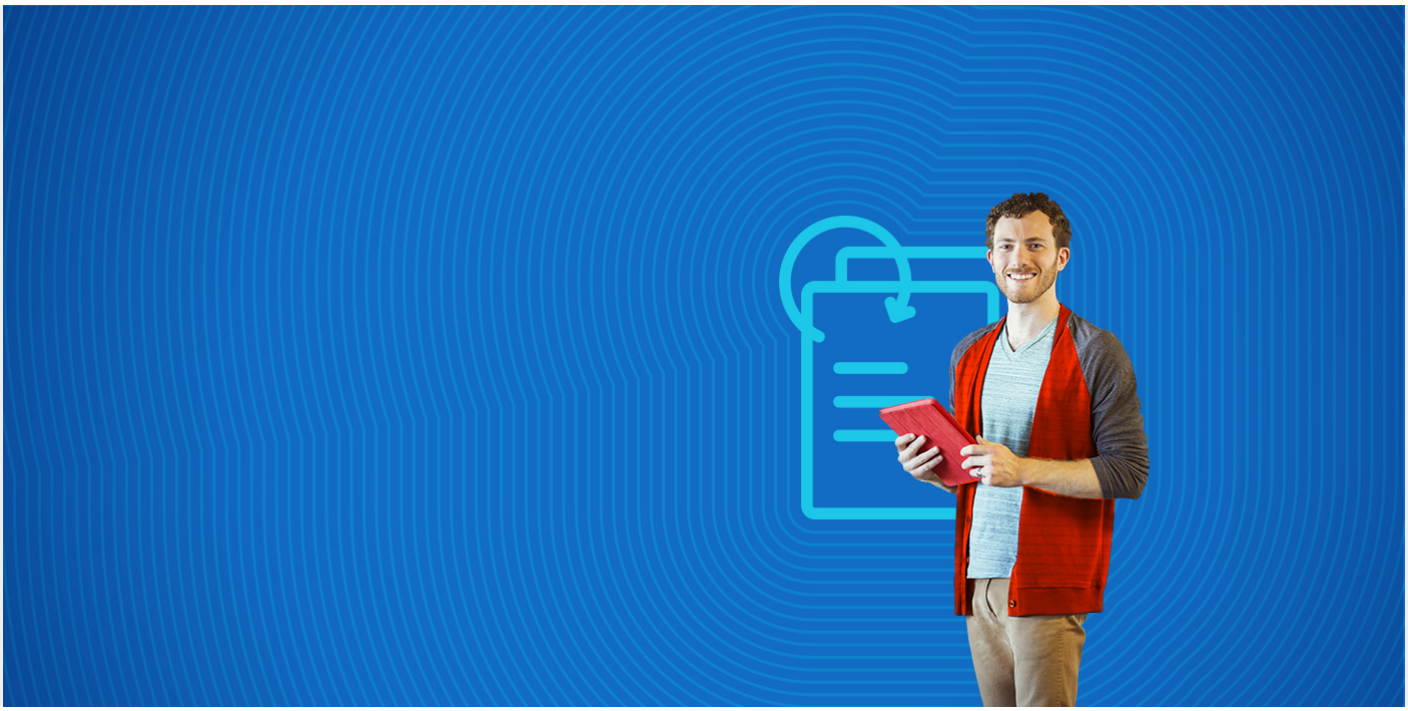
# USER DEFINED FUNCTIONS (UDFS) IN SNOWFLAKE



In this tutorial, we show you how to create user defined functions (UDF) in Snowflake.

In Snowflake, you can create:

- Functions in SQL and JavaScript languages
- Functions that return a single value (scalar)
- Functions that return multiple values (table)

## Create data

If you want to follow the tutorials below, use the instructions from this tutorial on statistical functions to load some data into Snowflake. The data is 41 days of hourly weather data from Paphos, Cyprus.

## Snowflake UDF SQL function

We start with a SQL example.

The code below takes the input weather conditions, described in the table column **main**, and converts that to an integer. This solves a common problem with machine learning: converting categorical data to an integer.

Notice that the function has parameters **(dt varchar(20))** and a return value **(int)**. The rest of it is just a SQL select statement.

The code below uses the **iff()** and **regex()** statement to see whether the word **rain**, **cloud**, etc., is found in the **main** column. It works by adding the numbers from 1 to 9. Since only one of these if

statements will be true, then the sum will be one of the values 1 to 9, thus giving the weather conditions.

```
create or replace function weathercategorical (dt varchar(20) )
  returns int
  as $$select (iff(main regexp '.*Clear.*',1,0) +
      iff(main regexp '.*Clouds.*',2,0) +
     iff(main regexp '.*Rain.*',3,0) +
     iff(main regexp '.*Thunderstorm.*', 4,0) +
     iff(main regexp '.*Mist.*', 5, 0) +
     iff(main regexp '.*Fog.*', 6, 0) +
     iff(main regexp '.*Squall.*',7,0) +
     iff(main regexp '.*Tornado.*', 8, 0) +
     iff(main regexp '.*Haze.*', 9, 0))
        from weather as w where w.dt = dt$$
```

The date and time is in epoch time format. The SQL statement below calls the function **weathercategorical** for the date January 1, 2000, returning the scalar value 1, meaning clear weather.

```
sselect weathercategorical (946684800) from weather where dt = 946684800
```

# Snowflake table function

Here we show how to return more than one value, which Snowflake calls a **table**.

Create these two tables:

```
CREATE TABLE customers
  (
     customernumber     varchar(100) PRIMARY KEY,
    customername varchar(50),
    phonenumber varchar(50),
    postalcode varchar(50),
    locale varchar(10),
    datecreated date,
    email varchar(50)
  );


CREATE TABLE orders
  (
     customernumber    varchar(100) ,
    ordernumber varchar(100) PRIMARY KEY,
    comments varchar(200),
    orderdate date,
    ordertype varchar(10),
    shipdate date,
```

```
discount float,
quantity int,
    productnumber varchar(50)
);
```

Then copy and paste this data.

# Scalar vs table function

Now we create a function to look up the customer name and email given a record from the order table. Orders don't contain customer information, so it's like doing a join. But since it's a function, it's far less wordy and more convenient than creating a join every time you need customer information with the order.

```
create or replace function getcustomer (customernumber number )
returns table (customername varchar, email varchar)
as 'select customername, email from customers
    where customers.customernumber = customernumber';
```

Given the customer number from the orders table, this statement gets:

- The customer's name
- Order number
- Email

```
select  getcustomer (948 ), ordernumber from orders where customernumber =
948;
```

# JavaScript UDFs

You can use JavaScript in a user defined function. Just put **language javascript**.

Let's calculate n factorial (n!) since Snowflake does not have that math function. n factorial n! is n * (n-1) * (n-2) * .. ** (n - (n + 1))). For example: 3!=3*2*1=6.

Notice below that we use **variant** as data type since JavaScript does not have integer types.

```
CREATE OR REPLACE FUNCTION factorial(n variant)
  RETURNS variant
  LANGUAGE JAVASCRIPT
  AS '
     var f=n;
     for (i=n-1; i>0; i--) {
  f=f*i
    }
 return f';
```

Run it and it calculates the value 6.

```
select factorial(3)
```

Note that 33 is the largest number that function can handle. 33! = 8683317618811886495518194401280000000

## Additional resources

For more tutorials like this, explore these resources:

- BMC Machine Learning & Big Data Blog
- How To Import Amazon S3 Data to Snowflake
- Snowflake Window Functions: Partition By and Order By
- Snowflake Lag Function and Moving Averages
- Amazon Braket Quantum Computing: How To Get Started