STORE PROCEDURE

WHAT IS THE STORE PROCEDURE

- A stored procedure contains logic you write so you can call it from SQL.
- A stored procedure’s logic typically performs database operations by executing SQL statements.

With a stored procedure, you can also:

1. Dynamically create and execute SQL statements.
2. Execute code with the privileges of the role that owns the procedure, rather than with the privileges of the role that runs the procedure.

- This allows the stored procedure owner to delegate the power to perform specified operations to users who otherwise could not do so.

- A stored procedure is created with a **CREATE PROCEDURE** command and is executed with a **CALL** command.

What are the Benefits of using a Stored Procedure in SQL?

Stored procedures provide some crucial benefits, which are:

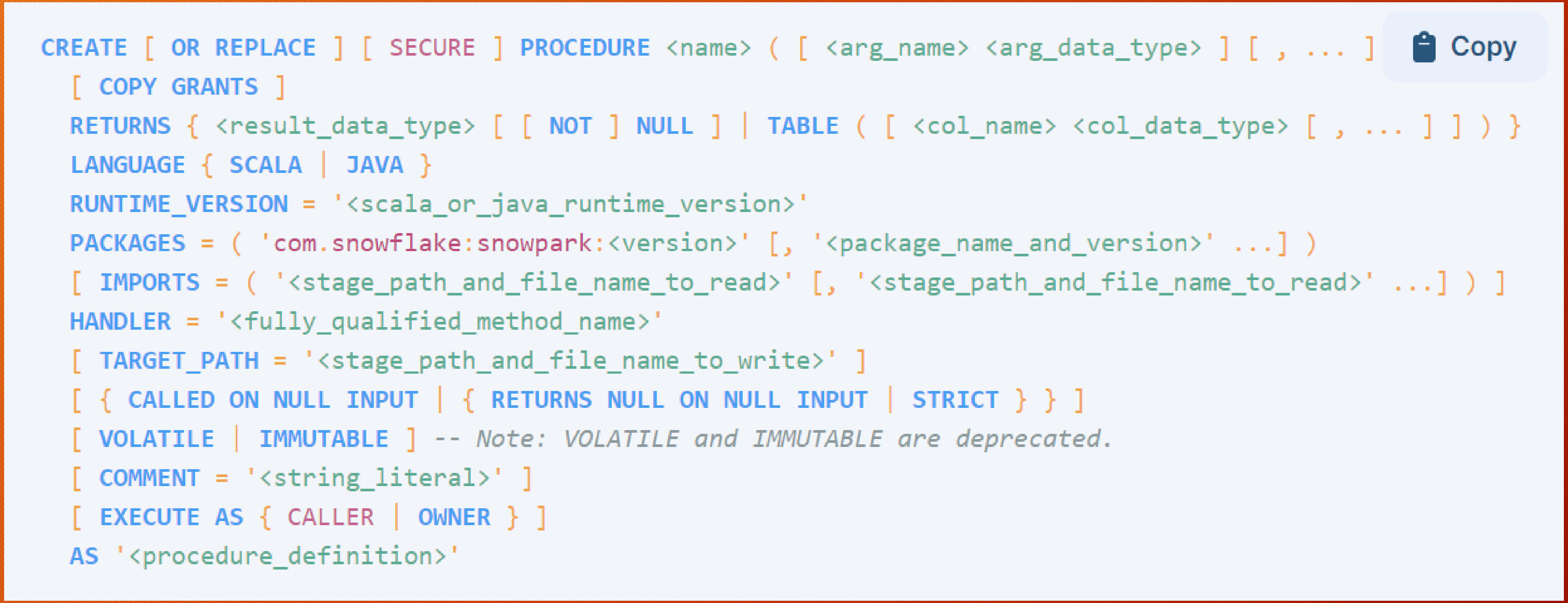
- Reusable: As mentioned, multiple users and applications can easily use and reuse stored procedures by merely calling it.
- Easy to modify: You can quickly change the statements in a stored procedure as and when you want to, with the help of the ALTER TABLE command.
- Security: Stored procedures allow you to enhance the security of an application or a database by restricting the users from direct access to the table.

