Zero to Snowflake: Simple SQL Stored Procedures

One question we often get when a customer is considering moving to Snowflake from another platform, like Microsoft SQL Server for instance, is what they can do about migrating their SQL stored procedures to Snowflake.

TL;DR: Below is a basic procedure template you can customize and extend for your use case:

```
CREATE OR REPLACE PROCEDURE simple_stored_procedure_example()
returns string not null
language javascript
as
$$
var cmd = `
<SOME SUPER SWEET SQL STATEMENT>
`
var sql = snowflake.createStatement({sqlText: cmd});
var result = sql.execute();
return '';
$$;
```

Drop your SQL query into the `<SOME SUPER SWEET SQL STATEMENT>` space and have fun. Note that you can use backticks around the SQL statement to keep it nicely formatted for readability.

Let's break down the example above:

1. Define the procedure without arguments.
2. Tell the procedure to return a string.
3. Make sure the runtime language is javascript ... duh.
4. Copy some SQL to the `cmd` variable.
5. Add the `cmd` variable to the `snowflake.createStatement()` function.
6. Execute the prepared statement in the `sql` variable, and store the results in a new variable called `result`.
7. Return a string (see step 2) on successful execution.
8. Profit. Note the moneybag.

This is another example where we pass in an argument to the procedure:

```
CREATE OR REPLACE PROCEDURE
simple_stored_procedure_example_with_arguments(awesome_argument VARCHAR)
returns string not null
language javascript
as
$$
var your_awesome_arg = AWESOME_ARGUMENT;
var result = `${your_awesome_arg}`
return result;
$$;
```

Here's how you call the procedure with any text you want in the argument:

```
CALL simple_stored_procedure_example_with_arguments('Howdy! ');
```

Giddy up!

Building a Snowflake Data Pipeline with a Stored Procedure, Task and Stream

Let's build a *slightly* more realistic scenario with a Snowflake task and stream. Snowflake's tasks are simply a way to run some SQL on a schedule or when triggered by other tasks. One problem with defining raw SQL in a task is that if you have to update the SQL definition---let's say by adding a column to a select statement---you must also remember to run an alter statement to resume the task when created or modified tasks are suspended by default.

By using a stored procedure like the template above, you can modify the SQL without having to remember to resume the task. So from the task's perspective, it's a transparent change, and if we combine a stream with a task, we can ensure that our procedure only executes when new data is added to the table.

Here's a more complete example that will only insert new records from a source table into a target table on a 15-minute schedule using a procedure, task and table stream for flagging new rows in the source table. First, let's create some tables and a stream for our data:

```
CREATE OR REPLACE TABLE source_table (
  uuid varchar
  ,inserted_at timestamp
);

CREATE OR REPLACE STREAM source_table_stream
ON TABLE source_table APPEND_ONLY=TRUE;

CREATE OR REPLACE TABLE target_table (
  uuid varchar
  ,source_inserted_at timestamp
  ,target_inserted_at timestamp
);
```

Next, we'll create a stored procedure that will insert a record into our source table and then select and insert that record into our target table from the stream:

```
CREATE OR REPLACE PROCEDURE source_table_stream_procedure()
returns string not null
language javascript
as
$$
var insert_cmd = `
INSERT INTO source_table
SELECT uuid_string(), current_timestamp::timestamp_ntz;
`
var sql_insert = snowflake.createStatement({sqlText: insert_cmd});
var insert_result = sql_insert.execute();

var stream_select_cmd = `
INSERT INTO target_table
SELECT
  uuid
  ,inserted_at
  ,current_timestamp::timestamp_ntz
FROM
```

```
  source_table_stream
WHERE
  metadata$action = 'INSERT';
`
var sql_select_stream = snowflake.createStatement({sqlText: stream_select_cmd});
var select_stream_result = sql_select_stream.execute();
return '';
$$;
```

We'll create the task:

```
CREATE OR REPLACE TASK source_table_stream_procedure_task
  WAREHOUSE = my_task_warehouse
  SCHEDULE = '15 MINUTE'
WHEN
  system$stream_has_data('source_table_stream')
AS
  CALL source_table_stream_procedure();
```

And, finally, set it to run:

```
ALTER TASK source_table_stream_procedure_task RESUME;
```

Let's check when the task is scheduled to run by looking at the `SCHEDULED_TIME` column by querying the task history:

```
SELECT *
FROM table(
  information_schema.task_history(
    task_name=>'source_table_stream_procedure_task'
    ,scheduled_time_range_start=>dateadd('hour',-1,current_timestamp())
  )
);
```

Congrats! You've just written a Snowflake data pipeline.