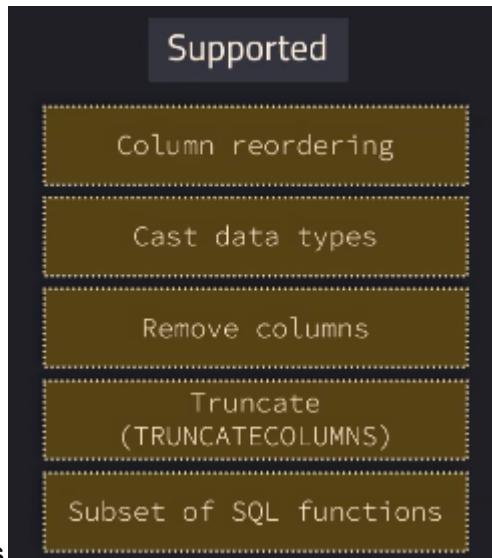


## Transformations and Functions

1. When loading the data ..we can apply some transformations
2. So when we want to load data, we can also, instead of just using an entire file, we can query and specify a specific select statement in the copy command.



3. What we can do is



4. What we cannot do

5. But in general we can say that there are a lot of functions available when we want to query data in Snowflake so we can apply and use functions in Snowflake. And most of the SQL functions that are defined in SQL 1999 are also supported.

Supports most standard SQL functions defined in SQL:1999 and parts of SQL:2003 extensions

6. Scalar functions

**Scalar functions**

Returns one value per invocation (one value per row)

```
SELECT DAYNAME('2023-12-31')
```

	DAYNAME('2023-12-31')
1	Sun

```
SELECT DAYNAME("effective_date")
FROM LOAN_PAYMENT
```

	DAYNAME("EFFECTIVE_DATE")
1	Thu
2	Thu
3	Thu
4	Thu
5	Fri

- 7.
8. Aggregate functions

**Aggregate functions**

Mathematical calculations such as max & min across rows

```
SELECT MAX(amount)
FROM orders
```

	MAX(AMOUNT)
1	98

9. Window functions

**Window functions**

Aggregate functions that operate on a subset of rows

```
SELECT ORDER_ID, SUBCATEGORY,
MAX(AMOUNT) OVER (PARTITION BY SUBCATEGORY)
FROM ORDERS
```

ORDER_ID	SUBCATEGORY	MAX(AMOUNT)
1	Stole	97
2	Electronic Games	98
3	Phones	977

10. Table Functions

**Table functions**

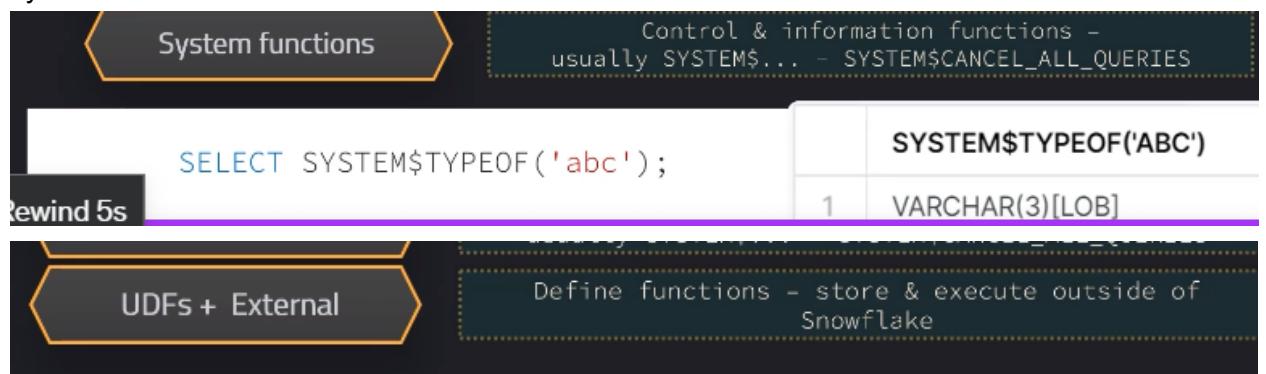
Return a set of rows for each input row - used to obtain information about Snowflake features

```
SELECT * FROM TABLE(VALIDATE( ORDERS , JOB_ID => '_last' ))
```

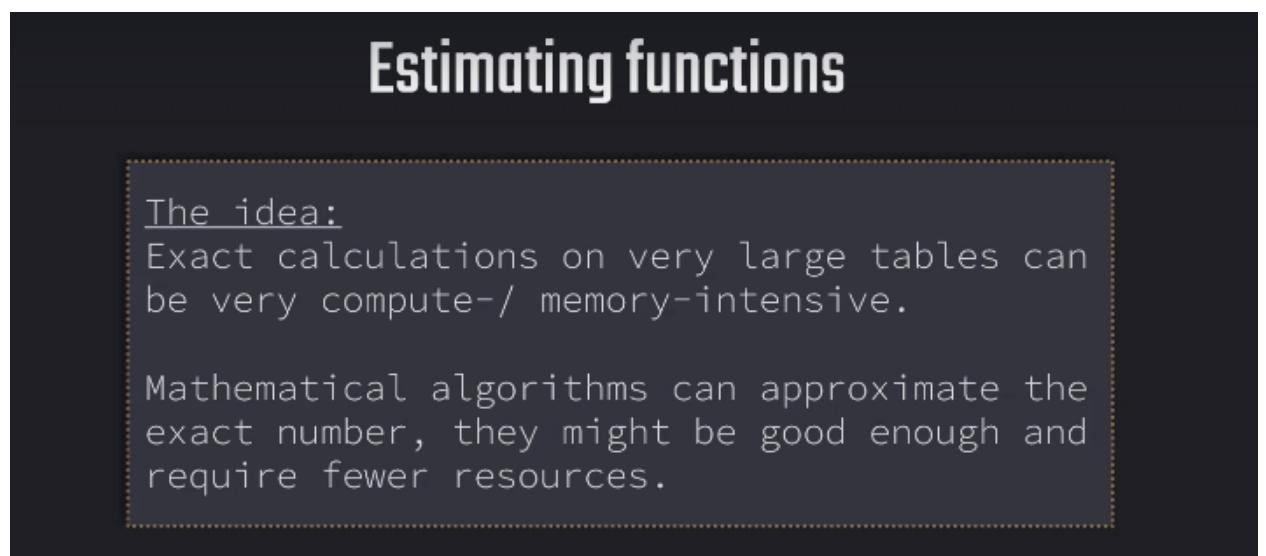
ERROR	FILE
1 Numeric value 'one thousand' is not recognized	returnfailed/OrderDetails_error.csv
2 Numeric value 'two hundred twenty' is not recognized	returnfailed/OrderDetails_error.csv

11. Table functions : <https://docs.snowflake.com/en/sql-reference/functions-table>

## 12. System function



## Estimation Functions



- 1.
2. Here we have 4 types of functions



3. HLL

### The idea:

HyperLogLog (cardinality estimation algorithm) is used to estimate the number of distinct values.

### Situation:

COUNT(DISTINCT (COLUMN1,...))

Large input

HLL(COLUMN1,...)

