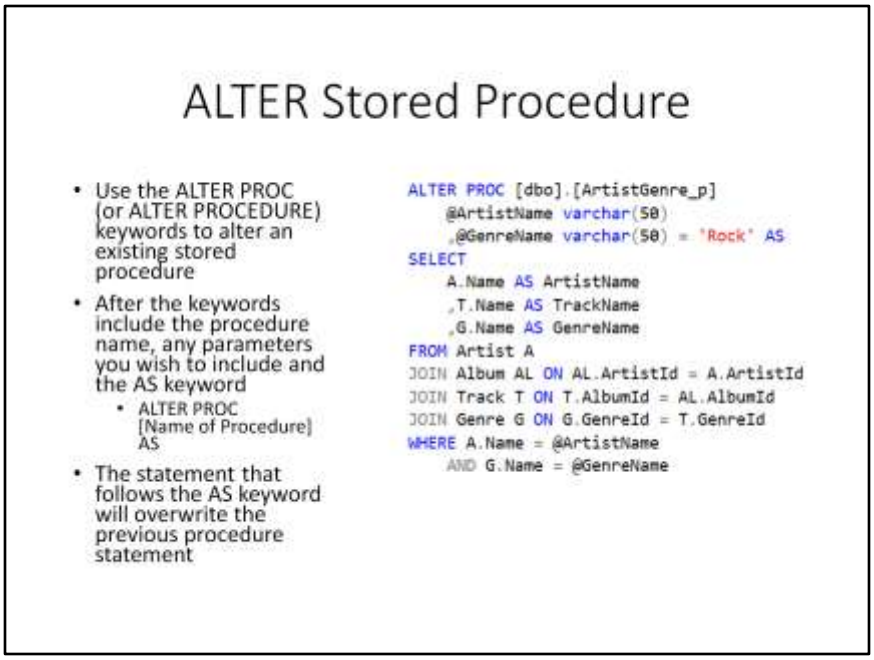
If you have more than one parameter then they need to be separated by commas. Values are required for all parameters. However you have the option of declaring a default value for one or more of your parameters. If a parameter has a default value, you are not required to assign it a value when executing the procedure, instead the default value will be used. The parameter values must be entered in the order they were created in the stored procedure. However you can include the parameter names when executing your procedure. If you do so then the parameters can be in any order. In the first example I only included “U2” as a value for the Artist name. Since I didn’t provide a value for the Genre name the default of “Rock” was used in the WHERE clause. In the second example I explicitly provided values for both parameters. Therefore rows with “U2” and “Pop” were returned instead.

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



You can manage your stored procedures by using the Object Explorer. Expand the Programmability folder, then the Stored Procedure folder, and right-click on the procedure you wish you edit. Select the Modify option from the menu. This will open a new tab with the script of the stored procedure ready to be altered. You can also use the sp_helptext stored procedure in the same way you used it for a view. Execute the sp_helptext stored procedure followed by the procedure name, and you will see the stored procedure script in the result set. You may want to change your result output to text view for easier viewing.



The syntax for altering a stored procedure is almost the same as that for altering a view. You enter the ALTER PROCEDURE keywords followed by the procedure name. Any parameters you have will come after the procedure name then followed by the AS keyword. The select statement you write after the AS keyword will replace the previous statement stored in the database. The ALTER PROCEDURE syntax is set up for you automatically when you use the Object Explorer to alter a procedure.

