Snowflake

Stage,a location where data files are stored (staged) for loading into or unloading from Snowflake tables .

1.Internal Stage :  A storage location within Snowflake for staging data files before loading them into tables.

1.a Table Stage : An “**implicit storage location**” tied to a specific table for staging data files before loading them into that table.(@%)

|  |
| --- |
| Query to see Table Stages : show stages like ‘@%’; |

1.b User Stage: An “**implicit storage location**” tied to a specific user for staging data files can upload and manage their files for data loading and unloading(@~).

|  |
| --- |
| Query to see User Stages : show stages like ‘@~’; |

1.c Named Internal Stages : A database object created explicitly that acts as a temporary holding area for data files, offering flexibility for data loading and unloading, and allowing multiple users to access and manage data for various tables.

|  |
| --- |
| Creation of Named Internal Stages:  Method 1: Without File Format  CREATE OR REPLACE STAGE my\_internal\_stage  **internalStageParams**;  Method 2: With File Format  CREATE OR REPLACE STAGE my\_internal\_stage  **internalStageParams**  FILE\_FORMAT = (TYPE = 'CSV')  DIRECTORY = (ENABLE = TRUE);  Command to Stage Files to the Internal Stage  **PUT** file://<absolute\_path\_to\_file>/<filename> **internalStage**  Method 3: Cloning other Stage  CREATE OR REPLACE STAGE <name>  CLONE <source\_stage>; |

2.External Stage : An external stage in Snowflake is a reference to an external storage location (like AWS S3, Azure Blob Storage, or Google Cloud Storage) where data files are stored.

|  |
| --- |
| Creation of External Stage:  1.Using AWS S3  CREATE OR REPLACE STAGE my\_s3\_stage  URL = 's3://mybucket/path/'  CREDENTIALS=(AWS\_KEY\_ID='our\_aws\_key\_id') AWS\_SECRET\_KEY='your\_aws\_secret\_key')  FILE\_FORMAT = (TYPE = 'CSV');    a.URL of the S3 bucket and the path where the data files are stored.  b. AWS\_KEY\_ID unique identifier associated with your AWS account or IAM user.  c.AWS\_SECRET\_KEY is your AWS secret access key. This works like a password.  2.Using Azure Blob Storage    CREATE OR REPLACE STAGE my\_azure\_stage  URL ='azure://myaccount.blob.core.windows.net/mycontainer/path/'  CREDENTIALS = (AZURE\_SAS\_TOKEN='your\_sas\_token')  FILE\_FORMAT = (TYPE = 'CSV');  a. A Shared Access Signature (SAS) token is a URI that grants restricted access rights to Azure Storage resources.  b. URL of the Azure Blob Storage container and the path where the data files are stored. |

**Storage Integration**: A configuration that securely connects Snowflake to external cloud storage using IAM entities, allows users to access storage locations without directly supplying credentials when creating stages or loading/unloading data.

|  |
| --- |
| Creation of External Stages using Storage Integration:  1.Using AWS S3:   1. Creation of Storage Integration:   CREATE OR REPLACE STORAGE INTEGRATION s3\_int  TYPE = EXTERNAL\_STAGE  STORAGE\_PROVIDER = 'S3'  ENABLED = TRUE  STORAGE\_AWS\_ROLE\_ARN = '<iam\_role>'  STORAGE\_ALLOWED\_LOCATIONS=('s3://bucket/path/', 's3://bucket/path2/');   1. Creation of External Stage:   CREATE OR REPLACE STAGE my\_s3\_stage  STORAGE\_INTEGRATION = s3\_int  URL = 's3://bucket1/path1'  FILE\_FORMAT = my\_csv\_format; |

|  |
| --- |
| 2. Using Azure Blob Storage:   1. Creation of Storage Integration:   CREATE OR REPLACE STORAGE INTEGRATION azure\_int  TYPE = EXTERNAL\_STAGE  STORAGE\_PROVIDER = 'AZURE'  ENABLED = TRUE  AZURE\_TENANT\_ID = '<tenant\_id>'  STORAGE\_ALLOWED\_LOCATIONS= ('azure://account.blob.core.windows.net/container/path/', 'azure://account.blob.core.windows.net/container/path2/');  a. The Tenant ID is used to identify your Azure AD instance. It is essential for authentication and authorization processes, by navigating to Azure Active Directory > Properties. The Tenant ID is listed in the Basic information.     1. Creation of External Stage   CREATE OR REPLACE STAGE my\_azure\_stage  STORAGE\_INTEGRATION = azure\_int  URL = 'azure://myaccount.blob.core.windows.net/container1/path1'  FILE\_FORMAT = my\_csv\_format; |

