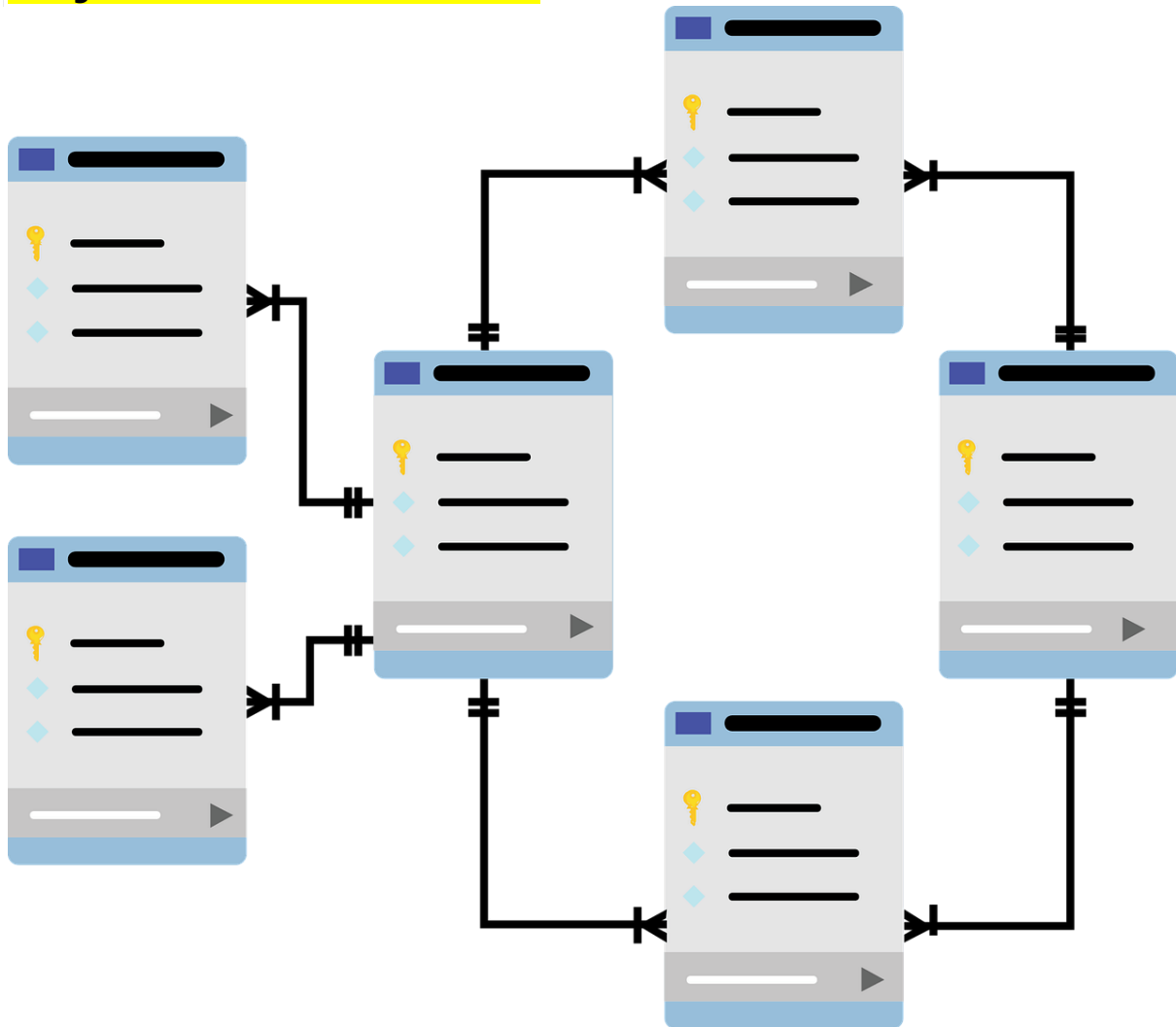


USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

Using Stored Procedures with Matillion



Stored procedures have been a part of database technology for a long time. In a typical modern [data fabric](#), you are likely to find stored procedures both in source databases and in your target [cloud data warehouse](#).