CONTINUE…

- Low network traffic: The server only passes the procedure name instead of the whole query, reducing network traffic.

- Increases performance: Upon the first use, a plan for the stored procedure is created and stored in the buffer pool for quick execution for the next time.

SYNTAX IN SNOWFLAKE



```
CREATE [ OR REPLACE ] [ SECURE ] PROCEDURE <name> ( [ <arg_name> <arg_data_type> ] [ , ... ] )
  [ COPY GRANTS ]
  RETURNS { <result_data_type> [ [ NOT ] NULL ] | TABLE ( [ <col_name> <col_data_type> [ , ... ] ] ) }
  LANGUAGE { SCALA | JAVA }
  RUNTIME_VERSION = '<scala_or_java_runtime_version>'
  PACKAGES = ( 'com.snowflake:snowpark:<version>' [, '<package_name_and_version>' ...] )
  [ IMPORTS = ( '<stage_path_and_file_name_to_read>' [, '<stage_path_and_file_name_to_read>' ...] ) ]
  HANDLER = '<fully_qualified_method_name>'
  [ TARGET_PATH = '<stage_path_and_file_name_to_write>' ]
  [ { CALLED ON NULL INPUT | { RETURNS NULL ON NULL INPUT | STRICT } } ]
  [ VOLATILE | IMMUTABLE ] -- Note: VOLATILE and IMMUTABLE are deprecated.
  [ COMMENT = '<string_literal>' ]
  [ EXECUTE AS { CALLER | OWNER } ]
  AS '<procedure_definition>'
```

SYNTAX IN SNOWFLAKE

```
CREATE OR REPLACE PROCEDURE <name> ( [ <arg_name> <arg_data_type> ]
[ , ... ] )
  RETURNS <result_data_type>
  LANGUAGE SQL
AS
  $$
    <procedure_body>
  $$
;
```

Note that you must use string literal delimiters (‘ or $$) around the procedure definition(body) if you are creating a Snowflake Scripting procedure in Classic Web Interface or SnowSQL. The string literal delimiters (‘ or $$) are not mandatory when writing procedures in SnowSight.

SYNTAX IN MYSQL

```
DELIMITER
&&
CREATE PROCEDURE NAME
  BEGIN
        SQL STATEMENT
  END &&
DELIMITER ;
```

Various parameters in the stored procedure

NAME <name>

Specifies the name of the stored procedure.

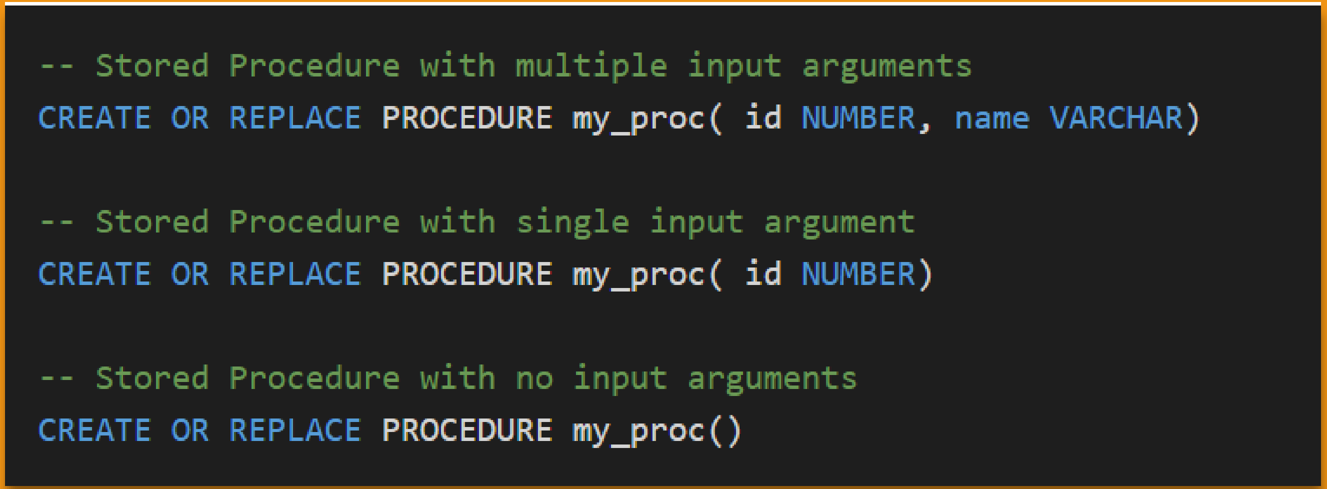
- The name must start with an alphabetic character and cannot contain spaces or special characters unless the entire identifier string is enclosed in double quotes (e.g. “My Procedure”). Identifiers enclosed in double quotes are also case-sensitive.

