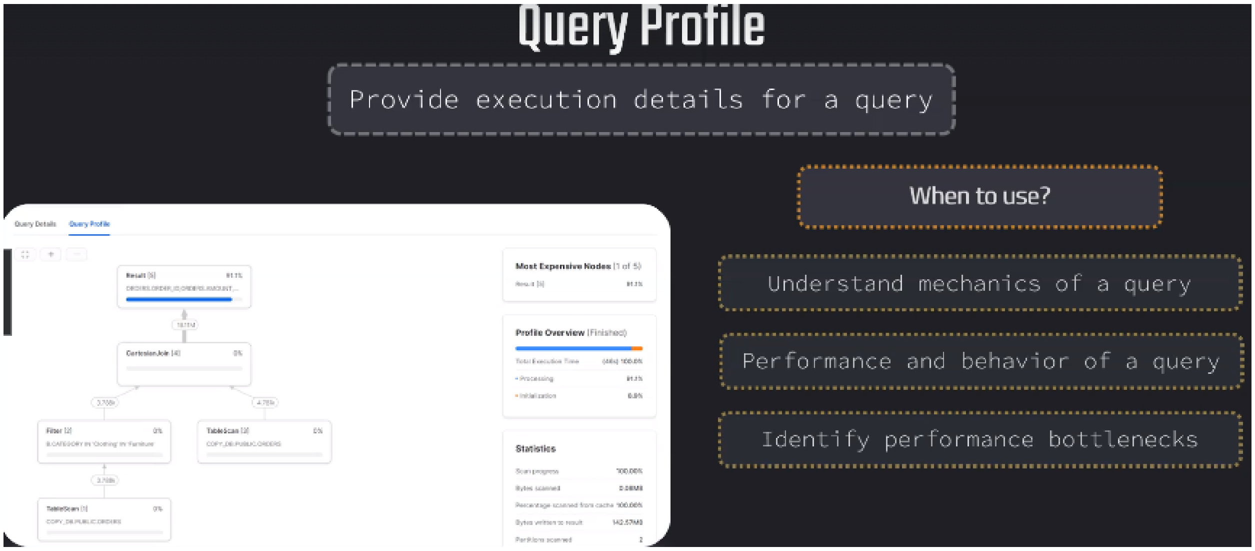
Query Profile & history

1. Query profile gives execution details
2. This is a graphical representation that we can use to get additional execution details for a given query.
3. Using this we can understand the mechanics of the query
4. 

Query Profile

Provide execution details for a query

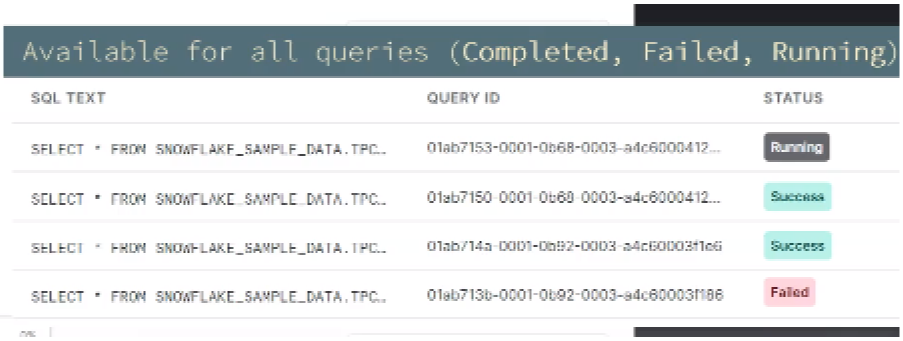
When to use?

Understand mechanics of a query

Performance and behavior of a query

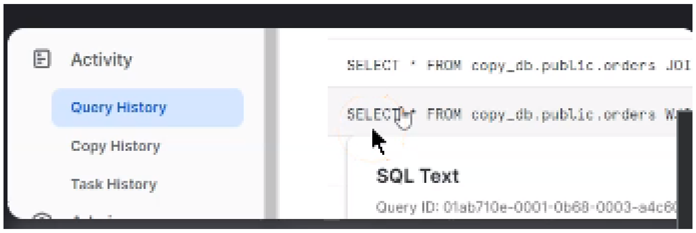
Identify performance bottlenecks

5. We can find the bottlenecks(point of lag) ..and can try to remove bottlenecks
6. It is available for all types of queries…like running,failed etc



Available for all queries (Completed, Failed, Running)

SQL TEXT	QUERY ID	STATUS
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPC...	01ab7153-0001-0b68-0003-a4c6000412...	Running
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPC...	01ab7150-0001-0b68-0003-a4c6000412...	Success
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPC...	01ab714a-0001-0b92-0003-a4c60003f1e6	Success
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPC...	01ab713b-0001-0b92-0003-a4c60003f186	Failed

7. We can find query profile at 

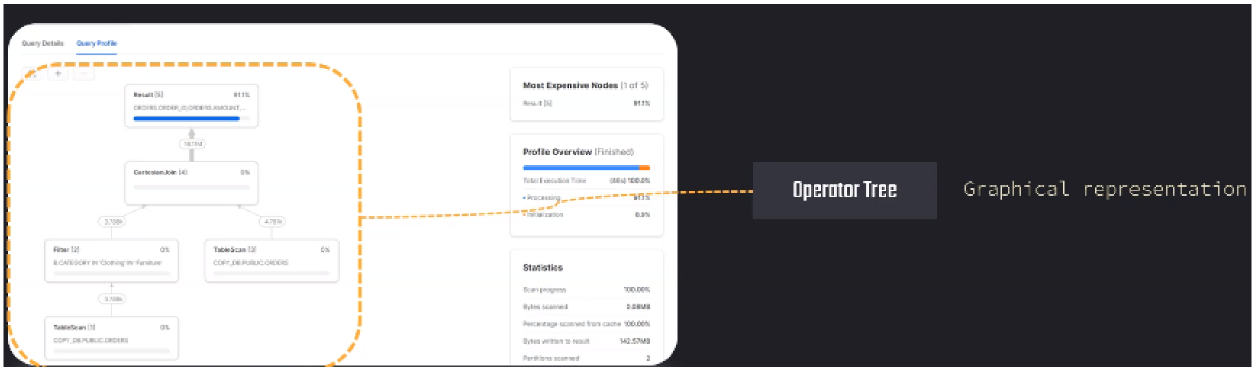
SELECT * FROM copy_db.public.orders JOI

SELECT * FROM copy_db.public.orders W

SQL Text

Query ID: 01ab710e-0001-0b68-0003-a4c6

8. Now lets look into query profile…first we have a operator tree

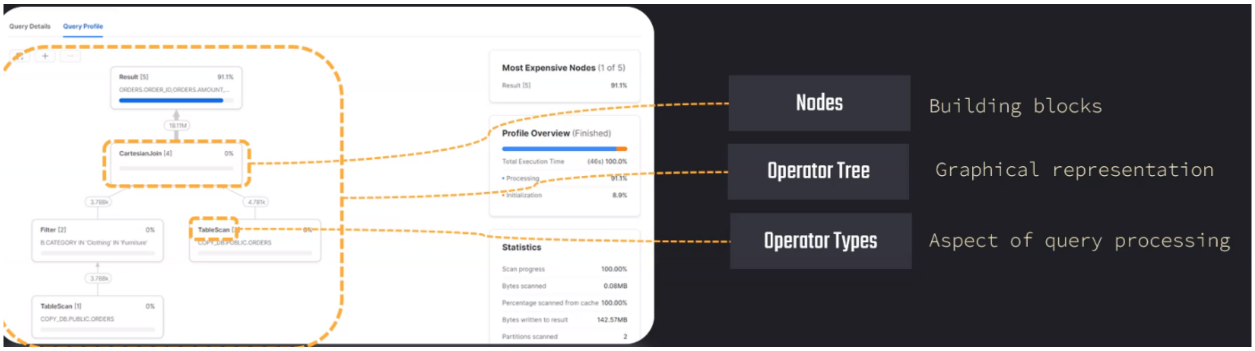


Most Expensive Nodes (1 of 5)
Result [5] 91.1%

Profile Overview (Finished)
Total Execution Time (46s) 100.0%
• Processing 91.1%
• Initialization 8.9%

Statistics
Scan progress 100.00%
Bytes scanned 0.08MB
Percentage scanned from cache 100.00%
Bytes written to result 142.57MB
Partitions scanned 2

9. In herewe have different nodes..which are building blocks of our query…and each node have diff operator type

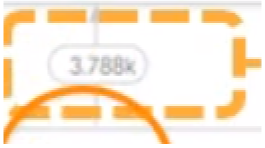


Most Expensive Nodes (1 of 5)
Result [5] 91.1%

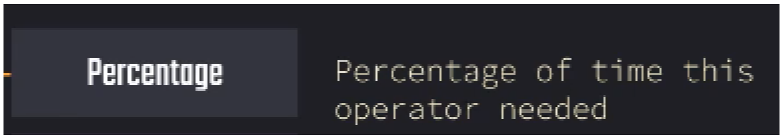
Profile Overview (Finished)
Total Execution Time (46s) 100.0%
• Processing 91.1%
• Initialization 8.9%

Statistics
Scan progress 100.00%
Bytes scanned 0.08MB
Percentage scanned from cache 100.00%
Bytes written to result 142.57MB
Partitions scanned 2

