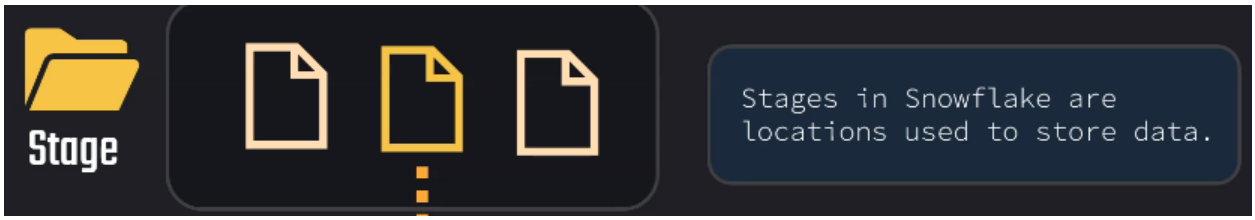
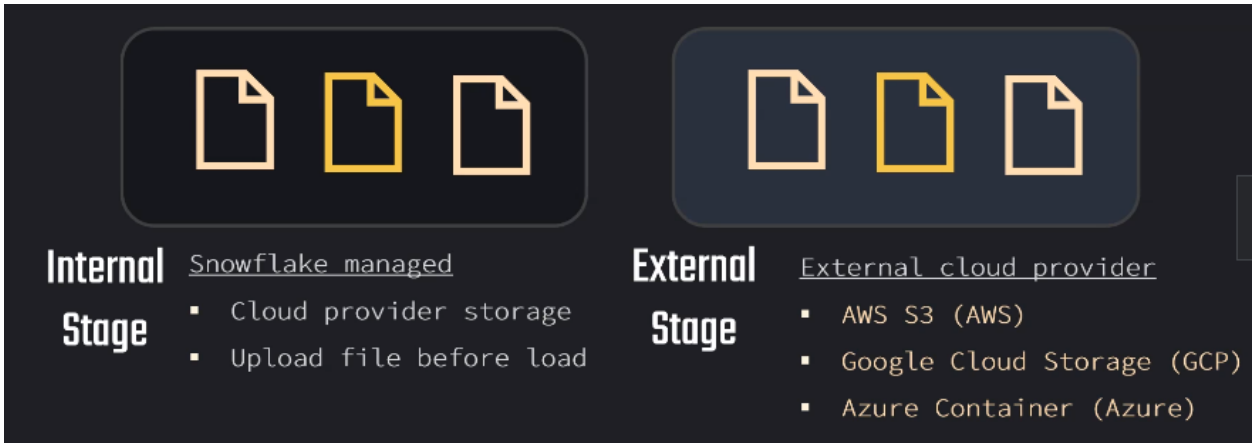
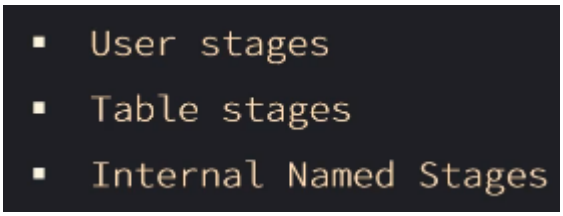
Internal Stage

1. 

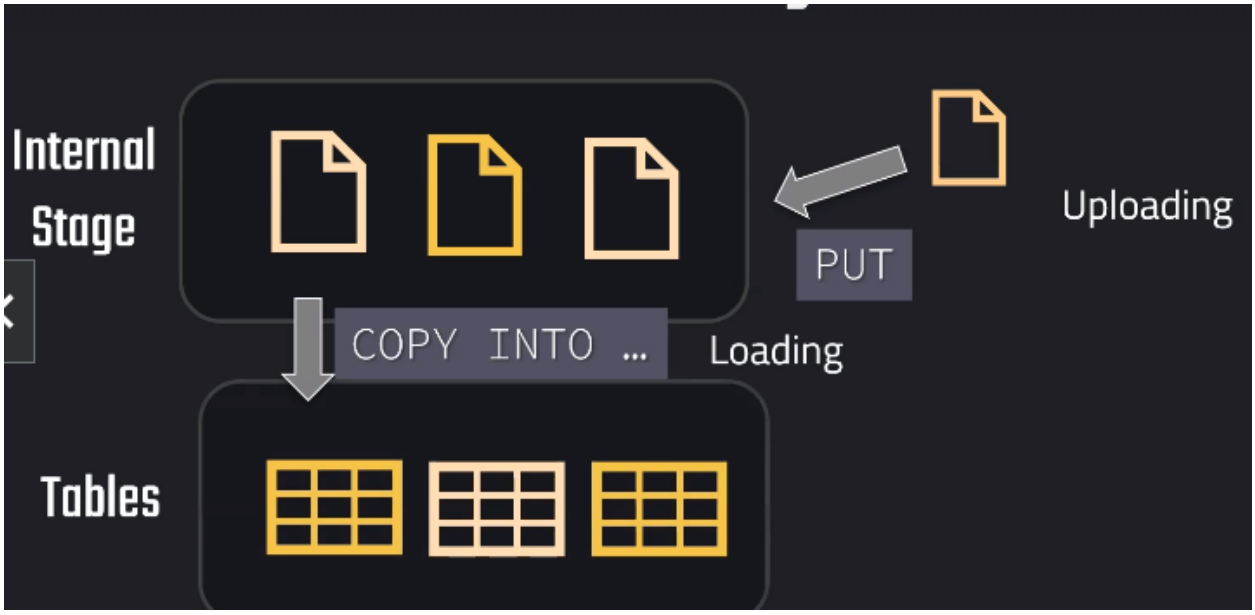
2. So whenever we want to load some data into a table, we need to have the files first available in a stage.
3. We can also send some files(tables) into stage…it is called unloading
4. This stage is not the stage in data warehouse..it is completely separate in snowflake
5. So we have 2 types of stage



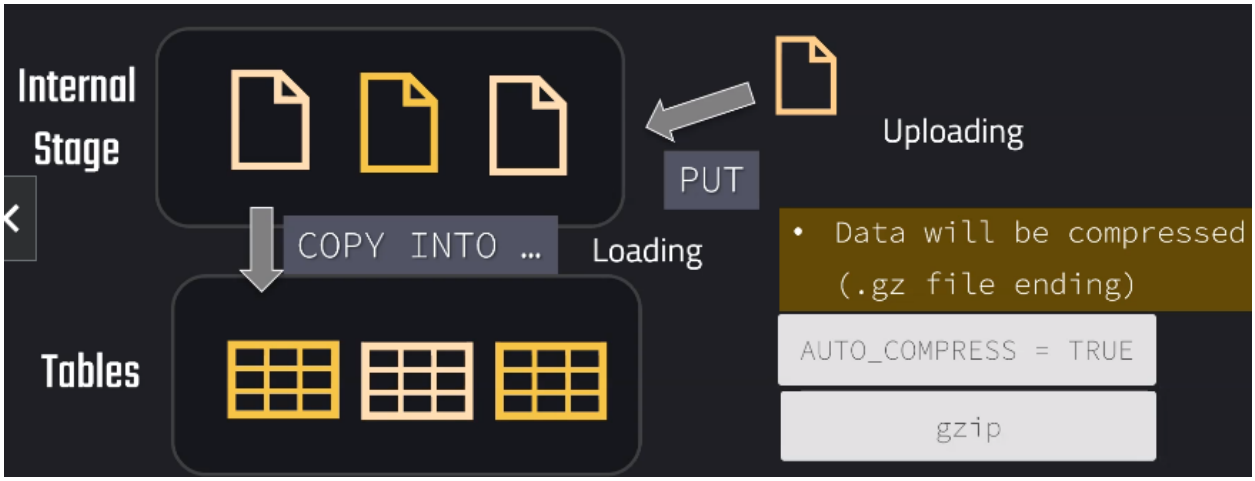
6. Coming to internal stage..it has 3 diff types



7. Internal stages
8. if we want to copy some data into a table, we first need to upload the file into this internal stage.
9. We can do this by connecting to SQL and using the Put command, and then we can use the copy into command to load the data into our tables.

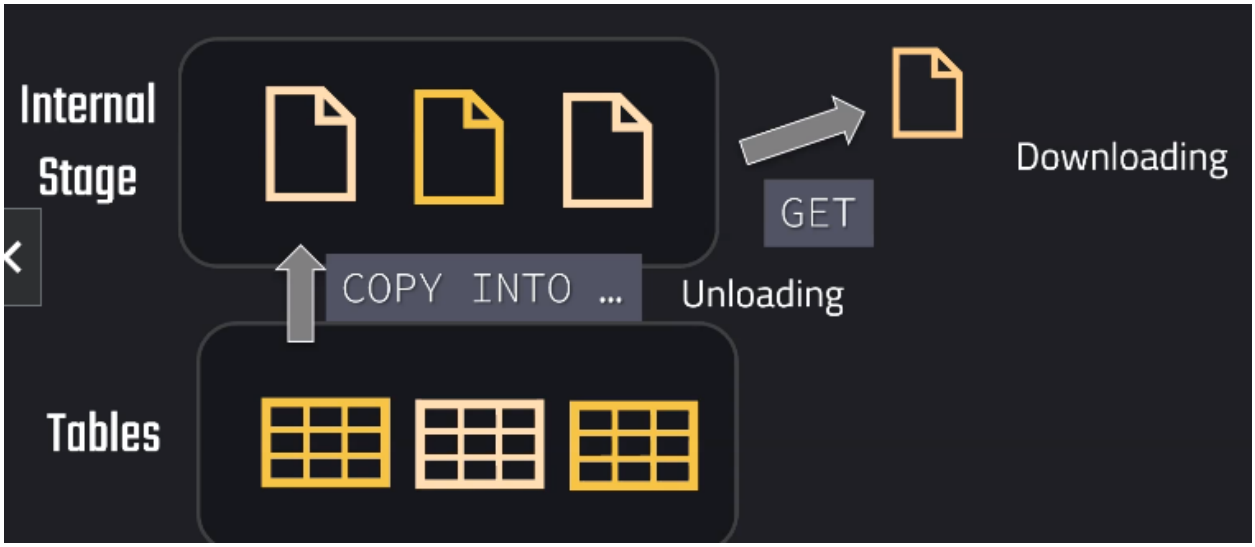
10. 

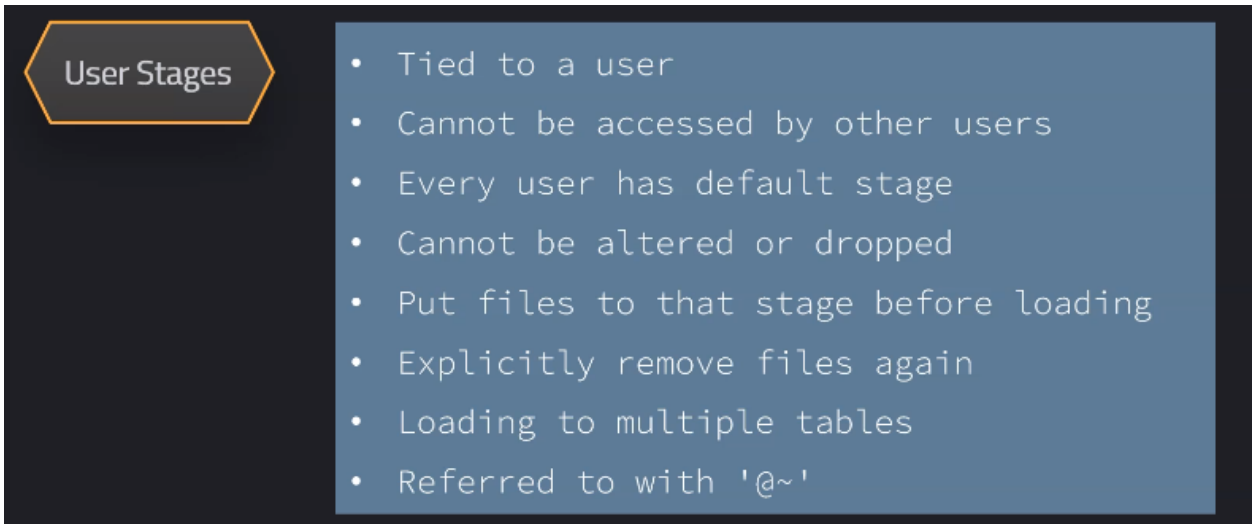
11. Put command will automatically compress the data..and data is automatically encrypted with 128bit or 256bit key



12. If we want to download the data(table)..first we have to copy our data into ..internal stage…which is called unloading..

13. After unloading…we can download data using GET command



14. User stage
15. So this can be used when we want to put some files before loading them into a table and we don't have
16. 

17. Table stage

18. A table stage in Snowflake is a special type of internal stage that is associated with a specific table. Table stages are created automatically when a table is created, and they are used to store data files that are being loaded into the table.

Characteristics of table stages:

- Table stages are referenced using the `@%` prefix. For example, to list the files in the table stage for a table named `my_table`, you would use the following command: `LIST @%my_table`
- Table stages cannot be altered or dropped.
- Table stages do not support setting file format options. Instead, you must specify file format and copy options as part of the `COPY INTO <table>` command.

Limitations of table stages:

- Table stages can only be accessed by users who have the `OWNERSHIP` privilege on the table.
- Table stages are not suitable for storing large files or files that need to be accessed by multiple users.