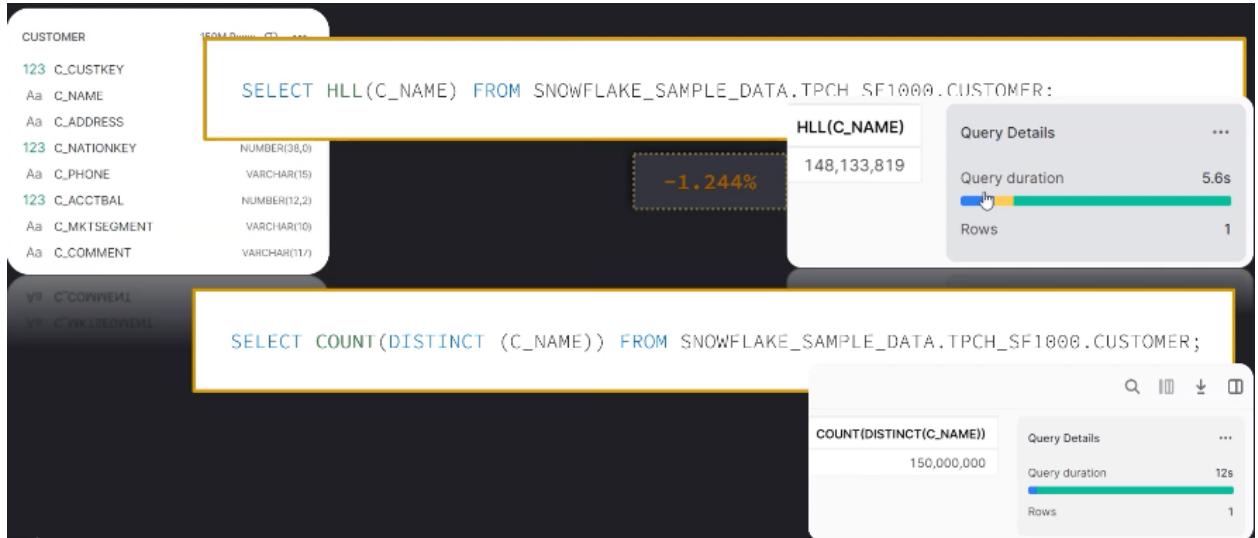
Average error is acceptable

1.62338%

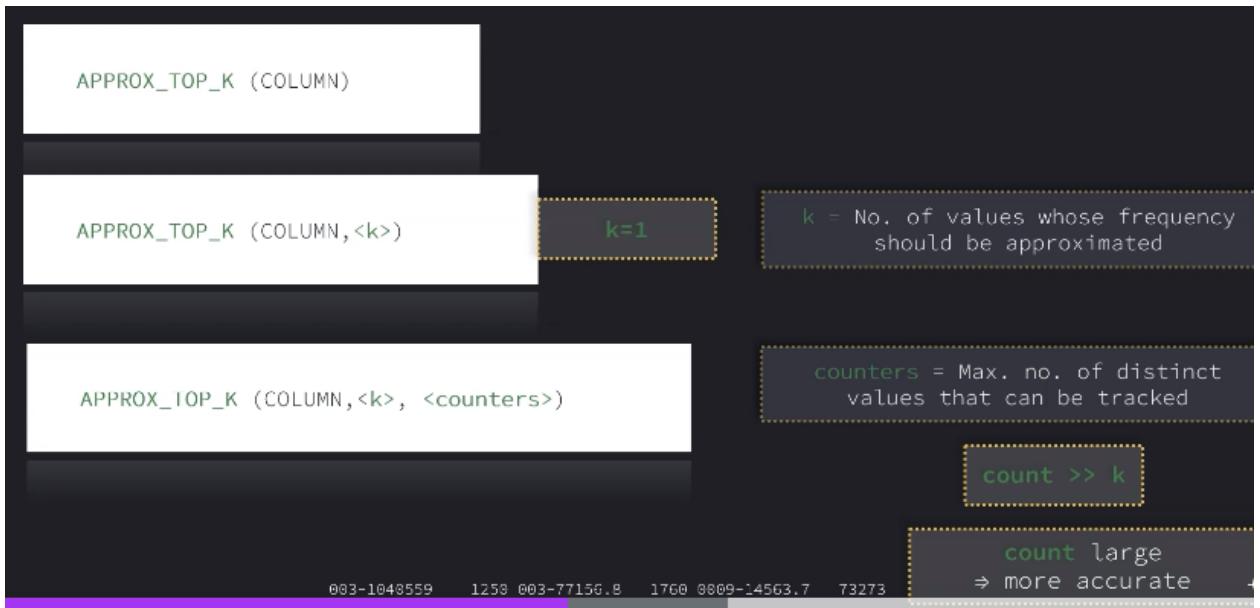
APPROX\_COUNT\_DISTINCT (COLUMN1,...)

4.

5. HLL gives approximate count of distinct columns in our table
6. Difference between using HLL and count ...with a table of 1.5mil rows



7. Frequent values



8.

9. Here if  $k = 10$ ..returns top 10 frequent values

10. Example ..here using

The screenshot shows the Snowflake UI with two queries:

```
SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL.STORE_SALES 28.8B rows
```

**Query 1 (Top):**

```
SELECT SS_CUSTOMER_SK,COUNT(SS_CUSTOMER_SK)
FROM STORE_SALES
GROUP BY SS_CUSTOMER_SK
```

**Query 2 (Bottom):**

```
SELECT APPROX_TOP_K (SS_CUSTOMER_SK,5,20)
FROM STORE_SALES
```

Both queries have their own "Query Details" panels on the right side:

- Query 1 (Top) Details:**
  - Query duration: 17m 43s
  - Rows: 65.0M
- Query 2 (Bottom) Details:**
  - Query duration: 2m 59s
  - Rows: 1

11. ⏪

12. Result form aprrox\_top\_k..will look like an list of list

	APPROX_TOP_K (SS_CUSTOMER_SK,5,20)
1	[ [ 20952969, 174160540 ], [ 44412221, 174160540 ], [ 30221701, 174160530 ], [ 30594069, 174160530 ], [ 53273150, 174160530 ] ]

13. Percentile value

#### 14. Percentile\_cont in sql

The `PERCENTILE_CONT()` function in SQL is an aggregate function that returns the value at a specified percentile within a group of rows. The percentile value is a number between 0 and 1, where 0 represents the lowest value in the group and 1 represents the highest value.

The `PERCENTILE_CONT()` function uses linear interpolation to calculate the percentile value. This means that if the percentile value falls between two rows in the group, the function will return a weighted average of the values of those two rows.

#### 15. Diff bw percentile cont and approx\_percentile

The screenshot shows a Snowflake query interface with two queries and their results.

**Top Query:**

```
SELECT APPROX_PERCENTILE(O_TOTALPRICE, 0.5)
FROM ORDERS;
```

Result: APPROX\_PERCENTILE(O\_TOTALPRICE, 0.5)  
144,288.43314293

**Bottom Query:**

```
SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY O_TOTALPRICE)
FROM ORDERS;
```

Result: PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY O\_TOTALPRICE)  
144,288.015

**Query Details:**

Query Details	...
Query duration	20s
Rows	1

Query Details	...
Query duration	1m 26s
Rows	

#### 16. Similarity between two data sets

# Similarity of Two or More Sets

The idea:

Uses MinHash to estimate the similarity between two or more data sets.

Typically the Jaccard similarity coefficient is used to compare similarity.

$$J(A, B) = (A \cap B) / (A \cup B)$$

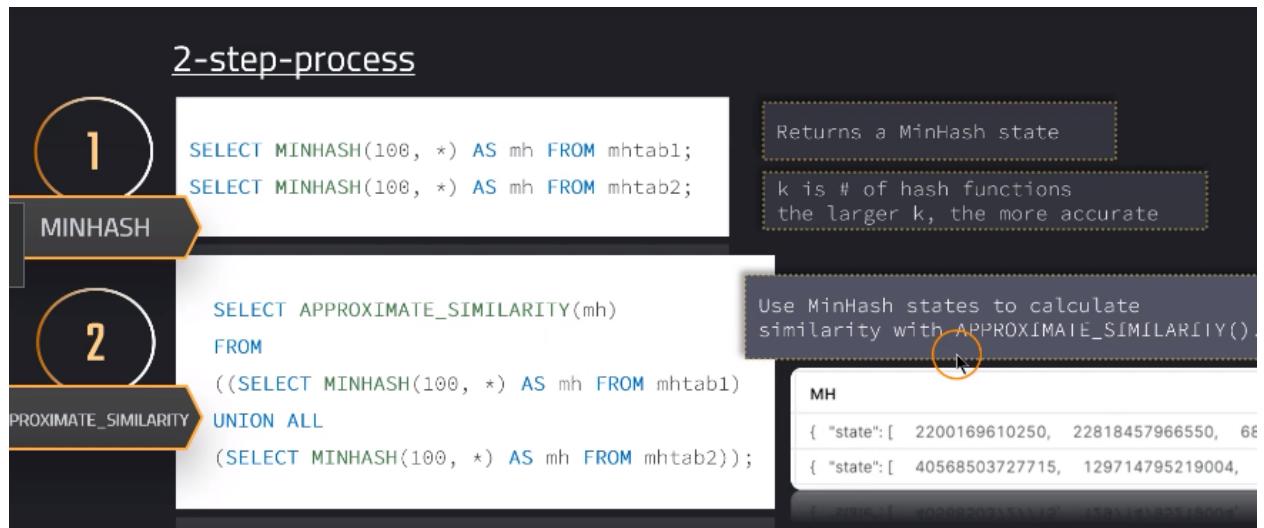
MinHash can estimate  $J(A, B)$  quickly.



Computationally expensive!

17.

18. Snowflake implemented MinHash..which lowers computation required



19.

20. Refer online too

21. Result of that 2nd query would be 0 or 1.0 means no similarity...1 means similarity

## User-defined Functions

Way of extending functionality with additional functions.

1.

Supported languages:

- o SQL
- o Python
- o Java
- o JavaScript

2. It supports writing in
3. Writing an UDFs using SQL (add\_two function)
4. First we create Function with the parameter..and we shud specify the datatype of parameter..then next we have to return datatype..what the function returns

```
create function add_two(n int)
returns int
```

5. Then we use "as" to define our func..and two \$\$ represents the body of our logic

```
create function add_two(n int)
returns int
as
$$
n+2
$$;
```

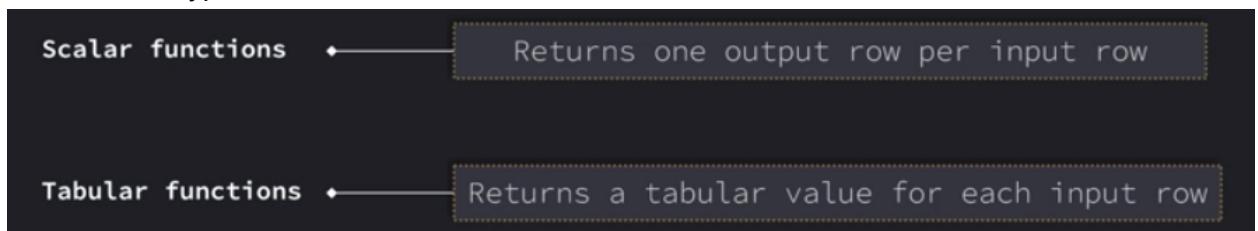


	status
1	Function ADD_TWO successfully created.