10. We can also see dataflow…it says that how many rows have been processed

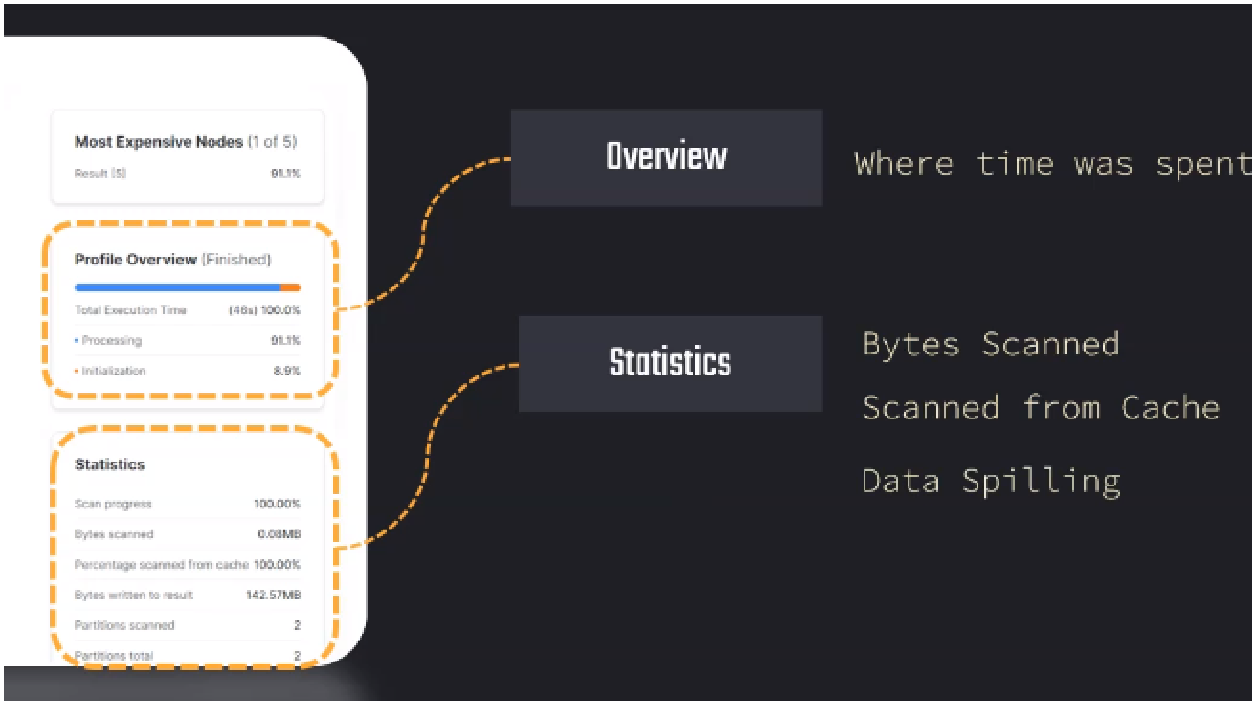


11. Then we have a percentage



Percentage of time this operator needed for our query

12. And to the right we have a dashboard



Most Expensive Nodes (1 of 5)

Result (5)	91.1%

Profile Overview (Finished)

Total Execution Time	(48s) 100.0%
• Processing	91.1%
• Initialization	8.9%

Statistics

Scan progress	100.00%
Bytes scanned	0.08MB
Percentage scanned from cache	100.00%
Bytes written to result	142.57MB
Partitions scanned	2
Partitions total	2

Overview

Where time was spent

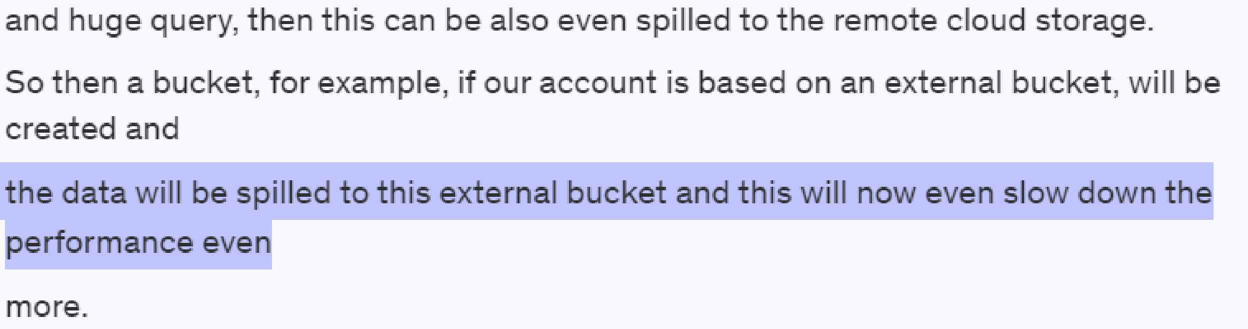
Statistics

Bytes Scanned

Scanned from Cache

Data Spilling

13. What is data spilling?
14. First we will see when does it occur
15. Lets suppose we have a small warehouse…when our data doesnt fit in this warehouse memory….it spills the memory to local storage….and in the end ..it decreases the performance
16. And even if our local storage maxes out…then it goes to the remote storage and spills data…

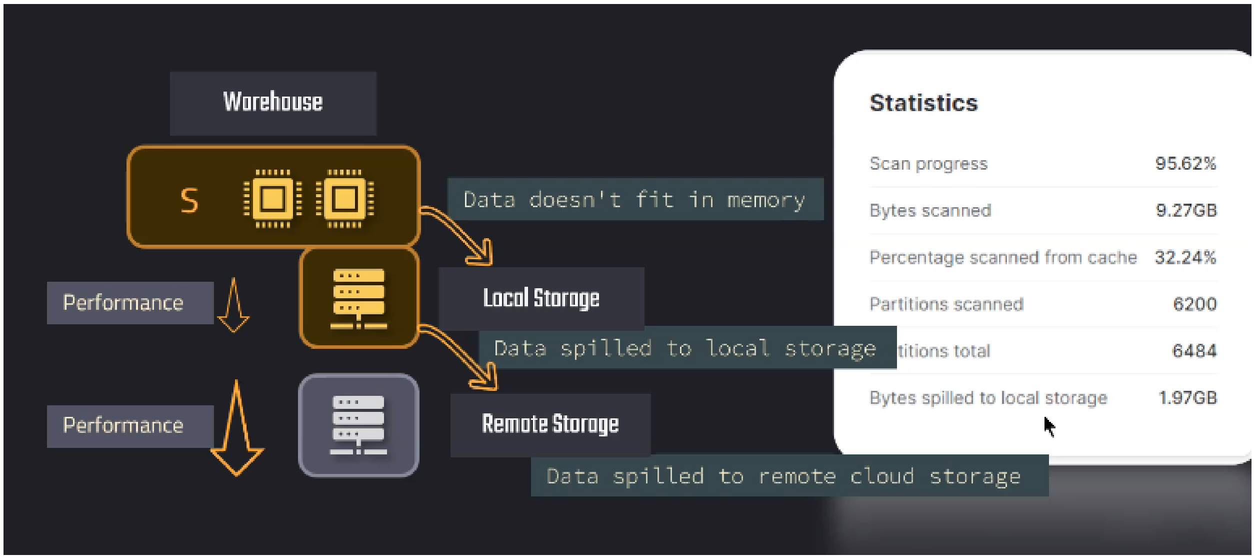


and huge query, then this can be also even spilled to the remote cloud storage.

So then a bucket, for example, if our account is based on an external bucket, will be created and

the data will be spilled to this external bucket and this will now even slow down the performance even

more.

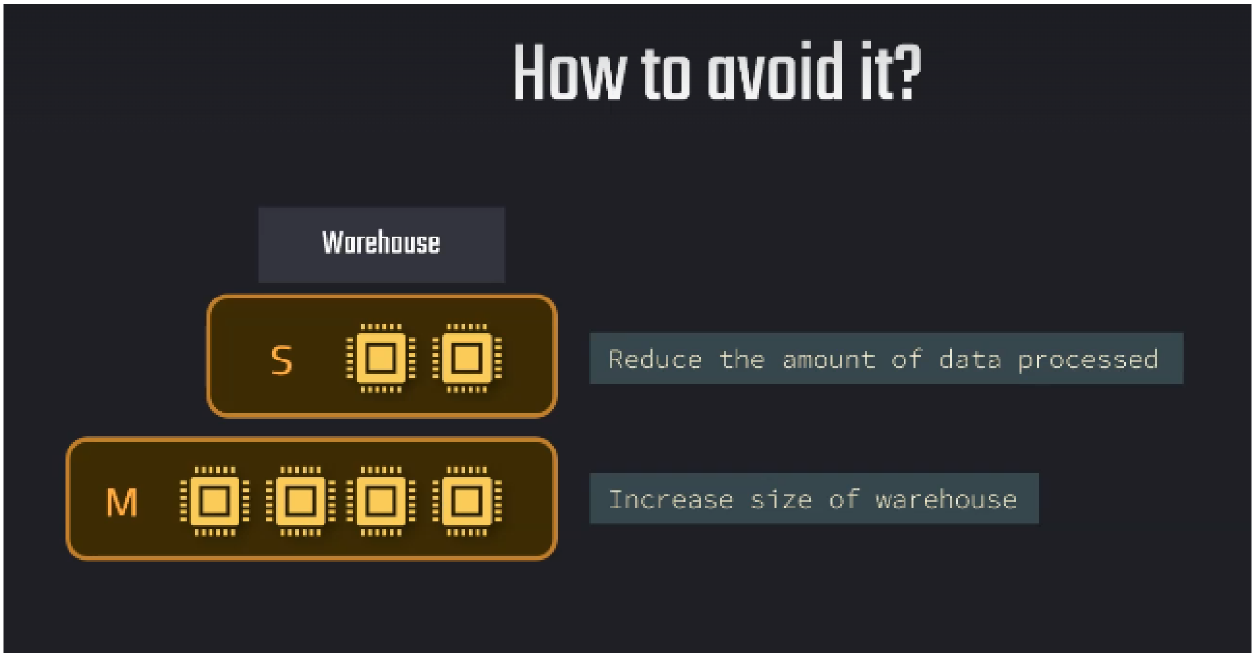


Warehouse
S
Data doesn't fit in memory
Performance
Local Storage
Data spilled to local storage
Performance
Remote Storage
Data spilled to remote cloud storage

Statistics

Scan progress	95.62%
Bytes scanned	9.27GB
Percentage scanned from cache	32.24%
Partitions scanned	6200
titions total	6484
Bytes spilled to local storage	1.97GB

