A Stored Procedure is a set/series of SQL statements

It has some inputs

It has some outputs

We can write Stored procedure in python.

You can use the snowflake Snowpark library within stored procedure to perform queries, updates, and other work on tables in Snowflake.

With Snowpark Stored Procedures, you can build and run your data pipeline within Snowflake, using a Snowflake warehouse as the compute framework.

You must use version 0.4.0 or a more recent version of the Snowpark library.

Snowflake currently supports writing Stored Procedures in Python version 3.8.

Setting Up Your Development Environment for Snowpark

You can either create an in-line stored procedure or a stored procedure with code uploaded from a stage.

In an in-line stored procedure, you write your Python code in the **AS clause** of the CREATE PROCEDURE statement.

In a stored procedure created with code uploaded from a stage, you write your Python code in a `.py` source file. For example you could put the following code into a file called `my_py_file.py`

You then upload the file to a stage and execute the CREATE PROCEDURE command, pointing to the file on the stage. For example:

```
CREATE OR REPLACE PROCEDURE MYPROC(from_table STRING, to_table STRING, count
INT)
  RETURNS INT
  LANGUAGE PYTHON
  RUNTIME_VERSION = '3.8'
  PACKAGES = ('snowflake-snowpark-python')
  IMPORTS = ('@mystage/my_py_file.py')
  HANDLER = 'my_py_file.run';
```

When writing the Python Code for the Stored Procedure

- Limit the Amount of Memory Consumed(avoid consuming too much memory)
- Write Thread-Safe Code
- Understand the Security Restrictions
- Decide on Using Owner's Rights or Caller's Rights
- Be Aware of snowflake-snowpark-python Version

When writing the method or function for the stored procedure

- Specify the Snowpark `Session` object as the first argument of your method or function. When you call your stored procedure, Snowflake automatically creates a `Session` object and passes it to your stored procedure. (You cannot create the `Session` object yourself.)
- For the rest of the arguments and for the return value, use the Python types that correspond to Snowflake data types. Snowflake supports the Python data types listed in SQL-Python Data Type Mappings for Parameters and Return Types

The following is an example of a Python method that copies a specified number of rows from one table to another table. The method takes the following arguments:

- A Snowpark `Session` object
- The name of the table to copy the rows from
- The name of the table to save the rows to
- The number of rows to copy

You can also display all packages available and their version information by querying the PACKAGES view in the Information Schema.

select * from information_schema.packages where language = 'python';

Set the CREATE PROCEDURE parameters as listed in the table below.

Parameter	Description
`[ arg_name arg_data_type [, ... ] ]`	• Omit the argument for the Snowpark `Session` object. As mentioned earlier, this is not a formal parameter that you specify in CREATE PROCEDURE or CALL. When you call your stored procedure, Snowflake creates a `Session` object and passes it to your stored procedure. • For the data types of the rest of the arguments, use the Snowflake data types that correspond to the Python types of the arguments in your method or function.
`RETURNS result_data_type`	Specify RETURNS with the Snowflake data type of your return value.
`LANGUAGE PYTHON`	You must specify this to indicate that your stored procedure code is written in Python.
`RUNTIME_VERSION = '3.8'`	You must specify this to indicate that your stored procedure uses Python 3.8.
`PACKAGES = ( 'package_name' )`	In this list, include the package for the version of the Snowpark library that you want to use.

Parameter	Description
	Specify the fully qualified package name for the Snowpark library in the following format: `snowflake-snowpark-python==<version>` For example: • To use the latest version of Snowpark, use: • `PACKAGES = ('snowflake-snowpark-python')` • To use the 0.4.0 version of Snowpark, use: • `PACKAGES = ('snowflake-snowpark-python==0.4.0')`
`IMPORTS = ( 'file' [, 'file' ... ] )`	If you are writing an in-line stored procedure, you can omit this clause, unless your code depends on classes defined outside the stored procedure or resource files. If you are creating a stored procedure with code uploaded from a stage, include that file in this list. If your stored procedure depends on any files that you uploaded to a stage location, include those files in this list.
`HANDLER = 'method_name'`	Set this to the fully qualified name of your Python method or function.
`EXECUTE AS CALLER`	If you plan to set up the stored procedure to use caller's rights, add this parameter. Otherwise, if you want to use owner's rights, omit this parameter.

The following examples create stored procedures in Python.

Example 1: In-line stored procedure:

```
CREATE OR REPLACE PROCEDURE myProc(from_table STRING, to_table STRING, count
INT)
RETURNS STRING
LANGUAGE PYTHON
RUNTIME_VERSION = '3.8'
PACKAGES = ('snowflake-snowpark-python')
HANDLER = 'run'
AS
$$
def run(session, from_table, to_table, count):
  session.table(from_table).limit(count).write.save_as_table(to_table)
  return "SUCCESS"
$$;
```

Example 2: A stored procedure that uses code uploaded from a stage in the Python file `myfile.py` on the internal stage `mystage`:

```
CREATE OR REPLACE PROCEDURE myProc(from_table STRING, to_table STRING, count
INT)
RETURNS STRING
LANGUAGE PYTHON
RUNTIME_VERSION = '3.8'
PACKAGES = ('snowflake-snowpark-python')
IMPORTS = ('@mystage/myfile.py')
HANDLER = 'myfile.run'
```

Calling Your Stored Procedure

In order for a user to call a stored procedure, the user's role must have the USAGE privilege for the stored procedure.

Once you have the privileges to call the stored procedure, you can use the CALL statement to call the stored procedure. For example:

```
CALL myProc('table_a', 'table_b', 5);
```