This article will explain exactly what stored procedures are, with examples showing how to take best advantage of them as a Matillion user.

Prerequisites

The prerequisites for using stored procedures with Matillion are:



USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

- Access to **Matillion ETL**

Plus either:

- Privilege to run stored procedures in your source database, or
- Privilege to create and use stored procedures in your Cloud Data Warehouse

What is a Stored Procedure?

Stored procedures are a high-code orchestration technology, designed to help with code re-use. Declare once: run repeatedly.

The code inside stored procedures mostly consists of SQL statements. In addition, different database vendors offer a variety of extension features such as variables, loops, conditional branching and exception handling.

In other words, stored procedures combine the convenience of bundling lots of related logic into a single named wrapper, plus encapsulation by allowing you to parameterize it.

Just like SQL itself, stored procedures exist entirely inside their host database. For data processing this brings two unique advantages:

1. Proximity to the data - which means top performance. As far as the SQL engine inside the database is concerned, it makes no difference whether a statement came from a stored procedure or not. The SQL runs just the same.
2. Tight integration with the host database - which means taking best advantage of all the specialized features provided by the database.

For Matillion users there are two main cases for stored procedures:

1. Inside a source database (such as SQL Server or Oracle), stored procedures offer the ability to get hold of data in a more sophisticated



USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

way than just running a SELECT query. When used for this purpose, stored procedures are conceptually similar to database views.

2. Inside a cloud data warehouse, stored procedures offer ways to encapsulate orchestration and administrative tasks. This especially applies where the task is frequently repeated and is parameterized.

In both the above cases, stored procedures also provide ways to govern access to database objects. A stored procedure may be declared to run either with the privileges of its owner or with the privileges of the caller. This distinction is also sometimes called definer vs invoker rights. There will be an example later in the article.

Stored Procedures in Source Databases

When your stored procedure exists in a source database, use a [Database Query component](#).

From a DataOps perspective, it can be helpful to verify in advance that the network route to the source database is open.



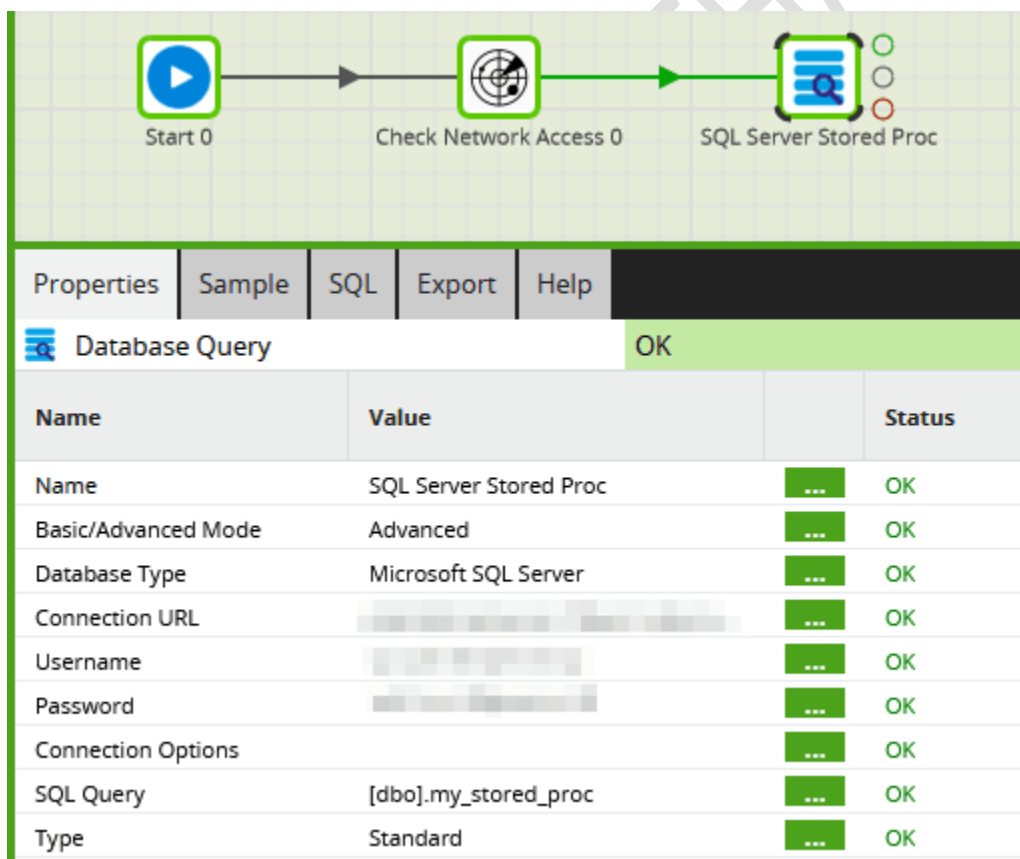
USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION



Use a [Check Network Access](#) component immediately before a Database Query

SQL Server Stored Procedures

From SQL Server you can call a stored procedure using a Database Query component in Advanced Mode. Just type the name of the stored procedure into the SQL Query property, optionally with its schema name.



The screenshot shows a workflow with three components: 'Start 0', 'Check Network Access 0', and 'SQL Server Stored Proc'. Below the workflow, the 'Database Query' component is configured in 'Advanced Mode' for 'Microsoft SQL Server'. The configuration table is as follows:

Name	Value	Status
Name	SQL Server Stored Proc	OK
Basic/Advanced Mode	Advanced	OK
Database Type	Microsoft SQL Server	OK
Connection URL	[REDACTED]	OK
Username	[REDACTED]	OK
Password	[REDACTED]	OK
Connection Options	[REDACTED]	OK
SQL Query	[dbo].my_stored_proc	OK
Type	Standard	OK

Depending on your Matillion ETL version, the sample tab may report a syntax error, but the component should nevertheless run successfully.



USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

SQL Server stored procedures can contain more than one SELECT statement. If there is a choice, Matillion will use the first one.

Some versions of SQL Server permit OLEDB access to stored procedures using OPENROWSET, with a SELECT statement like this:

```
SELECT * FROM OPENROWSET( 'SQLOLEDB', 'Trusted_Connection=Yes;  
Server=(local); Database=MyDatabase', 'exec my_stored_proc')
```

If OPENROWSET is enabled in your SQL Server source database, you can use the above as the SQL Query of your Matillion Database Query component.

Oracle Stored Procedures

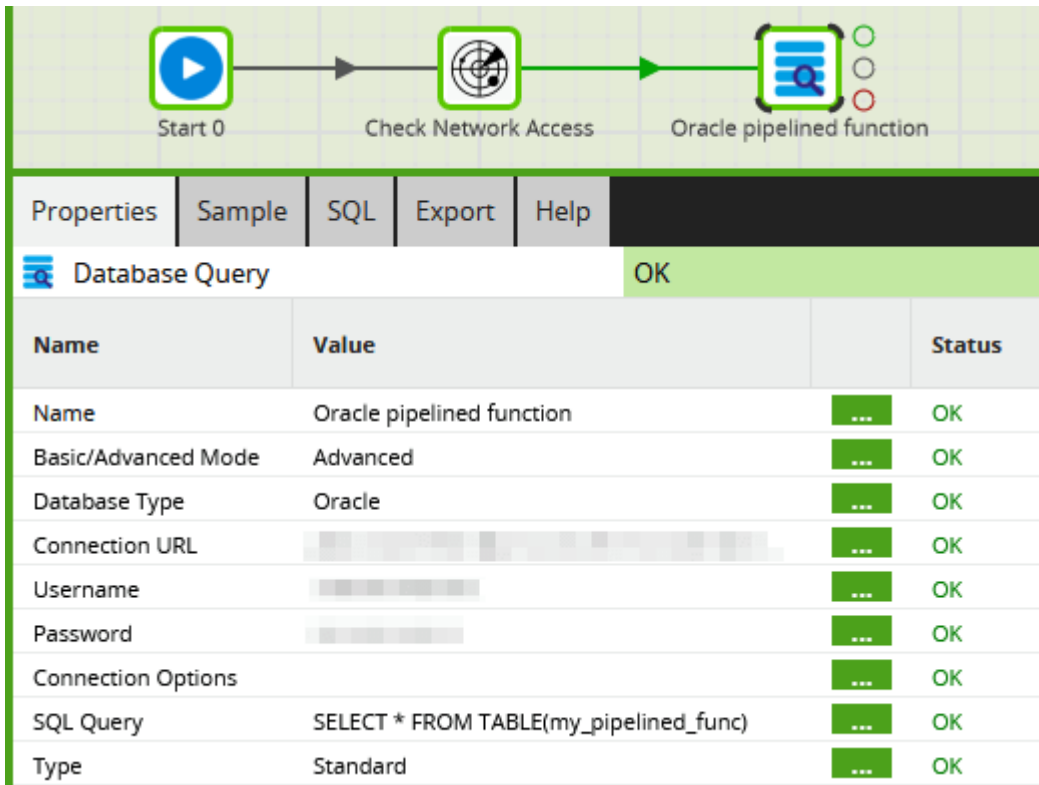
Oracle differentiates strongly between two kinds of stored subprograms:

- Stored Procedures - these are statements, and they do not return data.
- Stored Functions - these are expressions that return a value once the function has finished running. Stored functions that can return multiple records are known as pipelined functions.

Pipelined functions are the best way to read Oracle data defined by procedural logic. Switch your Matillion Database Query component into Advanced Mode, and use a SELECT with the following syntax:

```
SELECT * FROM TABLE(subprogram_name)
```

USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION



The screenshot shows a Matillion workflow with three jobs: 'Start 0', 'Check Network Access', and 'Oracle pipelined function'. Below the workflow, the 'Database Query' properties window is open, displaying the following configuration:

Name	Value		Status
Name	Oracle pipelined function	...	OK
Basic/Advanced Mode	Advanced	...	OK
Database Type	Oracle	...	OK
Connection URL	[REDACTED]	...	OK
Username	[REDACTED]	...	OK
Password	[REDACTED]	...	OK
Connection Options		...	OK
SQL Query	SELECT * FROM TABLE(my_pipelined_func)	...	OK
Type	Standard	...	OK

Stored Procedures in the Cloud Data Warehouse

Within a Cloud Data Warehouse, stored procedures can be useful for administrative operations in exactly the same way as Matillion's inbuilt low-code option: shared jobs.



USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION



Look for cases that involve several related SQL statements, and where the task needs to be repeated often with different parameters.

In data processing, a common example is a job audit table. Matillion requires the privilege to write to the audit table. But from a security perspective, interaction with the audit table must be carefully governed. A good solution is:

- Create the audit table in a private database location, and grant no access to it
- Create a stored procedure to manage the audit table, defined with Owner's rights
- Grant Matillion users the privilege to use the stored procedure

Snowflake Scripting example

For an audit table with columns named run_history_id, start_ts, end_ts, status and message, this is a simple stored procedure for managing the table:

```
CREATE OR REPLACE PROCEDURE  
"${environment_database}}.${examples_schema}."audit"( p_acti
```