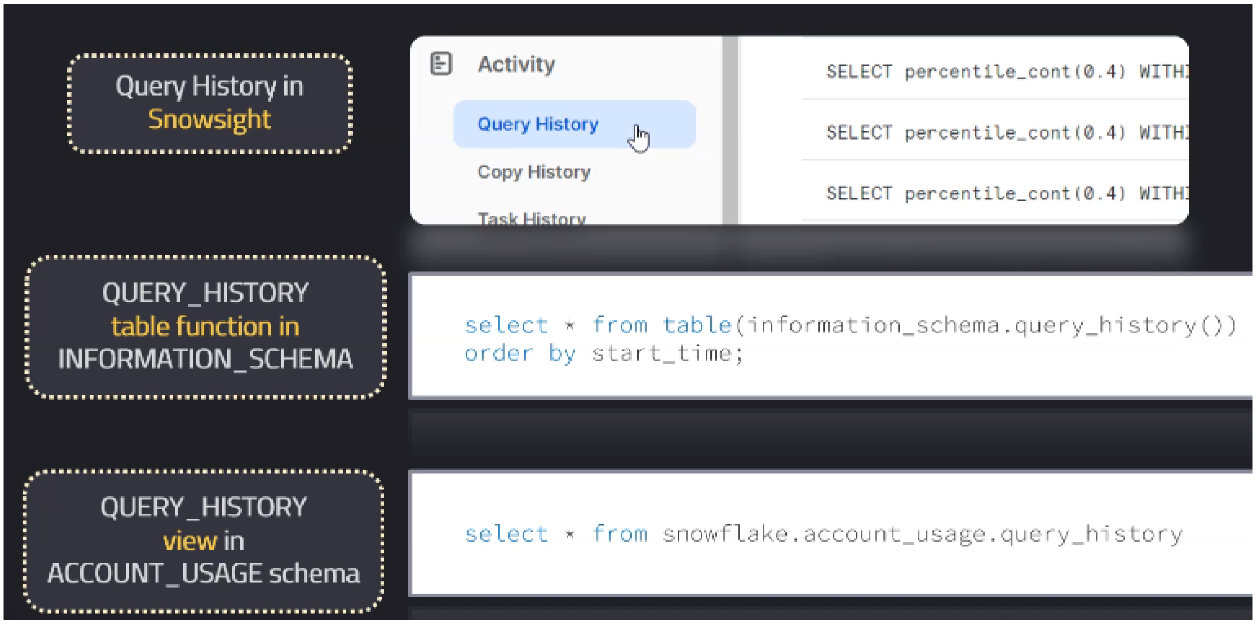
17. How can we avoid this data spilling?
18. We can reduce the number of queries running at same time and



How to avoid it?

Warehouse
S
Reduce the amount of data processed
M
Increase size of warehouse

19. How to access query history?

20. 

Query History in Snowsight

```
SELECT percentile_cont(0.4) WITHI
SELECT percentile_cont(0.4) WITHI
SELECT percentile_cont(0.4) WITHI
```

QUERY_HISTORY table function in INFORMATION_SCHEMA

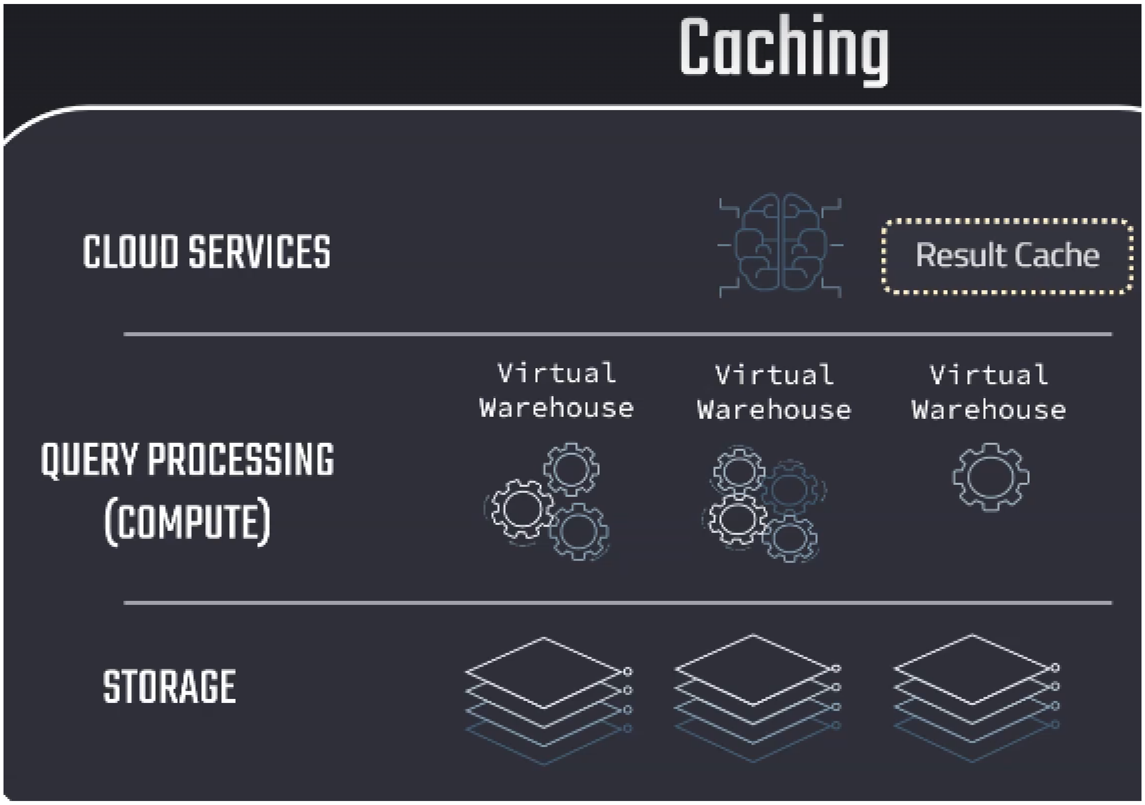
```
select * from table(information_schema.query_history())
order by start_time;
```

QUERY_HISTORY view in ACCOUNT_USAGE schema

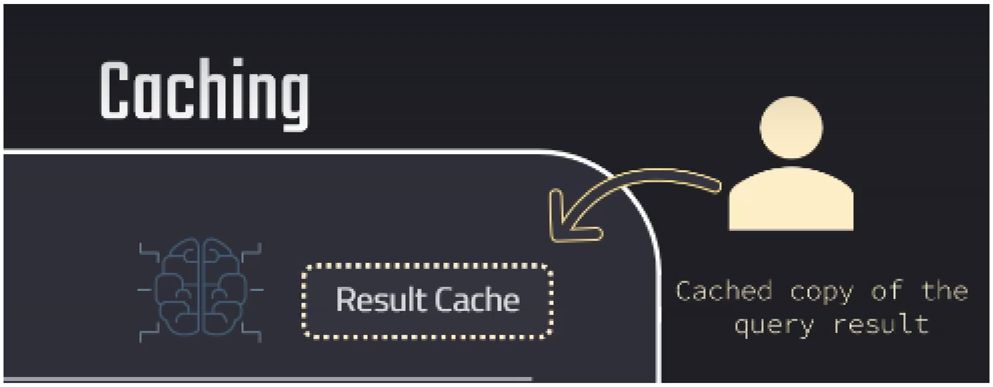
```
select * from snowflake.account_usage.query_history
```

Caching

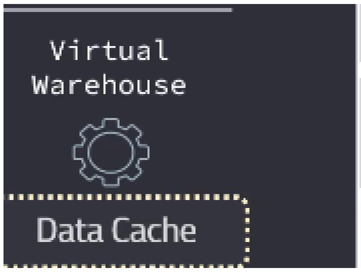
1. In Snowflake, we have three mechanisms for caching. They are in different locations and we need to understand those three caching mechanisms.

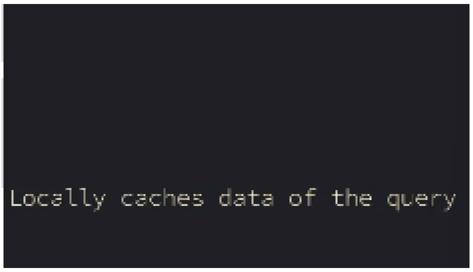


2. Here first we have result cache….it is located in cloud service layer..so we dont need any storage or compute resources for this
3. When we execute a query…a copy of this result..is stored in result cache

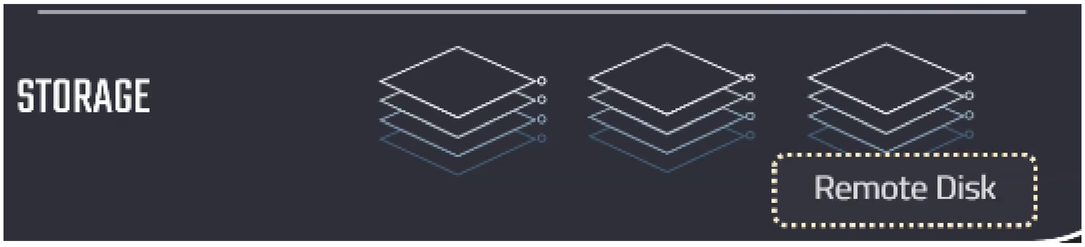


4. Next we have data cache.,..also called as virtual warehouse cache..and it is connected directly with our local SSD

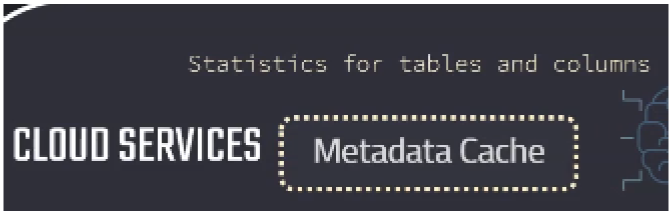




So this cannot be shared with another warehouse.

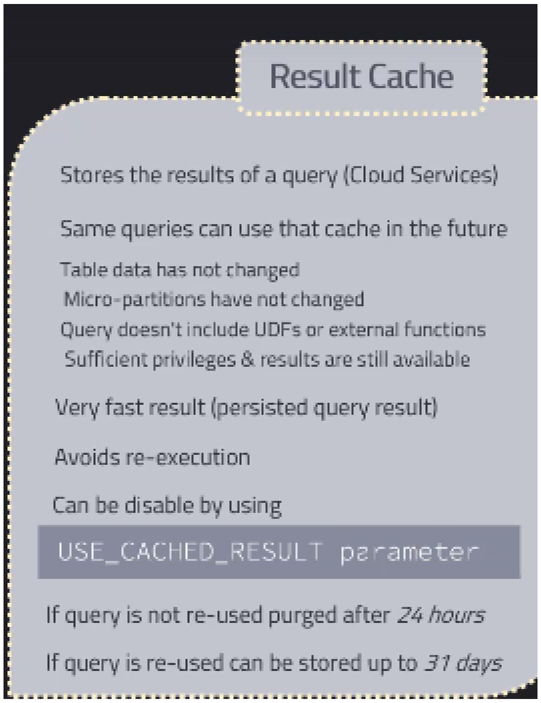
5. In the query profile, we can see this under the name of local disk I/O.
6. Next we have remote disk…if the data is not cached..then we need to retrieve this data from storage layer..this is not a cache mechanism
7. 