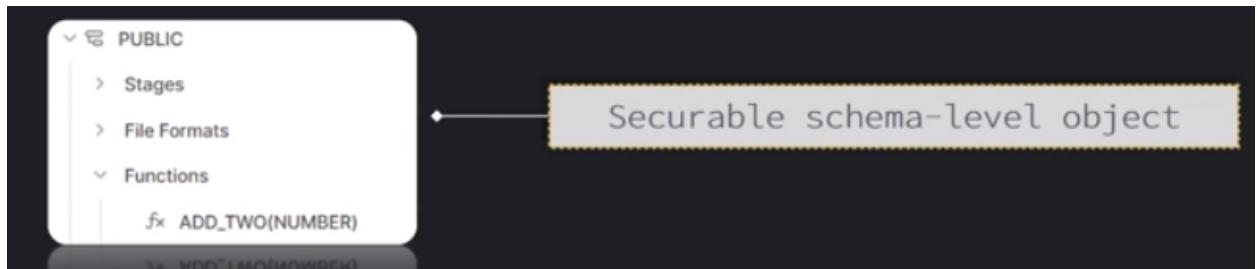
6. To use python..first two lines would be same for every language
7. Then if we are using python...we have to specify language,version and handler
8. And in the \$\$ (body) we have to use python function

```
create function add_two(n int)
returns int
language python
runtime_version = '3.8'
handler = 'addtwo'
as
$$
def add_two(n):
    return n+2
$$;
```

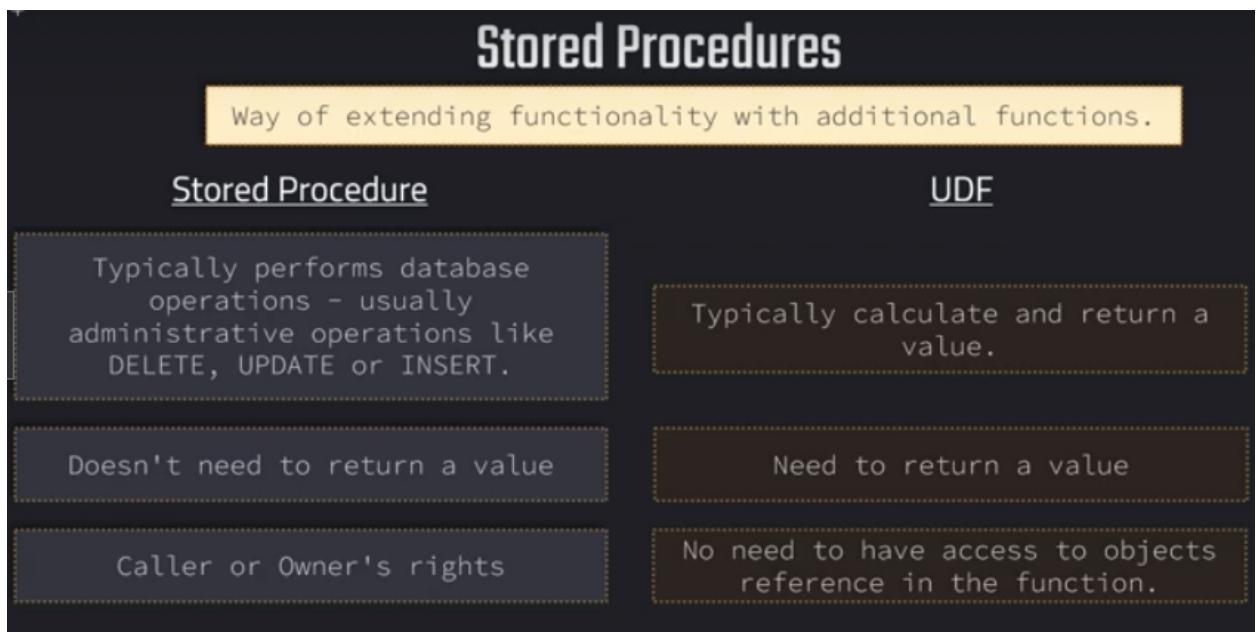
9. We have two types of function in UDF



10. The functions are schema level objects



Stored Procedure



1.

2. Supported languages for stored procedures in Snowflake

The screenshot shows a list of supported languages for stored procedures in Snowflake:

- o Snowflake Scripting  
(Snowflake SQL + procedural logic)
- o JavaScript
- o Snowpark API  
(Python, Scala, Java)

The screenshot shows the creation of a stored procedure named "find\_min". The code is as follows:

```
create procedure find_min(n1 int, n2 int)
returns int
language sql
as
BEGIN
    IF (n1 < n2)
        THEN RETURN n1;
        ELSE RETURN n2;
    END IF;
END;
```

To the right, a sidebar titled "Securable objects" lists various objects:

- > File Formats
- > Functions
- ▼ Procedures
  - ↳ CONCAT\_TEXT(VARCHAR, ...)
  - ↳ UPDATE\_FUNCTION()
  - ↳ UPDATE\_FUNCTION(NUMB...
  - ↳ BOUNDARY\_DATEPART(...)

3.

4. It is also a secure schema object..that means..we can give permission to users to use our stored procedure

The screenshot shows the execution of the stored procedure "find\_min" with arguments 5 and 7:

```
call find_min(5, 7);
```

To the right, a "Result" section displays the output of the procedure:

	FIND_MIN
1	5

5.

Define one or multiple operations.

6.

```
create procedure update_test_table()
returns varchar
language sql
as
BEGIN
UPDATE manage_db.public.test1
SET test_col = 3;
UPDATE manage_db.public.test1
SET test_col2 = 4;
END;
```

If argument is used in SQL statement ⇒ use :argument

```
create procedure update_test_table(new_value varchar)
returns int
language sql
as
BEGIN
    UPDATE manage_db.public.test1
    SET test_col = :new_value;
END;
```

7.

Runs either with caller's or owner's rights

Runs with the  
caller's privileges

Runs with the  
owner's privileges

Can make use of user information  
(e.g session variables)

Delegation of  
Administrative tasks

When creating specify rights  
⇒ use execute as caller/owner

DEFAULT  
OWNER

```
create procedure update_table(new_v varchar,table_name varchar)
execute as caller
returns int
language sql
as
BEGIN
    UPDATE IDENTIFIER(:table_name)
    SET test_col = :new_value;
END;
```

8.

9. Should learn more about snowflake stored procedure

## External Functions

# External functions

User-defined functions that are stored and executed outside of Snowflake.

Calls code that is executed outside of Snowflake.

- 1.
2. So for example, it will be some Azure function that we can just reference in our external function and it will be then executed outside of Snowflake.

- ✓ No code is stored in function definition
- ✓ Reference third-party libraries, services and data

```
create external function my_az_func(string_col VARCHAR)
returns variant
api_integration = azure_external_api_integration
as 'https://my-api-management-svc.azure-api.net/my-api-url/my_http_function'
```

- 3.
  - 4.
  5. This can be included in external functions
- ✓ Remotely executed code ⇒ "remote service"
  - ✓ Security related information are stored in API integration
  - ✓ Schema-level object

### Examples:

- AWS Lambda function
- Microsoft Azure function
- HTTPS server

### Advantages:

- ✓ Additional languages including GO and C#
- ✓ Accessing 3<sup>rd</sup>-party libraries such as machine learning scoring libraries
- ✓ Can be called from Snowflake and from other software

### Limitations:

- Must be scalar
- Slower performance (overhead + fewer optimizations)
- Not sharable

6.

## Secure UDFs & Stored Procedure

1. It may be possible that UDF or stored procedure..may contain some sensitive information
2. Some user can use desc command to see some partial information about the table and its structure

The screenshot shows a terminal window with a black background and white text. At the top, the command `DESC FUNCTION MANAGE_DB.PUBLIC.ADD_TWO(NUMBER);` is entered. Below the command, there is a table with four rows, each containing a property name and its corresponding value. The table has two columns: `property` and `value`. The properties listed are `signature`, `returns`, `language`, and `body`. The values are `(N NUMBER)`, `NUMBER(38,0)`, `SQL`, and `n+2` respectively. The table is enclosed in a light gray border.

property	value
1	signature
2	returns
3	language
4	body

- 3.
4. The other way to access this underlying data indirectly is by using optimizer

5. So to hide those information..we have to use secure UDF

#### Secure UDFs & Stored Procedures

- ✓ Hide certain information, e.g. definition
- ✓ Prevent users from seeing underlying data

```
CREATE SECURE FUNCTION <function_name> ...
```

#### Trade-off:

Reduced query performance  $\Leftrightarrow$  Security

#### Use-case:

- Consider purpose of UDF / Stored Procedure
- NOT make it secure if it is define just for convenience
- Make it secure when the data is sensitive enough

- 6.

Sequences

A sequence database object in Snowflake is a special type of object that is used to generate unique numbers. Sequences are similar to an auto-incrementing field in a database table, but more powerful and flexible. A sequence object can be used to generate numbers in a specific range, with a specific increment, and with a specific starting point.

Sequences are often used to generate unique primary keys for database tables. They can also be used to generate unique identifiers for other types of data, such as customer numbers, order numbers, and transaction numbers.

