# Introduction to Data Quality and data metric functions

ENTERPRISE EDITION FEATURE

Data Quality and data metric functions (DMFs) require Enterprise Edition. To inquire about upgrading, please contact Snowflake Support.

Data Quality uses data metric functions (DMFs), which include Snowflake-provided system DMFs and user-defined DMFs, to monitor the state and integrity of your data. You can use DMFs to measure key metrics, such as, but not limited to, freshness and counts that measure duplicates, NULLs, rows, and unique values.

## About Data Quality and DMFs

Data Quality focuses on knowing the state and integrity of your data, which includes data freshness and accuracy with respect to true data values compared to null values or blank fields in a column, to make data-driven decisions. You can measure the quality of your data by using DMFs. Snowflake provides built-in system DMFs in the SNOWFLAKE.CORE schema to measure common metrics without having to define them. You can also define your own custom DMFs to fine-tune your data quality measurements more precisely, and these DMFs are stored in the database and schema of your choice.

Whether you use system DMFs, custom DMFs, or both, after you assign a DMF to a table or view, Snowflake records the results of scheduling the DMF in a dedicated event table for data metric functions. You can specify the frequency for how often the DMF is called. For example, you can schedule the DMFs on a particular table to run three times daily. You can modify the frequency as needed based on your own internal data quality requirements. All DMFs that are set on the table follow the same schedule.

After you schedule the DMFs to run, you can configure alerts to notify you when changes to data quality occur. By combining the DMF and alert functionality, you can have consistent threshold notifications for data quality on the tables that you measure. These insights enhance your data governance posture by enabling the following:

- Data stewards to know the current state of their data based on a particular metric.
- Data engineers to take immediate action on important tables and views.
- Platform administrators to ensure data quality monitoring is done with cost, consistency, and performance.

The data quality workflow of defining, measuring, and monitoring data can then be applied to additional workloads.

# Access control setup

To complete this tutorial, use a single custom role that has all of the required access, which includes the following:

- Creating a database, which subsequently allows creating a schema, creating a DMF in the schema, and creating a table in the schema
- Creating a warehouse to perform query operations
- Querying the view that contains the results of calling the scheduled DMF
- Querying the view that contains serverless compute usage information

Create the `dq_tutorial_role` role to use throughout the tutorial:

```
USE ROLE ACCOUNTADMIN;
CREATE ROLE IF NOT EXISTS dq_tutorial_role;
```

Grant privileges, and grant the application role and database roles to the `dq_tutorial_role`:

```
GRANT CREATE DATABASE ON ACCOUNT TO ROLE dq_tutorial_role;
GRANT EXECUTE DATA METRIC FUNCTION ON ACCOUNT TO ROLE dq_tutorial_role;
GRANT APPLICATION ROLE SNOWFLAKE.DATA_QUALITY_MONITORING_VIEWER TO ROLE dq_tutorial_role;
GRANT DATABASE ROLE SNOWFLAKE.USAGE_VIEWER TO ROLE dq_tutorial_role;
GRANT DATABASE ROLE SNOWFLAKE.DATA_METRIC_USER TO ROLE dq_tutorial_role;
```

Create a warehouse to query the table that contains the data and grant the USAGE privilege on the role to the `dq_tutorial_role` role:

```
CREATE WAREHOUSE IF NOT EXISTS dq_tutorial_wh;
GRANT USAGE ON WAREHOUSE dq_tutorial_wh TO ROLE dq_tutorial_role;
```

Confirm the grants to the `dq_tutorial_role` role:

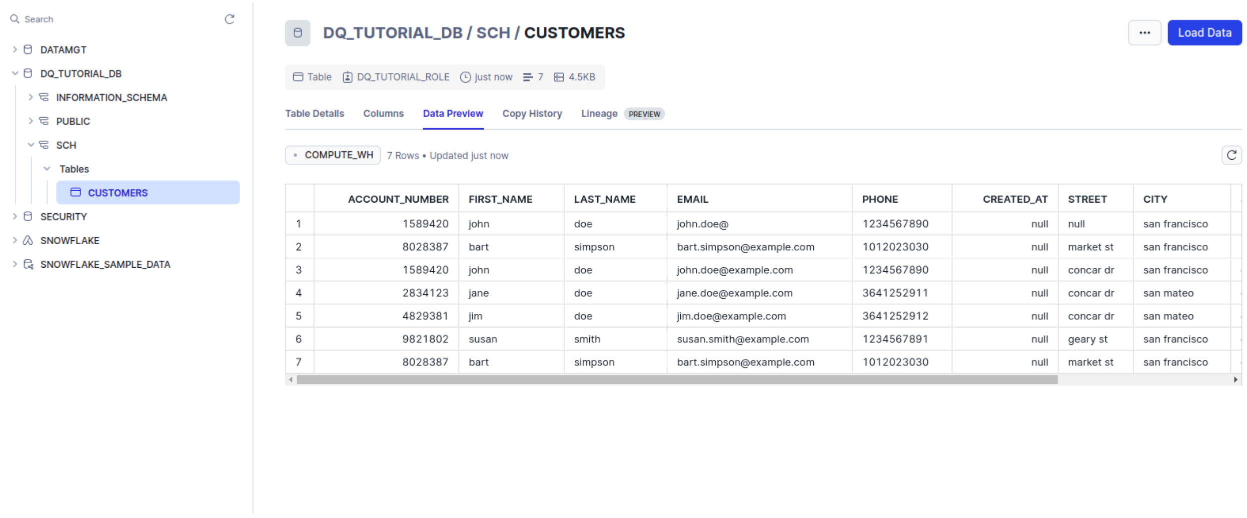
```
SHOW GRANTS TO ROLE dq_tutorial_role;
```

Establish a role hierarchy and grant the role to a user who can complete this tutorial (replace the `jsmith` value):

```
GRANT ROLE dq_tutorial_role TO ROLE SYSADMIN;
GRANT ROLE dq_tutorial_role TO USER jsmith;
```

Let's now create a database table with some sample data

```sql
USE ROLE dq_tutorial_role;
CREATE DATABASE IF NOT EXISTS dq_tutorial_db;
CREATE SCHEMA IF NOT EXISTS sch;

CREATE TABLE customers (
  account_number NUMBER(38,0),
  first_name VARCHAR(16777216),
  last_name VARCHAR(16777216),
  email VARCHAR(16777216),
  phone VARCHAR(16777216),
  created_at TIMESTAMP_NTZ(9),
  street VARCHAR(16777216),
  city VARCHAR(16777216),
  state VARCHAR(16777216),
  country VARCHAR(16777216),
  zip_code NUMBER(38,0)
);

USE WAREHOUSE dq_tutorial_wh;

INSERT INTO customers (account_number, city, country, email, first_name, last_name, phone, state, street, zip_code)
  VALUES (1589420, 'san francisco', 'usa', 'john.doe@', 'john', 'doe', 1234567890, null, null, null);

INSERT INTO customers (account_number, city, country, email, first_name, last_name, phone, state, street, zip_code)
  VALUES (8028387, 'san francisco', 'usa', 'bart.simpson@example.com', 'bart', 'simpson', 1012023030, null, 'market st', 94102);

INSERT INTO customers (account_number, city, country, email, first_name, last_name, phone, state, street, zip_code)
  VALUES
    (1589420, 'san francisco', 'usa', 'john.doe@example.com', 'john', 'doe', 1234567890, 'ca', 'concar dr', 94402),
    (2834123, 'san mateo', 'usa', 'jane.doe@example.com', 'jane', 'doe', 3641252911, 'ca', 'concar dr', 94402),
    (4829381, 'san mateo', 'usa', 'jim.doe@example.com', 'jim', 'doe', 3641252912, 'ca', 'concar dr', 94402),
    (9821802, 'san francisco', 'usa', 'susan.smith@example.com', 'susan', 'smith', 1234567891, 'ca', 'geary st', 94121),
    (8028387, 'san francisco', 'usa', 'bart.simpson@example.com', 'bart', 'simpson', 1012023030, 'ca', 'market st', 94102);
```



DQ_TUTORIAL_DB / SCH / CUSTOMERS

Table · DQ_TUTORIAL_ROLE · just now · 7 · 4.5KB

COMPUTE_WH 7 Rows • Updated just now

| | ACCOUNT_NUMBER | FIRST_NAME | LAST_NAME | EMAIL | PHONE | CREATED_AT | STREET | CITY |
|---|---|---|---|---|---|---|---|---|
| 1 | 1589420 | john | doe | john.doe@ | 1234567890 | null | null | san francisco |
| 2 | 8028387 | bart | simpson | bart.simpson@example.com | 1012023030 | null | market st | san francisco |
| 3 | 1589420 | john | doe | john.doe@example.com | 1234567890 | null | concar dr | san francisco |
| 4 | 2834123 | jane | doe | jane.doe@example.com | 3641252911 | null | concar dr | san mateo |
| 5 | 4829381 | jim | doe | jim.doe@example.com | 3641252912 | null | concar dr | san mateo |
| 6 | 9821802 | susan | smith | susan.smith@example.com | 1234567891 | null | geary st | san francisco |
| 7 | 8028387 | bart | simpson | bart.simpson@example.com | 1012023030 | null | market st | san francisco |