File Format : A file format defines the structure and format of data files that you want to load into or unload from Snowflake tables.

|  |
| --- |
| Creation of File Format :  CREATE OR REPLACE FILE FORMAT <Format\_Name>  TYPE = 'CSV' | 'JSON' | 'PARQUET'  **[FormatTypeOptions:]**  CREATE OR REPLACE FILE FORMAT my\_csv\_format  TYPE = 'CSV'  FIELD\_DELIMITER = ','  SKIP\_HEADER = 1  NULL\_IF = ('NULL', 'null'); |

AVRO,ORC,XML are the file formats which are not supported for data unloading from snowflake tables.

**Copy Into Command** : COPY INTO command in Snowflake enables loading data from files in a stage location to an existing table or unloading data from a table into a stage.

Copy Into <Table> : Loads data from files to an existing table .

|  |
| --- |
| Copy Into :  COPY INTO <target\_table>  FROM @stage\_name/file\_path  FILE\_FORMAT = (FORMAT\_NAME = 'my\_file\_format' | TYPE = 'CSV' FIELD\_OPTIONALLY\_ENCLOSED\_BY = '"')  [ copyOptions ]; |

|  |
| --- |
| Copy Into with Validation mode :  COPY INTO <table\_name>  FROM <stage>/file\_path  [ FILES = ( '<file\_name>' [ , '<file\_name>' ] [ , ... ] ) ]  [ FILE\_FORMAT = ( { FORMAT\_NAME = '<file\_format\_name>' | TYPE = { CSV | JSON | AVRO | ORC | PARQUET | XML } [ formatTypeOptions ] } ) ]  [ COPY\_OPTIONS ]  [ VALIDATION\_MODE = { RETURN\_<n>\_ROWS | RETURN\_ERRORS | RETURN\_ALL\_ERRORS } ]; |

VALIDATION\_MODE : Validates the data and returns rows with errors without loading them into the table. This helps in identifying and handling errors before loading them.

Copy Options :

1.ON\_ERROR=CONTINUE|SKIP\_FILE|SKIP\_FILE\_num|'SKIP\_FILE\_num'| ABORT\_STATEMENT.

2. SIZE\_LIMIT = num , Number (> 0) that specifies the maximum size (in bytes) of data to be loaded for a given COPY statement.

3. PURGE = TRUE | FALSE , Boolean that specifies whether to remove the data files from the stage automatically after the data is loaded successfully.

4. FORCE = TRUE | FALSE , Boolean that specifies to load all files, regardless of whether they’ve been loaded previously and have not changed since they were loaded.

5.MATCH\_BY\_COLUMN\_NAME=CASE\_SENSITIVE|CASE\_INSENSITIVE | NONE , String that specifies whether to load semi-structured data into columns in the target table that match corresponding columns represented in the data.

Copy Into <Location> : Unloads data from a table (or query) into one or more files in stages or locations.

|  |
| --- |
| COPY INTO { internalStage | externalStage | externalLocation }  FROM { [<namespace>.]<table\_name> | ( <query> ) }  [ PARTITION BY <expr> ]  [FILE\_FORMAT = ( { FORMAT\_NAME = '[<namespace>.]<file\_format\_name>' |TYPE = { CSV | JSON | PARQUET } [ formatTypeOptions ] } ) ]  [ copyOptions ]  [ VALIDATION\_MODE = RETURN\_ROWS ] |

Copy Options :

1.OVERWRITE = TRUE | FALSE , Boolean that specifies whether the COPY command overwrites existing files with matching names.

2. SINGLE = TRUE | FALSE , Boolean that specifies whether to generate a single file or multiple files. If FALSE, a filename prefix must be included in path.

3. MAX\_FILE\_SIZE = num , that specifies the upper size limit (in bytes) of each file to be generated in parallel per thread.

Error Logs Handling in Copy Command : Using Copy Command in Validation helps us to maintain error handling and the error details can be stored.