1.

```
CREATE TABLE sequence_test(  
    id int DEFAULT my_seq.nextval,  
    first_name varchar);
```

```
INSERT INTO sequence_test(first_name)  
VALUES  
('Maria'),('Frank');
```

ID	FIRST_NAME
1	Maria;
2	Frank

2.

```
CREATE OR REPLACE SEQUENCE my_seq  
START = 1  
INCREMENT = 1;
```

3. Hands on first we create sequence

```
SELECT my_seq.nextval;  
SELECT my_seq.nextval,my_seq.nextval;
```

4. Now if runs this 2 queries

5. For first query we get 1...for 2nd query we get 2 and 3 as output
6. Values of the sequence may not be consecutive..like 1,2,3,4..etc

7. Now we create table with id (int) and first\_name ..for the id we gave sequence

```
CREATE OR REPLACE TABLE sequence_test(
    id int DEFAULT my_seq.nextval,
    first_name varchar);

INSERT INTO sequence_test(first_name)
VALUES
('Maria'),
('Frank');

SELECT * FROM sequence_test;
```

8. Now if insert values into the table..id will get random numbers from mySeq

## SemiStructured Data

1. Very imp to know how snowflake tackles semistructured data
2. Because Snowflake actually has really great native support for these type of data formats.
3. Examples of structured and semistruc data

The slide features three main sections: 1. A title 'What is semi-structured data?' with a list of two bullet points: 'no fixed schema' and 'contains tags/labels and a nested structure'. 2. A title 'What is structured data?' with a single bullet point: 'Data has a well-defined structure'. 3. A table titled 'loan' showing three rows of loan data. To the right of the table is a JSON representation of the data:

```
{  
  "courses": [  
    {  
      "topic": "Snowflake",  
      "level": "All levels"  
    },  
    {  
      "topic": "SQL",  
      "language": ["English", "German"]  
    },  
    {  
      "topic": "Azure",  
      "level": "Beginner"  
    }  
  ]  
}
```

- 4. Supported format for semiStructures
- 5. We have 3 data types in snowflake semistru

#### Supported formats:

- JSON
- XML
- PARQUET
- ORC
- Avro

### OBJECT

Unordered set of name value pairs.

```
{  
    "topic":"Snowflake",  
    "level":"All levels"  
},
```

### ARRAY

Consists of 0 or more pieces of data.

```
["USA", "India", "Canada"]
```

VARIANT	Use-cases
Can store values of <b>any other data type</b> including ARRAY and OBJECT.	Explicitely define hierarchy of ARRAYS and OBJECTs
Suitable to <b>store and query semi-structured data</b> .	Let Snowflake convert semi-structured data into hierarchy of ARRAY, OBJECT, and VARIANT data stored into VARIANT
6.	<pre>CREATE FILE FORMAT my_fileformat TYPE = {JSON   AVRO   XML   PARQUET   ORC}</pre>

7. And third is variant data type...which store all the other data types including itself

The VARIANT data type in Snowflake is a special data type that can store any other data type, including ARRAY, OBJECT, and other VARIANTs. This makes it a very versatile and powerful data type.

VARIANTs can be used to store a variety of different types of data, such as:

- JSON data
- XML data
- Semi-structured data
- Complex data structures

Here is an example of how to create a table with a VARIANT column:

SQL

```
CREATE TABLE customers (
    customer_id INTEGER NOT NULL,
    customer_data VARIANT
);
```

8. INSERT INTO customers (customer\_id, customer\_data) VALUES (12345, JSON\_PARSE('{ "name": "John Doe", "email": "john.doe@example.com" }'));

SQL

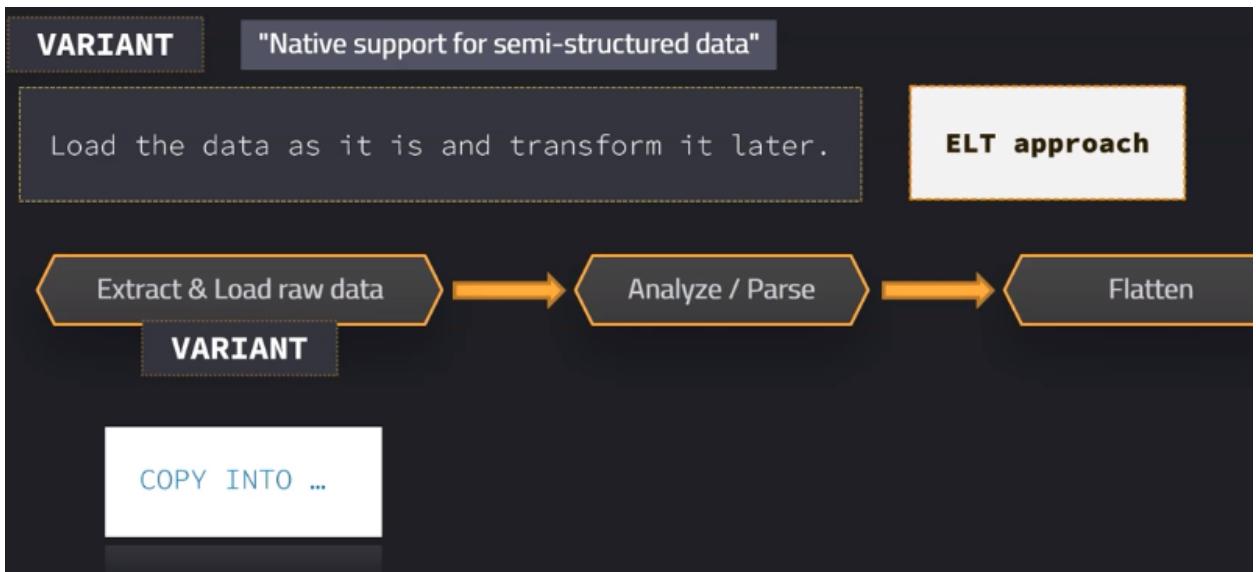
```
SELECT customer_data
FROM customers;
```

Use code with caution. [Learn more](#)

This will return the JSON object that was inserted into the `customer_data` column.

9.

10. Variant will load the data as it is..and we will transform this data later(ELT)



Hands on- semi structured data

1. Refer online

Query semi structured data

1. Lets assume we have this kind of data in variant column

```
{"courses": {  
    "snowflake": {  
        "module1": {  
            "topic": "Introduction",  
            "difficulty": "All levels"  
        },  
        "module2": {  
            "topic": "Loading data",  
            "formats": [  
                "video lectures"  
            ],  
            "difficulty": "All levels"  
        },  
        "module3": {  
            "topic": "Machine learning",  
            "formats": [  
                "video lectures",  
                "hands-on",  
                "quizzes"  
            ]  
        }  
    },  
    "azure": {  
        "module1": {  
            "topic": "Introduction",  
            "formats": [  
                "video lectures",  
                "hands-on",  
                "quizzes"  
            ]  
        }  
    }  
}
```

2. Now if we want to access the courses ...

To access elements of VARIANT column use :

```
SELECT raw_column:courses  
FROM variant_table
```

RAW\_COLUMN:COURSES

```
{ "azure": { "module1": { "formats": [ "video lectures", "hands-on", "quizzes" ], "topic": "Introduction" } } }
```

[[ RAW\_COLUMN:COURSES

```
{ "courses": {  
    "snowflake": {  
        "module1": {  
            "topic": "Introduction",  
            "difficulty": "All levels"  
        },  
        "module2": {  
            "topic": "Loading data",  
            "formats": [  
                "video lectures"  
            ],  
            "difficulty": "All levels"  
        },  
        "module3": {  
            "topic": "Machine learning",  
            "formats": [  
                "video lectures",  
                "hands-on",  
                "quizzes"  
            ]  
        }  
    },  
    "azure": {  
        "module1": {  
            "topic": "Introduction",  
            "formats": [  
                "video lectures",  
                "hands-on",  
                "quizzes"  
            ]  
        }  
    }  
}
```

3. We can also use column number..instead of col name

```
SELECT $1:courses  
FROM variant_table
```

RAW\_COLUMN:COURSES

```
{ "azure": { "module1": { "formats": [ "video lectures", "hands-on", "quizzes" ], "topic": "Introduction" } }, "module2": { "formats": [ "video lectures", "hands-on", "quizzes" ], "topic": "Machine Learning" } }
```

4. If we want to access the snowflake in the variant object..then we have use .

```
SELECT column_name:courses.snowflake  
FROM variant_table
```

RAW\_COLUMN:COURSES.SNOWFLAKE

```
{ "module1": { "difficulty": "All levels", "topic": "Introduction" }, "module2": { "difficulty": "All levels", "formats": [ "video lectures", "hands-on", "quizzes" ], "topic": "Machine Learning" } }
```

5. Here in azure course...we used arrays for storing formats type..to access the arrays we

To access elements of VARIANT column use :  
Use [] to access array elements

use

```
SELECT column_name: courses.azure.module1.formats[1]  
FROM variant_table
```

RAW\_COLUMN:COURSES.AZURE.MODULE1.FORMATS[1]

1	"hands-on"

6. If we want to remove semicolons for hands -on we use

```
SELECT  
    column_name: courses.azure.module1.formats[1]::VARCHAR  
FROM variant_table
```

## HandsOn query semi structured data

1. Will practice hands on

## Flatten Hierarchical data

Data flattening is the process of converting complex, nested data into a simpler, tabular format. This can be useful for a variety of reasons, such as:

1.
  - To make data easier to query and analyze
  - To improve the performance of data processing applications
  - To reduce the storage space required to store data

2.