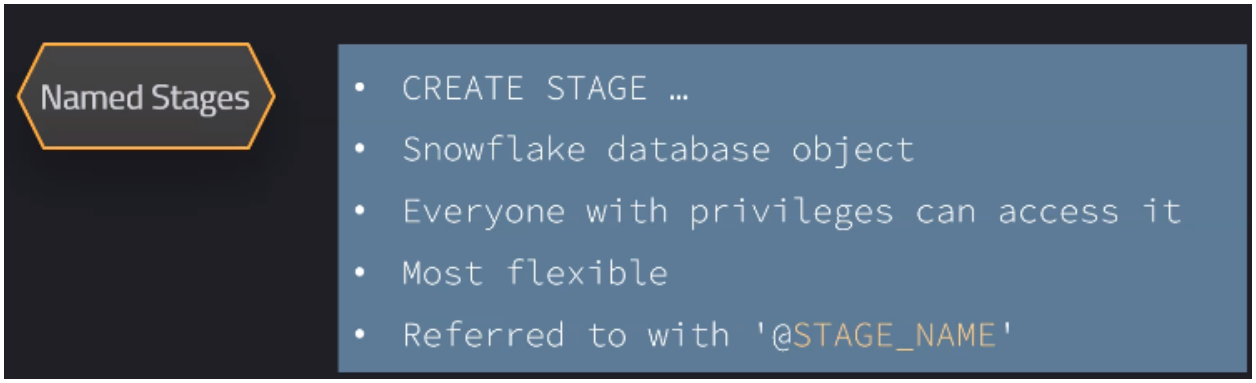
19. **Use cases for table stages:**

- Loading data from one or more files into a single table.
- Staging files that are being processed by a Snowflake job that loads data into a table.

Examples of using a table stage:

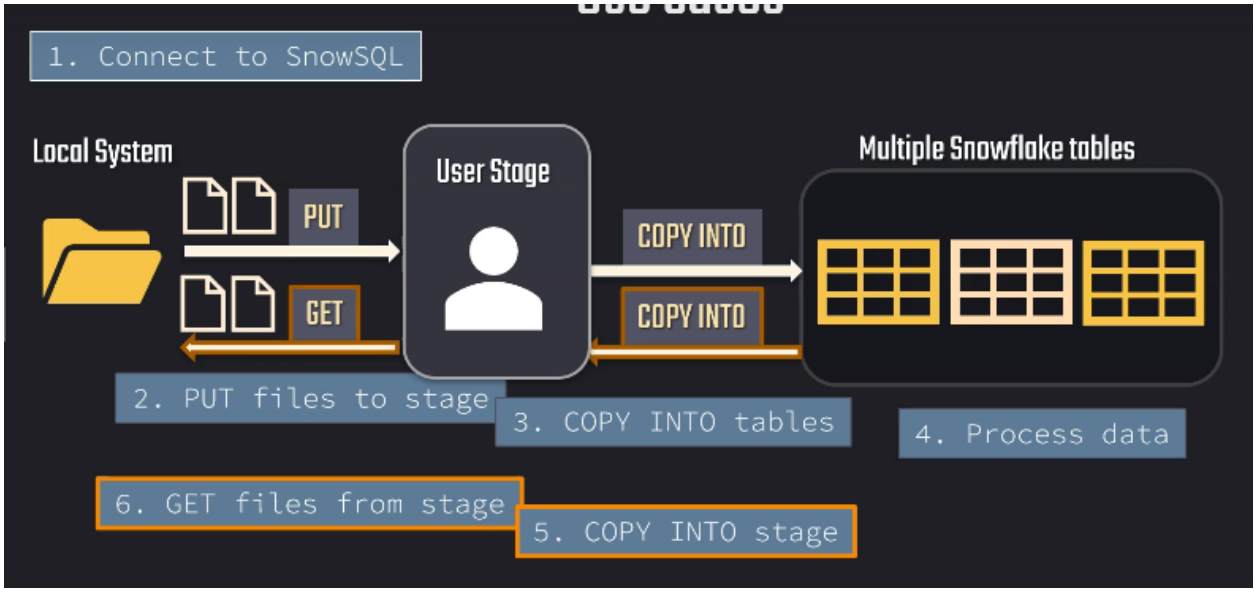
```
-- Create a table and its associated table stage.
CREATE TABLE my_table (
  column1 INT,
  column2 STRING
);

-- Put a data file into the table stage.
PUT @%my_table/myfile.csv;

-- Copy the data from the table stage into the table.
COPY INTO my_table FROM @%my_table/myfile.csv;
```

20. Named stage

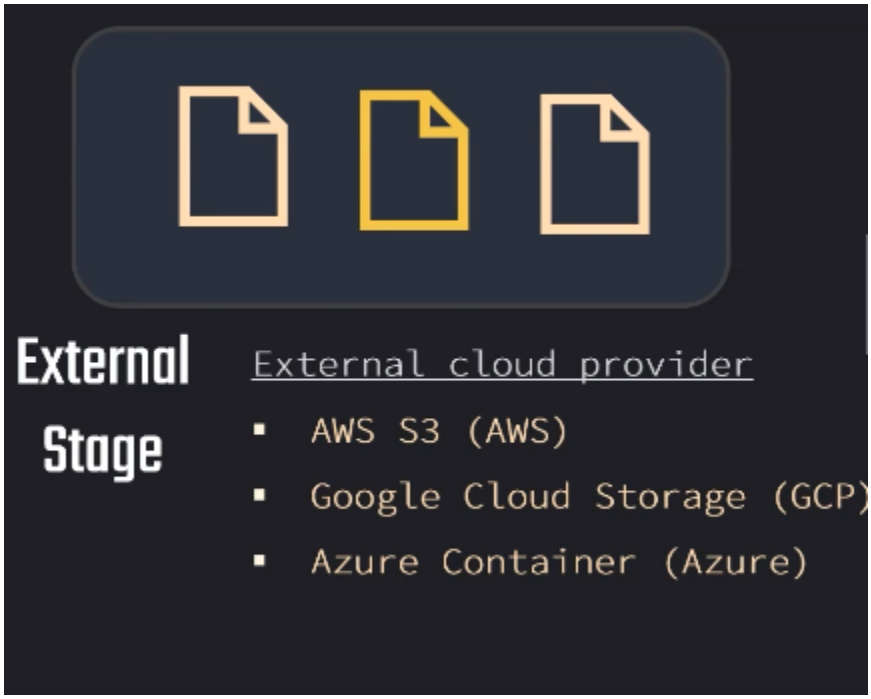
21. 

Type of internal stage	Characteristics	Limitations
User stage	Temporary, can only be accessed by the user who created it	Cannot be altered or dropped, does not support setting file format options
Table stage	Temporary, associated with a specific table, can only be accessed by users with the `OWNERSHIP` privilege on the table	Cannot be altered or dropped, does not support setting file format options
Named internal stage	Permanent, user-defined, can be accessed by users with the `USAGE` privilege on the stage	Subject to Snowflake's storage quotas

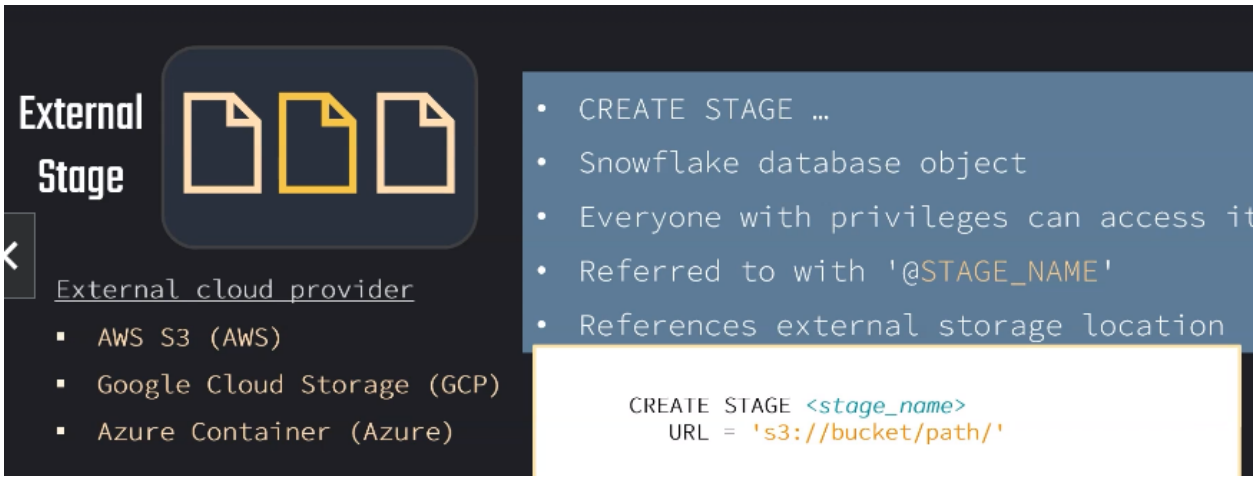
22.
23. Use cases
24. We can get transformed/compressed table if we load our data into snowflake table ..and then by retrieving it

25. 

External Stage

1. 

2. It is similar named stage in internal stages…just the diff is ..we have to add region location(URL) while using external stage

3. 

4. AWS we use s3 bucket as URL
5. For azure we use

```
CREATE STAGE <stage_name>
URL = 'azure://account.blob.core.windows.net/container/path/'
```

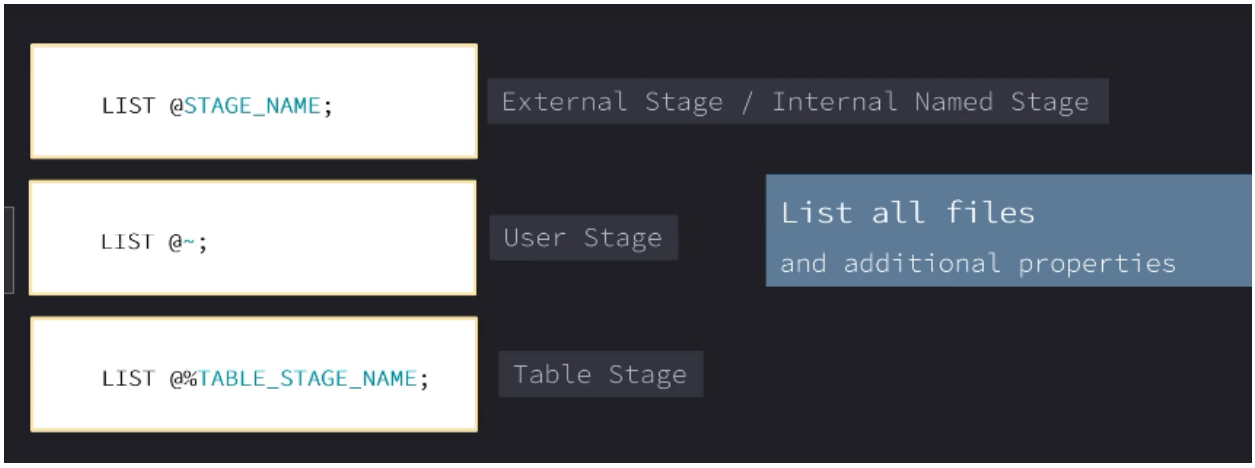
6. We can also add keys(to access data in s3 or azure) while creating stages..but it is not

```
CREATE STAGE <stage_name>
URL = 's3://bucket/path/'
CREDENTIALS = …
FILE_FORMAT = …
```

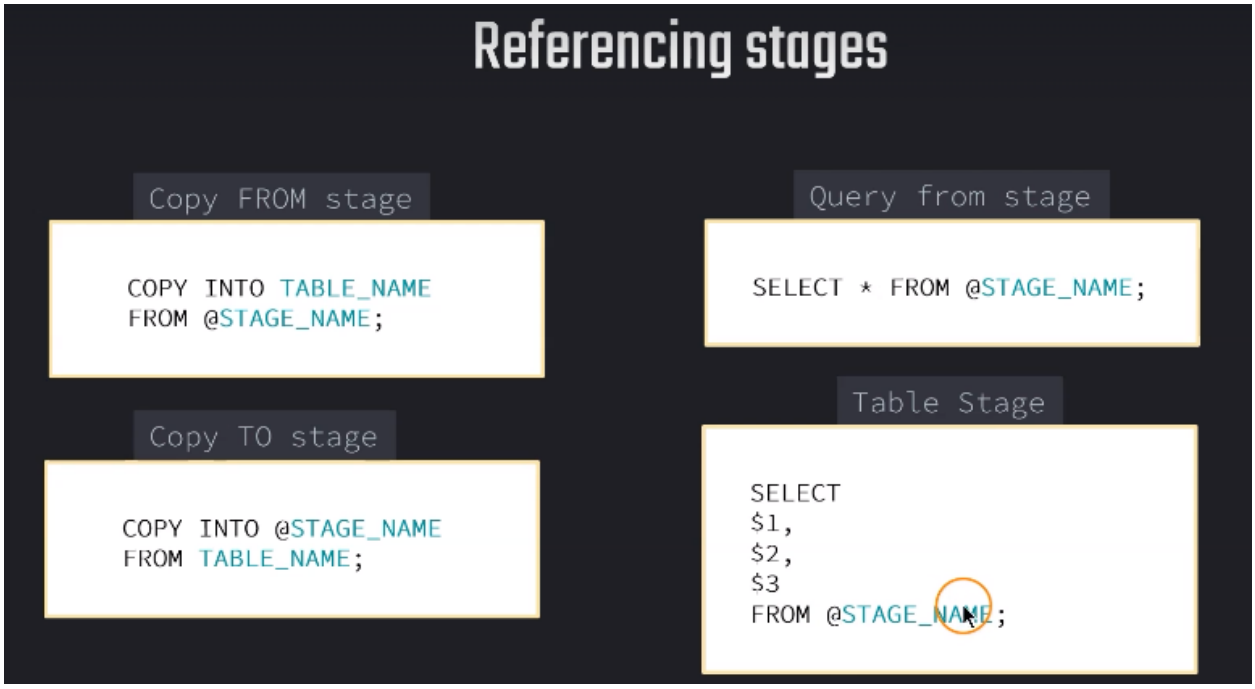
recommended
7. Instead we can use storage integration…which stores these credentials to a separate

```
CREATE STAGE <stage_name>
URL = 's3://bucket/path/'
STORAGE_INTEGRATION = …
FILE_FORMAT = …
```

object
8. If we want to see files in our stages…then we have to use LIST …



9. If we want to load the data from the stages..we use copy into command

10. 