8. Then we have metadata cache…it contains statistics for tables,cloumns and micro partitions



9. Lets now look into result cache…
10. It stores the result of a query ..when executed…
11. And if the same query runs again..we get the result faster…but the table data must not be modified..
12. The results in result cache are stored for 24hrs….and if run our query again in this 24hrs….then the result will be stored for next 24hrs

13. If query is reused…result cache can store this result for 31days



Result Cache

Stores the results of a query (Cloud Services)

Same queries can use that cache in the future

Table data has not changed
Micro-partitions have not changed
Query doesn't include UDFs or external functions
Sufficient privileges & results are still available

Very fast result (persisted query result)

Avoids re-execution

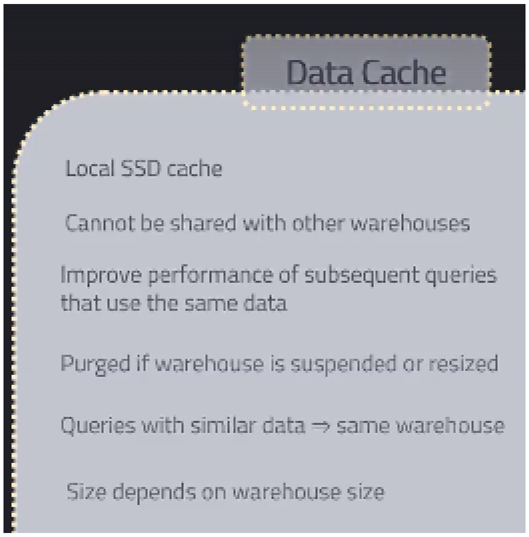
Can be disable by using

`USE_CACHED_RESULT parameter`

If query is not re-used purged after *24 hours*

If query is re-used can be stored up to *31 days*

14. Data Cache
15. It is a local SSD of a given warehouse
16. We can optimize this by using the queries with similar data..in the same warehouse



Data Cache

Local SSD cache

Cannot be shared with other warehouses

Improve performance of subsequent queries that use the same data

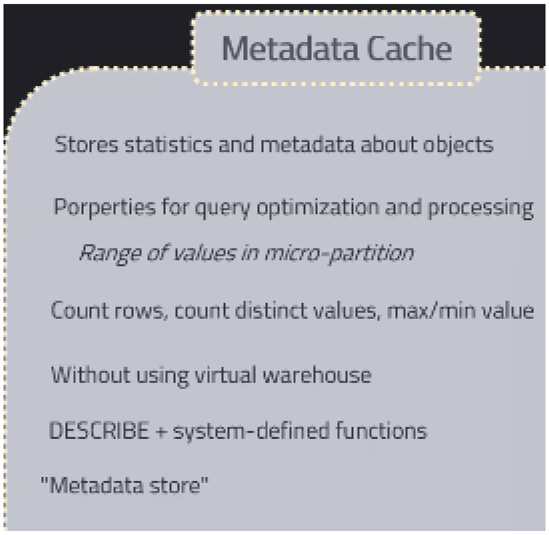
Purged if warehouse is suspended or resized

Queries with similar data ⇒ same warehouse

Size depends on warehouse size

17. Metadata cache
18. It stores the pre calculated statistic of the object…like count of rows, max/min values etc
19. For example it can be range of values in micro-partition
20. And also for commands like DESCRIBE …this metadata will be helpful

21. This is sometimes referred as a metadata store



Metadata Cache

Stores statistics and metadata about objects

Porperties for query optimization and processing

Range of values in micro-partition

Count rows, count distinct values, max/min value

Without using virtual warehouse

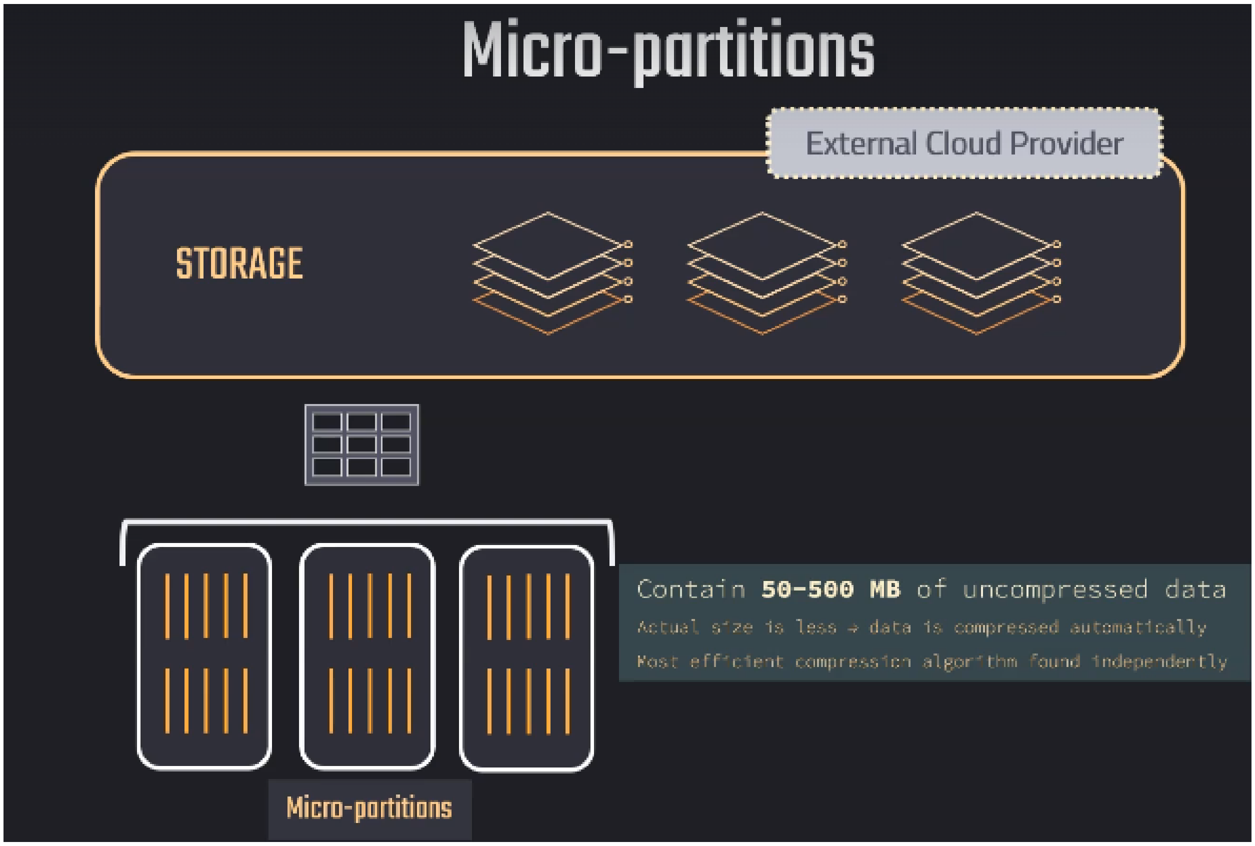
DESCRIBE + system-defined functions

"Metadata store"

22. And for the virtual private edition, there can be a dedicated metadata store that is not shared with any other customers.
23. For hands on refer file

Micro Partitions

1. Actually our data is stored in the external cloud provider
2. But the question is how is now the data in a table actually stored with this cloud provider?
3. Data is stored in micro-partitions..which is very specific to snowflake architecture

4. 

Micro-partitions

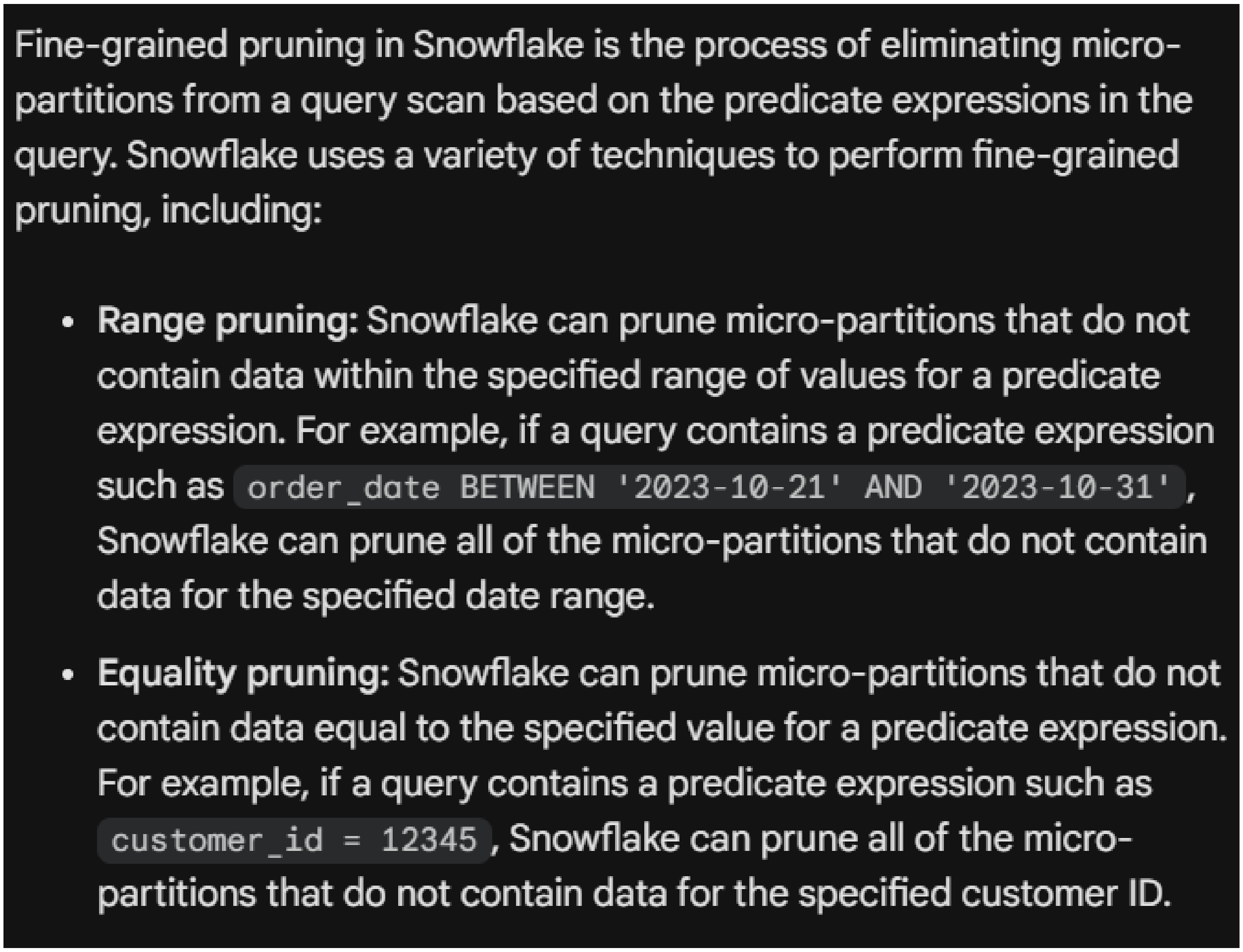
Contain **50-500 MB** of uncompressed data