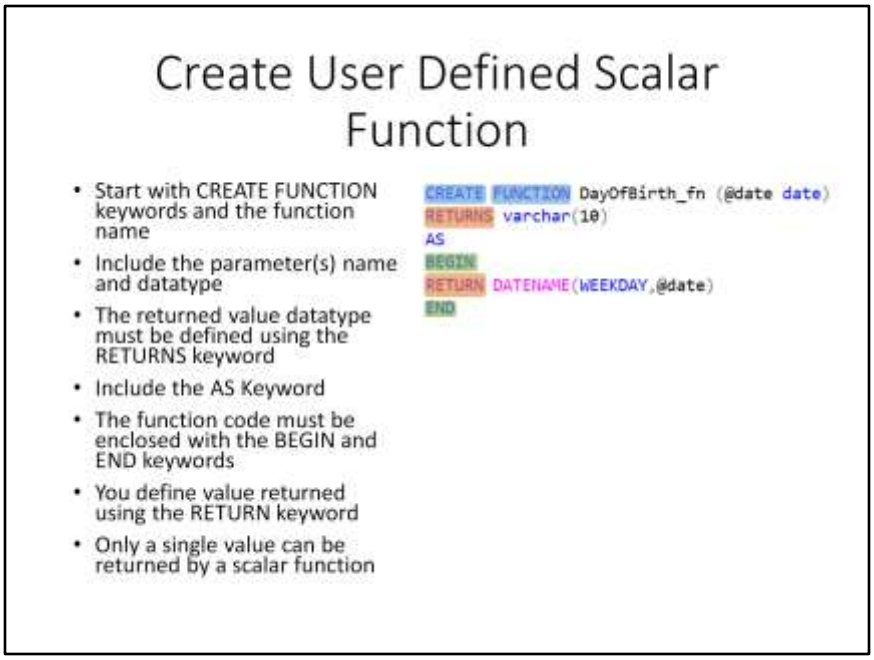
User Defined Scalar Functions

- 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

A user defined scalar function is very similar to the built-in scalar functions I covered earlier in the course. The main difference is you build the function yourself. User defined functions are useful when you have a specific modification you wish to display for a column that isn't covered by any of the built-in functions provided by SQL Server. Remember scalar functions are only allowed to return a single value per row.



Creating a user defined scalar function is similar to creating a stored procedure. You start with the CREATE FUNCTION keywords followed by the function name and any parameters with their datatypes enclosed in parenthesis. All functions require a return value. You first have to declare the datatype of the return value, and you do this by typing the RETURNS keyword followed by the datatype. You then enter the AS keyword. After the AS keyword you write your script, however functions require that the script be enclosed between the BEGIN and END keywords. Finally you must define the value to be returned with the RETURN keyword.

In the example I am creating a function called DayOfBirth_fn with a parameter of @date. The value returned will be a varchar datatype. I take the @date parameter and insert it inside a DATENAME function. I then tell my function to return the result of the DATENAME function.