11. If we want to query columns from stage..we use $col_name..see pic

Stage HandsOn

1.
```
// Show all named stages
SHOW STAGES;

// List files in user stage;
LIST @~;

// List files in table stage;
LIST @%LOAN_PAYMENT;
```
2. Next we have created a new db for stage

```
// Database to manage stage objects, fileformats etc.

CREATE OR REPLACE DATABASE manage_db;

CREATE OR REPLACE SCHEMA external_stages;
```

3. Using replace is not mandatory..
4. Next we created external stage…and gave credentials..but it is not recommended method
5.
```
CREATE OR REPLACE STAGE manage_db.external_stages.aws_stage
    url='s3://bucketsnowflakes3'
    credentials=(aws_key_id='ABCD_DUMMY_ID' aws_secret_key='1234abcd_key');
```

6. To get the description of external stage..we use

```
// Description of external stage

DESC STAGE manage_db.external_stages.aws_stage;
```

7. To alter external stage

```
// Alter external stage

ALTER STAGE aws_stage
    SET credentials=(aws_key_id='XYZ_DUMMY_ID' aws_secret_key='987xyz');
```

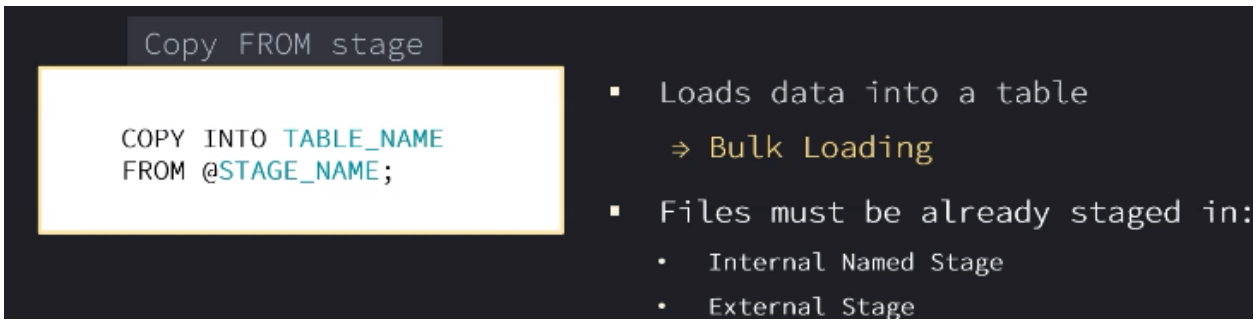
8. As we already have some files in aws cloud…if we use LIST @aws_stage ..we get

```
43  LIST @aws_stage;
44
45
46
47
```

Objects | Editor | Results | Chart

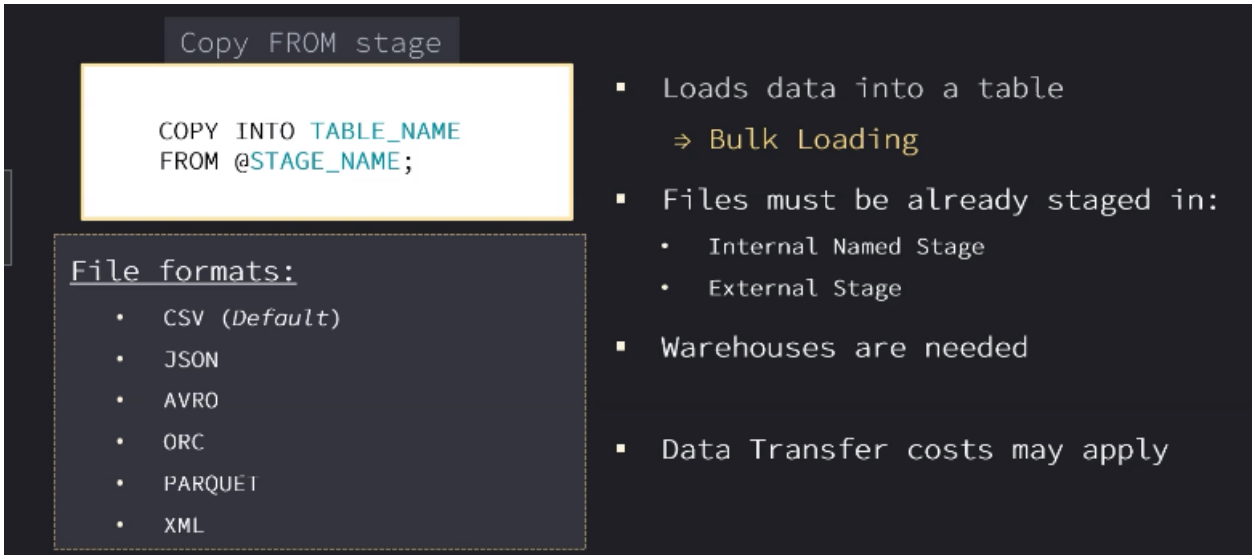
	name	size	md5
1	s3://bucketsnowflakes3/Loan_payments_data.csv	44,417	f354864a37b2dbec4951900
2	s3://bucketsnowflakes3/OrderDetails.csv	54,600	1a1c4a47a8e8e43ecef5bf8a
3	s3://bucketsnowflakes3/sampledata.csv	158	462259b257e0238ff9f9fb08

COPY INTO

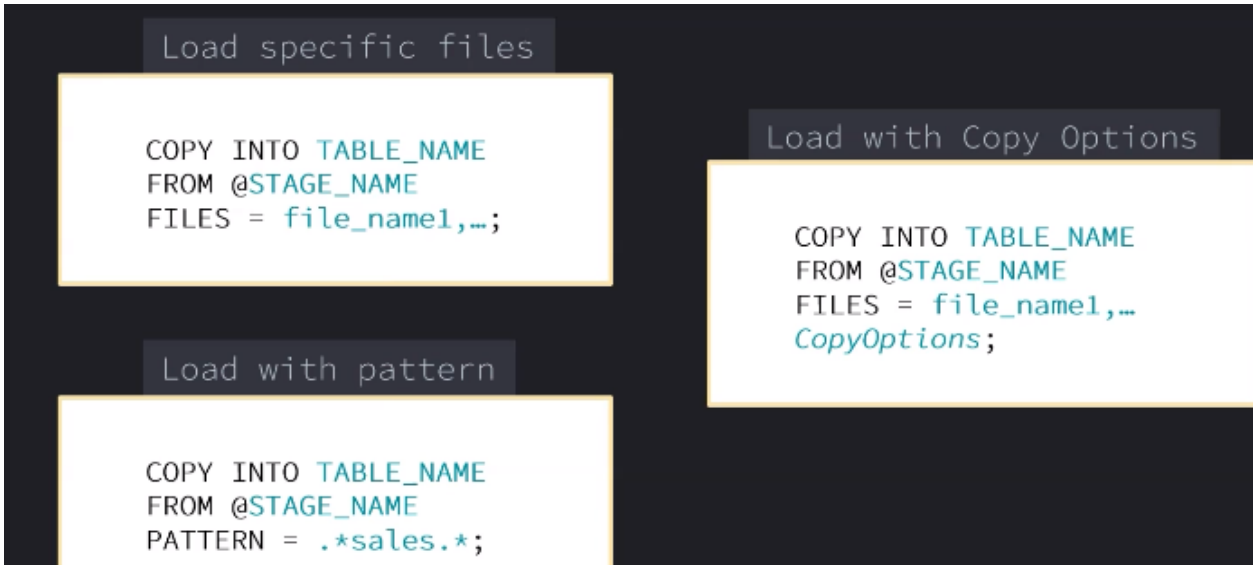
1. This is the most important command to load the data into our tables.
2. 

3. And we need a warehouse for computing the commands

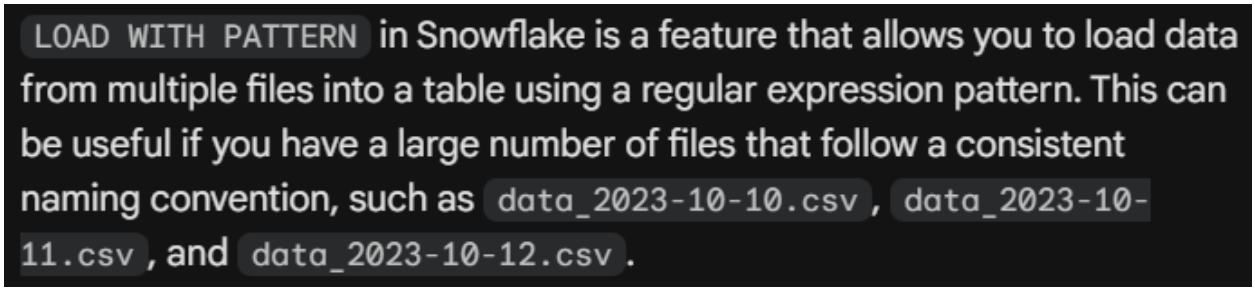
4. Some costs may apply if the location where we load the data from is a different region or a different cloud provider

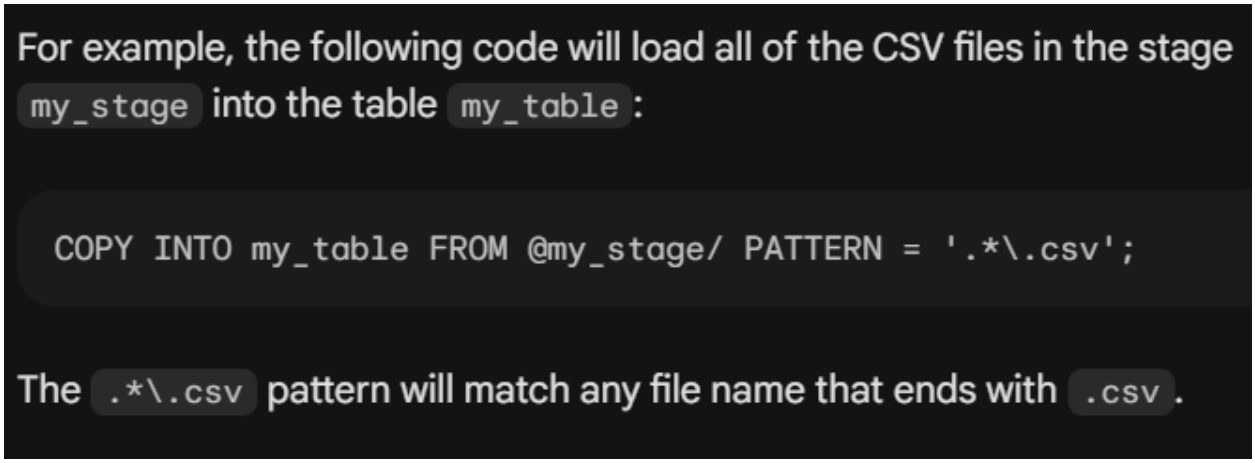


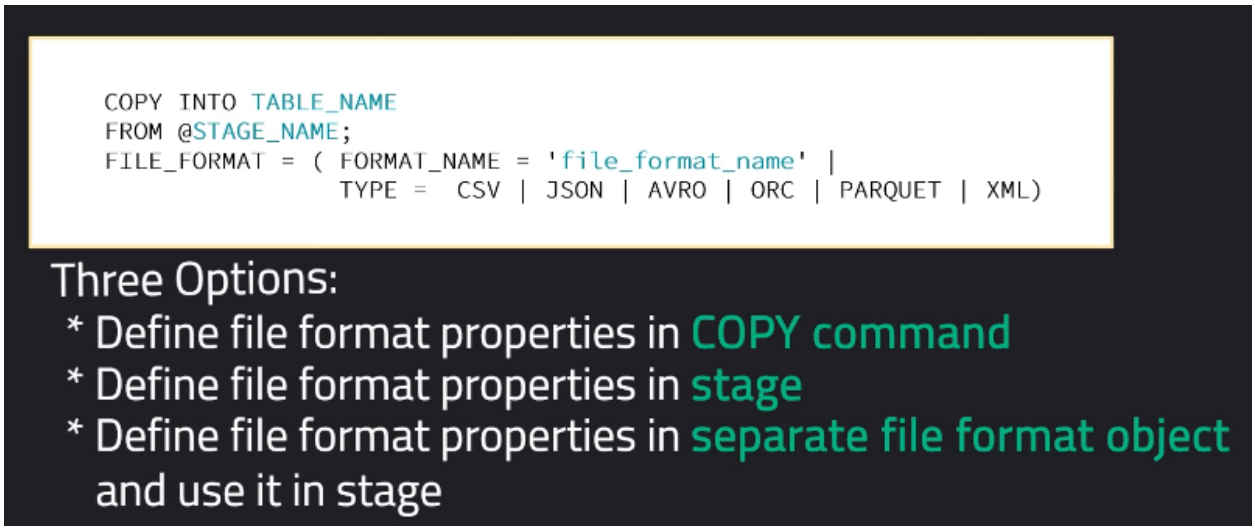
5. Copy into options



6. Load with pattern





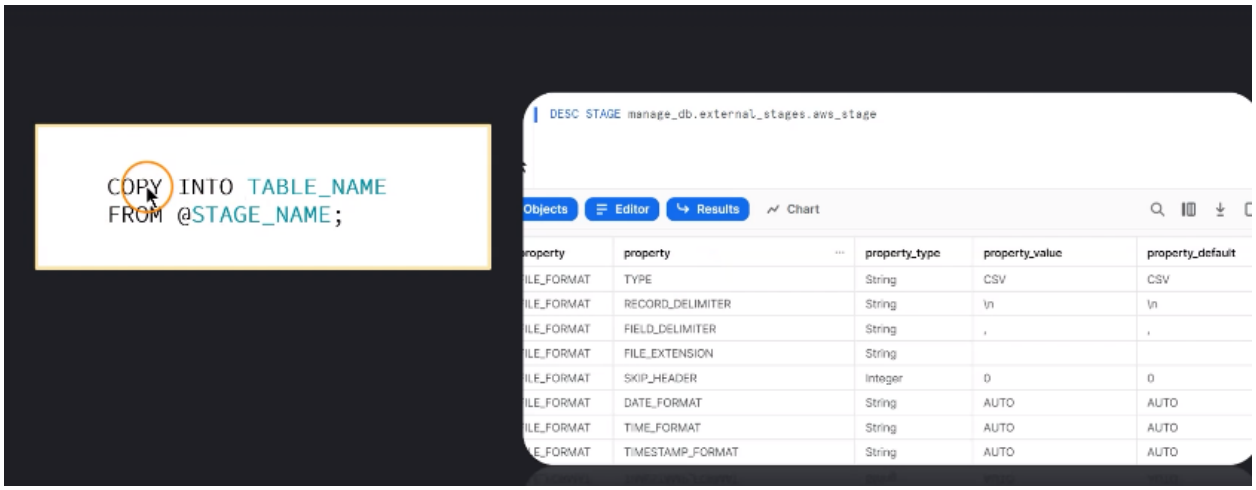
7. 