Actual size is less → data is compressed automatically

Most efficient compression algorithm found independently

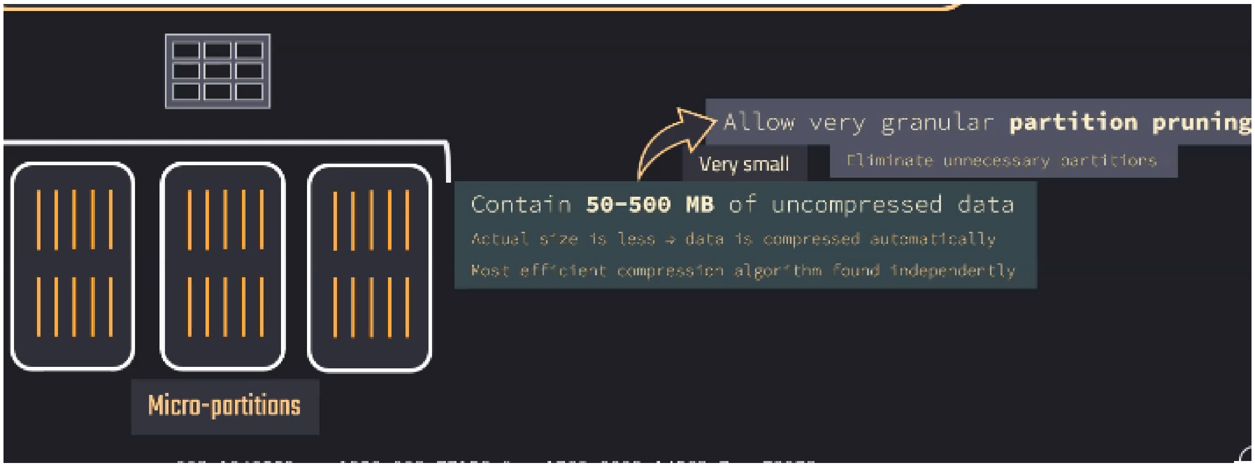
5. But actual size of this data is very less…snowflake finds the best compression algo for each partition and compresses each partition
6. Also it allow granular partition pruning…means it eliminates the unnecessary partition required forn the query

7. Bard on granular pruning



Fine-grained pruning in Snowflake is the process of eliminating micro-partitions from a query scan based on the predicate expressions in the query. Snowflake uses a variety of techniques to perform fine-grained pruning, including:

- **Range pruning:** Snowflake can prune micro-partitions that do not contain data within the specified range of values for a predicate expression. For example, if a query contains a predicate expression such as `order_date BETWEEN '2023-10-21' AND '2023-10-31'`, Snowflake can prune all of the micro-partitions that do not contain data for the specified date range.
- **Equality pruning:** Snowflake can prune micro-partitions that do not contain data equal to the specified value for a predicate expression. For example, if a query contains a predicate expression such as `customer_id = 12345`, Snowflake can prune all of the micro-partitions that do not contain data for the specified customer ID.

8. 

Allow very granular **partition pruning**

Very small

Eliminate unnecessary partitions

Contain **50-500 MB** of uncompressed data

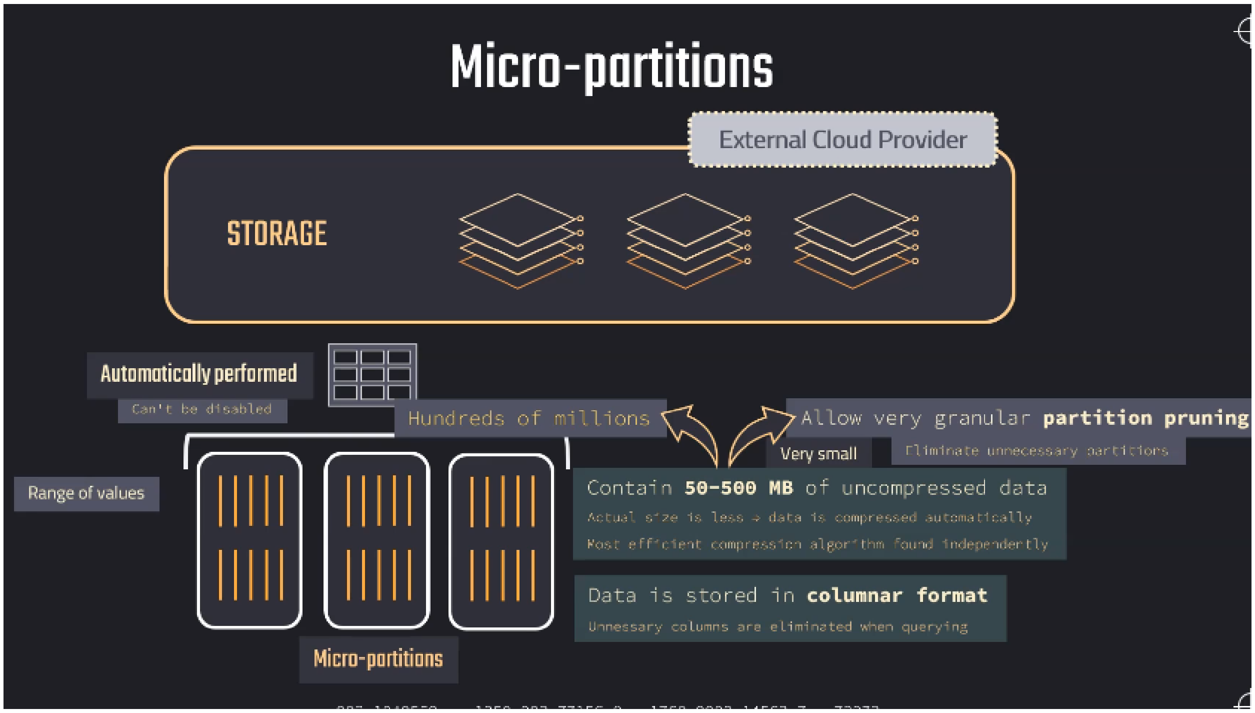
Actual size is less → data is compressed automatically

Most efficient compression algorithm found independently

Micro-partitions

9. It may contain hundred of millions of partitions And also the data is stored not in rows but in a so-called columnar format.

10. And this micro-partition is automatically done by snowflake..we cannot disable it



Micro-partitions

External Cloud Provider

STORAGE

Automatically performed

Can't be disabled

Hundreds of millions

Very small

Allow very granular **partition pruning**

Eliminate unnecessary partitions

Range of values

Contain **50-500 MB** of uncompressed data

Actual size is less → data is compressed automatically

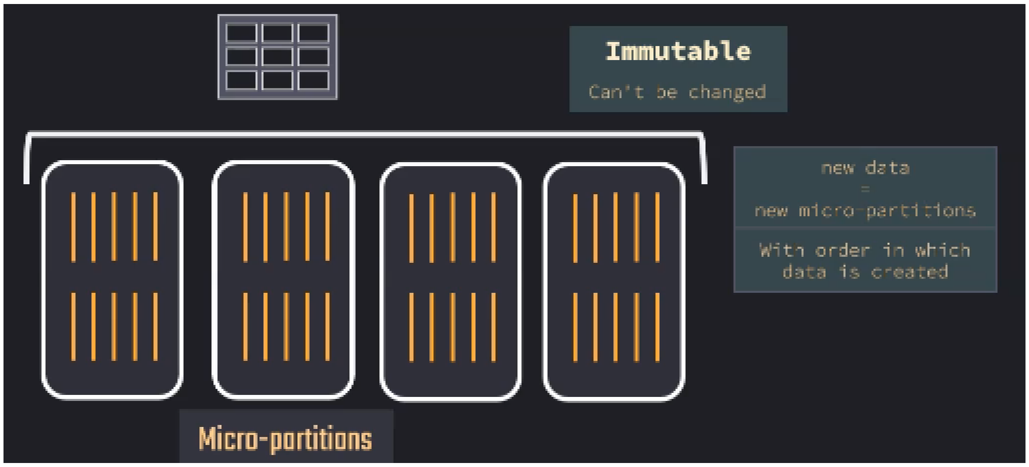
Most efficient compression algorithm found independently

Data is stored in **columnar format**

Unnessary columns are eliminated when querying

Micro-partitions

11. This partitions are immutable and..if theres any new data…this will create new micro partitions and this will happen just with the natural order with which the data is inserted.