RESULT\_SCAN : RESULT\_SCAN function allows you to process the output of a previous query.

|  |
| --- |
| Steps :  1. CREATE OR REPLACE TABLE TEST\_TABLE  (  EMP\_ID NUMBER,  NAME VARCHAR(),  DEPT VARCHAR(),  SALARY NUMBER  );  2. CREATE OR REPLACE TABLE ERROR\_LOG (  ERROR\_ROW VARCHAR(),  ERROR\_MESSAGE STRING,  FILE\_NAME STRING  );  3. COPY INTO TEST\_TABLE  FROM '@"TEST\_PURPOSE"."TEST\_PURPOSE\_SCHEMA"."MY\_TEST\_STAGE"/empdata.csv'  FILE\_FORMAT = (FORMAT\_NAME='MY\_TEST\_FORMAT')  VALIDATION\_MODE=RETURN\_ERRORS;  4. INSERT INTO ERROR\_LOG (ERROR\_ROW, ERROR\_MESSAGE, FILE\_NAME)  SELECT  T.$3 AS ERROR\_ROW,  T.$1 AS ERROR\_MESSAGE,  T.$2 AS FILE\_NAME  FROM TABLE(RESULT\_SCAN(LAST\_QUERY\_ID())) T;  5.SELECT \* FROM ERROR\_LOG;  A white rectangular object with a black border  AI-generated content may be incorrect. |

SNOW PIPE : Snowpipe enables loading data from files as soon as they’re available in a stage. This means you can load data from files in micro-batches, making it available to users within minutes, rather than manually executing COPY statements on a schedule to load larger batches.

Snowpipe is serverless. This means that Snowflake manages the compute resources required for data loading automatically.

|  |
| --- |
| Creation of PIPE :  CREATE [ OR REPLACE ] PIPE [ IF NOT EXISTS ] <name>  [ AUTO\_INGEST = [ TRUE | FALSE ] ]  [ ERROR\_INTEGRATION = <integration\_name> ]  [ AWS\_SNS\_TOPIC = '<string>' ]  [ INTEGRATION = '<string>' ]  [ COMMENT = '<string\_literal>' ]  AS <copy\_statement>;  PIPE with Copy Command :  CREATE OR REPLACE PIPE my\_pipe  AUTO\_INGEST = TRUE  AWS\_SNS\_TOPIC = 'arn:aws:sns:us-west-2:123456789012:my\_sns\_topic'  INTEGRATION = 'my\_s3\_integration'  COMMENT = 'Pipe for loading data from S3'  AS  COPY INTO my\_table  FROM @my\_s3\_stage  FILE\_FORMAT = (TYPE = 'CSV'); |

1.Create an S3 Bucket: Store your data files in an S3 bucket.

2.Configure SNS Topic: Set up an SNS topic to send notifications when new files are added.

3.Create Storage Integration: Create a storage integration in Snowflake to access the S3 bucket.

4.Create Pipe: Use the CREATE PIPE statement to define the pipe with the SNS topic and storage integration.

Streams : A stream object records data manipulation language (DML) changes made to tables, including inserts, updates, and deletes, as well as metadata about each change, so that actions can be taken using the changed data. This process is referred to as change data capture (CDC).

|  |
| --- |
| Creation of Stream :  CREATE OR REPLACE TABLE employees  (  id NUMBER,  name VARCHAR,  salary NUMBER  );  CREATE OR REPLACE STREAM employee\_stream ON TABLE employees; |

Offset Storage : When a stream is created, it takes an initial snapshot of the source table and records an offset.

Table Versioning : Each transaction that includes DML statements creates a new table version. Streams track changes between these versions, allowing you to query the changes made between specific points in time.

Repeatable Read Isolation : Streams support repeatable read isolation, meaning multiple SQL statements within a transaction see the same set of records in a stream. This ensures consistency.

Types of Streams :

1. Standard : A standard (i.e. delta) stream tracks all DML changes to the source object, including inserts, updates, and deletes .Supports standard tables, dynamic tables, Snowflake-managed Apache Iceberg tables, directory tables, or views.

2. Append – Only : Supported for streams on standard tables, dynamic tables, Snowflake-managed Apache Iceberg tables, or views. An append-only stream exclusively tracks row inserts.

3.Insert-Only : Supported for streams on Apache Iceberg™ or external tables. An insert-only stream tracks row inserts only.

|  |
| --- |
| 1.Standard Stream :  CREATE OR REPLACE STREAM my\_standard\_stream ON TABLE my\_table;  2.Append-Only :  CREATE OR REPLACE STREAM my\_append\_only\_stream ON TABLE my\_table APPEND\_ONLY = TRUE;  3. Insert-Only :  CREATE OR REPLACE STREAM my\_insert\_only\_stream ON TABLE my\_table INSERT\_ONLY = TRUE; |

TASKS : A Snowflake Task is a scheduled job that automates the execution of SQL statements, stored procedures, or scripts at specified intervals.