8. For hands on refer resources

FILE FORMAT

1. We need to specify the file format..of the file..which we are going to copy into the table from stage

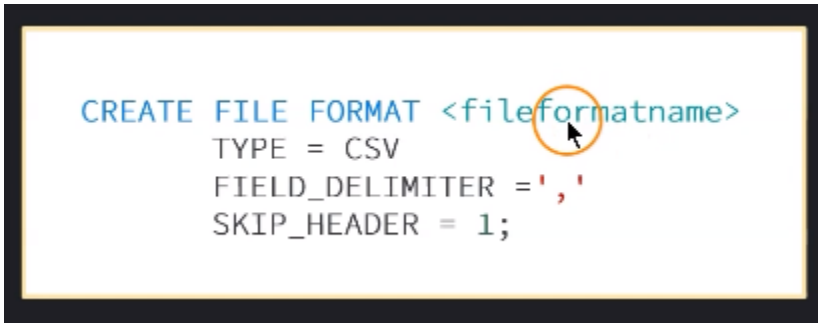
2. If do not specify any fule type..default will be CSV..if the file is not CSV..then it throws error



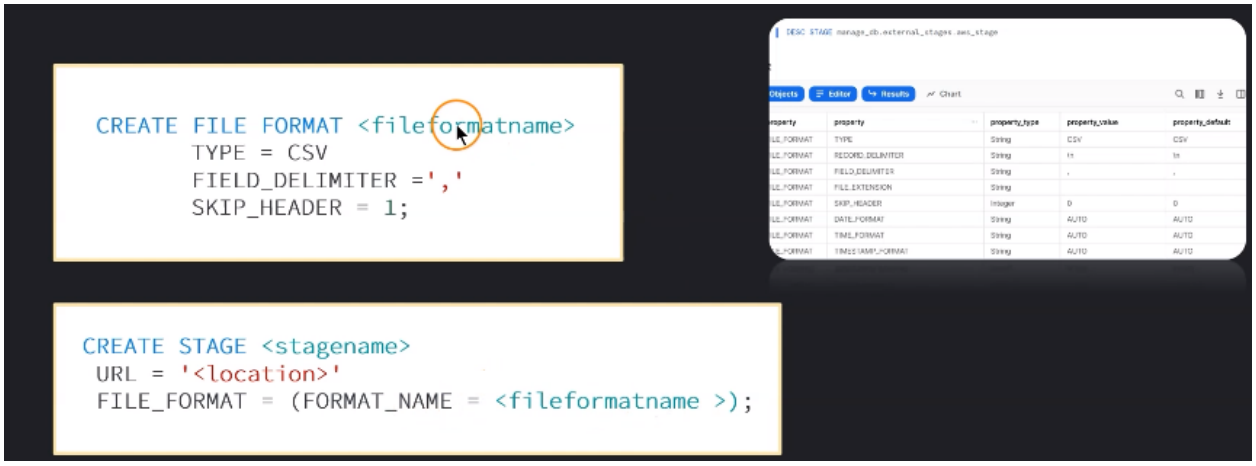
3. To get rid of error..we have to specify the file format in our command

```
COPY INTO TABLE_NAME
FROM @STAGE_NAME
FILE_FORMAT = (TYPE = CSV …);
```

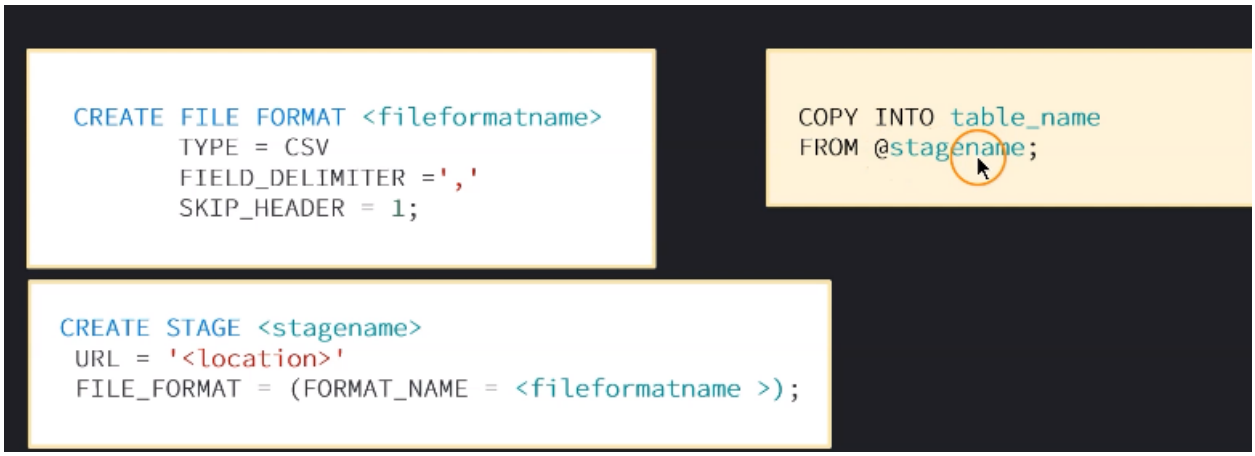
4. The recommended method of specifying file format is



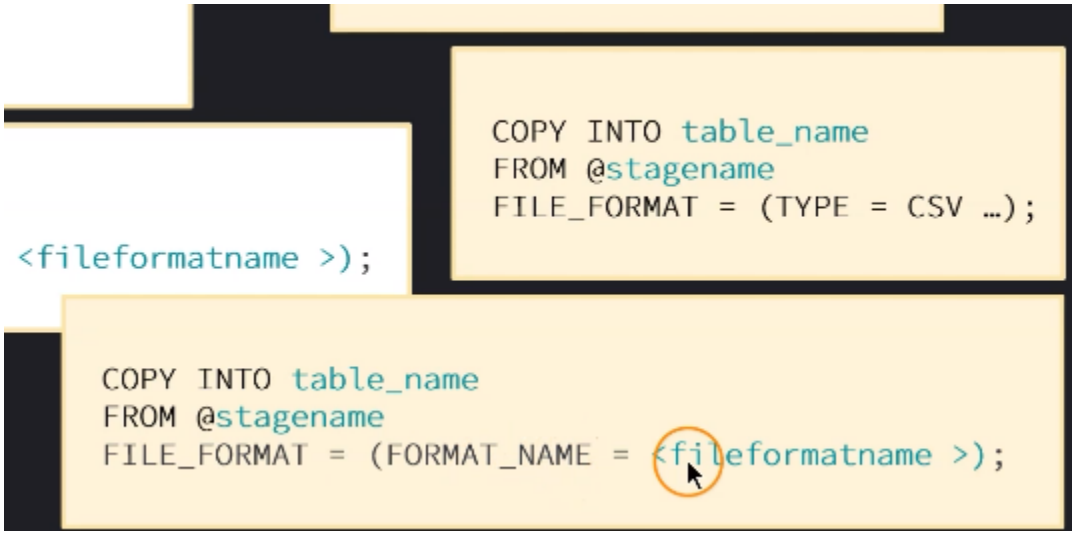
5. First we have to define the file format..with file type,field delimiter..
6. Then we have to use this file format object while creating stage



7. Now while copying from stage to table..we can ignore file format…because we have created stage with file format



8. We can also overrule file format by specifying it in query…or we can also use another file format object



9. For hands on refer online

Insert and update

1. To manually insert data into table..first we create a table

```
CREATE OR REPLACE TABLE OUR_FIRST_DB.PUBLIC.ORDERS (
        ORDER_ID VARCHAR (30),
        AMOUNT INT,
        PROFIT INT,
        QUANTITY INT,
        CATEGORY VARCHAR(30),
        SUBCATEGORY VARCHAR(30));
```

2. Then to insert one row we use

```
// Insert single row
INSERT INTO OUR_FIRST_DB.PUBLIC.ORDERS
VALUES (1,0,0,0, 'None','None');
```

3. To insert multiple rows at once …we use

```
// Insert multiple rows
INSERT INTO OUR_FIRST_DB.PUBLIC.ORDERS
VALUES
(2,12,4,1, 'Garden','Flowers'),
(3,15,6,2, 'House','Kitchen'),
(4,11,2,1, 'House','Sleeping');
```

4. Just check online..these are same as sql
5. Insert overwrite ..it truncates the entire table..and adds this values which are in query

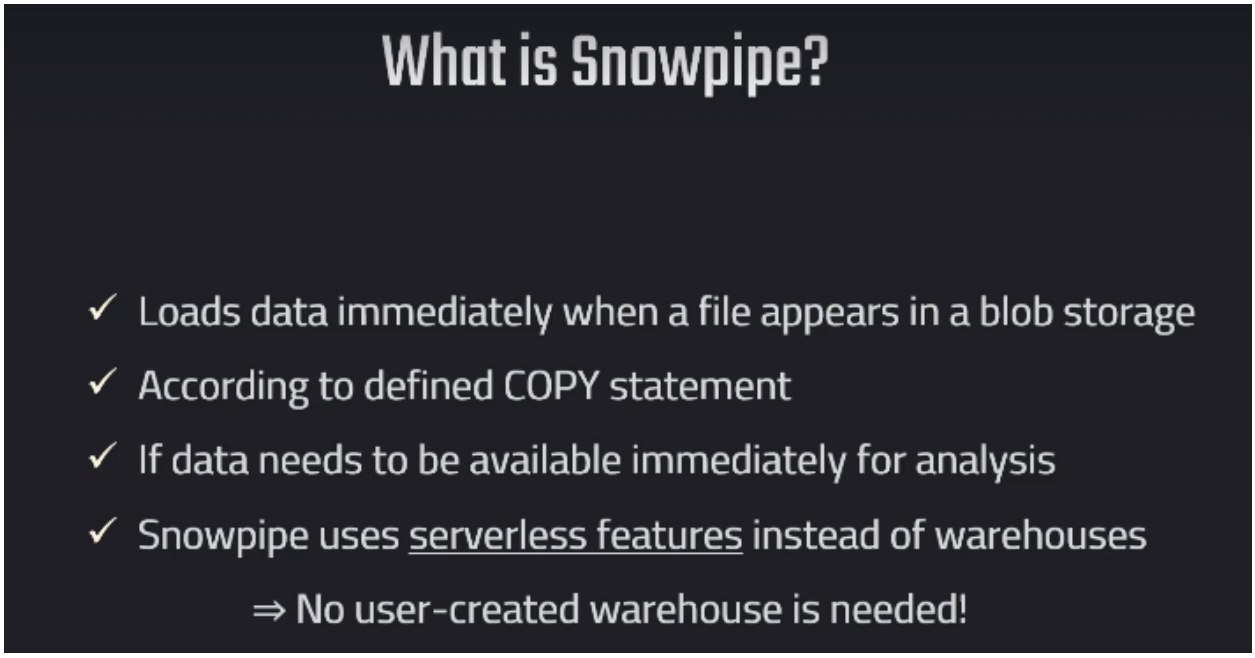
```
//INSERT OVERWRITE - Truncates the table
INSERT OVERWRITE INTO OUR_FIRST_DB.PUBLIC.ORDERS (ORDER_ID,SUBCATEGORY)
VALUES
(20,'Flowers'),
(30,'Kitchen'),
(40,'Sleeping');
```

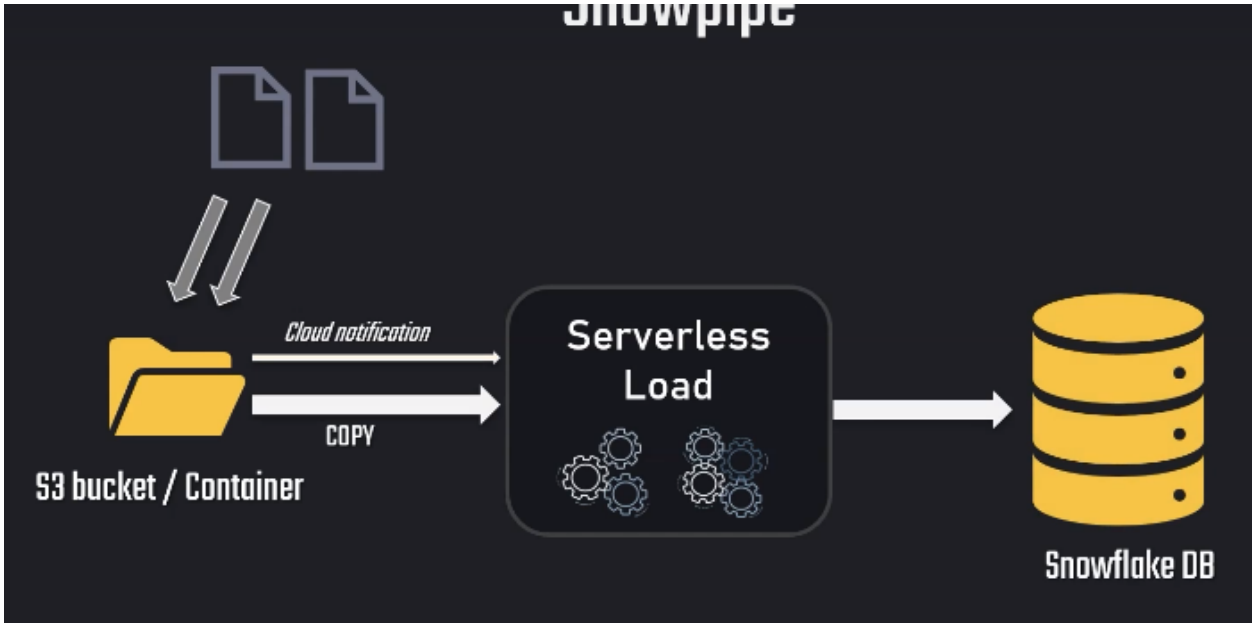
6.