CONTINUE…

INPUT PARAMETERS ([<arg_name> <arg_data_type>] [, …])

A Stored Procedure can be built which takes one or more arguments as input parameters or even without any input parameters.

- The <arg_name> specifies the name of the input argument.
- The <arg_data_type> specifies the SQL data type of the input argument.

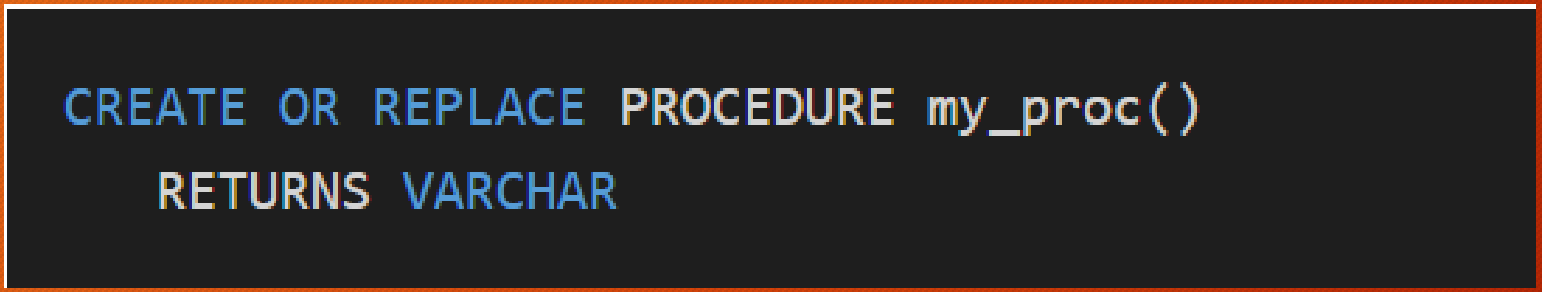


```sql
-- Stored Procedure with multiple input arguments
CREATE OR REPLACE PROCEDURE my_proc( id NUMBER, name VARCHAR)

-- Stored Procedure with single input argument
CREATE OR REPLACE PROCEDURE my_proc( id NUMBER)

-- Stored Procedure with no input arguments
CREATE OR REPLACE PROCEDURE my_proc()
```

CONTINUE…

RETURNS <result_data_type>

- Specifies the type of the result returned by the stored procedure.

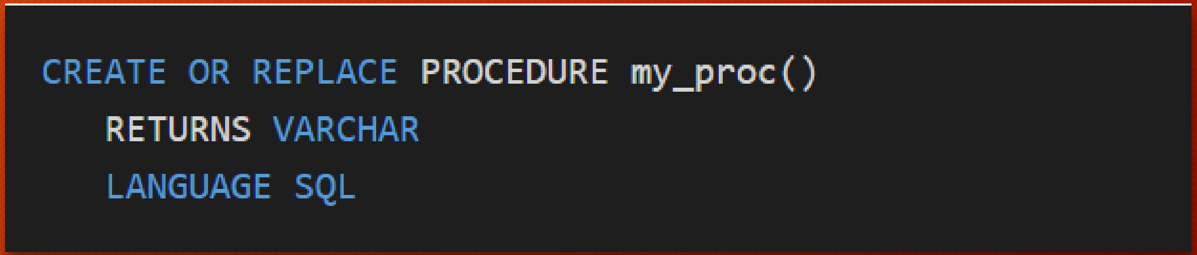


```
CREATE OR REPLACE PROCEDURE my_proc()
   RETURNS VARCHAR
```

