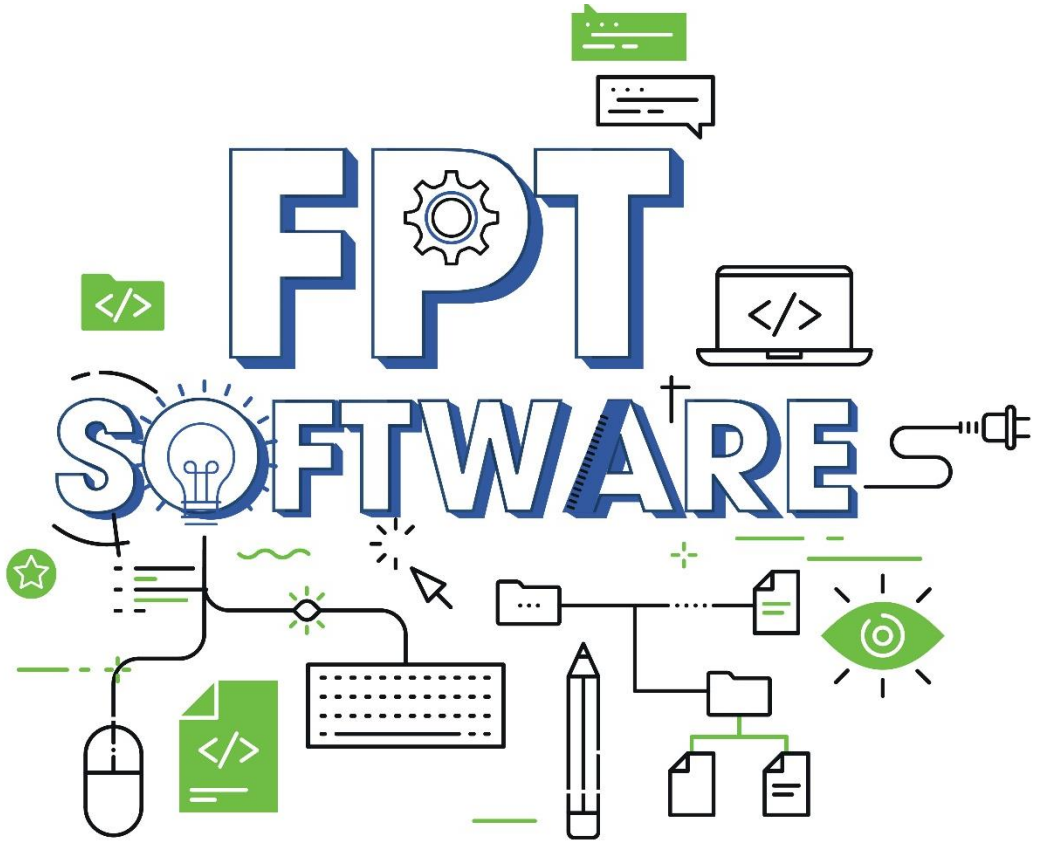
User Stored Procedures and User-Defined Functions



Fsoft Academy

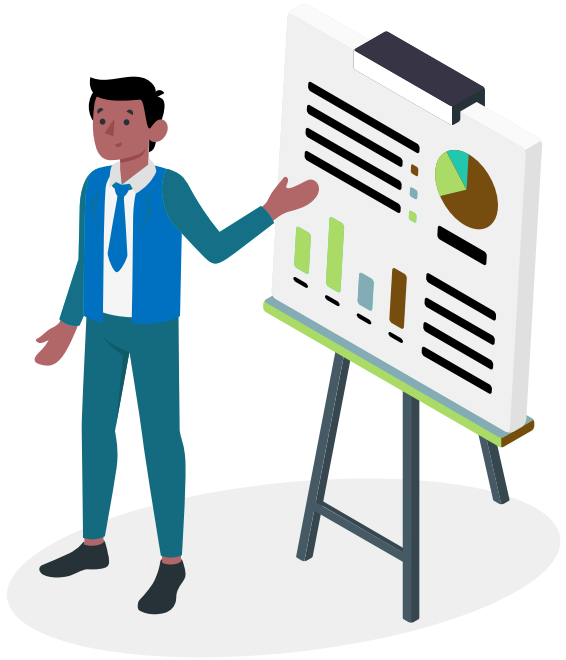
Lesson Objectives

01 Understand about USP, UDF and **SQL Code Practice** in SQL Server.

02 Using smoothly them and apply to project.

Agenda

1. Basic of programming SQL
2. User Defined Functions
3. User Stored Procedures
3. Demo



Section 1

Basic of Programming SQL

Comments

- Indicate user-provided text
  - Double Dash:
    - **Ex**: SELECT * FROM Orders -- This is a comment
  - Block Comment:
    - **Ex**: /*Multi-line comments here*/