Use tasks to automate, schedule, and optimize business procedures on your data pipeline.

|  |
| --- |
| Creation of Task :   1. CREATE TASK my\_interval\_task   WAREHOUSE = 'my\_warehouse'  SCHEDULE = '5 MINUTES'  AS  INSERT INTO my\_table (column1, column2)  SELECT column1, column2 FROM another\_table; |

Severless Tasks : Tasks execute on Snowflake-managed compute. Snowflake automatically resizes the resources as required for each workload. This model is recommended for tasks that always start at the same time and tasks that only run when new data arrives.

|  |
| --- |
| Creation of Severless Task :   1. CREATE TASK my\_interval\_task   SCHEDULE = '5 MINUTES'  AS  INSERT INTO my\_table (column1, column2)  SELECT column1, column2 FROM another\_table; |

|  |
| --- |
| Different ways for creation of Tasks :   1. CREATE TASK my\_interval\_task   WAREHOUSE = 'my\_warehouse'  SCHEDULE = '5 MINUTES'  AS  INSERT INTO my\_table (column1, column2)  SELECT column1, column2 FROM another\_table;   1. CREATE OR REPLACE TASK every\_5\_minutes\_task   WAREHOUSE = 'my\_warehouse'  SCHEDULE = 'USING CRON \*/5 \* \* \* \* UTC'  AS  INSERT INTO my\_table (column1, column2)  SELECT column1, column2 FROM another\_table;   1. CREATE TASK my\_when\_task   WAREHOUSE = 'my\_warehouse'  WHEN SYSTEM$STREAM\_HAS\_DATA('my\_stream')  AS  INSERT INTO my\_table (column1, column2)  SELECT column1, column2 FROM another\_table; |

Task Graphs : Task graphs in Snowflake, also known as directed acyclic graphs (DAGs), allow you to manage dependencies and automate sequences of tasks.

A task graph is a series of tasks organized by their dependencies.

Root Task: The initial task that triggers the sequence.

Child Tasks: Subsequent tasks that depend on the completion of parent tasks.

|  |
| --- |
| Creation of Task Graphs:  CREATE OR REPLACE TASK root\_task  WAREHOUSE = 'my\_warehouse'  SCHEDULE = 'USING CRON 0 0 \* \* \* UTC' -- Runs daily at midnight UTC  AS  CALL my\_stored\_procedure();  -- Child Task 1  CREATE OR REPLACE TASK child\_task\_1  WAREHOUSE = 'my\_warehouse'  AFTER root\_task  AS  INSERT INTO my\_table (column1, column2)  SELECT column1, column2 FROM another\_table;  -- Child Task 2  CREATE OR REPLACE TASK child\_task\_2  WAREHOUSE = 'my\_warehouse'  AFTER child\_task\_1  AS  UPDATE my\_table SET column3 = 'updated\_value';  -- Child Task 3  CREATE OR REPLACE TASK child\_task\_3  WAREHOUSE = 'my\_warehouse'  AFTER child\_task\_1, child\_task\_2  AS  DELETE FROM my\_table WHERE column1 IS NULL; |

Single Direction Flow: Tasks flow in one direction; a task later in the sequence cannot trigger an earlier task1.

Dependencies: Each task can have multiple parent and child tasks.

A task graph is limited to a maximum of 1000 tasks.

A single task can have a maximum of 100 parent tasks and 100 child tasks.

When tasks run in parallel on the same user-managed warehouse, the compute resources must be sized to handle the concurrent task runs.

TIME TRAVEL : Snowflake Time Travel is a powerful feature that allows you to access historical data at any point within a defined retention period.

The data retention period is the duration for which historical data is preserved, allowing you to use Time Travel features.

Access Historical Data: Query data as it existed at a specific point in the past.

Data Recovery: Restore dropped tables, schemas, and databases.

Cloning: Create clones of tables, schemas, or databases at a specific historical point.

The retention period defines how long historical data is preserved. It varies based on the Snowflake edition:

Standard Edition: Up to 1 day.

Enterprise Edition and higher: Up to 90 days.

|  |
| --- |
| 1.SELECT \* FROM my\_table AT (TIMESTAMP => '2025-04-01 12:00:00');  2. SELECT \* FROM my\_table AT (OFFSET => -3600); 1 hour ago  3. SELECT \* FROM my\_table BEFORE (TIMESTAMP => '2025-04-01 12:00:00');  4. SELECT \* FROM my\_table BEFORE(STATEMENT => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726'); |

FAIL SAFE : Fail-safe provides a (non-configurable) 7-day period during which historical data may be recoverable by Snowflake. This period starts immediately after the Time Travel retention period ends.