CONTINUE…

LANGUAGE SQL

- Since Snowflake supports stored procedures in multiple languages, the **LANGUAGE** parameter specifies the language of the stored procedure definition. For Snowflake scripting, the value to the LANGUAGE parameter is passed as **SQL**.



```
CREATE OR REPLACE PROCEDURE my_proc()
  RETURNS VARCHAR
  LANGUAGE SQL
```

CONTINUE…

PROCEDURE BODY

The body defines the code executed by the stored procedure. The procedure definition is mentioned after the **AS** clause in the stored procedure construct. As mentioned earlier the body is wrapped between $$ string literal delimiters if the procedure scripting is not done in SnowSight.

Understanding various sections in Stored Procedure Body

The Stored Procedure Body is made up of multiple sections. The various sections in the stored procedure body are as follows.

```
DECLARE ... (variable declarations, cursor declarations, etc.) ...
BEGIN ... (Snowflake Scripting and SQL statements) ...
EXCEPTION
... (statements for handling exceptions) ...
END;
```

CONTINUE…

DECLARE

- The DECLARE section is used to define any variables, cursors etc. used in the body. Alternatively, they can be declared in the BEGIN…END section of the body also.

BEGIN…END

- The SQL statements and scripting constructs are written between the BEGIN and END sections of the body.

EXCEPTION

- The EXCEPTION section of the body is used to hold any exception handling code you wanted to add.

CONTINUE…

- ***Note that DECLARE and EXCEPTION sections are not mandatory in every procedure definition.***

A simple stored procedure body just requires to BEGIN and END sections.

```
BEGIN
   CREATE TABLE employees(id NUMBER, firstname VARCHAR);
END;
```

Creating a Stored Procedure in Snowflake

- Consider a use case where the requirement is to purge the inactive employees' data from a database table. Let us build a Stored Procedure which performs this activity.

- The below-Stored Procedure deletes all records with status field value as 'INACTIVE' from the employee's table.

CONTINUE…

```
CREATE OR REPLACE PROCEDURE purge_data()
   RETURNS VARCHAR
   LANGUAGE SQL
AS
 $$
   DECLARE
      message VARCHAR;
   BEGIN
      DELETE FROM employees WHERE status = 'INACTIVE';
      message := 'Inactive employees data deleted successfully';
      RETURN message;
   END;
 $$
;
```

Let us break down each block of the stored procedure below to understand better

- The name of the stored procedure is purge_data and do not take any input parameters.
- The data type of the return value from the store procedure is defined as varchar.
- The language is defined as SQL, the language in which the procedure body is defined.
- A variable named message of type varchar is defined under DECLARE section of the body.
- Between BEGIN…END section of the procedure body,
- The statement to delete the records with INACTIVE status is defined. The variable message is assigned a string value. The assignment operator used is:= for assigning value to the variable.
- The variable message is returned as the output from the stored procedure.

Creating a Stored Procedure with Input Parameters

- Consider another scenario where you wanted to purge the data from a table based on an input you passed. Let us understand with an example.

- The Stored Procedure deletes all records with a status value that matches the value passed as input through an input parameter in_status from the employees table.