Immutable

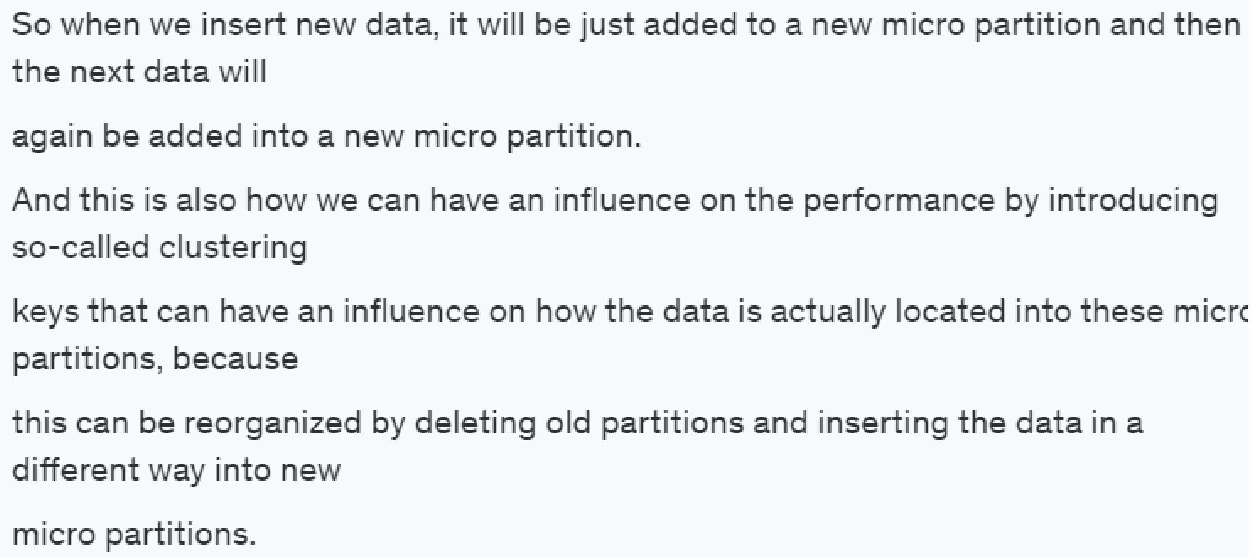
Can't be changed

new data
=
new micro-partitions

With order in which data is created

Micro-partitions

12. We can actually locate the data ..using clustering keys



So when we insert new data, it will be just added to a new micro partition and then the next data will

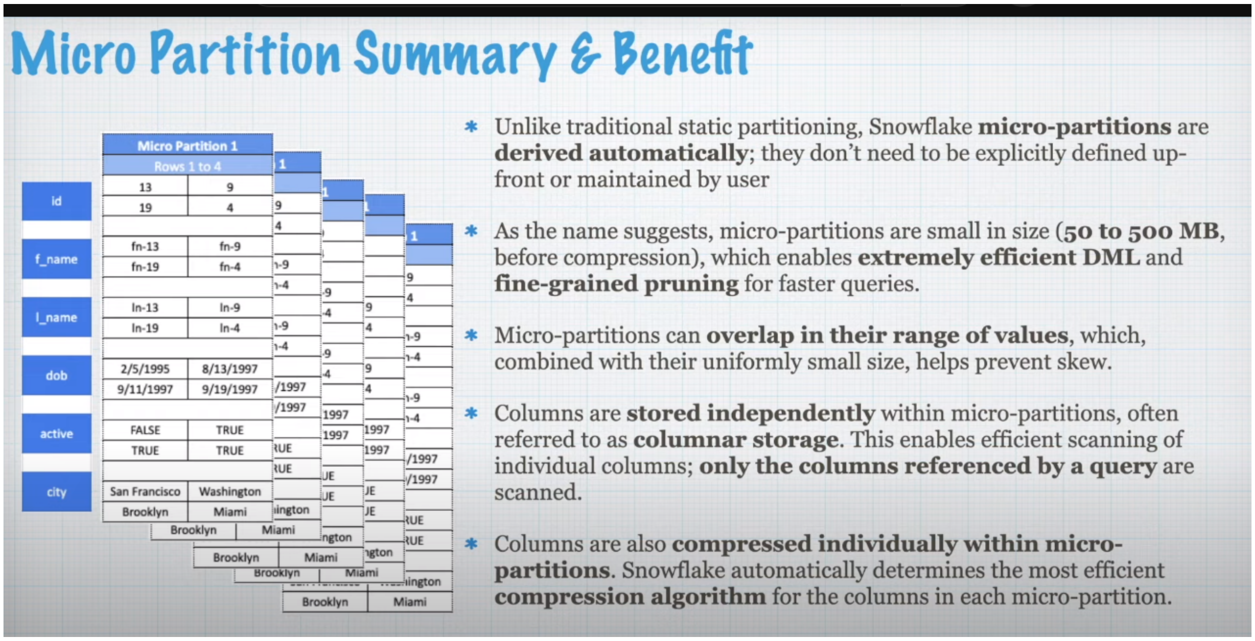
again be added into a new micro partition.

And this is also how we can have an influence on the performance by introducing so-called clustering

keys that can have an influence on how the data is actually located into these micro partitions, because

this can be reorganized by deleting old partitions and inserting the data in a different way into new

micro partitions.



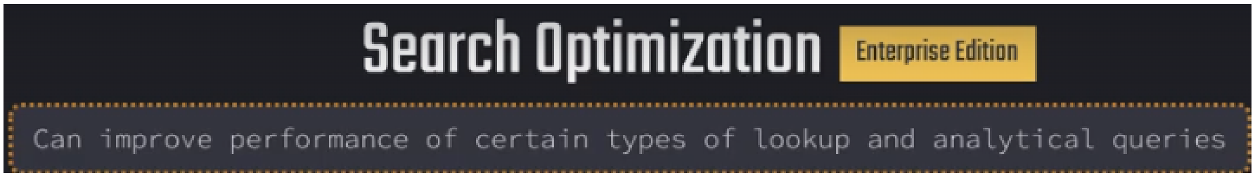
Micro Partition Summary & Benefit

- Unlike traditional static partitioning, Snowflake **micro-partitions** are **derived automatically**; they don't need to be explicitly defined up-front or maintained by user
- As the name suggests, micro-partitions are small in size (**50 to 500 MB**, before compression), which enables **extremely efficient DML** and **fine-grained pruning** for faster queries.
- Micro-partitions can **overlap in their range of values**, which, combined with their uniformly small size, helps prevent skew.
- Columns are **stored independently** within micro-partitions, often referred to as **columnar storage**. This enables efficient scanning of individual columns; **only the columns referenced by a query** are scanned.
- Columns are also **compressed individually within micro-partitions**. Snowflake automatically determines the most efficient **compression algorithm** for the columns in each micro-partition.

Imp for exam

For micropartitioning and clustering :
https://www.youtube.com/watch?v=UNBysn1M9Vg

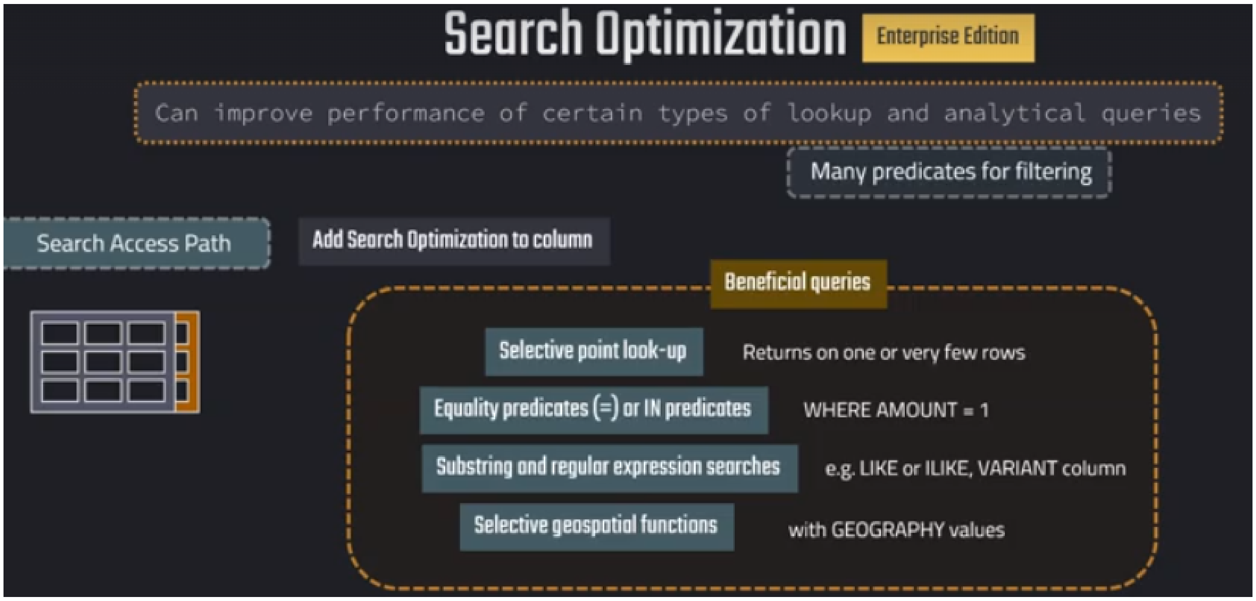
Search Optimization Service

1. 

   Search Optimization

   Enterprise Edition

   Can improve performance of certain types of lookup and analytical queries
2. It can improve performance for filtering techniques using where etc
3. It is very beneficial for queries which returns one or very few rows(Imagine a data analyst lookin for some specific data in a large table)
4. Also for queries which uses “=” and IN operator

   Search Optimization

   Enterprise Edition

   Can improve performance of certain types of lookup and analytical queries

   Many predicates for filtering

   Search Access Path

   Add Search Optimization to column

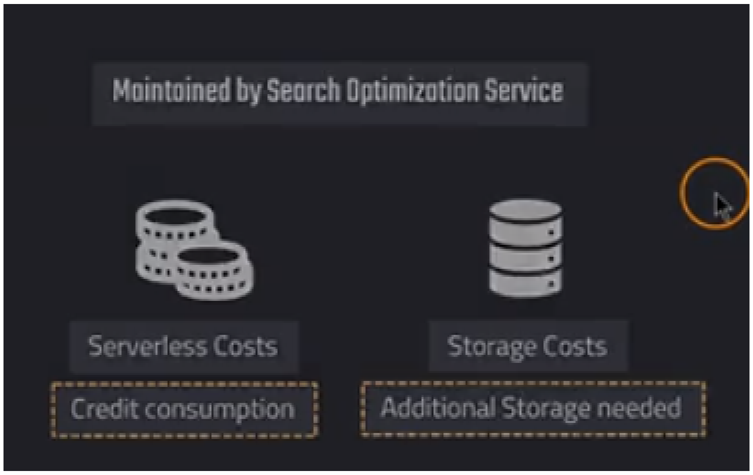
   Beneficial queries

   Selective point look-up — Returns on one or very few rows

   Equality predicates (=) or IN predicates — WHERE AMOUNT = 1

   Substring and regular expression searches — e.g. LIKE or ILIKE, VARIANT column

   Selective geospatial functions — with GEOGRAPHY values
5. It is maintained by snowflake..once we enable this..and this also consumes credits Storage is needed for maintaining search access

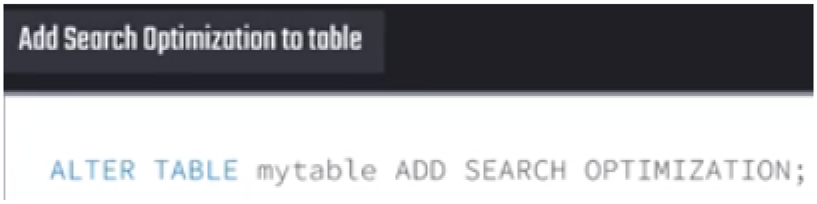
   Maintained by Search Optimization Service