Storage Integration

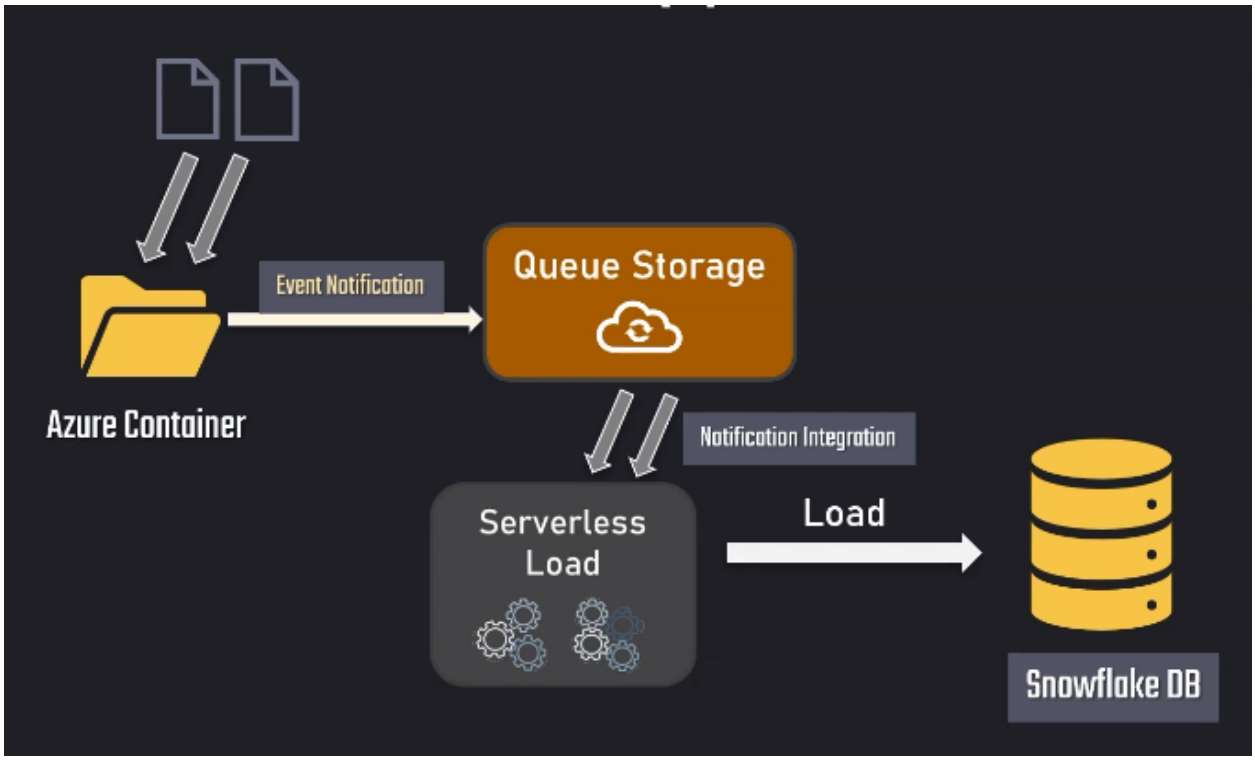
1. Refer Bard and online resources ..not useful for exam

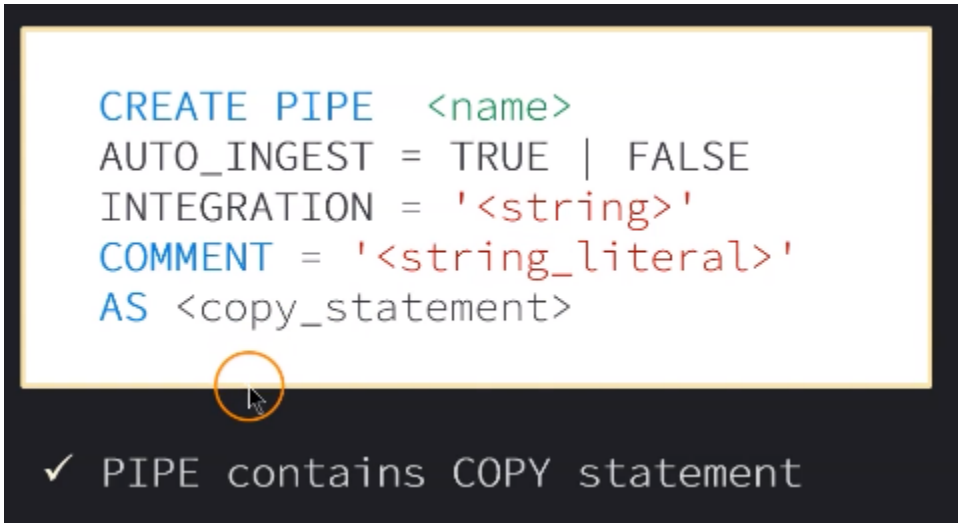
Snowpipe

1. Till now we have used Bulk loading(manual execution) but there's also continuous data loading and this uses so-called snowpipe
2. A pipe is an object that is created in our snowflake account. And with that we can load data files immediately and automatically when they appear in blob storage
3. 

4. So in the definition there will be the copy statement already present inside the Snowpipe and then the data will be loaded according to this copy statement.
5. 

6. So basically how this will work is…if there are any new files in our S3bucket/blob storage…then..we can setup a cloud notification…and with the help of this notification we can trigger copy statement..which runs in snowflake servers…then we can copy files from s3 to snowflake DB
7. Implementation using azure



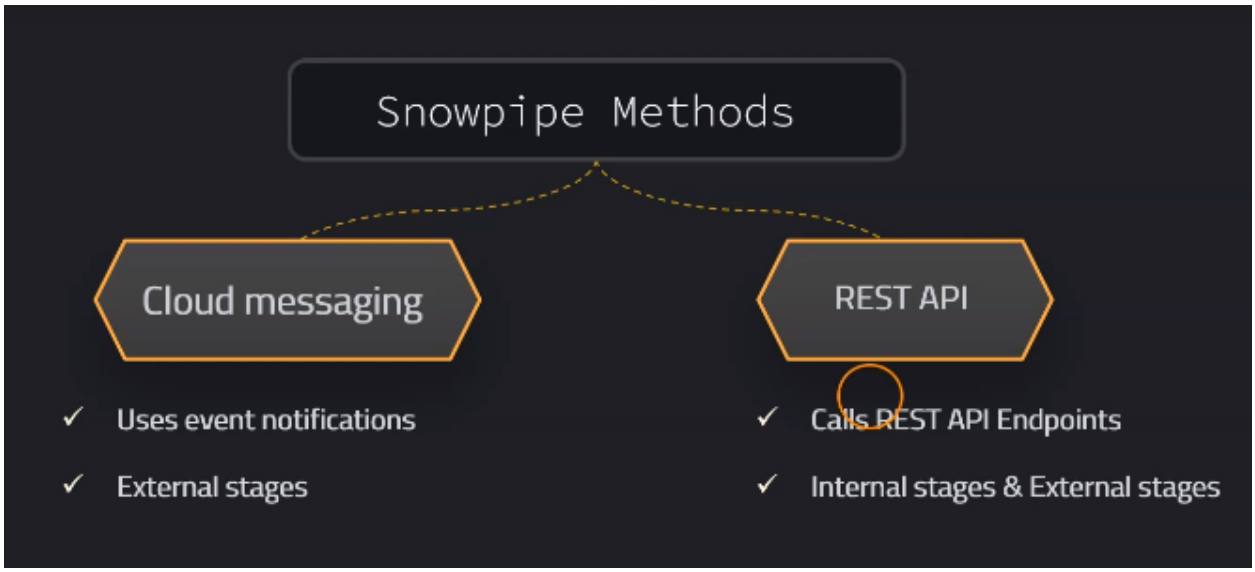
8. 

9. We can then also define the integration and this is, as mentioned, the notification integration that is watching the cloud messages in the queue storage and this can then trigger the pipe

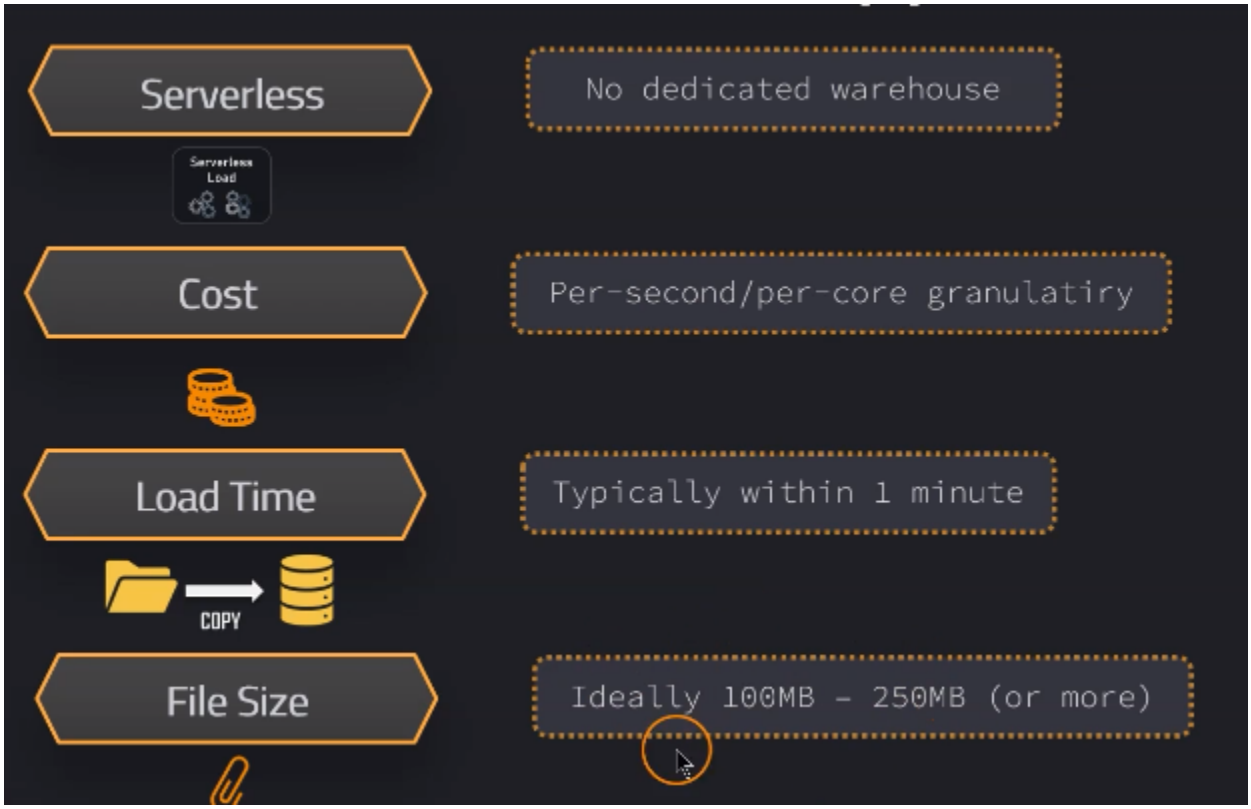
10. Here we have a copy statement ..which will load data according to copy statement

```
CREATE PIPE  <name>
AUTO_INGEST = TRUE | FALSE
INTEGRATION = '<string>'
COMMENT = '<string_literal>'
AS COPY INTO table_name
FROM @stage_name
```

✓ PIPE contains COPY statement

11. There are two types of notification triggering in snowpipe
12. 

13. Things to remember for snowpipe



14. It is serverless…that means it does not require a warehouse..and everything is done by snowflake
15. Cost will be per second/per- core…then this will turn into credits

16. The cost of Snowpipe is calculated based on the following factors:

    - **Compute:** Snowpipe uses compute resources to load data into Snowflake. You are charged for the amount of compute time that Snowpipe uses.
    - **Storage:** Snowpipe stores data in the Snowflake storage layer. You are charged for the amount of storage that you use.
    - **Network:** Snowpipe uses the Snowflake network to transfer data from the source to the target table. You are charged for the amount of network bandwidth that you use.

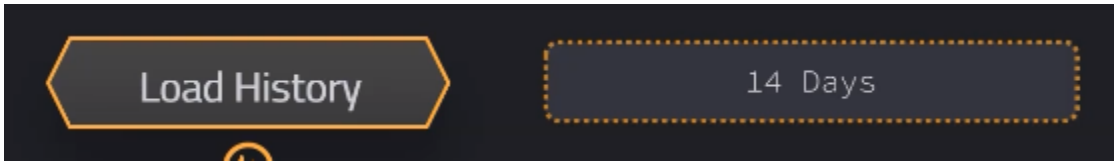
    In addition to these factors, there is also a per-file overhead cost associated with Snowpipe. This cost is charged regardless of the amount of data that is loaded in each file.