FLATTEN(INPUT => <expression> )

Used to convert semi-structured data into relational table view

3. The flatten function is a table function.
4. This means we can also query from it when we include it in table.

FLATTEN(INPUT => <expression> )	Used to convert semi-structured data into relational table view																												
SELECT * FROM TABLE(FLATTEN(INPUT => [2,4,6]))	Cannot be used in COPY command!																												
<table border="1"> <thead> <tr> <th></th><th>SEQ</th><th>KEY</th><th>PATH</th><th>INDEX</th><th>VALUE</th><th>THIS</th></tr> </thead> <tbody> <tr> <td>1</td><td>1</td><td>null</td><td>[0]</td><td>0</td><td>2</td><td>[ 2, 4, 6 ]</td></tr> <tr> <td>2</td><td>1</td><td>null</td><td>[1]</td><td>1</td><td>4</td><td>[ 2, 4, 6 ]</td></tr> <tr> <td>3</td><td>1</td><td>null</td><td>[2]</td><td>2</td><td>6</td><td>[ 2, 4, 6 ]</td></tr> </tbody> </table>		SEQ	KEY	PATH	INDEX	VALUE	THIS	1	1	null	[0]	0	2	[ 2, 4, 6 ]	2	1	null	[1]	1	4	[ 2, 4, 6 ]	3	1	null	[2]	2	6	[ 2, 4, 6 ]	Produces a lateral view: Contains references to other tables in FROM clause
	SEQ	KEY	PATH	INDEX	VALUE	THIS																							
1	1	null	[0]	0	2	[ 2, 4, 6 ]																							
2	1	null	[1]	1	4	[ 2, 4, 6 ]																							
3	1	null	[2]	2	6	[ 2, 4, 6 ]																							

5.

```
SELECT
  RAW_FILE:id::int as id,
  RAW_FILE:first_name::STRING as first_name,
  VALUE::STRING as prev_company
FROM OUR_FIRST_DB.PUBLIC.JSON_RAW
,TABLE(flatten ( input => RAW_FILE:prev_company ));
```

cts Editor Results Chart			
ID	FIRST_NAME	PREV_COMPANY	...
2	Dag	MacGyver, Kessler and Corwin	
2	Dag	Gerlach, Russel and Moen	
3	Heath	Schaden LLC	
3	Heath	Reynolds LLC	
6	Austina	Rath Inc	
8	Remington	Kuhlman LLC	
8	Remington	Lockman, Kunze and Bartoletti	

6.

## Insert JSON Data

- Lets see how we can manually insert json data in variant

2. First we use parse\_join..to convert our data into JSON format

```
SELECT parse_json(' { "key1": "value1", "key2": "value2" } '');
```

3. Now we will create a table with VARIANT data

4. Then we insert

```
CREATE OR REPLACE TABLE semi_structured (data variant);

-- Insert data using PARSE_JSON
INSERT INTO semi_structured SELECT parse_json(' { "key1": "value1", "key2": "value2" } '');
```

-- Query from table

5. **SELECT DATA:key1 FROM semi\_structured** It gives output “value1”

6. Now we insert data into our existing table JSON\_RAW

```
INSERT INTO OUR_FIRST_DB.PUBLIC.JSON_RAW SELECT (parse_json('
{
    "id":9,
    "first_name":"Kelcey",
    "last_name":"Pavlenko",
    "gender":"Male",
    "city":"Zhuyeping",
    "job":[{"title":"Pharmacist","salary":31100},
    "spoken_languages":["Nepali","English"]},
    '))
;
```

7. Now we query this semi structured data by flatten it

```
SELECT
    RAW_FILE:id::int as id,
    RAW_FILE:first_name::STRING as first_name,
    VALUE::STRING as prev_company
FROM OUR_FIRST_DB.PUBLIC.JSON_RAW
,TABLE(flatten ( input => RAW_FILE:prev_company ));
```

```
SELECT
    RAW_FILE:id::int as id,
    RAW_FILE:first_name::STRING as first_name,
    VALUE::STRING as prev_company
FROM OUR_FIRST_DB.PUBLIC.JSON_RAW
,lateral flatten ( input => RAW_FILE:prev_company ));
```

- 8.

## 9. Flatten function

The FLATTEN function in Snowflake is a table function that explodes (converts) nested VARIANT, OBJECT, and ARRAY data into rows. It can be used to convert semi-structured data to a relational representation, which makes it easier to query and analyze.

Here is an example of how to use the FLATTEN function to flatten an ARRAY column:

SQL

```
CREATE OR REPLACE TABLE my_table (
    id INT,
    names ARRAY<STRING>
);

INSERT INTO my_table (id, names) VALUES
(1, ARRAY['Alice', 'Bob', 'Carol']),
(2, ARRAY['Dave', 'Eve', 'Frank']);

SELECT id, name
FROM my_table
LATERAL FLATTEN(names) AS t;
```

Output:

id	name
--	---
1	Alice
1	Bob
1	Carol
2	Dave
2	Eve
2	Frank

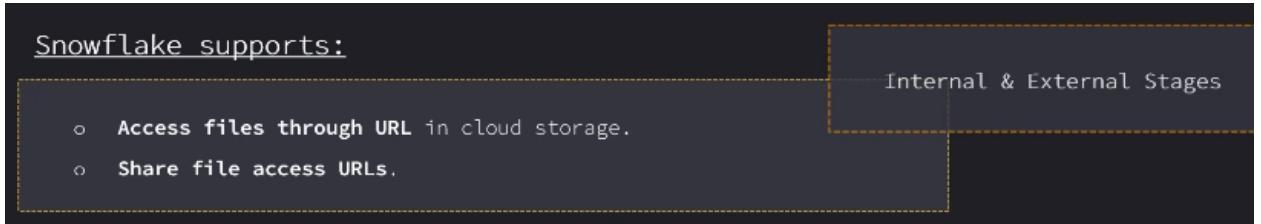
## Unstructured Data

What is unstructured data?

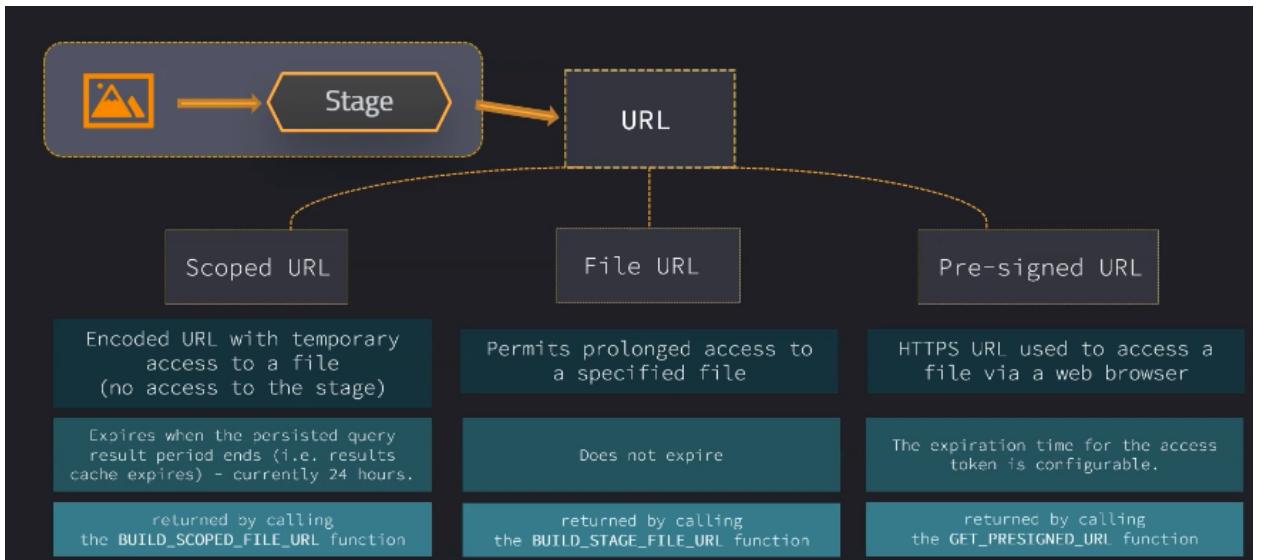
- Does not fit into any pre-defined data model
  - video files 
  - audio files 
  - documents 

1.

- And in Snowflake, we can set up URLs through which we can access those files in the cloud storage. And we can also use these URLs to share these files with other people.