   Serverless Costs

   Credit consumption

   Storage Costs

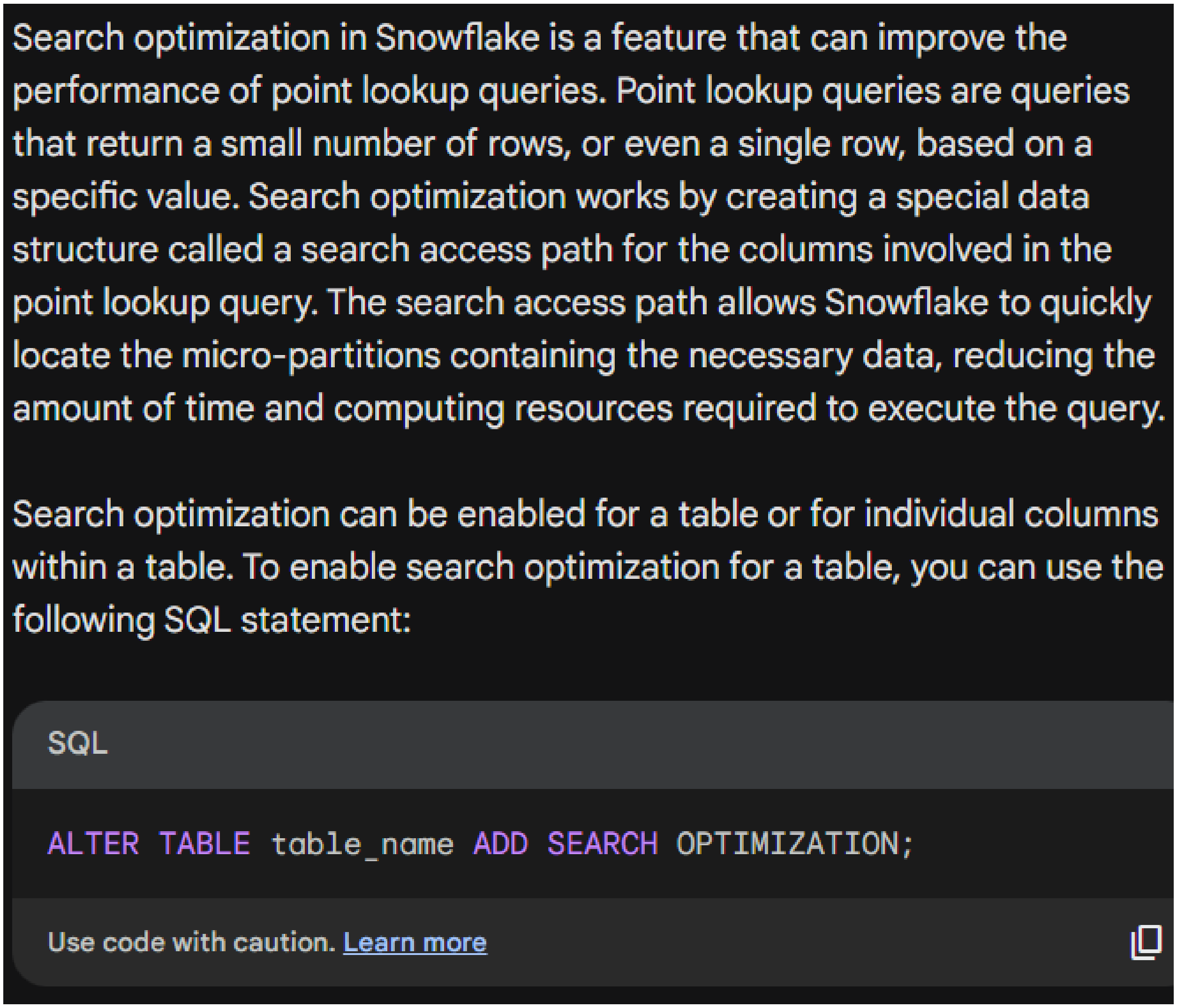
   Additional Storage needed

6. 

Add Search Optimization to table

```
ALTER TABLE mytable ADD SEARCH OPTIMIZATION;
```

7. We need to have ownership privilege or ADD Search optimization privilege …to add search optimization to a table

8. 

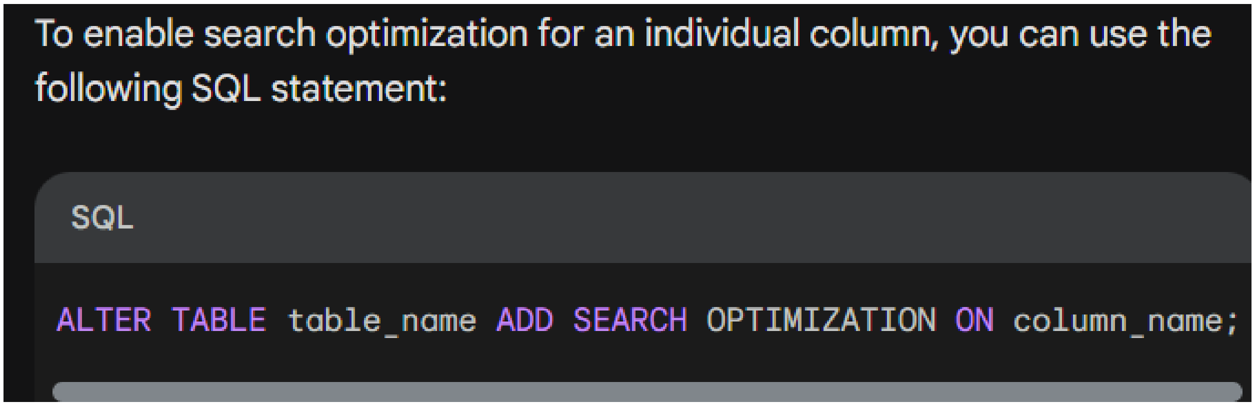
Search optimization in Snowflake is a feature that can improve the performance of point lookup queries. Point lookup queries are queries that return a small number of rows, or even a single row, based on a specific value. Search optimization works by creating a special data structure called a search access path for the columns involved in the point lookup query. The search access path allows Snowflake to quickly locate the micro-partitions containing the necessary data, reducing the amount of time and computing resources required to execute the query.

Search optimization can be enabled for a table or for individual columns within a table. To enable search optimization for a table, you can use the following SQL statement:

SQL

```
ALTER TABLE table_name ADD SEARCH OPTIMIZATION;
```

Use code with caution. Learn more

9. 

To enable search optimization for an individual column, you can use the following SQL statement:

```SQL
ALTER TABLE table_name ADD SEARCH OPTIMIZATION ON column_name;
```

Materialized Views

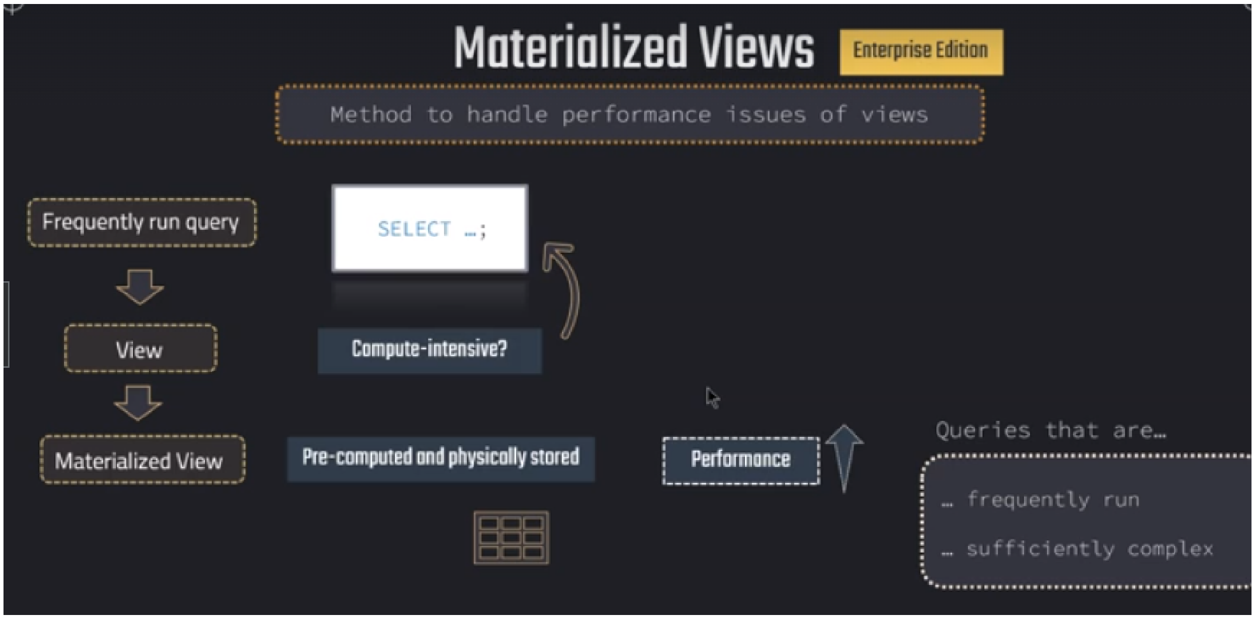
1.

   Materialized Views

   Enterprise Edition

   Method to handle performance issues of views

2. Lets assume we have query which is frequently runned to see the data..so we use views
3. But sometimes it takes lot of compute resources…and this causes compute costs and performance
4. The sol to the above prob is Materialized views…which basically precomputes and stores the data physically



Materialized Views

Enterprise Edition

Method to handle performance issues of views

Frequently run query

View

Materialized View

SELECT …;

Compute-intensive?