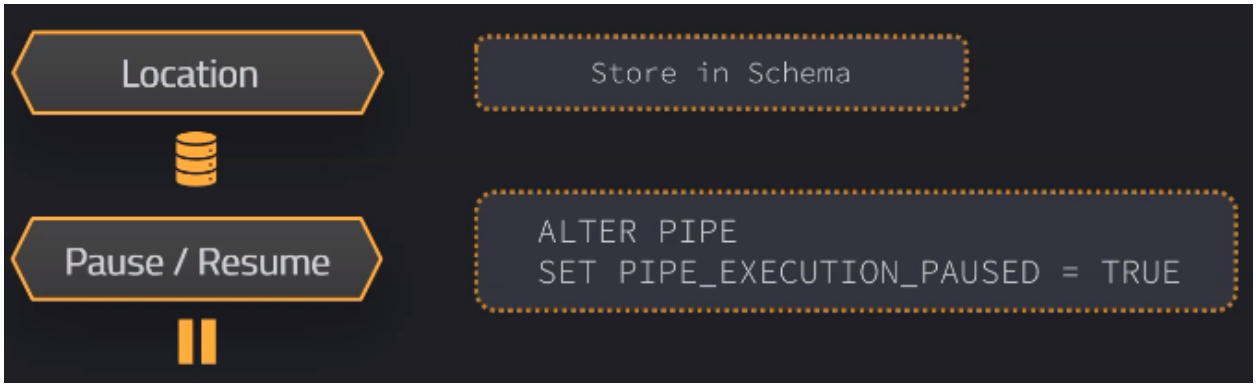
    The specific cost of Snowpipe will vary depending on the amount of data that you load, the frequency of your loads, and the size of your files. However, Snowpipe is generally a very cost-effective way to load data into Snowflake.

17. Load time..depends on size and number of files...usually it takes 1 min

18. 

19. Load history is 14 days



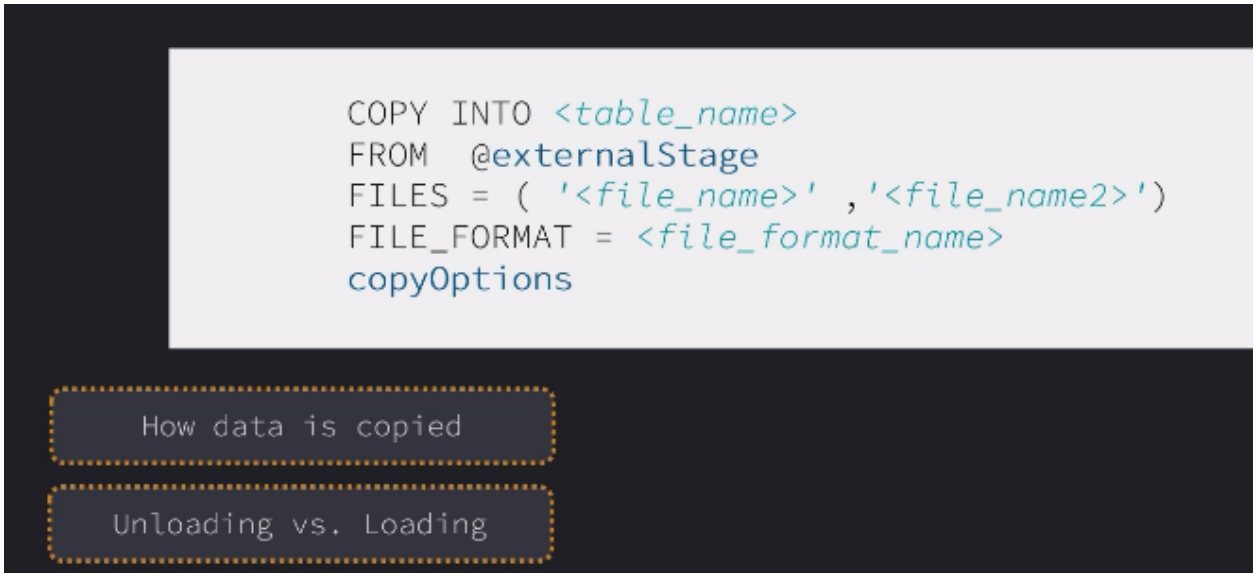
20. So Snowflake and the pipe knows which files have been already loaded by using this pipe and this metadata will be stored in the pipe so that no duplicate data is loaded as long as it is in retention period
21. 

22. The pipe can be paused and resumed just by altering the pipe and changing the pipe execution post property.
23. This is oftentimes done when we want to transfer the ownership of the pipe.

Copy Options

1. Copy options define how the data is copied into our tables..

2. Can also be used to unload the data



3. Bard definition of copy tables

Snowflake COPY options allow you to control how data is loaded from or unloaded to a Snowflake table. You can specify options such as the file format, compression, and error handling.

To use COPY options, you specify them in the COPY command. For example, the following command loads data from a CSV file in an internal stage, using the `ON_ERROR=ABORT` option to abort the load if any errors occur:

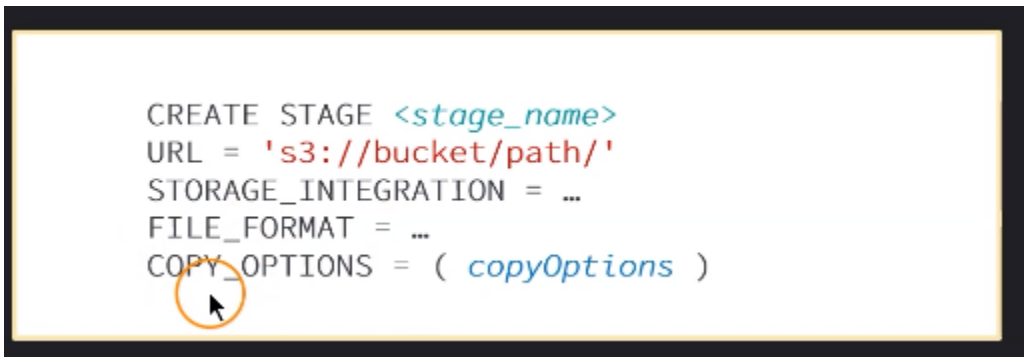
SQL

```
COPY INTO my_table FROM @my_stage/my_data.csv
ON_ERROR=ABORT;
```

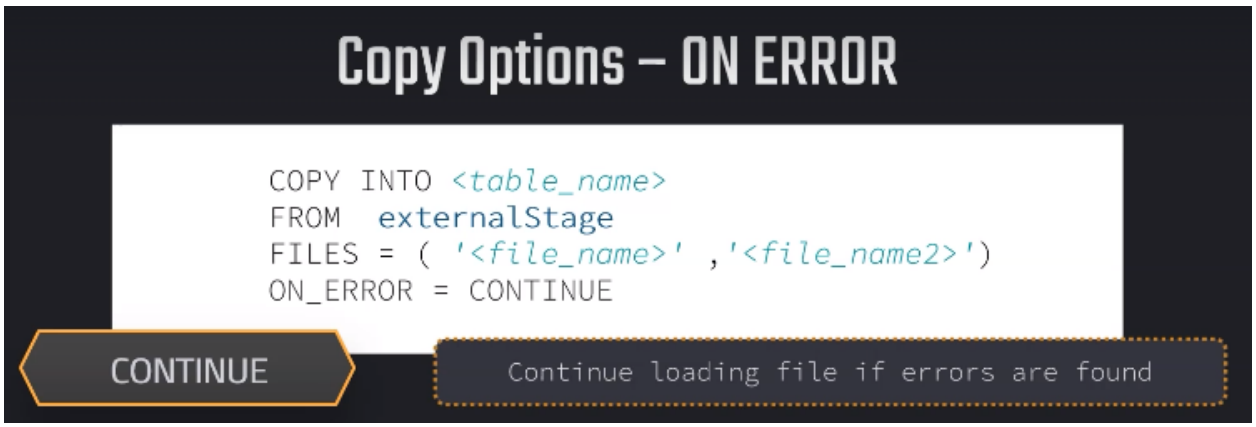
4. If we didnt specified any copy option..then it will take default stage properties

	parent_property	property	property_type	property_value	property_default
1	STAGE_FILE_FORMAT	FORMAT_NAME	String	fileformat_azure	
2	STAGE_COPY_OPTIONS	ON_ERROR	String	ABORT_STATEMENT	ABORT_STATEMENT
3	STAGE_COPY_OPTIONS	SIZE_LIMIT	Long		
4	STAGE_COPY_OPTIONS	PURGE	Boolean	false	false
5	STAGE_COPY_OPTIONS	RETURN_FAILED_ONLY	Boolean	false	false
6	STAGE_COPY_OPTIONS	ENFORCE_LENGTH	Boolean	true	true
7	STAGE_COPY_OPTIONS	TRUNCATECOLUMNS	Boolean	false	false
8	STAGE_COPY_OPTIONS	FORCE	Boolean	false	false
9	STAGE_LOCATION	URL	String	["azure://demosnowflak	

5. While creating a stage…we can mention copy options..which overrides the default copy options

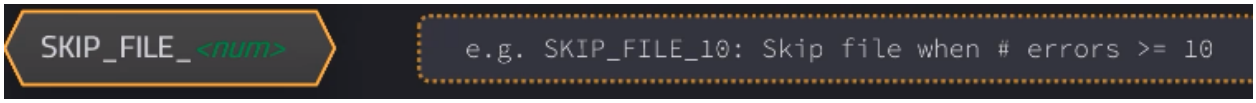


6. Copy options on error



7. We have skip file..which skips the file if the errors are found..continoius with another file…it is default copy option for snowpipe