Identifiers

❖**The database object name is referred to as its identifier.**

- ✓ An object identifier is created when the object is defined
- ✓ The identifier is used to reference the object

❖**There are 2 types of Identifiers:**

- ✓ *Regular Identifiers:*
  - Example: Orders, Customers, Employee…
- ✓ *Delimited Identifiers:* Are enclosed in double quotation marks (") or brackets ([ ])
  - [My Table]
  - [1Person]
- ✓ *For Example*:

```
SELECT * FROM [My Table]
WHERE [Order]=40
```

Variables

❖Declare a variable

Must be DECLARE and start with @ symbol

```
DECLARE @limit money
DECLARE @min_range int, @hi_range int
```

❖ Assign a value into a variable using SET

```
SET @min_range = 0, @hi_range = 100
SET @limit = $10
```

❖ Assign a value into a variable using SELECT

```
SELECT @price = price FROM titles
WHERE  title_id = 'PC2091'
```

Control-of-flow

- **The T-SQL control-of-flow language keywords are:**
  - ✓BEGIN…END
  - ✓IF…ELSE
  - ✓CASE … WHEN
  - ✓TRY…CATCH
  - ✓WHILE
  - ✓BREAK / CONTINUE
  - ✓GOTO
  - ✓RETURN

Control-of-flow/BEGIN…END

- **BEGIN…END**
  - ✓Define a statement block
  - ✓Other Programming Languages:
    - C#, Java, C: **{...}**
    - Pascal, Delphi: BEGIN … END

Control-of-flow/IF…ELSE

- **IF…ELSE**
  - ✓ Define conditional and, optionally, alternate execution when a condition is false
  - ✓ **Syntax:**

```
IF Boolean_expression
        SQL_statement|block_of_statements
[ELSE
        SQL_statement|block_of_statements]
```

  - ✓ **Example:**

```
USE AdventureWorks2022;
GO

IF (SELECT COUNT(*) FROM Production.Product WHERE Name LIKE 'Touring-3000%' ) > 5
        PRINT 'There are more than 5 Touring-3000 bicycles.'
ELSE
        PRINT 'There are 5 or less Touring-3000 bicycles.' ; GO
```

Control-of-Flow

- **CASE … WHEN**
  - ✓ Evaluate a list of conditions and returns one of multiple possible result expressions
  - ✓ **Syntax:**

```
CASE input_expression
        WHEN when_expression1 THEN result_expression1
        WHEN when_expression2 THEN result_expression2
        ...
        ELSE else_result_expression
END
```

Control-of-Flow

- **CASE … WHEN**

```
USE AdventureWorks2022;

GO

SELECT ProductNumber,
        Category = CASE ProductLine
                WHEN 'R' THEN 'Road'
                WHEN 'M' THEN 'Mountain'
                WHEN 'T' THEN 'Touring'
                WHEN 'S' THEN 'Other sale items'
                ELSE 'Not for sale' END,
        Name
FROM Production.Product
ORDER BY ProductNumber;
```

Control-of-flow

- **TRY… CATCH**
  - ✓ Provide error handling for T-SQL that is similar to the exception handling in the C# / Java
  - ✓ **Syntax:**

```
BEGIN TRY
                { sql_statement | statement_block }
END TRY
BEGIN CATCH
                [ { sql_statement | statement_block } ]
END CATCH
```

Control-of-flow

- **WHILE**
  - ✓Set a condition for the repeated execution of an statement block
  - ✓The statements are executed repeatedly as long as the specified condition is true
  - ✓The execution of statements in the WHILE loop can be controlled from inside the loop with the BREAK and CONTINUE keywords
  - ✓**Syntax**

```
WHILE Boolean_expression
                { sql_statement | statement_block | BREAK |
CONTINUE }
```