# Create and work with DMFs

In the following sections, we will create a user-defined DMF to measure the count of invalid email addresses and subsequently do the following:

- Schedule the DMF to run every 5 minutes.
- Check the DMF table references (find the tables the DMF is set on).
- Query a built-in view that contains the result of calling the scheduled DMF.
- Unset the DMF from the table to avoid unnecessary serverless credit usage.

## Create a DMF

Create a data metric function (DMF) to return the number of email addresses in a column that don't match the specified regular expression:

```sql
CREATE DATA METRIC FUNCTION IF NOT EXISTS
  invalid_email_count (ARG_T table(ARG_C1 STRING))
  RETURNS NUMBER AS
  'SELECT COUNT_IF(FALSE = (
    ARG_C1 REGEXP ''^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$''))
    FROM ARG_T';
```

## Set the schedule on the table

The DMF schedule defines when all DMFs on the table run. Currently, 5 minutes is the shortest possible time interval:

```sql
ALTER TABLE customers SET DATA_METRIC_SCHEDULE = '5 MINUTE';
```

**Note**
For the purpose of the tutorial, the schedule is set for 5 minutes. However, after you optimize your DMF use cases, experiment with the other schedule settings, such as cron expressions or trigger events associated with DML operations that affect the table.

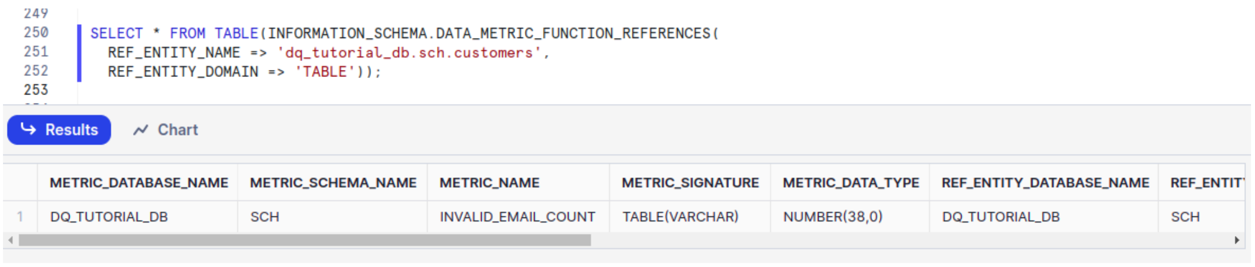
## Set the DMFs on the table and check the references

Associate the DMF to the table:

```
ALTER TABLE customers ADD DATA METRIC FUNCTION
  invalid_email_count ON (email);
```

Because the schedule is set for 5 minutes, we need to wait 5 minutes in order for Snowflake to call the DMF and process the results. For now, we can check to see that the DMF is associated with the table by calling the DATA_METRIC_FUNCTION_REFERENCES Information Schema table function:

```
SELECT * FROM TABLE(INFORMATION_SCHEMA.DATA_METRIC_FUNCTION_REFERENCES(
  REF_ENTITY_NAME => 'dq_tutorial_db.sch.customers',
  REF_ENTITY_DOMAIN => 'TABLE'));
```



```
SELECT * FROM TABLE(INFORMATION_SCHEMA.DATA_METRIC_FUNCTION_REFERENCES(
  REF_ENTITY_NAME => 'dq_tutorial_db.sch.customers',
  REF_ENTITY_DOMAIN => 'TABLE'));
```

| | METRIC_DATABASE_NAME | METRIC_SCHEMA_NAME | METRIC_NAME | METRIC_SIGNATURE | METRIC_DATA_TYPE | REF_ENTITY_DATABASE_NAME | REF_ENTIT |
|---|---|---|---|---|---|---|---|
| 1 | DQ_TUTORIAL_DB | SCH | INVALID_EMAIL_COUNT | TABLE(VARCHAR) | NUMBER(38,0) | DQ_TUTORIAL_DB | SCH |