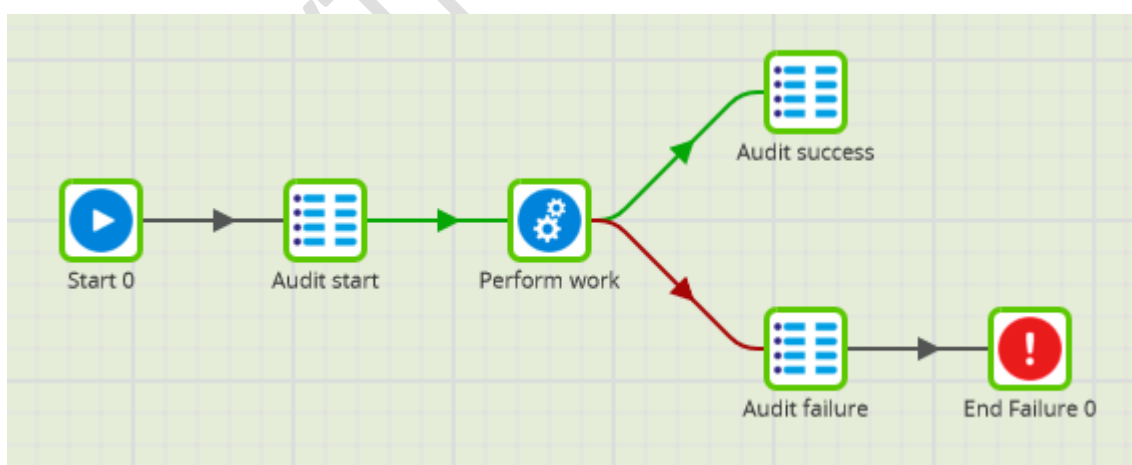


USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

```
on VARCHAR, p_run_history_id INTEGER, p_message VARCHAR)
RETURNS INTEGER LANGUAGE SQL EXECUTE AS OWNER AS $$ BEGIN   IF
(p_action = 'START') THEN      INSERT INTO
"${environment_database}"."${examples_schema}"."audit_log"
      ("run_history_id", "start_ts", "status")      VALUES
(:p_run_history_id, CURRENT_TIMESTAMP(),
'STARTED');   ELSE      UPDATE
"${environment_database}"."${examples_schema}"."audit_log"
      SET "end_ts" = CURRENT_TIMESTAMP(),      "status" =
:p_action,      "message" = :p_message      WHERE
"run_history_id" = :p_run_history_id;   END IF; END; $$ ;
```

The **EXECUTE AS OWNER clause** declares that the SQL will run with the privileges of the database user that owns the stored procedure - rather than whichever Matillion user happens to be running it. This is a great way of giving tightly controlled access to the protected audit table.

The START action creates a new record. When the job has finished, SUCCESS or FAILURE actions update the record and set the timing and optional error message. An implementation taking advantage of Matillion ETL's default **error handling pattern** would look like this:



For illustration purposes, all the audit tasks above are just **SQL Script components**. In production, it would be better to implement them as a **Shared Job**.

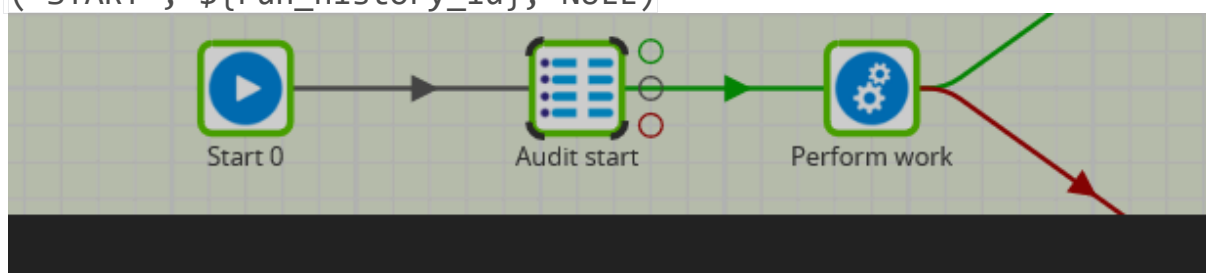


USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

Audit start

As the very first task of the job, the SQL uses the run_history_id automatic variable like this to create a new audit record:

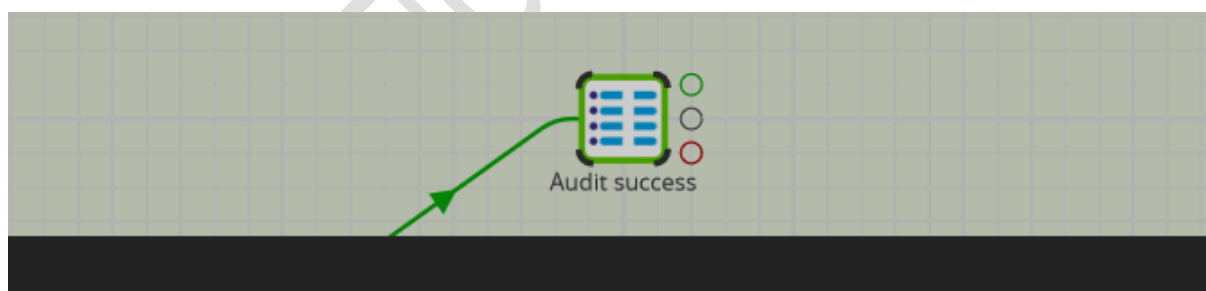
```
CALL "${environment_database}"."${examples_schema}"."audit"('START', ${run_history_id}, NULL)
```



```
1 CALL "${environment_database}"."${examples_schema}"."audit"('START', ${run_history_id}, NULL)
```