BREAK / CONTINUE

- The **WHILE** statement **repeats a statement or block of statements** as long as a specified condition remains true.
- Two Transact-SQL statements are commonly used with WHILE: BREAK or CONTINUE.
  - ✓ The BREAK statement **exits the innermost** WHILE loop and the CONTINUE statement **restarts a WHILE loop**.
  - ✓ A program might execute a BREAK statement if, for example, there are no other rows to process. A CONTINUE statement could be executed if, for example, the execution of the code should continue.
- **Example**:

```
USE AdventureWorks2022;


WHILE (SELECT AVG(ListPrice) FROM Production.Product) < $300
BEGIN
   UPDATE Production.Product SET ListPrice = ListPrice * 2

   SELECT MAX(ListPrice) FROM Production.Product
   IF (SELECT MAX(ListPrice) FROM Production.Product) > $500
      BREAK
   ELSE
      CONTINUE
END
PRINT 'Too much for the market to bear';
```

Control-of-flow

- **GOTO**
  - ✓ Alter the flow of execution to a label. The Transact-SQL statement or statements that follow **GOTO** are skipped and processing continues at the label
  - ✓ **Syntax:**

```
--Define the label
label :
--Alter the execution:
GOTO label
```

Control-of-flow

- **RETURN**
  - ✓ Exit unconditionally from a query or procedure
  - ✓ This will be discussed more details in Stored Procedure section.
  - ✓ **Syntax**

```
RETURN [ integer_expression ]
```

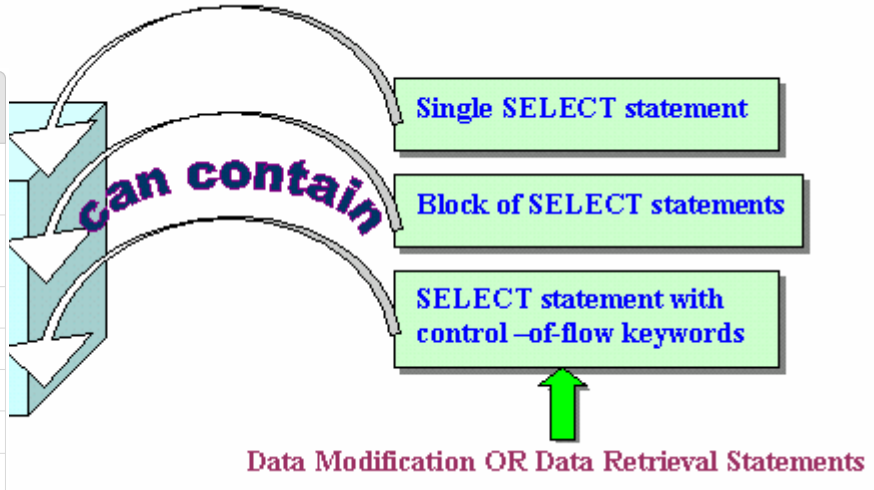
Section 2

Stored Procedures

Overview

- A stored procedure (SP) is a **collection of SQL statements** that SQL Server compiles into a single execution plan.
- It can accept input parameters, return output values as parameters, or return success or failure status messages.

Tiêu chí	Stored Procedure (SP)	Function (UDF - User Defined Function)
Trả về giá trị	Có thể trả về hoặc không	Luôn trả về giá trị
Dùng trong `SELECT`	❌ Không thể dùng trực tiếp	✅ Có thể gọi trong `SELECT`
Gọi trong `FROM`	❌ Không thể dùng	✅ Có thể dùng như bảng
Chứa `INSERT`, `UPDATE`, `DELETE`	✅ Có thể	❌ Không thể
Chứa `TRY...CATCH` để xử lý lỗi	✅ Có thể	❌ Không thể
Dùng `EXEC` để gọi	✅ Có thể	❌ Không thể
Dùng trong JOIN	❌ Không thể	✅ Có thể
Có thể tạo Transaction (`BEGIN TRANSACTION` ... `COMMIT`)?	✅ Có thể	❌ Không thể



Types of Stored Procedures

- **User-defined**
  - ✓ **A user-defined procedure** can be created in a user-defined database or in all system databases except the **Resource** database.
- **Temporary**
  - ✓ **Temporary procedures** are a form of user-defined procedures. The temporary procedures are like a permanent procedure, except temporary procedures are stored in **tempdb**.
  - ✓ There are **two types** of temporary procedures: local and global. They differ from each other in their names, their visibility, and their availability.