CONTINUE…

```
CREATE OR REPLACE PROCEDURE purge_data_by_status(in_status  VARCHAR)
    RETURNS VARCHAR
    LANGUAGE SQL
AS
 $$
    DECLARE
            message VARCHAR;
    BEGIN
        DELETE FROM employees WHERE status = :in_status;
        message := in_status ||' employees data deleted sucessfully’;
        RETURN message;
    END;
$$
;
```

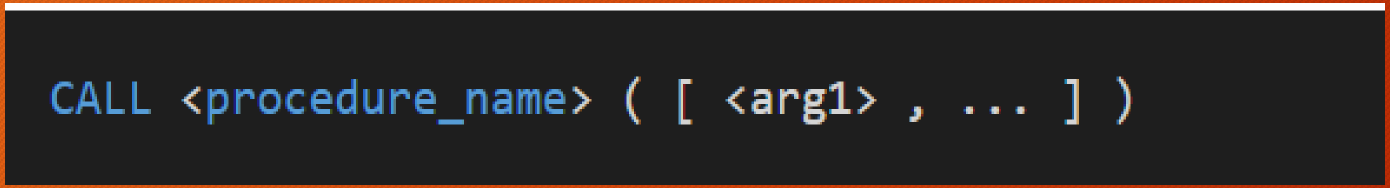
CONTINUE…

- The name of the procedure is purge_data_by_status and accepts input through a parameter named in_status of type varchar.
- The input parameter is used in the SQL statement which deletes the data from employees table. Prefix the input parameter with a colon (:in_status) to use in a SQL statement.
- The same input parameter is also used in the string value assigned to the message variable indicating records with which status are deleted.

Calling a Stored Procedure in Snowflake

- Use the CALL command to execute a stored procedure in Snowflake and MYSQL.

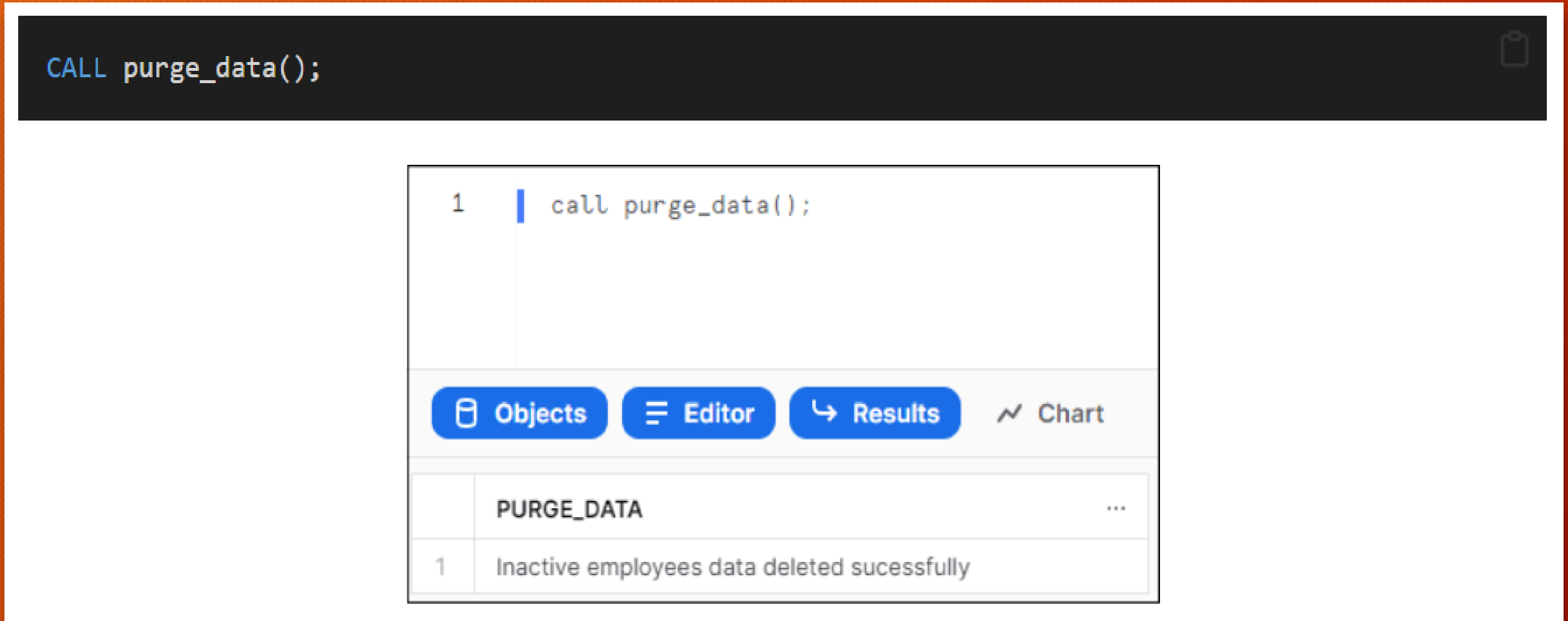
- The following is the syntax for to CALL command.



```
CALL <procedure_name> ( [ <arg1> , ... ] )
```