Using a User Defined Scalar Function

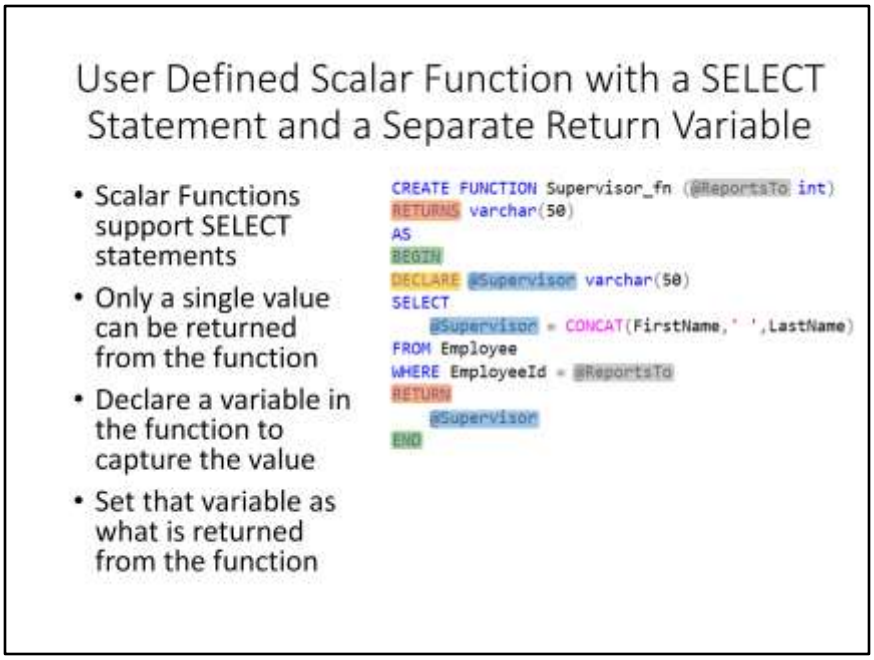
- Use it the same way you would a built in scalar function
- Must include schema name before the function name
- Schema name is dbo by default

```
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

When using a user defined scalar function, you call the function the same way you would call a built-in scalar function. The only difference is that SQL Server requires that you include the schema name with the function name. The schema name is "dbo" by default. In the example I applied the DayOfBirth_fn function to the column BirthDate, and it returned the day of the week that the birthdate fell on.



Scalar functions can support SELECT statements as long as you only return a single value. When working with SELECT statements you will probably have to declare a separate variable to store the result in. In the example I created a scalar function that takes the parameter @ReportsTo as an integer and returns the first and last name of that employee's supervisor. In the function I declared a new variable called @Supervisor which will be the value returned by the function. I then set the @Supervisor variable equal to the column that concatenates the first and last name. The SELECT statement is filtered using the value input into the @ReportsTo variable. I then tell the function to return the value of the @Supervisor variable. If this SELECT statement returned more than one row, only the last row's value would be returned.

Using the 2nd UDF Example

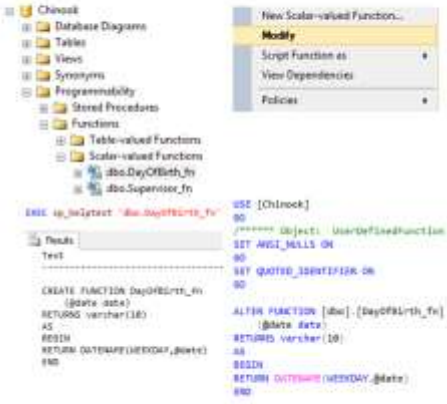
- The dbo.Supervisor_fn scalar function returns the name of the employee's supervisor
- The function saves you the trouble of having to write the self-join and concatenation code

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

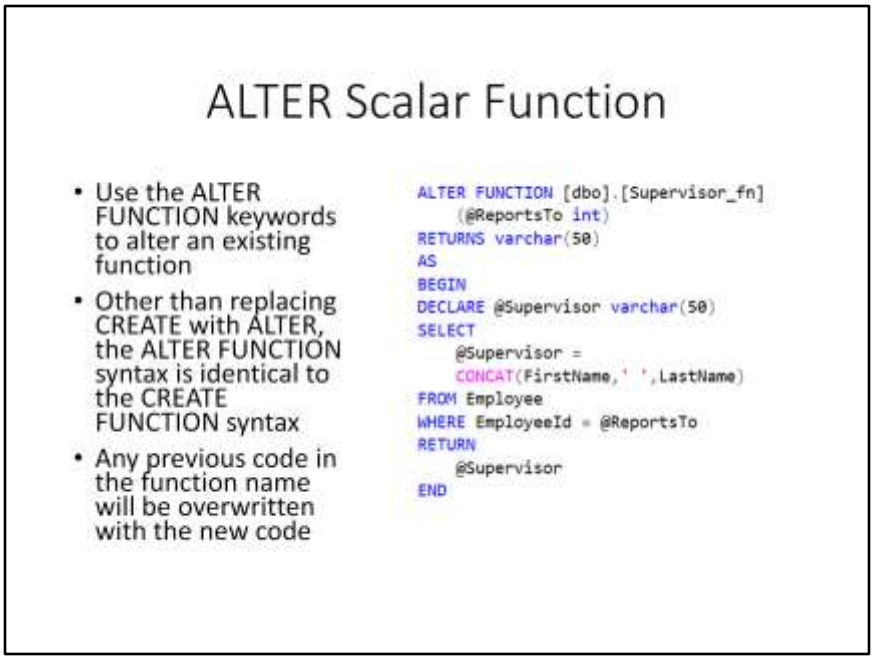
In this example I take the Supervisor_fn function and apply it to the ReportsTo column in the Employee table. The returned result of the function is the first and last name of that employee's supervisor. This example shows the practical application of a user defined scalar function. If I didn't have this function I would have needed to add a self join and concatenation function to the SELECT statement.