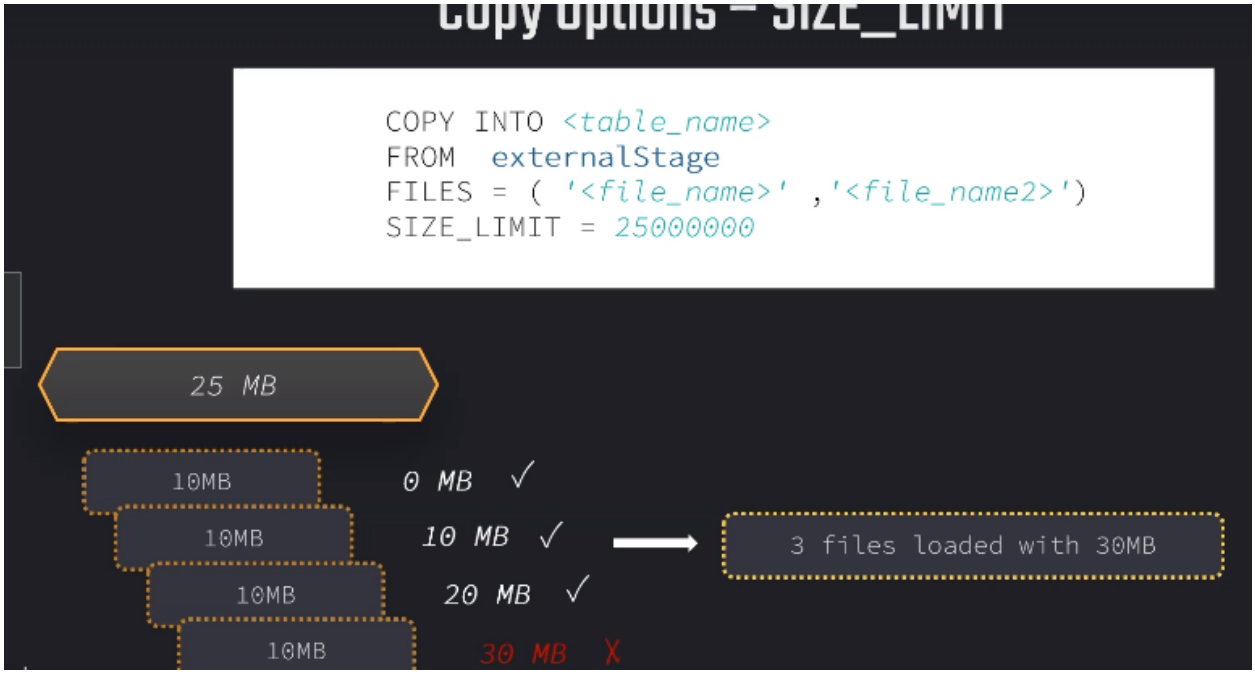
8. Skip_file_num

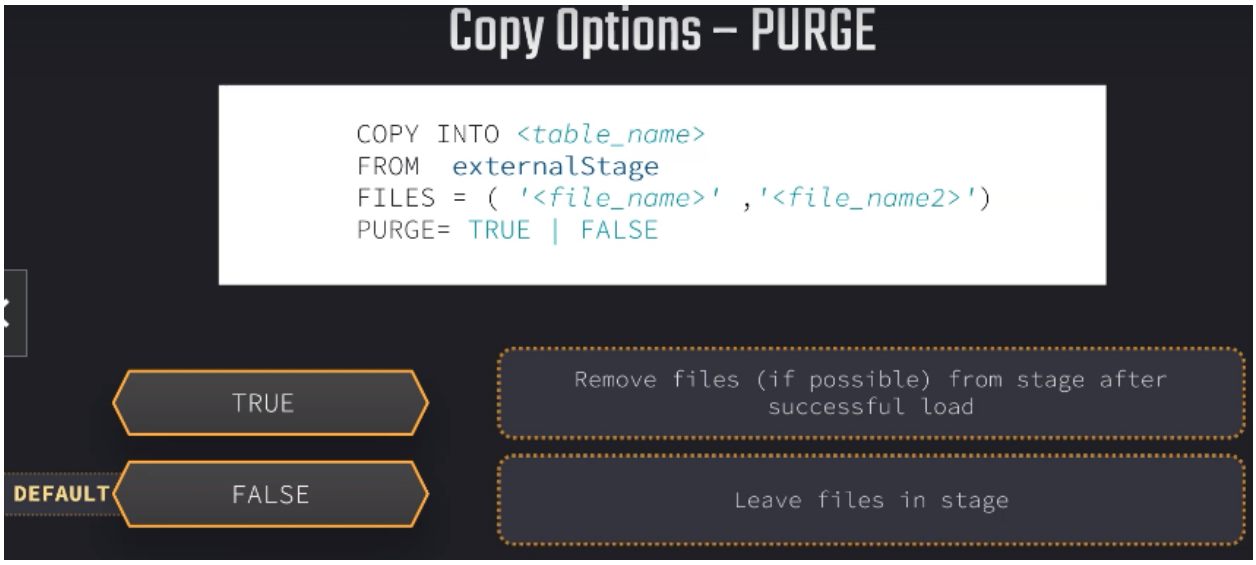


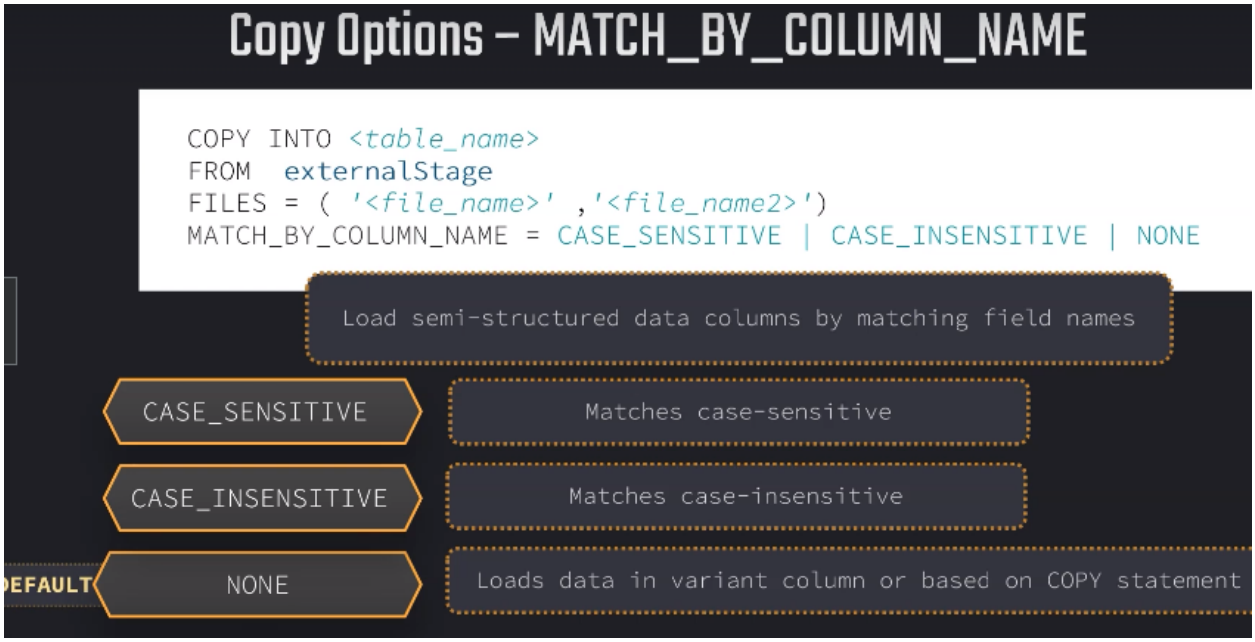
9. skip_file_num%

10. Default is abort statement..it is default co for bulk load

11. 

12. 

13. This will load until the size_limit is exceeded and then it stops loading files
14. 

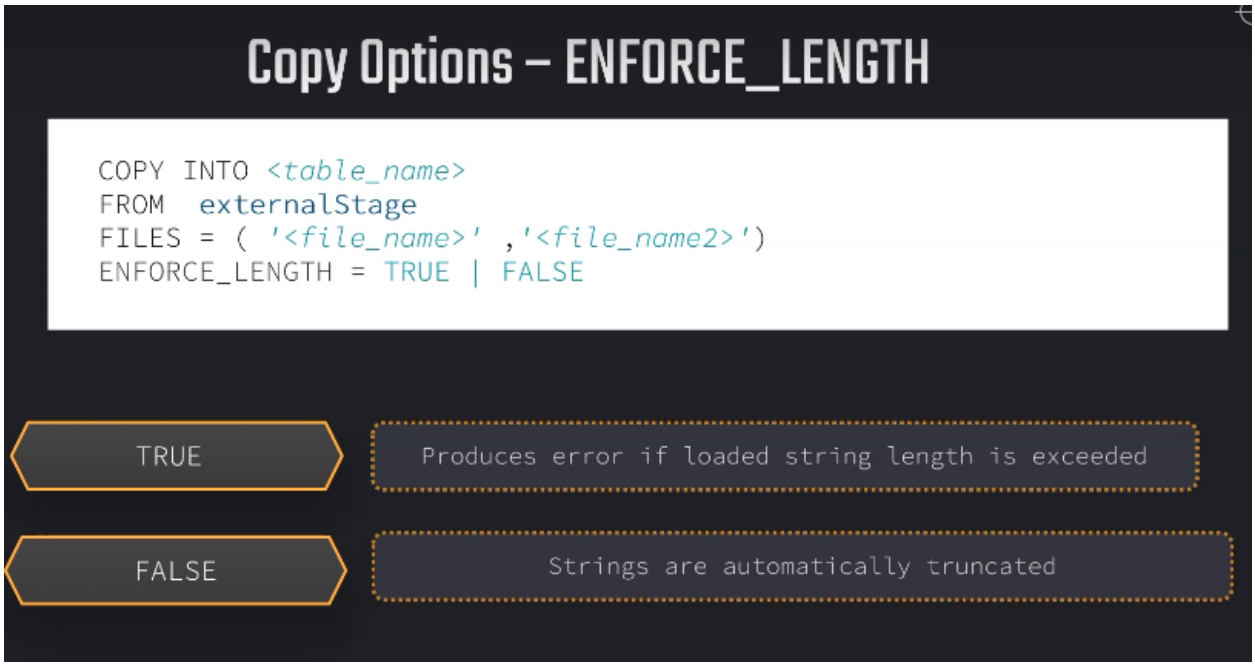
15. 

16. Bard example

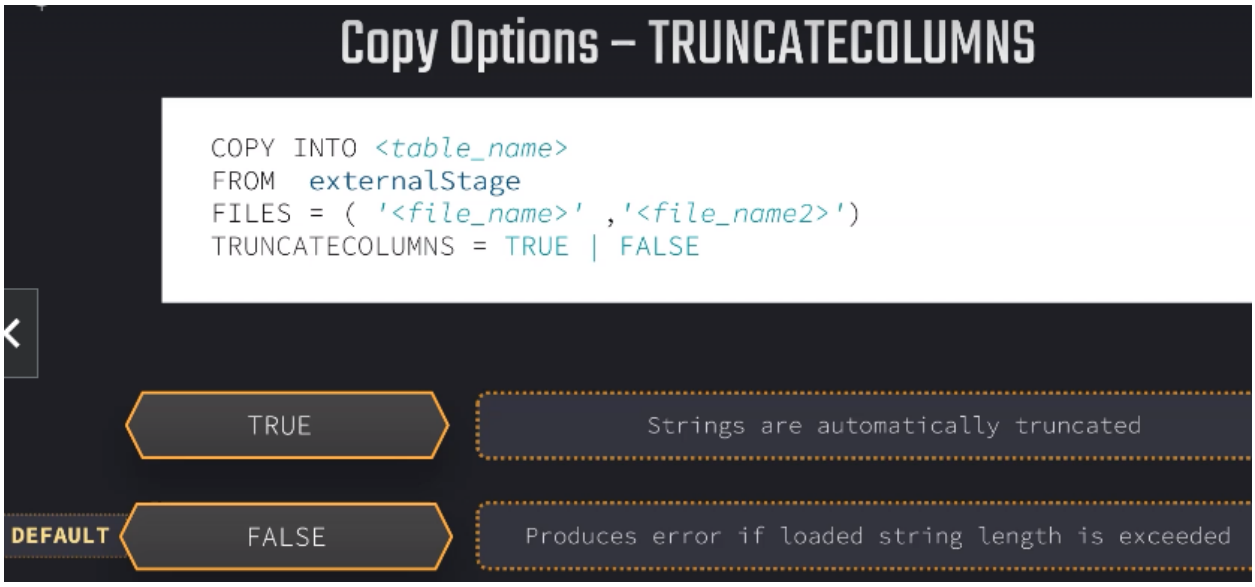
The `MATCH_BY_COLUMN` COPY option in Snowflake allows you to load data from semi-structured files into separate columns in the target table that match corresponding columns represented in the source data. This option is useful for loading data from files that do not have a fixed column order, or that have a different column order than the target table.

To use the `MATCH_BY_COLUMN` option, you specify a list of column names in the COPY command. Snowflake will then try to match the column names in the source data to the column names in the target table. If a match is found, the data from the source column will be loaded into the corresponding target column. If no match is found, the data from the source column will be ignored.

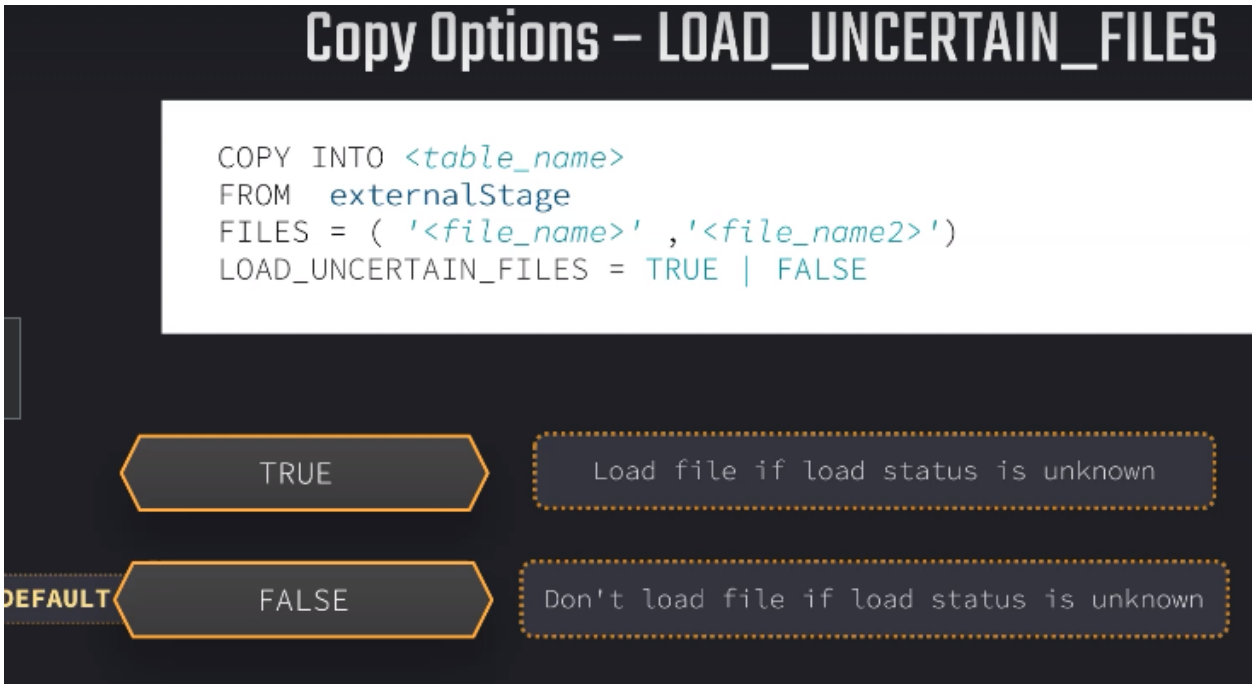
```
COPY INTO my_table FROM @my_stage/my_data.json
FILE_FORMAT=JSON
MATCH_BY_COLUMN=('id', 'name', 'age');
```

17. 

default is true

18. 