Types of Stored Procedures

- **System**
  - ✓ System procedures are included with SQL Server.
  - ✓ They are physically stored in the internal, hidden **Resource** database and logically appear in the **sys** schema of every system- and user-defined database.

- Because **system procedures** start with the prefix sp_, we recommend that you **do not use** this prefix when naming user-defined procedures.
- In this topic we will learn about User Stored Procedure.

Create a SP- Syntax

- **Basic Syntax to Create a SP:**

```
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
        -- Add the parameters for the stored procedure here
        <@Param1, sysname, @p1> <Datatype_For_Param1, , int> = <Default_Value_For_Param1, , 0>,
        <@Param2, sysname, @p2> <Datatype_For_Param2, , int> = <Default_Value_For_Param2, , 0>
AS
BEGIN
        -- SET NOCOUNT ON added to prevent extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;
        -- Insert statements for procedure here

        SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>

END

GO
```

User Stored Procedures

- **Stored procedures return data in four ways:**
  - ✓ ***Output parameters***, which can return either data (such as an integer or character value) or a cursor variable (cursors are result sets that can be retrieved one row at a time).
  - ✓ *Example 1*:

```
USE AdventureWorks2022;

GO
CREATE PROCEDURE GetImmediateManager
        @employeeID INT,
        @managerID INT OUTPUT
AS
BEGIN
        SELECT @managerID = ManagerID FROM HumanResources.Employee
        WHERE EmployeeID = @employeeID
END
```

User Stored Procedures

- *Example 2:*

```
USE AdventureWorks2022;
GO

CREATE PROCEDURE HumanResources.uspGetEmployeesTest
        @LastName nvarchar(50),
        @FirstName nvarchar(50)
AS
        SET NOCOUNT ON;

        SELECT FirstName, LastName, Department
        FROM HumanResources.vEmployeeDepartmentHistory
        WHERE FirstName = @FirstName AND LastName = @LastName AND EndDate IS NULL;
GO
```

- *Run the procedure:*

```
EXECUTE/EXEC HumanResources.uspGetEmployeesTest N'Ackerman', N'Pilar';
```

User Stored Procedures

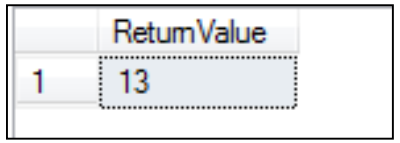
- **Stored procedures return data in four ways:**
  - ✓ *Return codes*, which are always an integer value.
  - ✓ *Example*:

```
CREATE PROC dbo.TestReturn (@InValue int)
AS
        Return @Invalue + 10
GO

-- Call SP
DECLARE @ReturnValue INT

EXEC @ReturnValue = dbo.TestReturn 3
SELECT ReturnValue=@ReturnValue
```

  - ✓ *Result*:



	ReturnValue
1	13

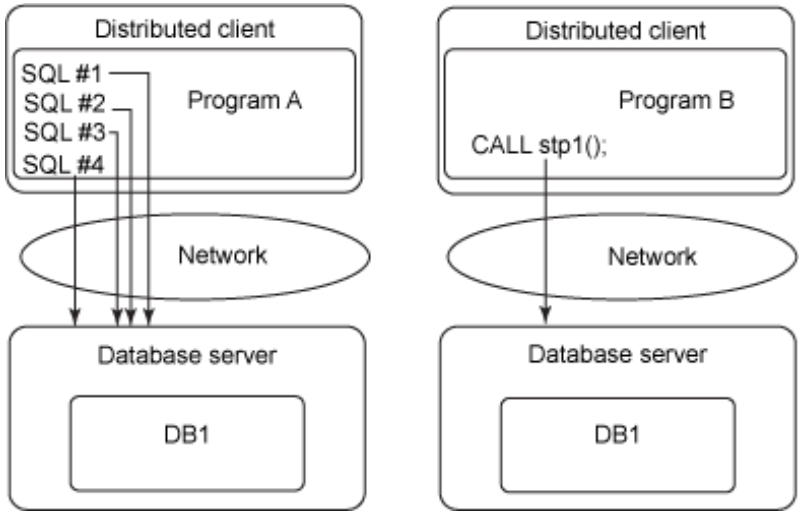
User Stored Procedures

❖ **Stored procedures return data in four ways:**

- ✓A result set for each SELECT statement contained in the stored procedure or any other stored procedures called by the stored procedure.
- ✓A global cursor that can be referenced outside the stored procedure.