Pre-computed and physically stored

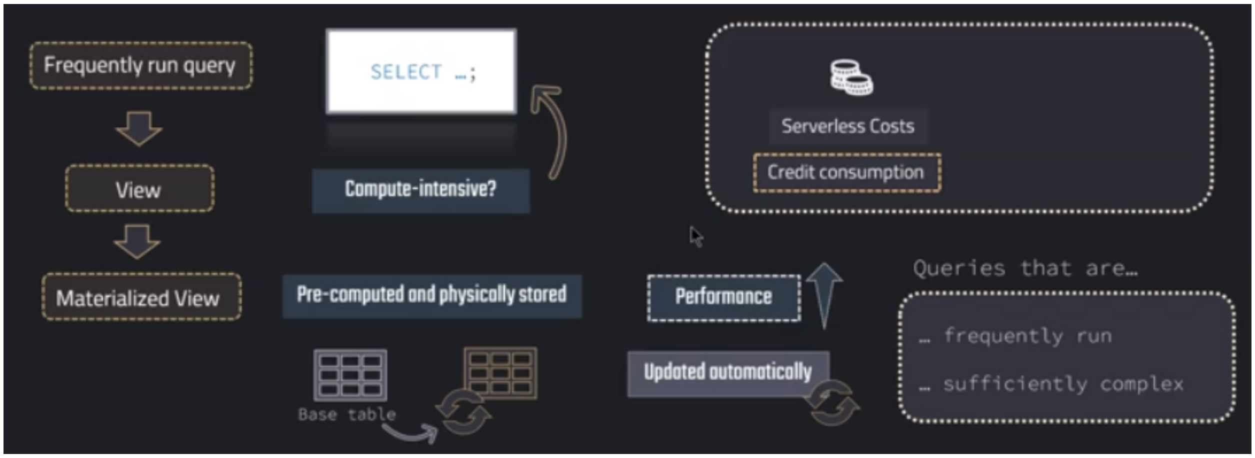
Performance

Queries that are…

… frequently run

… sufficiently complex

5. And if we update our table in the base location…then the data which is in materialized view..will also gets updated automatically…it is done by snowflake server…and we will charged for computing



Frequently run query

View

Materialized View

SELECT …;

Compute-intensive?

Pre-computed and physically stored

Base table

Serverless Costs

Credit consumption

Performance

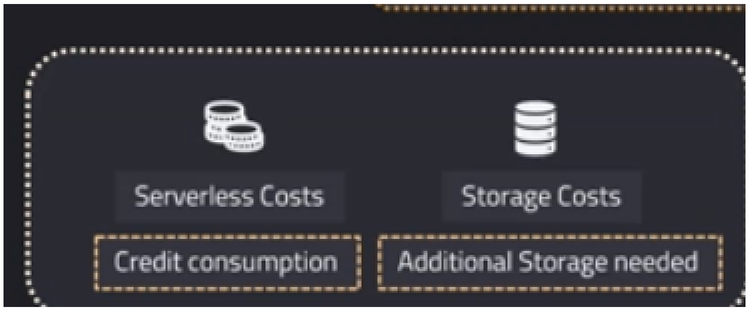
Updated automatically

Queries that are…

… frequently run

… sufficiently complex

6. And also we get charged for storage costs..as we are physically storing data now



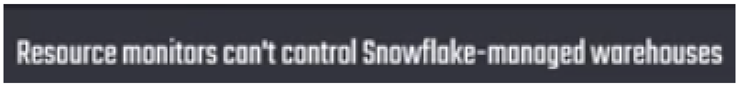
Serverless Costs

Credit consumption

Storage Costs

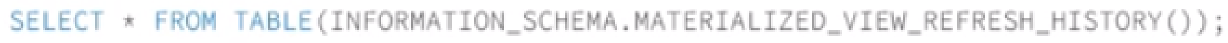
Additional Storage needed

7. As this materialized views is creating additional costs…we have to be careful and have to start slow..by using one materialized view

8. 

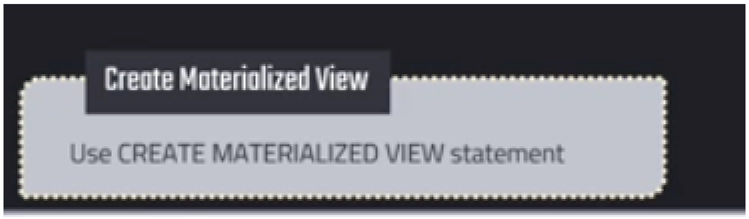
Resource monitors can't control Snowflake-managed warehouses

9. To get addition information we use



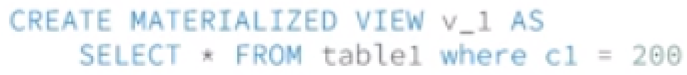
```
SELECT * FROM TABLE(INFORMATION_SCHEMA.MATERIALIZED_VIEW_REFRESH_HISTORY());
```

10. Syntax



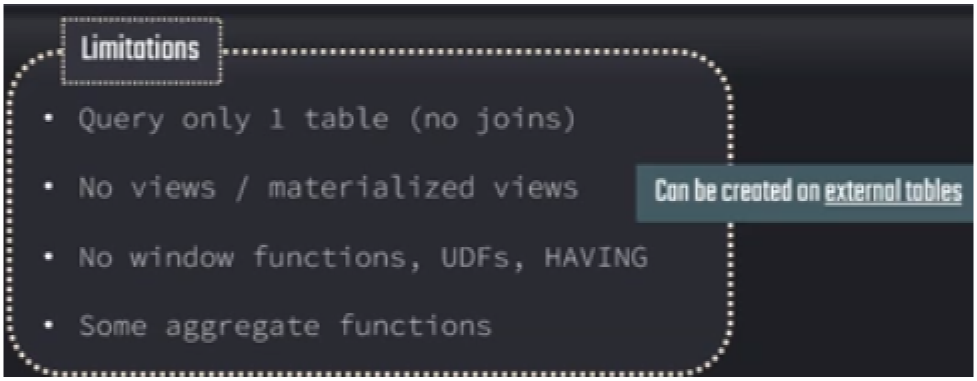
Create Materialized View

Use CREATE MATERIALIZED VIEW statement

11. 

```
CREATE MATERIALIZED VIEW v_1 AS
    SELECT * FROM table1 where c1 = 200
```

12. Limitations

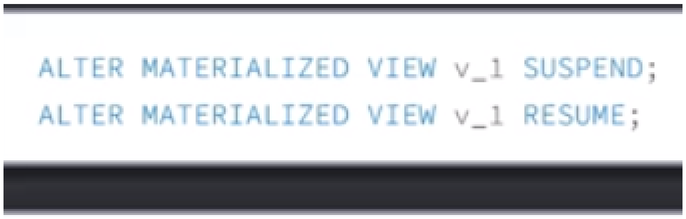
13. We cannot use joins while creating MV(mater view)..
14. And cannot create a MV on a view or MV…but we can create MV on external table
15. 

Limitations

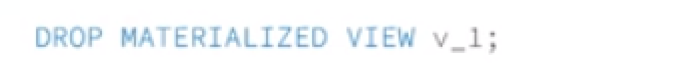
- Query only 1 table (no joins)
- No views / materialized views
- No window functions, UDFs, HAVING
- Some aggregate functions

Can be created on external tables

16. To decrease some costs we can suspend the MV..or if we dont want this MV..we can drop this



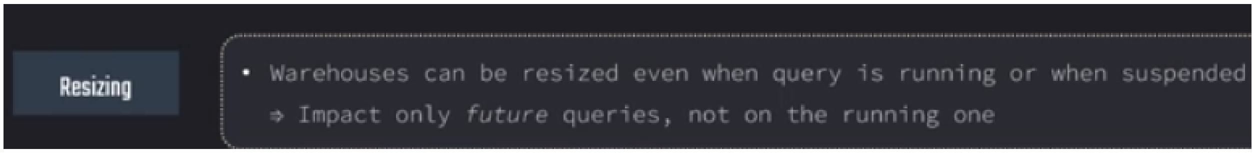
```
ALTER MATERIALIZED VIEW v_1 SUSPEND;
ALTER MATERIALIZED VIEW v_1 RESUME;
```



```
DROP MATERIALIZED VIEW v_1;
```

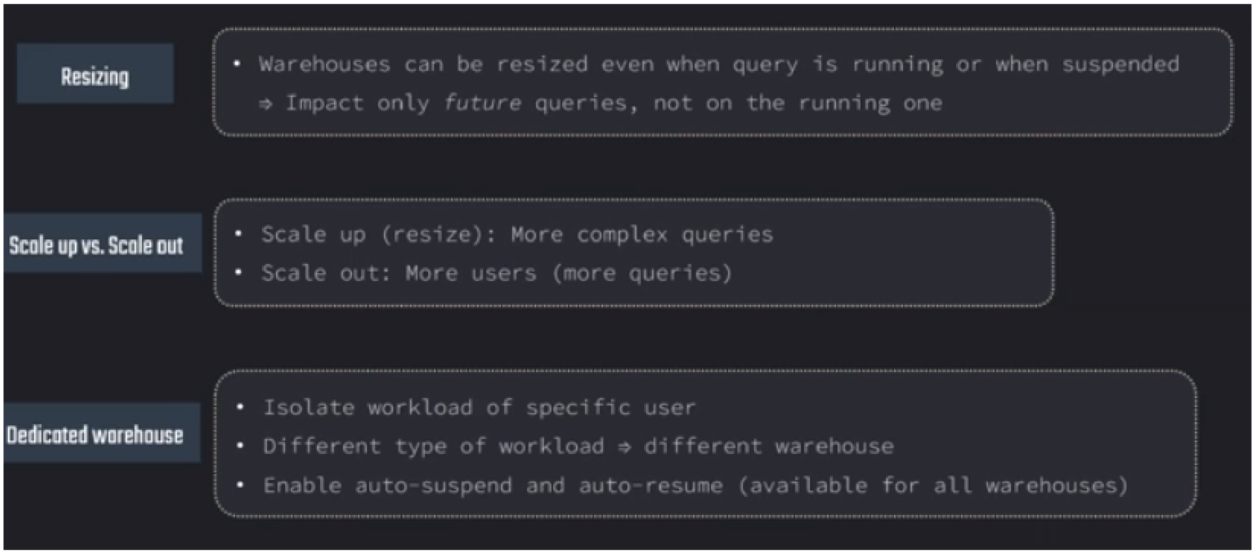
Warehouse Considerations

1. Resizing
2. If we resize(increase compute etc) our warehouse when there are multiple queries running..then the queries which are running will get the old size…only the new queries will get updated warehouse power



Resizing

- Warehouses can be resized even when query is running or when suspended
  ⇒ Impact only *future* queries, not on the running one

3. 

Resizing

- Warehouses can be resized even when query is running or when suspended
  ⇒ Impact only *future* queries, not on the running one

Scale up vs. Scale out

- Scale up (resize): More complex queries
- Scale out: More users (more queries)

Dedicated warehouse

- Isolate workload of specific user
- Different type of workload ⇒ different warehouse
- Enable auto-suspend and auto-resume (available for all warehouses)