Audit success

On the success branch, the same run_history_id is used to close the open audit record.



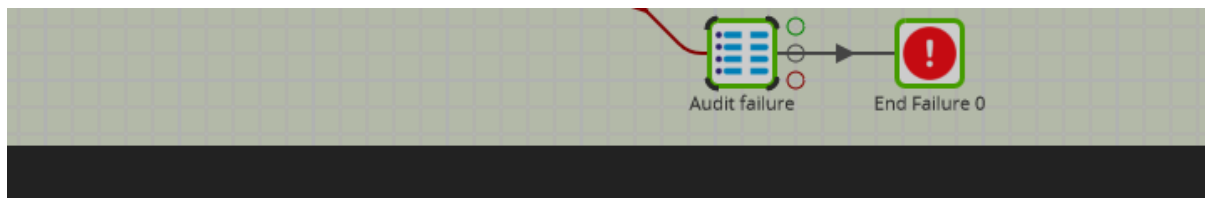
```
1 CALL "${environment_database}"."${examples_schema}"."audit"('SUCCESS', ${run_history_id}, NULL)
```

Audit failure

The failure branch is almost identical, but includes the detailed_error automatic variable to record the stack trace while the audit record is closed.



USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION



```
1 CALL "${environment_database}}.${examples_schema}."audit('FAILURE', ${run_history_id}, '${detailed_error}')
```

It is advisable to add an End Failure component afterwards because the audit task is on the failure branch. The job should still fail overall even after the audit task succeeds.

Practical Implementation :

Use State_Region zip files uploaded previously for all the data load into initial source table or click below link :

https://drive.google.com/drive/folders/1KnNJ71p2M6x6kYc_jAlgZSKDyWqZbjCL?usp=sharing

--Execute the following commands in Snowflake by creating a respective source table

```
CREATE OR REPLACE DATABASE TEST_SRC;
```

```
CREATE OR REPLACE SCHEMA TEST_SCHEMA;
```

```
USE SCHEMA TEST_SCHEMA;
```

```
CREATE OR REPLACE SCHEMA DEST_SCHEMA;
```

--First Create a respective table in the desired source database & source schema

```
CREATE OR REPLACE TABLE BROKER CLONE DEMO_DATABASE.DEMO_SCHEMA.AJ_BROKER;
```

```
CREATE OR REPLACE TABLE CATEGORIES CLONE DEMO_DATABASE.DEMO_SCHEMA.AJ_CATEGORIES;
```

```
CREATE OR REPLACE TABLE COMPLAIN CLONE DEMO_DATABASE.DEMO_SCHEMA.AJ_COMPLAIN;
```

--then using stored procedure we will create the table in destination database (respective destination schema) and load the records

```

graph LR
    A[Start 0] --> B[Create proc]
    B --> C[Create tables]
  
```

The diagram illustrates the initial steps of a process. It starts with a blue play button icon labeled "Start 0". An arrow points to a green box icon labeled "Create proc". Another arrow points from "Create proc" to a green box icon labeled "Create tables".

```
CREATE OR REPLACE PROCEDURE actiontobeperformed(src_db STRING,  
                                                src_dbschema string,  
        tablename string,  
        tgt_db string,  
        tgt_dbschema string,  
        action_type string,  
        table_view char(1))
```



USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

return string

language

javascript execute

as caller as

\$\$

```
//VARIABLE DECLARATION    var
action_type = ACTION_TYPE;    var
src_db = SRC_DB;    var src_dbschema
= SRC_DBSCHEMA;    var tablename =
TABLENAME;    var tgt_db = TGT_DB;
    var tgt_dbschema = TGT_DBSCHEMA;
var table_view = TABLE_VIEW;
    var result = "";
    var sql_query_text = "";

    try
    {
        //CREATE TABLES    if (action_type.toUpperCase() == 'CREATE' &&
table_view.toUpperCase() == 'T')
    {
        var sql_query_text = 'CREATE OR REPLACE TABLE ' + tgt_db + '.' + tgt_dbschema + '.' +
tablename + ' like ' + src_db +
        '.' + src_dbschema + '.' + tablename + ';';
snowflake.createStatement({sqlText:sql_query_text}).execute();    result = "Table
created successfully: " + tgt_db + '.' + tgt_dbschema + '.' + tablename
    }
```



USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

```
//INSERT RECORDS      else if (action_type.toUpperCase() == 'INSERT' &&
table_view.toUpperCase() == 'I')
{
    var sql_query_text = 'INSERT INTO ' + tgt_db + '.' + tgt_dbschema + '.' + tablename + '
select * from ' + src_db +
    '.' + src_dbschema + '.' + tablename + ';'
snowflake.createStatement({sqlText:sql_query_text}).execute();      result = "Insertion
successfully completed: " + tgt_db + '.' + tgt_dbschema + '.' + tablename
    }

else
{
    result = "Fail: Please give proper parameter \n" + "1.If INSERTing the data then parameter
should be only Table(t/T)\n"
    }
}

catch (err)
{
    result += "\n Code: " + err.code + "\n State: " +
err.state;      result += "\n Message: " + err.message;
result += "\n Stack Trace:\n" + err.stackTraceTxt;      result
+= "Failed Query Text: " +sql_query_text;      throw result;
    }

return result;
```



USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

\$\$

;

Variables

Name	Default value
------	---------------

Manage Variables ▾

```
1 USE TEST_SRC;
2
3 CREATE OR REPLACE procedure actiontobeperformed(src_db STRING,
4 src_dbschema string,
5 tablename string,
6 tgt_db string,
7 tgt_dbschema string,
8 action_type string,
9 table_view char(1))
10 returns string
11 language javascript
12 execute as caller
13 as
14 $$
15
16 //VARIABLE DECLARATION
17 var action_type = ACTION_TYPE;
18 var src_db = SRC_DB;
19 var src_dbschema = SRC_DBSCHEMA;
20 var tablename = TABLENAME;
21 var tgt_db = TGT_DB;
22 var tgt_dbschema = TGT_DBSCHEMA;
23 var table_view = TABLE_VIEW;
24 var result = "";
25 var sql_query_text = "";
26
27
```

Run

Press "Run" to test script execution.

Update Component OK Cancel

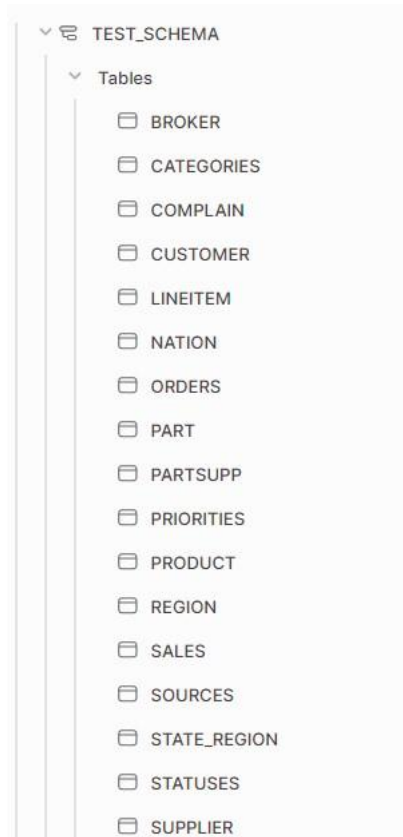
--Then drag and place SQL SCRIPT component again where we will be calling stored procedure to create table first and then will change the script and instead of CREATE will replace INSERT along with alias as I as shown below

```
1 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','CUSTOMER','TEST_DST','DEST_SCHEMA','CREATE','T');
2 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','LINEITEM','TEST_DST','DEST_SCHEMA','CREATE','T');
3 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','REGION','TEST_DST','DEST_SCHEMA','CREATE','T');
4 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','NATION','TEST_DST','DEST_SCHEMA','CREATE','T');
5 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','SUPPLIER','TEST_DST','DEST_SCHEMA','CREATE','T');
6 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','PARTSUPP','TEST_DST','DEST_SCHEMA','CREATE','T');
7 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','PART','TEST_DST','DEST_SCHEMA','CREATE','T');
8 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','ORDERS','TEST_DST','DEST_SCHEMA','CREATE','T');
9
```



USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

Once the table gets created in the destination database (resp schema) as shown below



Its time to load the records now by executing STORED PROCEDURE CALL by changing CREATE to INSERT with alias I



USING STORED PROCEDURE CONCEPT IN MATILLION
LOAD TABLE FROM SRC TO DESTINATION

Variables

Name	Default value
------	---------------

Manage Variables

```
1 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','CUSTOMER','TEST_DST','DEST_SCHEMA','INSERT','I');
2 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','LINEITEM','TEST_DST','DEST_SCHEMA','INSERT','I');
3 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','REGION','TEST_DST','DEST_SCHEMA','INSERT','I');
4 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','NATION','TEST_DST','DEST_SCHEMA','INSERT','I');
5 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','SUPPLIER','TEST_DST','DEST_SCHEMA','CREATE','I');
6 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','PARTSUPP','TEST_DST','DEST_SCHEMA','CREATE','I');
7 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','PART','TEST_DST','DEST_SCHEMA','CREATE','I');
8 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','ORDERS','TEST_DST','DEST_SCHEMA','CREATE','I');
9
```

Run

Press "Run" to test script execution.

Update Component OK Cancel

Variables

Name	Default value
------	---------------

Manage Variables

```
1 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','CUSTOMER','TEST_DST','DEST_SCHEMA','INSERT','I');
2 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','LINEITEM','TEST_DST','DEST_SCHEMA','INSERT','I');
3 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','REGION','TEST_DST','DEST_SCHEMA','INSERT','I');
4 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','NATION','TEST_DST','DEST_SCHEMA','INSERT','I');
5 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','SUPPLIER','TEST_DST','DEST_SCHEMA','INSERT','I');
6 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','PARTSUPP','TEST_DST','DEST_SCHEMA','INSERT','I');
7 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','PART','TEST_DST','DEST_SCHEMA','INSERT','I');
8 CALL actiontobeperformed('TEST_SRC','TEST_SCHEMA','ORDERS','TEST_DST','DEST_SCHEMA','INSERT','I');
9
```

Run

Press "Run" to test script execution.

Update Component OK Cancel

Finally Run the Jobs and we will be able to see the records in the destination

load tables using proc		6.5s	02:54:10	02:54:10	02:54:17	02:54:17		
load tables using proc	Start 0	0.0s	02:54:10	02:54:10	02:54:10			
load tables using proc	Create proc	1.5s	02:54:10	02:54:10	02:54:12	0	Successfully executed [2] queries.	
load tables using proc	Create tables	5.0s	02:54:12	02:54:12	02:54:17	0	Successfully executed [8] queries.	



USING STORED PROCEDURE CONCEPT IN MATILLION LOAD TABLE FROM SRC TO DESTINATION

practice_matillion

load tables using proc

Task - load tables using proc

Start 0 → Create proc → Create tables

Properties | SQL | Export | Help

SQL Script

Name	Value	Status
Name	Create tables	OK
SQL Script	CALL actiontobeperf...	OK

Tasks

Task	Envir...	Version	Job	Queu...	Job C...	Sched...
Validate	testin...	default	load t...	03:26...	03:26...	03:26...
Run	testin...	default	load t...	03:07...	03:08...	03:08...
Validate	testin...	default	load t...	03:07...	03:07...	03:07...
Sample	testin...	default	load t...	03:07...	03:07...	03:07...

*****HAPPY LEARNING*****