Benefit of Using SP

- **Benefit of Using SP**
  - ✓*Reduced server/client network traffic:* Only the call to execute the procedure is sent across the network



  - ✓ *Stronger security:* When calling a procedure over the network, only the call to execute the procedure is visible. Therefore, malicious users cannot see table and database object names, embed Transact-SQL statements of their own, or search for critical data

Benefit of Using SP

- **Benefit of Using SP**

✓*Reuse of code:*

- The code for any repetitious database operation is the perfect candidate for encapsulation in procedure (for instance, UPDATE data on a table)

✓*Improve Performance:*

- Procedure is stored in cache area of memory when the stored procedure is first executed so that it can be used repeatedly. SQL Server does not have to recompile it every time the stored procedure is run.

Stored Procedures vs. SQL Statement

SQL Statement	*Stored Procedure*
	Creating - *Check syntax* - *Compile*
First Time - *Check syntax* - *Compile* - *Execute* - *Return data*	**First Time** - *Execute* - *Return data*
Second Time - *Check syntax* - *Compile* - *Execute* - *Return data*	**Second Time** - *Execute* - *Return data*

Exec, Update, Delete a SP

- **Execute a Procedure:**

  EXEC[UTE] procedure_name

- **Update a Procedure**

```
ALTER PROC[EDURE] procedure_name
        [ @parameter_name data_type] [= default] [OUTPUT]
        [,...,n]
AS
SQL_statement(s)
```

- **Delete a Procedure**

  DROP PROC[EDURE] procedure_name

Stored Procedure Disadvantages

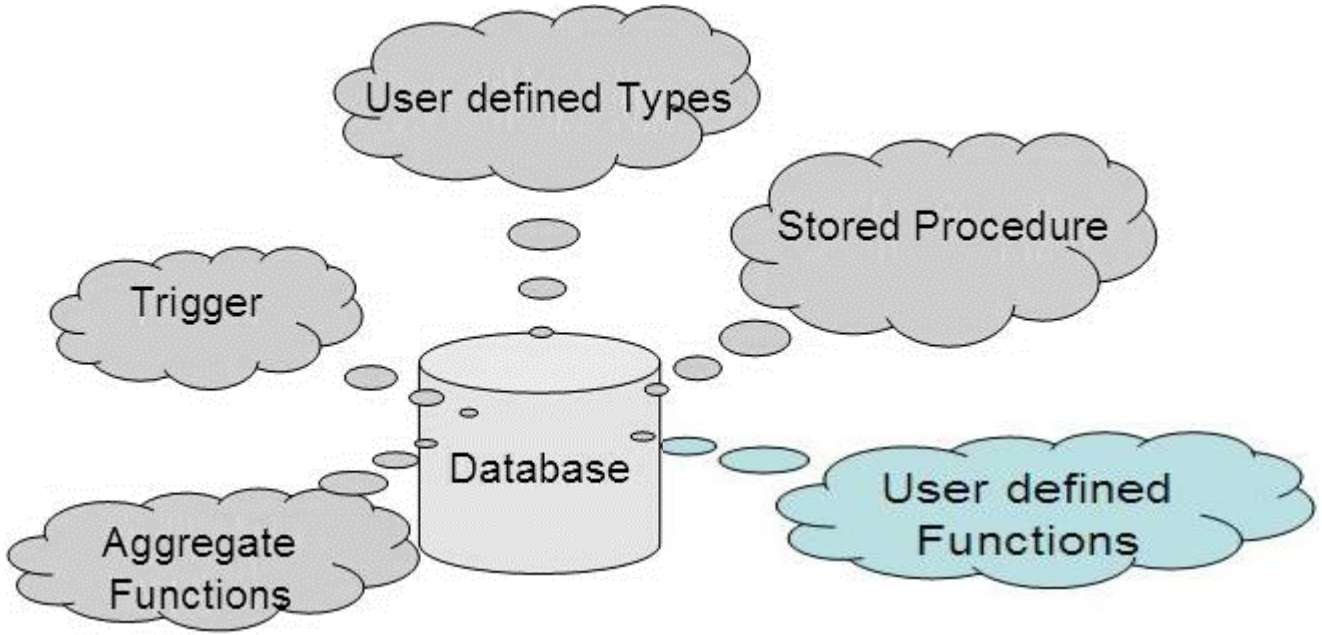
- Make the database server high load in both memory and processors
- Difficult to write a procedure with complexity of business logic
- Difficult to debug
- Not easy to write and maintain

Section 3

User-Defined Functions

What is a UDF?