Function Definition

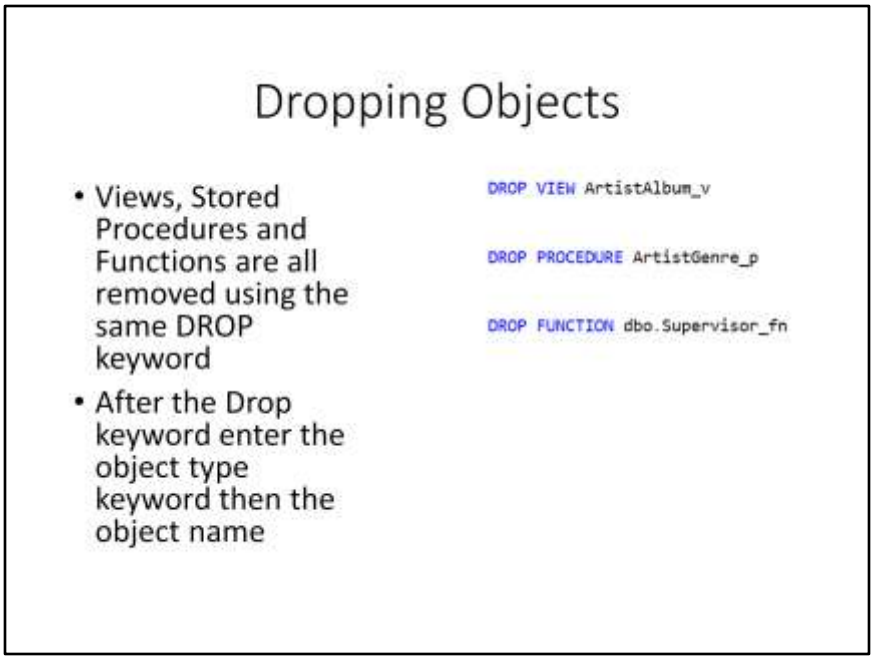
- Information on a scalar function is located in the Object Explorer
- Right-click the function name and select "Modify" to open the code in an alterable state
- Alternately use sp_helptext to display the function script



Viewing a function definition is almost identical to the steps for viewing a stored procedure. You can manage your scalar functions in the Object Explorer. First expand the Programmability folder, then the Functions folder, the Scalar-valued Functions folder, and then right-click on the function you wish you edit. Select the Modify option from the menu. This will open a new tab with the script of the function ready to be altered. You can also use the sp_helptext stored procedure. Execute the sp_helptext stored procedure followed by the function name, and you will see the function script in the result set.



The syntax for altering a scalar function is similar to that for a view and a stored procedure. You enter the ALTER FUNCTION keywords followed by the function name. The code you write after the AS keyword will replace the previous code stored in the database. The ALTER FUNCTION syntax is set up for you automatically when you use the Object Explorer to alter a function.



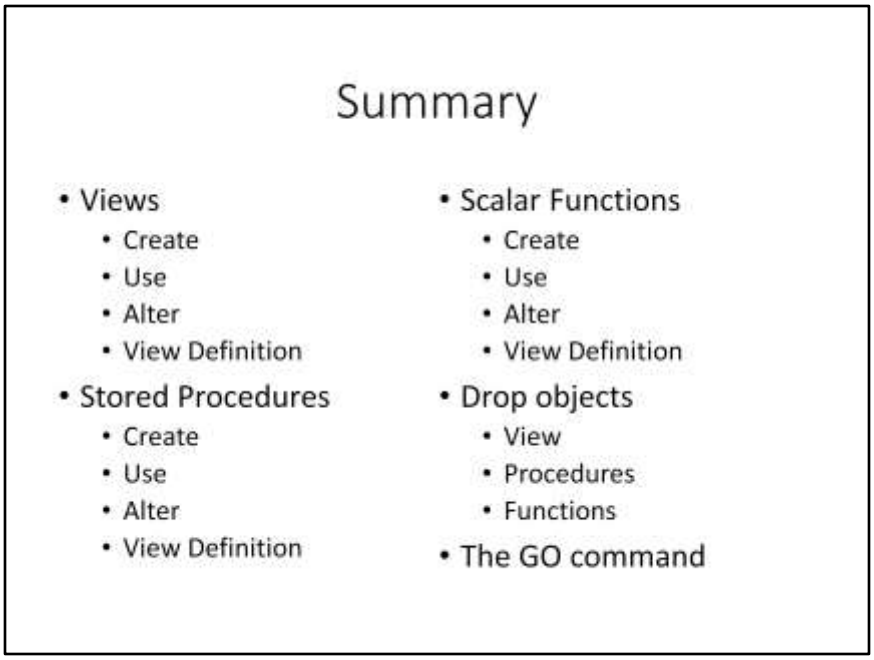
Dropping views, procedures and functions is simple in SQL Server. You start with the DROP keyword and follow it with the type of object you wish to drop. For example DROP VIEW, DROP PROCEDURE, or DROP FUNCTION. After the keywords you enter the name of the object. Then you are ready to execute the command.

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)
```

When you execute the code on the tab of your query window, all of the code gets sent to SQL Server as a single batch. However there are some commands that cannot run in the same batch together, especially on the Data Definition Language side of SQL. To get around potential syntax errors you can separate your code into batches by using the GO command. GO isn't a SQL statement but rather a command recognized by management studio. By using GO your script will be separated into batches that execute independently of one another.



This concludes the presentation on Views, Procedures and Functions.