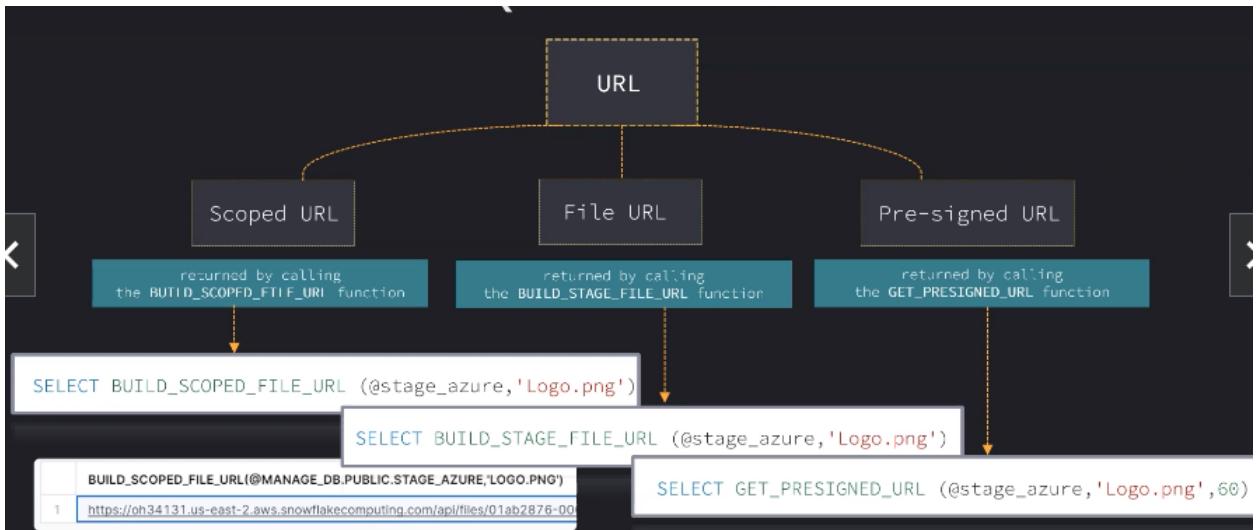
- Lets suppose we have an image in our stage
- Now we can generate URLs to access this file and we have 3 types of URLs available



- TO access them we use

```
SELECT BUILD_SCOPED_FILE_URL (@stage_azure, 'Logo.png')
```

- at `@stage_azure`  
...we have to give path to our file...if it is directly at the stage..then no need of giving it



7.

## 8. Example

`| SELECT GET_PRESIGNED_URL (@stage_azure,'Logo.png',60)`

Objects Editor Results Chart

GET\_PRESIGNED\_URL (@STAGE\_AZURE,'LOGO.PNG')

<https://demosnowflakeintegration.blob.core.windows.net/csvcontainer/Logo.png?sv=2021-04-10&se=2023-03-24T13%3A43%3A47Z&sko>

9. Now This url will expire in 60 sec

## Directory Tables

What is a directory table?

**Stores metadata of staged files**

- o Layered on a stage
- o Can be queried with sufficient privileges (on stage)
- o Retrieve file URLs to access files

1.

Needs to be enabled for stages

```
CREATE STAGE stage_azure  
URL = <'url'>  
STORAGE_INTEGRATION = integration  
DIRECTORY = (ENABLE = TRUE)
```

DEFAULT  
FALSE

```
ALTER STAGE stage_azure  
SET DIRECTORY = (ENABLE = TRUE)
```

2.

```
SELECT * FROM DIRECTORY(@stage_azure)
```

3. Now if we run

4. We get no results...thats because we have to refresh our stage using

```
ALTER STAGE stage_azure REFRESH;
```

5. Now if we query our directory we get

	RELATIVE_PATH	SIZE	LAST_MODIFIED	MD5	ETAG	FILE_URL
1	Logo.png	46,603	1:19:22.000 -0700	e24ca582347f57f	"0x8DB2C510"	<a href="https://oh34131.us">https://oh34131.us</a>
2	myfile_0_0_0.csv.gz	33,785	1:18:52.000 -0700	a3169775b4e20a0	"0x8DB29241"	<a href="https://oh34131.us">https://oh34131.us</a>

6. Now this file\_url is a scoped URL(refer prev lec)

Scoped URL

BUILD\_SCOPED\_FILE\_URL function

```

1   ALTER STAGE stage_azure
2     SET DIRECTORY = (ENABLE = TRUE)
3
4     SELECT * FROM DIRECTORY( @<stage_name> )
5
6     SELECT * FROM DIRECTORY( @stage_azure ) || SELECT * FROM DIRECTORY( @stage_azure ) ||
7
8   ALTER STAGE stage_azure REFRESH;
9
10
11

```

Objects    Editor    Results    Chart

	file	status	...	description
1	myfile_0_0_0.csv.gz	REGISTERED_NEW		File registered successfully.
2	Logo.png	REGISTERED_NEW		File registered successfully.

7.

## Data Sampling

1. Imagine we have 10TB of data..we need to query some data from this table...
2. Querying from 10tb table will require more time and compute resources
3. So we sample certain amount of data size ..from the table

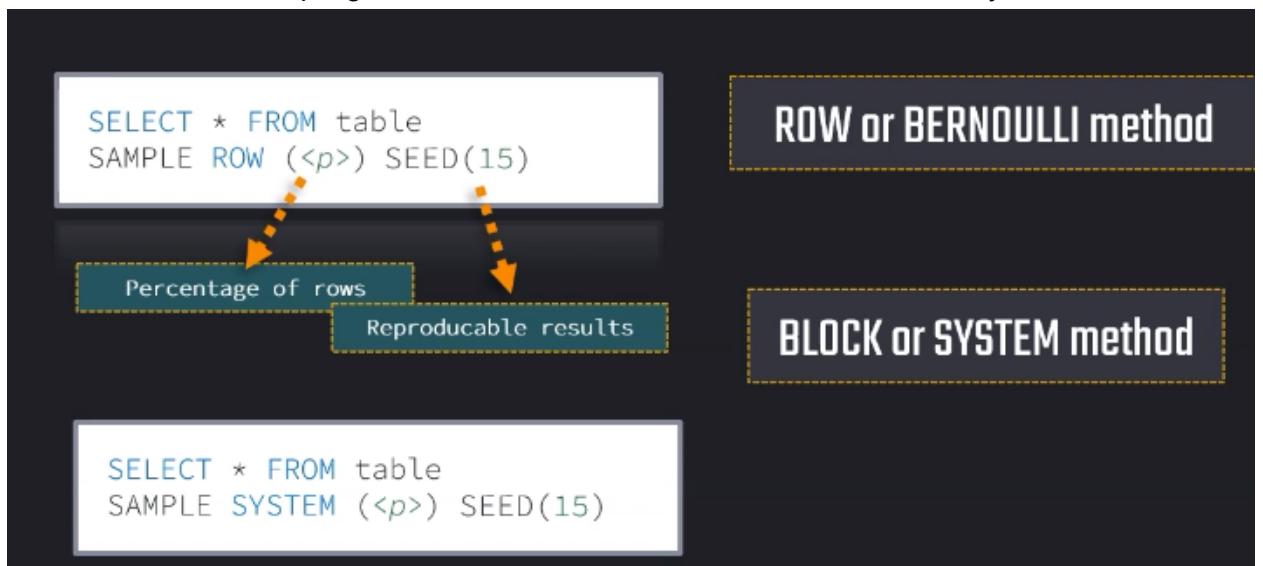


4. It is useful for testing ,data analysis and query dev

## Why Sampling?

- Use-cases: Query development, data analysis etc.
- Faster & more cost efficient (less compute resources)

5. We have two data sampling method ...one is row and next one is Block or System



- 6.

7. Here seed produces reproducible results so that other people can also execute the same query and get exactly the same results.

ROW or BERNOULLI method	BLOCK or SYSTEM method
Every row is chosen with percentage $p$	Every block is chosen with percentage $p$
More "randomness"	More effective processing
Smaller tables	Larger tables

- 8.

9. In bernoulli ..every row has 10% of chance to be in the sampled data table...in Block...there will be micro partitions of entire table..and it will choose p% from their

10. Bernoulli is more randomness..as it goes to every row in the table .....block is more effective processing..
11. As bernoilli goes to every row..we have to use that in smaller table...and BLOCK for larger tables
12. Once look thru hands on/check online

## Tasks

**Task**

- Used to schedule execution of SQL statement / stored procedures
- Often combined with **streams** to set up continuous ETL workflows

```
CREATE TASK my_task
WAREHOUSE = my_wh
SCHEDULE = '15 MINUTE'
AS
INSERT INTO my_table(time_col) VALUES(CURRENT_TIMESTAMP);
```

- 1.
2. So to setup a task we need to have execute manage task on account level
3. Then we also need to have create task privilege on the schema level and usage on the warehouse.

```
CREATE TASK my_task
WAREHOUSE = my_wh
SCHEDULE = '15 MINUTE'
AS
INSERT INTO my_table(time_col) VALUES(CURRENT_TIMESTAMP);
```