CONTINUE…

- The below image shows calling a stored procedure named purge_data and the output of the stored procedure call.



```
CALL purge_data();
```

```
call purge_data();
```

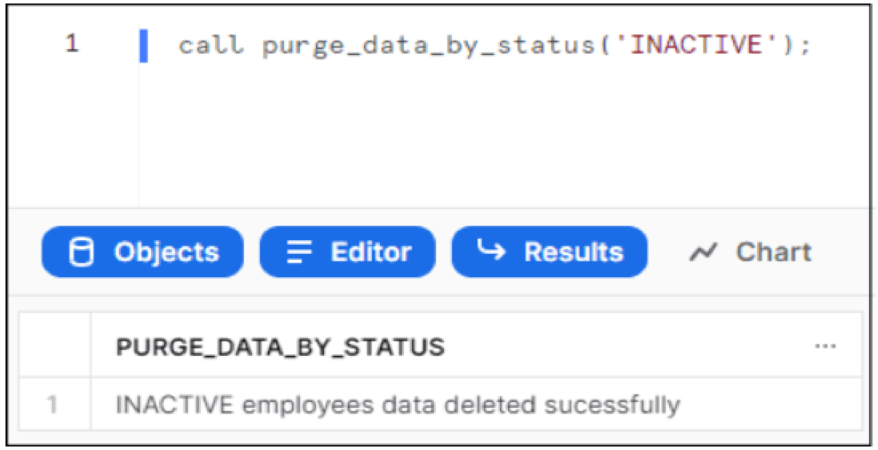
```
  PURGE_DATA
1 Inactive employees data deleted sucessfully
```

Call Stored Procedure without any Input Parameters

CONTINUE…

- The below image shows calling a stored procedure named purge_data_by_status with a string input parameter 'INACTIVE' and the output of the stored procedure call.

```
CALL purge_data_by_status('INACTIVE');
```



```
call purge_data_by_status('INACTIVE');
```

```
PURGE_DATA_BY_STATUS
1  INACTIVE employees data deleted sucessfully
```

Call Stored Procedure with Input Parameters

Usage Notes

- For all stored procedures:
- Stored procedures support overloading. Two procedures can have the same name if they have a different number of parameters or different data types for their parameters.
- Stored procedures are not atomic; if one statement in a stored procedure fails, the other statements in the stored procedure are not necessarily rolled back. For information about stored procedures and transactions, see Transaction Management.
- CREATE OR REPLACE <object> statements are atomic. That is, when the object is replaced, the old object deletion and the new object creation are processed in a single transaction.
- For JavaScript stored procedures:
- A JavaScript stored procedure can return only a single value, such as a string (for example, a success/failure indicator) or a number (for example, an error code). If you need to return more extensive information, you can return a VARCHAR that contains values separated by a delimiter (such as a comma), or a semi-structured data type, such as VARIANT.

Drawbacks of Using Stored Procedures

- If we use stored procedures, the memory of every connection that uses those stored procedures will increase substantially. Also, if we overuse many logical applications inside stored procedures, CPU usage will increase. It is because the database server is not well-designed for logical operations.

- Stored procedure constructs are not designed to develop complex and flexible business logic.

- It is challenging to debug stored procedures. Only a few database management systems allow us to debug stored procedures. Unfortunately, MySQL does not provide facilities for debugging stored procedures.

- It is not easy to develop and maintain stored procedures. Developing and maintaining stored procedures are often required a specialized skill set that not all application developers possess. It may lead to problems in both the application development and maintenance phases.