- User-Defined Function (UDF) are routines that accept parameters, perform an action and return the result of that action as a value.
  - ✓ The return value can be a single scalar value or a result set
  - ✓ Functions are computed values and cannot perform permanent environmental changes to SQL Server (i.e. no INSERT or UPDATE statements allowed).



Benefits of User-Defined Functions

Why use user-defined functions (UDFs)?

- **Modular programming:**
  - ✓ You can create the function once, store it in the database, and call it any number of times in your program.
  - ✓ User-defined functions can be modified independently of the program source code.
- **Faster execution**
  - ✓ Similar to stored procedures, Transact-SQL user-defined functions reduce the compilation cost of Transact-SQL code by **caching the plans** and **reusing** them **for repeated executions**.
  - ✓ This means the **user-defined function doesn't need to be reparsed** and **reoptimized** with each use resulting in much faster execution times.

Benefits of User-Defined Functions

Why use user-defined functions (UDFs)?

- **Reduce network traffic.**
  - ✓ An operation that filters data based on some complex constraint that can't be expressed in a single scalar expression can be expressed as a function.
  - ✓ The function can then be invoked in the WHERE clause to **reduce the number of rows sent to the client**.

!Important

Transact-SQL UDFs in queries can only be executed on a single thread (serial execution plan).

UDF's types

- **Scalar User-Defined Function:** can accept zezo to many input parameter and will return a single value:
  - A Scalar user-defined function returns one of the scalar (int, char, varchar etc) data types
  - Text, ntext, image, cursor and timestamp data types are not supported
- **Table-valued functions**
  - *Inline Table-valued Functions*: returns a variable of data type table whose value is derived from a single SELECT statement
  - *Multi-statement Table-Valued Functions*: returns a table

UDF Function

- **UDF Scalar Function Syntax**

```
CREATE FUNCTION [ schema_name. ] function_name
        ( [ { @parameter_name data_type [ = default ] [ READONLY ] } [ ,...n ] ] )

RETURNS return_data_type
[ AS ]
BEGIN
        function_body
        RETURN scalar_expression
END
```

UDF Function

- **UDF Scalar Function Example**

```
CREATE FUNCTION sales.udfNetSale(
    @quantity INT,
    @list_price DEC(10,2),
    @discount DEC(4,2)
)
RETURNS DEC(10,2)
AS
BEGIN
    RETURN @quantity * @list_price * (1 - @discount);
END;
```

sales.order_items

- * order_id
- * item_id
- product_id
- quantity
- list_price
- discount

- **Calling a scalar function**

```
SELECT sales.udfNetSale(10,100,0.1) net_sale;
```

- **Output:**

net_sale
900.00

UDF Function

- **UDF Inline Table-valued Functions Syntax**

```
CREATE FUNCTION [schema_name.]function_name
                    ( [ { @parameter_name data_type  [ = default ] } [ ,...n ] ] )
RETURNS TABLE
[ WITH < function_option > [ ,...n ] ]
[ AS ]
        RETURN [ ( ] select_statement [ ) ]
```

UDF Function

- **UDF Inline Table-valued Functions Example**

```
CREATE FUNCTION [dbo].[udfGetProductList](
        @SafetyStockLevel SMALLINT
)
RETURNS TABLE
AS
RETURN
    (SELECT Product.ProductID,
            Product.Name,
            Product.ProductNumber
     FROM Production.Product
     WHERE SafetyStockLevel >= @SafetyStockLevel)
```

- **Run the function:**

```
SELECT * FROM dbo.udfGetProductList( 100 )
```

Product table:

Results | Messages

	ProductID	Name	ProductNumber
1	1	Black Tire	AR-5381
2	2	Bearing Ball	BA-8327
3	3	BB Ball Bearing	BE-2349
4	4	Headset Ball Bearings	BE-2908
5	316	Blade	BL-2036
6	317	LL Crankarm	CA-5965
7	318	ML Crankarm	CA-6738
8	319	HL Crankarm	CA-7457
9	320	Chainring Bolts	CB-2903
10	321	Chainring Nut	CN-6137
11	322	Chainring	CR-7833
12	323	Crown Race	CR-9981
13	324	Chain Stays	CS-2812
14	325	Decal 1	DC-8732
15	326	Decal 2	DC-9824
16	327	Down Tube	DT-2377
17	328	Mountain End Caps	EC-M092
18	329	Road End Caps	EC-R098
19	330	Touring End Caps	EC-T209
20	331	Fork End	FE-3760
21	332	Freewheel	FH-2981
22	341	Flat Washer 1	FW-1000
23	342	Flat Washer 6	FW-1200
24	343	Flat Washer 2	FW-1400
25	344	Flat Washer 9	FW-3400
26	345	Flat Washer 4	FW-3800

UDF Functions Syntax

- **UDF Multi-statement Table-Valued Functions:**
  - ✓A multi-statement table-valued function or MSTVF is a table-valued function that **returns the result of multiple statements**.
  - ✓The multi-statement-table-valued function is **very useful** because you **can execute multiple queries** within the function and aggregate results into the returned table.

```
CREATE FUNCTION [ schema_name. ] function_name
(( [ { @parameter_name data_type [ = default ] [ READONLY ] } [ ,...n ] ] ))

RETURNS @return_variable TABLE <table_type_definition>
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
BEGIN
        function_body
        RETURN
END
```

UDF Functions Syntax

- **UDF Multi-statement Table-Valued Functions Example:**

```
CREATE FUNCTION udf_Contacts()
    RETURNS @contacts TABLE (
        first_name VARCHAR(50),
        last_name VARCHAR(50),
        email VARCHAR(255),
        phone VARCHAR(25),
        contact_type VARCHAR(20)    )
AS
BEGIN
    INSERT INTO @contacts
    SELECT first_name, last_name, email, phone,'Staff'
    FROM sales.staffs;

    INSERT INTO @contacts
    SELECT first_name, last_name, email, phone,'Customer'
    FROM sales.customers;
    RETURN;
END;
```

UDF Functions Syntax

- **Execute a multi-statement table-valued function:**

```
SELECT * FROM udfContacts();
```

- **Output:**

first_name	last_name	email	phone	contact_type
Fabiola	Jackson	fabiola.jackson@bikes.shop	(831) 555-5554	Staff
Mireya	Copeland	mireya.copeland@bikes.shop	(831) 555-5555	Staff
Genna	Serrano	genna.serrano@bikes.shop	(831) 555-5556	Staff
Virgie	Wiggins	virgie.wiggins@bikes.shop	(831) 555-5557	Staff
Jannette	David	jannette.david@bikes.shop	(516) 379-4444	Staff
Marcelene	Boyer	marcelene.boyer@bikes.shop	(516) 379-4445	Staff
Venita	Daniel	venita.daniel@bikes.shop	(516) 379-4446	Staff
Kali	Vargas	kali.vargas@bikes.shop	(972) 530-5555	Staff
Layla	Terrell	layla.terrell@bikes.shop	(972) 530-5556	Staff
Bernardine	Houston	bernardine.houston@bikes.shop	(972) 530-5557	Staff
Debra	Burks	debra.burks@yahoo.com	NULL	Customer
Kasha	Todd	kasha.todd@yahoo.com	NULL	Customer
Tameka	Fisher	tameka.fisher@aol.com	NULL	Customer
Daryl	Spence	daryl.spence@aol.com	NULL	Customer
Charolette	Rice	charolette.rice@msn.com	(916) 381-6003	Customer
Lyndsey	Bean	lyndsey.bean@hotmail.com	NULL	Customer
Latasha	Hays	latasha.hays@hotmail.com	(716) 986-3359	Customer
Jacquline	Duncan	jacquline.duncan@yahoo.com	NULL	Customer

User-Defined Function Demo

- **Demo**
  - ✓ Scalar function
  - ✓ Inline table function
  - ✓ Multi-statement table-valued function

SUMMARY

- Basic of Programming SQL
- User-Defined Function
  - ✓What is a Function?
  - ✓UDF's Type
  - ✓UDF Function Syntax
- User Stored Procedures

THANK YOU!