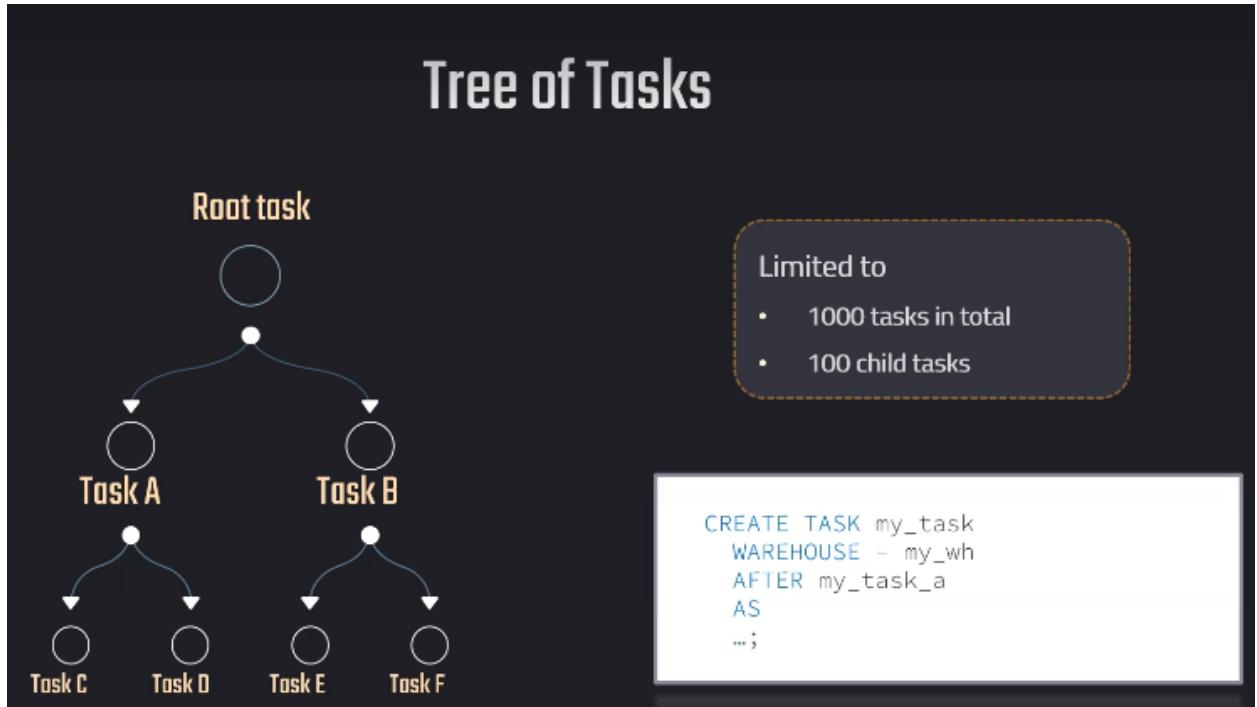
4. If we are not specifying any warehouse..then it will use snowflake managed compute

```
CREATE TASK my_task
WAREHOUSE = my_wh
SCHEDULE = '15 MINUTE'
AS
INSERT INTO my_table(time_col) VALUES(CURRENT_TIMESTAMP);
```

5. So to start a task ..we need to use resume...and if we want to stop..we use suspend

```
ALTER TASK my_task RESUME;  
ALTER TASK my_task SUSPEND;
```

6. We can also have DAG



A directed acyclic graph (DAG) in Snowflake tasks is a series of tasks that are organized by their dependencies. A DAG has a single root task, and each task can have multiple child tasks. The child tasks cannot depend on the root task, and there cannot be any cycles in the graph.

DAGs are useful for automating complex workflows that consist of multiple tasks. For example, you could use a DAG to automate the following workflow:

1. Load data from a source system into a Snowflake stage.
2. Transform the data in the stage.
3. Load the transformed data into a Snowflake table.
4. Run a machine learning model on the data in the table.
5. Generate a report based on the results of the machine learning model.

7.

## 8. Example code for dag

```
CREATE TASK load_data
WAREHOUSE = my_warehouse
SCHEMA = my_schema
STATEMENT = 'COPY INTO my_stage FROM @my_source_system';

CREATE TASK transform_data
WAREHOUSE = my_warehouse
SCHEMA = my_schema
STATEMENT = 'CREATE OR REPLACE TABLE my_table SELECT ... FROM my_stage';

CREATE TASK run_model
WAREHOUSE = my_warehouse
SCHEMA = my_schema
STATEMENT = 'CALL my_model(my_table)';

CREATE TASK generate_report
WAREHOUSE = my_warehouse
SCHEMA = my_schema
STATEMENT = 'CREATE OR REPLACE TABLE my_report SELECT ... FROM my_table';

ALTER TASK load_data
ADD DEPENDENCY NONE;

ALTER TASK transform_data
ADD DEPENDENCY load_data;

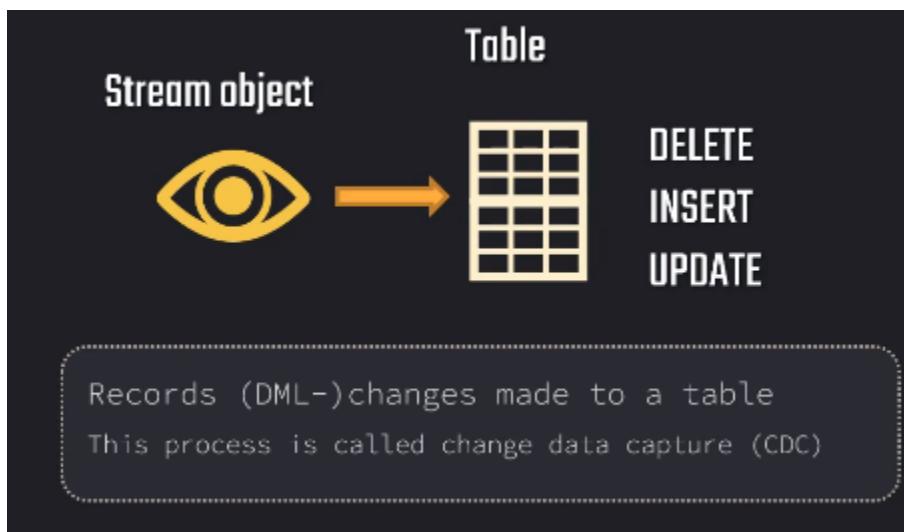
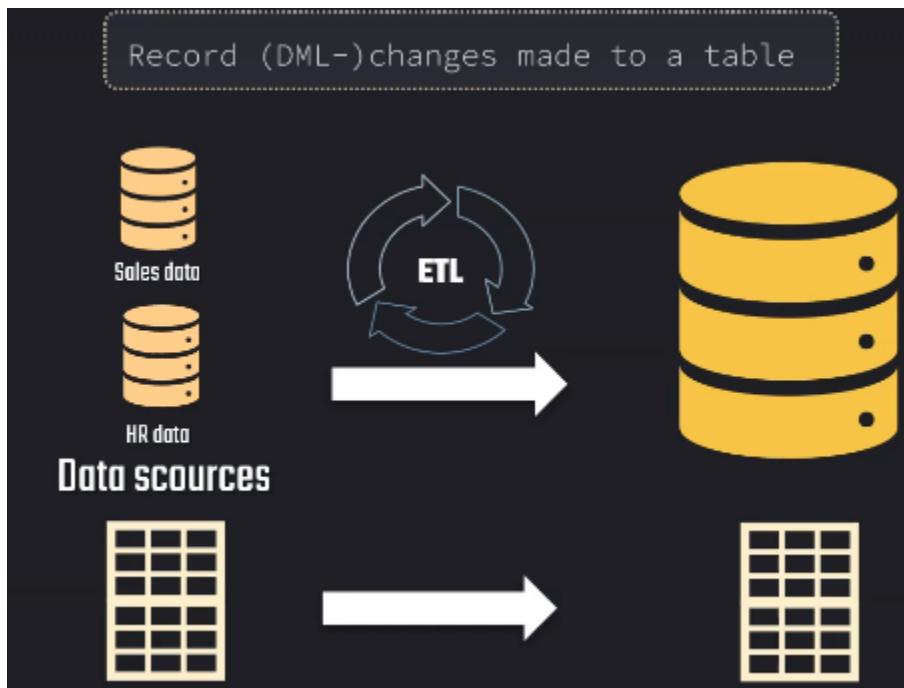
ALTER TASK run_model
ADD DEPENDENCY transform_data;

ALTER TASK generate_report
ADD DEPENDENCY run_model;
```

