**Examples**

1. **Create Square Function**

CREATE FUNCTION dbo.Square ( @Num INT )

RETURNS INT

AS

BEGIN

RETURN @Num \* @Num;

END;

select dbo.square(5)

1. **Create Diffenretn numbers**

CREATE FUNCTION dbo.GetEvenNumbers ( @Start INT, @End INT )

RETURNS TABLE

AS

RETURN (

WITH CTE\_Numbers AS (

SELECT @Start AS Number

UNION ALL

SELECT Number + 1 FROM CTE\_Numbers WHERE Number + 1 <= @End

)

SELECT Number FROM CTE\_Numbers WHERE Number % 2 = 0

);

select \* from dbo.GetEvenNumbers(5,50)

2. Aggregate functions: An aggregate function takes multiple input values and returns a single value, such as the sum or average of those values. Aggregate functions are often used in SELECT statements with GROUP BY clauses to compute summary statistics for groups of rows.

Here is an example of an aggregate function that calculates the median value of a set of input values:

sqlCopy code

CREATE FUNCTION dbo.Median ( @Values VARCHAR(MAX) ) RETURNS DECIMAL(18, 2) AS BEGIN DECLARE @Median DECIMAL(18, 2); SELECT @Median = AVG(Number) FROM ( SELECT ROW\_NUMBER() OVER (ORDER BY CONVERT(DECIMAL(18, 2), Value)) AS RowNum, CONVERT(DECIMAL(18, 2), Value) AS Number FROM STRING\_SPLIT(@Values, ',') ) AS T WHERE T.RowNum = (COUNT(\*) + 1) / 2 OR T.RowNum = COUNT(\*) / 2 + 1; RETURN @Median; END;

These are the three types of user-defined functions in MsSQL. Each type has its own syntax and use cases, depending on the specific needs of your application.

please create userdifiecd function with date time

Sure! Here is an example of a scalar-valued user-defined function that takes a datetime input parameter and returns the number of hours between the input datetime and the current datetime:

lessCopy code

CREATE FUNCTION dbo.HoursSinceDateTime ( @InputDateTime DATETIME ) RETURNS INT AS BEGIN DECLARE @Hours INT; SELECT @Hours = DATEDIFF(HOUR, @InputDateTime, GETDATE()); RETURN @Hours; END;

In this example:

* **dbo.HoursSinceDateTime** is the name of the function, which is prefixed with the schema name **dbo**.
* **@InputDateTime** is the input parameter of the function, which is of type **DATETIME**.
* **RETURNS INT** specifies the data type of the function's return value.
* **AS BEGIN** marks the beginning of the function's body, which contains the T-SQL statements that implement the function's logic.
* **DECLARE @Hours INT;** declares a local variable to hold the result of the calculation.
* **SELECT @Hours = DATEDIFF(HOUR, @InputDateTime, GETDATE());** calculates the number of hours between the input datetime and the current datetime using the **DATEDIFF** function, and stores the result in the **@Hours** variable.
* **RETURN @Hours;** returns the value of the **@Hours** variable as the function's output.

Once the function is created, you can call it from other T-SQL statements, just like any built-in function. Here is an example of calling the **dbo.HoursSinceDateTime** function:

sqlCopy code

SELECT dbo.HoursSinceDateTime('2022-02-28 09:00:00');

This statement returns the number of hours between the input datetime **2022-02-28 09:00:00** and the current datetime, which is the number of hours that have elapsed since that datetime.

create function which take dob of user and create Age of user

Sure, here's an example of a scalar-valued user-defined function in Microsoft SQL Server that takes a date of birth (DOB) as input and calculates the age of the user in years:

lessCopy code

CREATE FUNCTION dbo.CalculateAge ( @DOB DATE ) RETURNS INT AS BEGIN DECLARE @Age INT; SELECT @Age = DATEDIFF(YEAR, @DOB, GETDATE()) - CASE WHEN MONTH(@DOB) > MONTH(GETDATE()) OR (MONTH(@DOB) = MONTH(GETDATE()) AND DAY(@DOB) > DAY(GETDATE())) THEN 1 ELSE 0 END; RETURN @Age; END;

In this example:

* **dbo.CalculateAge** is the name of the function, which is prefixed with the schema name **dbo**.
* **@DOB** is the input parameter of the function, which is of type **DATE**.
* **RETURNS INT** specifies the data type of the function's return value.
* **AS BEGIN** marks the beginning of the function's body, which contains the T-SQL statements that implement the function's logic.
* **DECLARE @Age INT;** declares a local variable to hold the result of the calculation.
* **SELECT @Age = DATEDIFF(YEAR, @DOB, GETDATE()) - ...** calculates the age of the user in years by subtracting the year of the input DOB from the current year, and then adjusting the result by -1 if the user's birthday has not yet occurred in the current year. The calculation is performed using the **DATEDIFF** function and a **CASE** expression.
* **RETURN @Age;** returns the value of the **@Age** variable as the function's output.

Once the function is created, you can call it from other T-SQL statements, just like any built-in function. Here is an example of calling the **dbo.CalculateAge** function:

sqlCopy code

SELECT dbo.CalculateAge('1990-03-15');

This statement returns the age of the user whose DOB is **1990-03-15**, which is the number of full years between that DOB and the current date.