## View the DMF results

The results of calling the scheduled DMF are stored in the DATA_QUALITY_MONITORING_RESULTS view. To determine the number of invalid email addresses, query the DATA_QUALITY_MONITORING_RESULTS view to see the results of calling the scheduled DMF:

```
SELECT scheduled_time, measurement_time, table_name, metric_name, value
FROM SNOWFLAKE.LOCAL.DATA_QUALITY_MONITORING_RESULTS
WHERE TRUE
AND METRIC_NAME = 'INVALID_EMAIL_COUNT'
AND METRIC_DATABASE = 'DQ_TUTORIAL_DB'
LIMIT 100;
```

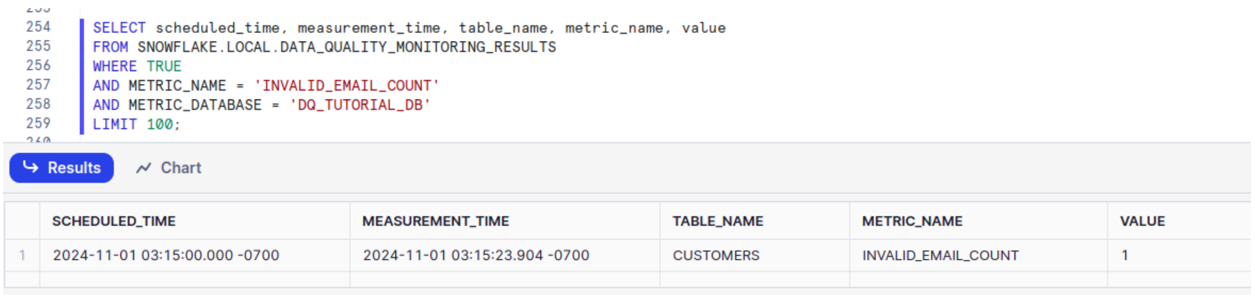
The results show that the `value` column contains `1`. This number corresponds to one improperly formatted email address, which corresponds to the first INSERT statement in the Insert values into a table section.

## Unset the DMFs from the table

You have established that the DMF is working as expected based on the definition of the DMF, the schedule, and the expected results.

To avoid unnecessary serverless credit usage, unset the DMF from the table:

```
ALTER TABLE customers DROP DATA METRIC FUNCTION
  invalid_email_count ON (email);
```



```
SELECT scheduled_time, measurement_time, table_name, metric_name, value
FROM SNOWFLAKE.LOCAL.DATA_QUALITY_MONITORING_RESULTS
WHERE TRUE
AND METRIC_NAME = 'INVALID_EMAIL_COUNT'
AND METRIC_DATABASE = 'DQ_TUTORIAL_DB'
LIMIT 100;
```

| | SCHEDULED_TIME | MEASUREMENT_TIME | TABLE_NAME | METRIC_NAME | VALUE |
|---|---|---|---|---|---|
| 1 | 2024-11-01 03:15:00.000 -0700 | 2024-11-01 03:15:23.904 -0700 | CUSTOMERS | INVALID_EMAIL_COUNT | 1 |

## View your serverless credit consumption

Calling scheduled data metric functions (DMFs) requires serverless compute resources. You can query the Account Usage view DATA_QUALITY_MONITORING_USAGE_HISTORY to view the DMF serverless compute cost.

Because the view has a latency of 1-2 hours, wait for that time to pass before querying the view. You can come back to this step later.

Query the view and filter the results to include the time interval of your scheduled DMF:

```
USE ROLE dq_tutorial_role;
SELECT *
FROM SNOWFLAKE.ACCOUNT_USAGE.DATA_QUALITY_MONITORING_USAGE_HISTORY
WHERE TRUE
AND START_TIME >= CURRENT_TIMESTAMP - INTERVAL '3 days'
LIMIT 100;
```

## Summary and key points

In summary, you learned how to do the following:

- Create a custom DMF to measure data quality and manage the DMF to optimize serverless credit usage.
- Monitor the serverless credit usage associated with calling the scheduled DMF.

## Drop the tutorial objects

If you plan to repeat the tutorial, you can keep the objects that you created.

Otherwise, drop the tutorial objects as follows:

```
USE ROLE ACCOUNTADMIN;
DROP DATABASE dq_tutorial_db;
DROP WAREHOUSE dq_tutorial_wh;
DROP ROLE dq_tutorial_role;
```