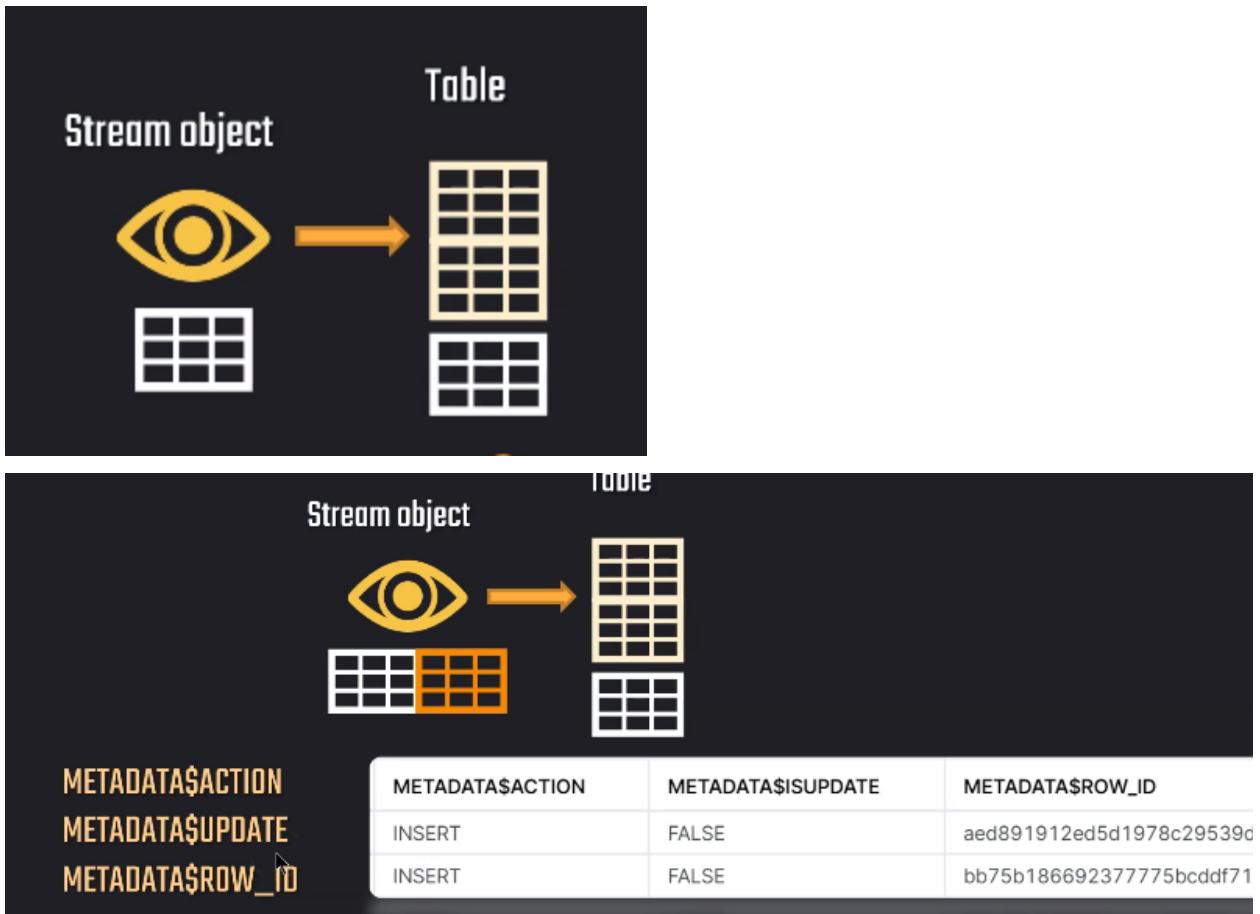
## Stream

1. A stream is an object that can be used to record data manipulation, language changes.
2. So if there are some inserts, updates or deletes made on a table. A stream can record those changes.

3. It is also useful to track changes...and update these changes in our warehouse



- 4.
5. For example...if there's any new rows inserted in our table...stream tracks it and stores the rows that are inserted..and it also stores the meta data



6. To setup a stream we use

```
CREATE STREAM my_stream
ON TABLE my_table
```

Create Stream

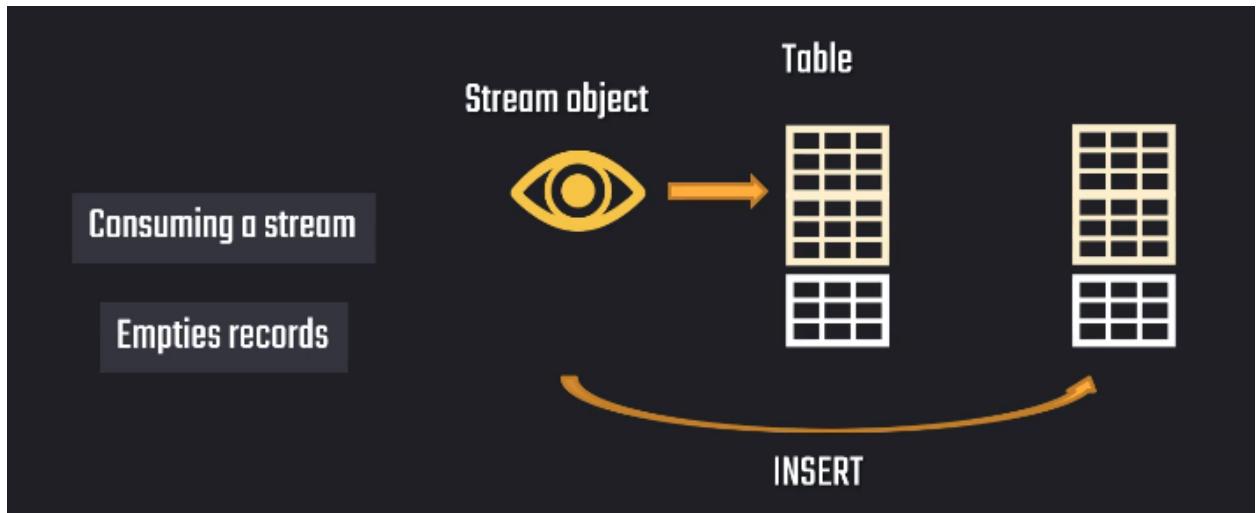
7. Also we can query(new data and metadata) using streams

```
SELECT * FROM my_stream
```

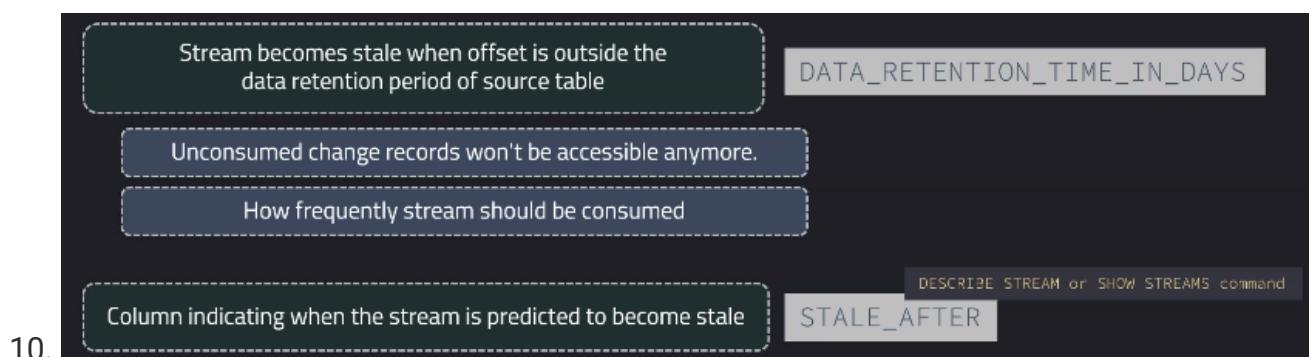
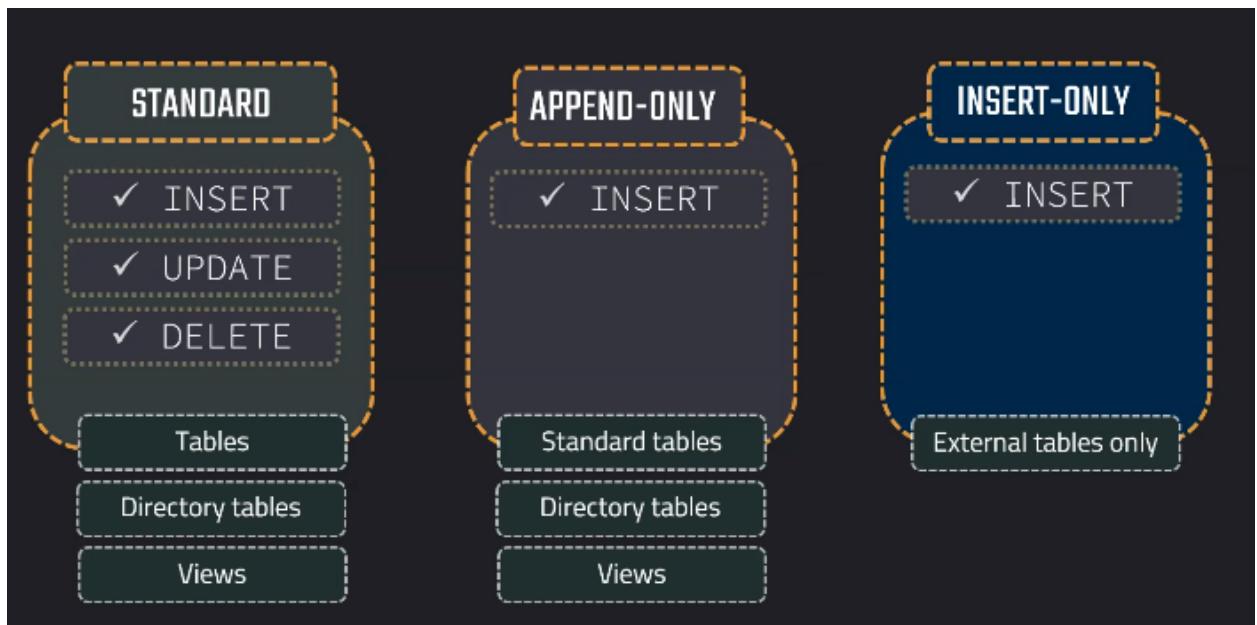
We can query from stream

8. In stream...after we insert this newly data to the table..or update the table with existing data...then this will empty the stream object becuz we have consumed

the changes in the stream..and we dont need them anymore



9. So we have 3 types of streams



10.

11. To check about stream staleness...we use describe stream or show stream command

12.	Stream extends retention to 14 days (default). Regardless of Snowflake edition	DEFAULT=14	MAX_DATA_EXTENSION_TIME_IN_DAYS
-----	---	------------	---------------------------------

To use streams and tasks together in Snowflake, you can use the following steps:

1. Create a stream on the table that you want to monitor for changes.
2. Create a task that executes the desired SQL statement on the stream data.
3. Configure a dependency between the stream and the task so that the task is executed whenever new data is added to the stream.

13.

```
CREATE TASK my_task
  WAREHOUSE = my_wh
  SCHEDULE = '15 MINUTE'
  WHEN SYSTEM$STREAM_HAS_DATA('MY_STREAM')
  AS
    INSERT INTO my_table(time_col) VALUES(CURRENT_TIMESTAMP);
```

## Handson streams and tasks

1.