19. 
COPY INTO <table_name>
FROM externalStage
FILES = ('<file_name>' ,'<file_name2>')
FORCE = TRUE | FALSE
TRUE
Load files even if they have been loaded before
DEFAULT
FALSE
Don't load file when they have been loaded before

20. 
Copy Options – LOAD_UNCERTAIN_FILES
COPY INTO <table_name>
FROM externalStage
FILES = ('<file_name>' ,'<file_name2>')
LOAD_UNCERTAIN_FILES = TRUE | FALSE
TRUE
Load file if load status is unknown
DEFAULT
FALSE
Don't load file if load status is unknown

21. overview

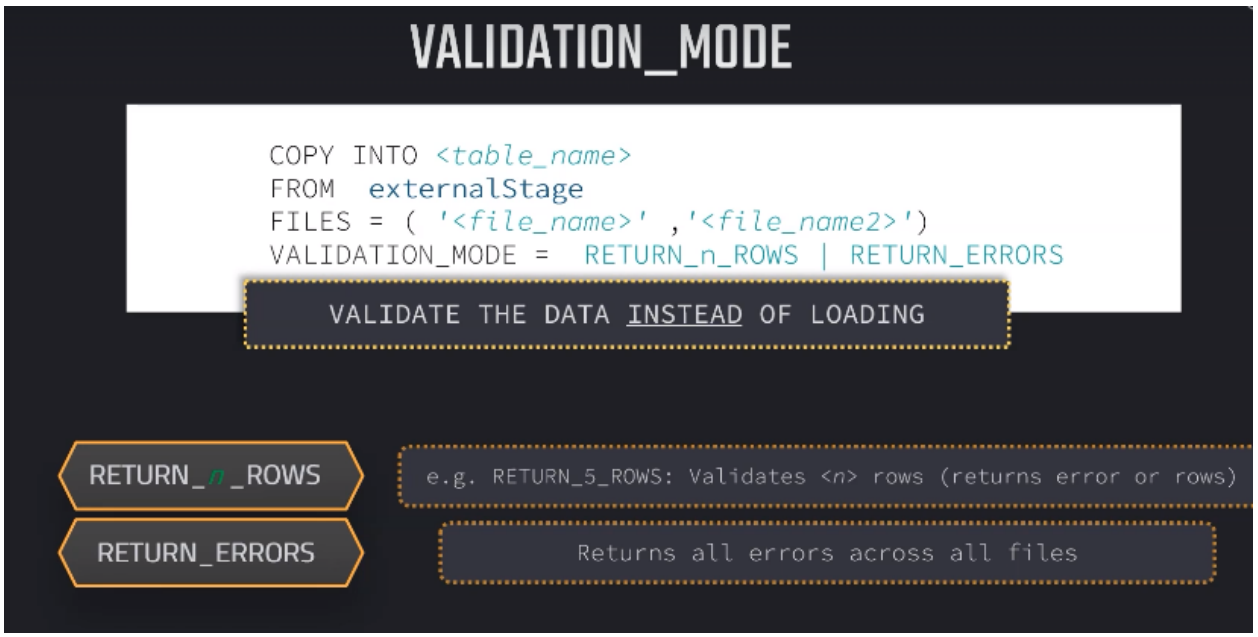
CopyOption	Description	Values	Default
ON_ERROR	Specifies the error handling for the load operation	CONTINUE | SKIP_FILE | SKIP_FILE_num | 'SKIP_FILE_num%' | ABORT_STATEMENT	ABORT_STATEMENT
SIZE_LIMIT	Specifies the maximum size (in bytes) of data to be loaded	*num*	null (no limit)
PURGE	Remove files after successful load	TRUE | FALSE	FALSE
RETURN_FAILED_ONLY	Return only files that have failed to load	TRUE | FALSE	FALSE
MATCH_BY_COLUMN_NAME	Load semi-structured data into columns in matching the columns names	CASE_SENSITIVE | CASE_INSENSITIVE | NONE	NONE
ENFORCE_LENGTH	Truncate text strings that exceed the target column length	TRUE | FALSE	TRUE
TRUNCATECOLUMNS	Truncate text strings that exceed the target column length	TRUE | FALSE	FALSE
FORCE	Load files even if loaded before	TRUE | FALSE	FALSE
LOAD_UNCERTAIN_FILES	Load files even if load status unknown	TRUE | FALSE	FALSE

Hands_on copy option

1. Refer online

Validation Mode

1. Lets check another parameter which can be used with copy_options ..validation mode
2. If we use the validation mode, this will just validate the data in a copy command instead of actually loading the data.
3. We use validation mode to check for errors..before using copy options



4. For hands on refer online

Validate

1. Lets check validate function

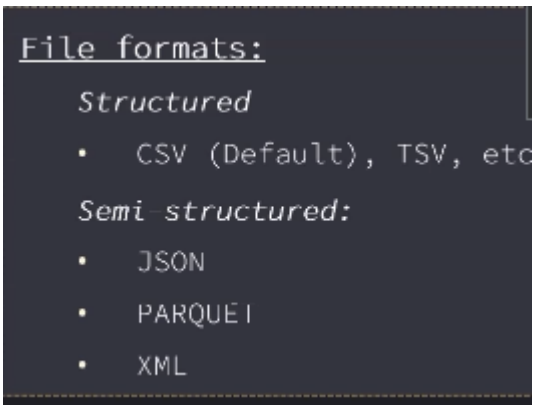
> The VALIDATE function in Snowflake tables allows you to validate the data that has been loaded into a table. This can be useful for ensuring that the data is accurate and complete, and that it meets the data quality requirements of your application.
>
> To use the VALIDATE function, you specify the name of the table that you want to validate in the function call. The VALIDATE function will then validate the data in the table and return any errors that it encounters.

2. For more on validate : https://www.youtube.com/watch?v=lE4mF8fgIOM

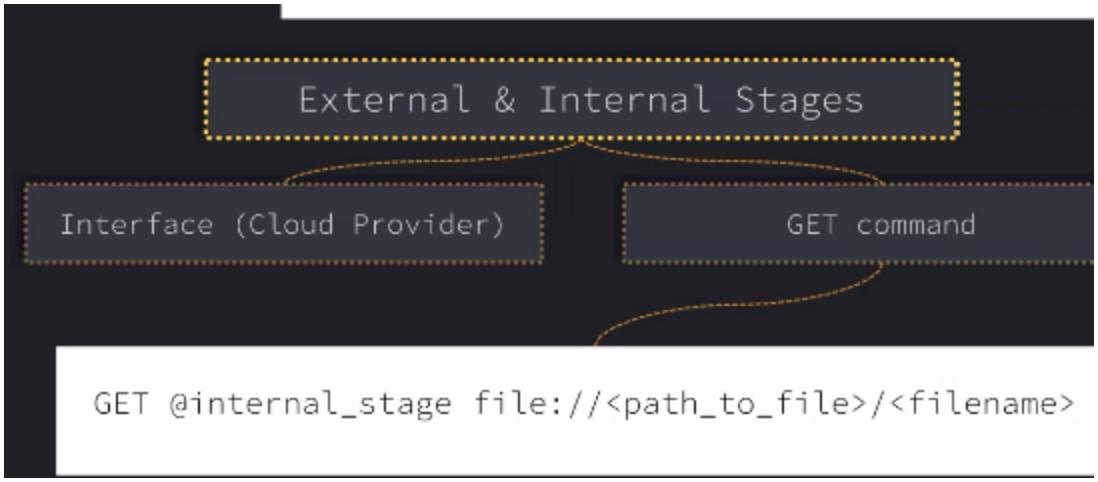
Unloading

```
COPY INTO @stage_name
FROM table_name
```

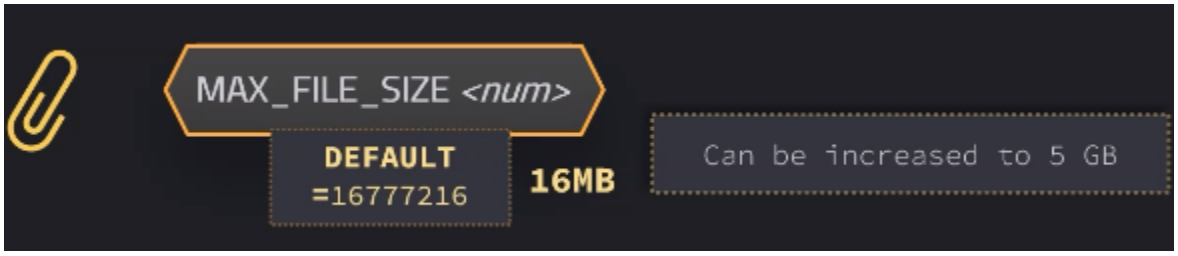
1. Here we will load data from table to stage
2. Unloading can be done to both internal and external stages

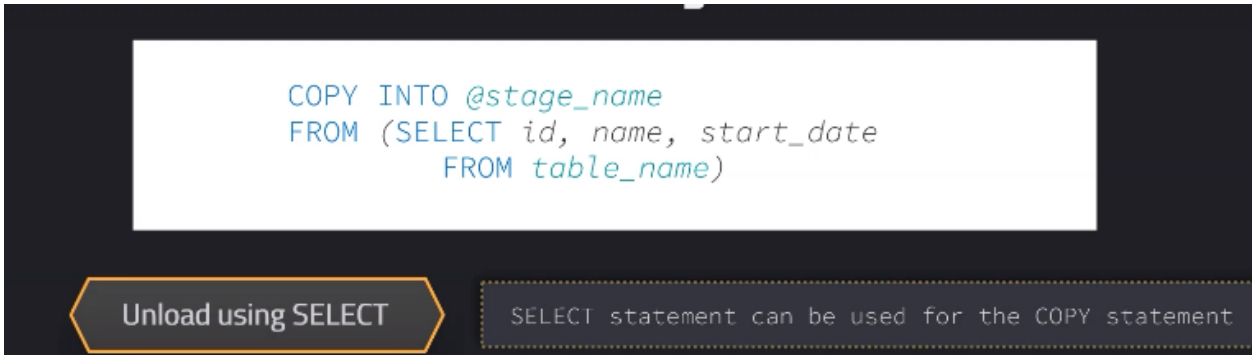


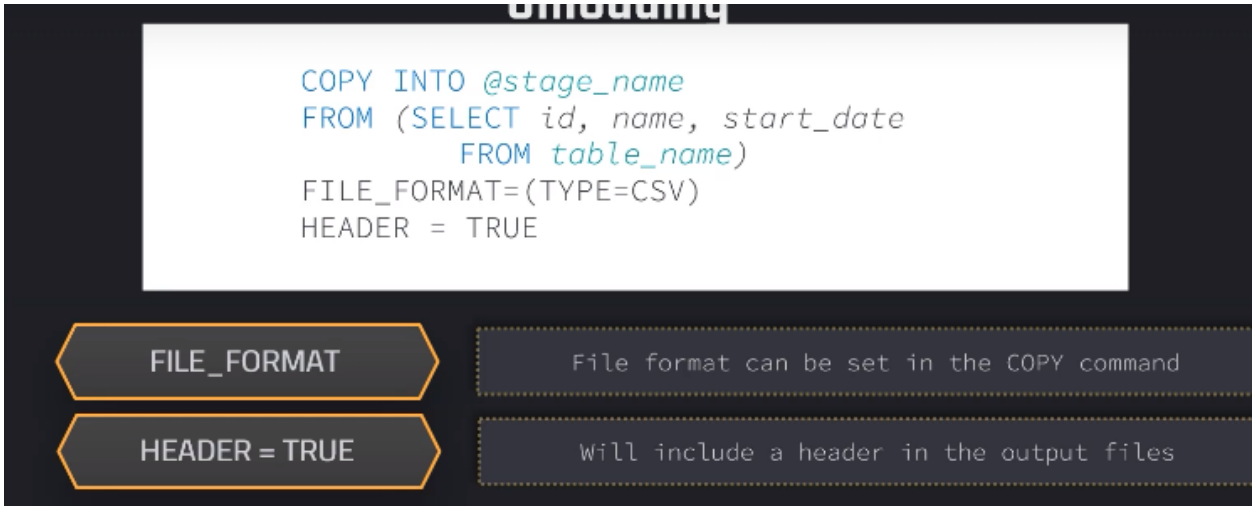
3. File format can be
4. After unloading into stage ..we can download these files to our local machines
5. To download from external stage..we have to use cloud provider interface

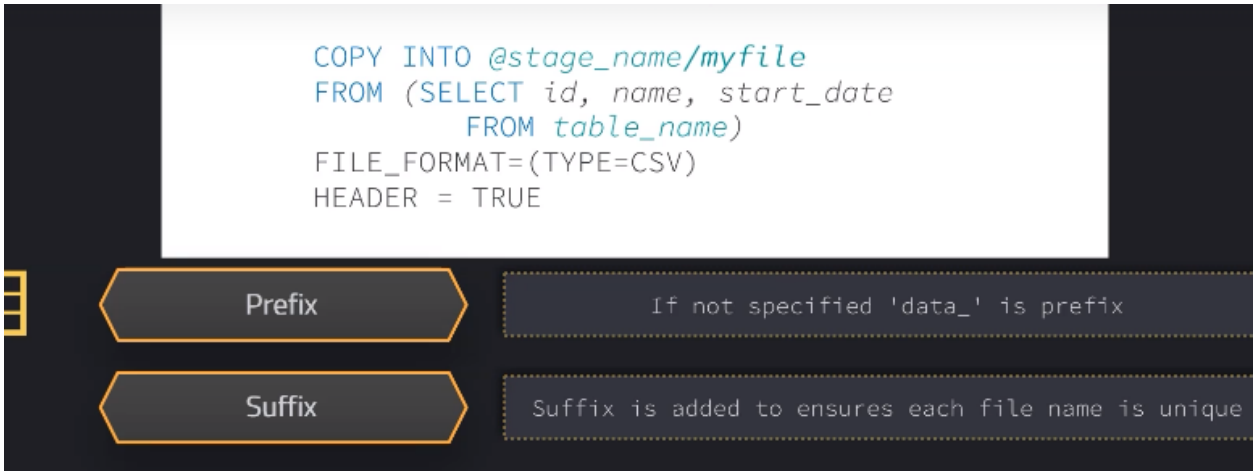
6. 

7. If the file is in internal stage…we can get by using GET
8. Here also we have copying options
9. 

10. 

11. 

12. 

13. 

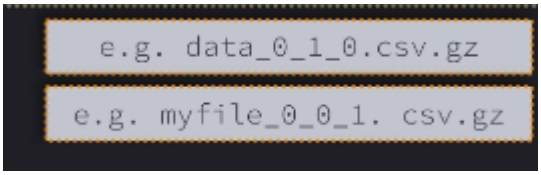
14. The `PREFIX` and `SUFFIX` copy options in Snowflake allow you to specify the prefix and suffix that will be used for the filenames of the unloaded data files.

The `PREFIX` option specifies the prefix that will be used for the filenames of all the unloaded data files. If a prefix is not specified, Snowflake prefixes the generated filenames with `data_`.

The `SUFFIX` option specifies the suffix that will be used for the filenames of all the unloaded data files. If a suffix is not specified, Snowflake appends a suffix that ensures each file name is unique across parallel execution threads.

15. Here is an example of how to use the `PREFIX` and `SUFFIX` options to unload data from a Snowflake table to a CSV file in an internal stage, using the `PREFIX='my_data_'` and `SUFFIX='.csv'` options to specify the prefix and suffix for the unloaded data files:

```sql
COPY INTO @my_stage/my_data_*.csv FROM my_table
FILE_FORMAT=CSV
PREFIX='my_data_'
SUFFIX='.csv';
```

16. Example 

..here files which are downloaded..will be compressed by default
17. For hands on refer online
18.