INTRODUCTION TO QUERY OPTIMIZATION

SNOWFLAKE QUERY OPTIMIZATION

Imagine you're a chef in a big kitchen preparing a delicious meal. You have a recipe that involves using different ingredients and cooking techniques. Now, let's say you have a list of tasks to complete to make the meal: chopping vegetables, marinating meat, boiling pasta, and so on.

In the world of databases, like Snowflake, when you run a query (which is like asking the database a question), it's like giving the kitchen chef a set of cooking instructions. However, just like the chef can find more efficient ways to complete cooking tasks to save time and effort, databases can also find smarter ways to execute queries.

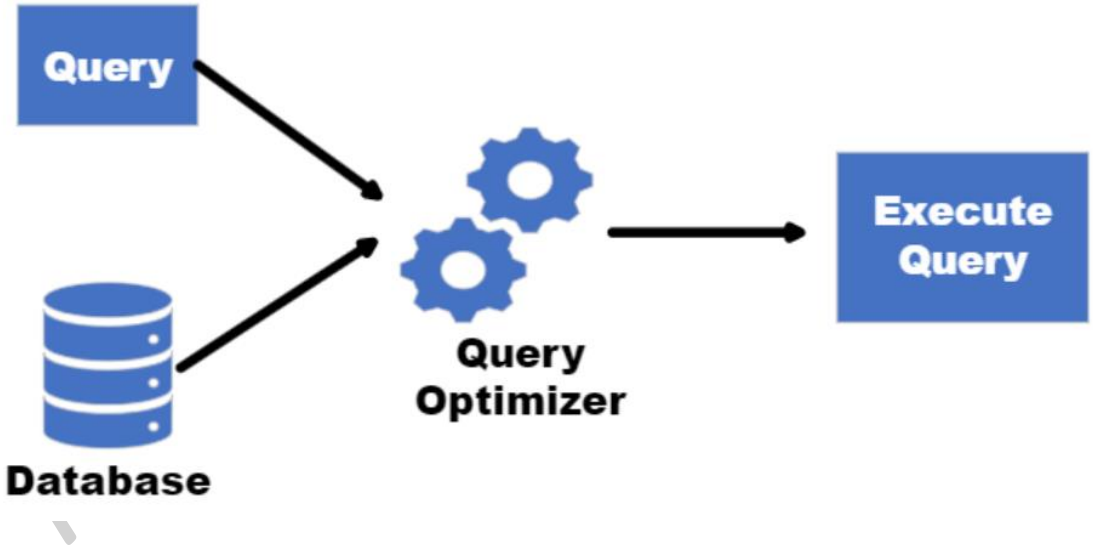
Query optimization in Snowflake is like the chef trying to cook the meal in the fastest and most efficient way. The database looks at your query and figures out the best path to retrieve and combine the data you want. It considers things like which parts of the data to fetch first, how to sort and filter them, and whether it can use special techniques to make things quicker.

For instance, if your query involves looking at data from different tables, the database might choose to combine the tables in a particular way to avoid unnecessary work. It's like the chef deciding to chop all the vegetables at once instead of doing it separately for each dish.

The goal of query optimization is to minimize the time it takes for your query to give you the results you want. Just like the chef aiming to serve a delicious meal quickly, Snowflake's query optimization helps you get your data faster and more efficiently, making your work smoother and more enjoyable

Query optimization is a process of defining the most efficient and optimal way and techniques that can be used to improve query performance based on the rational use of system resources and performance metrics.

The purpose of query tuning is to find a way to decrease the response time of the query, prevent the excessive consumption of resources, and identify poor query performance.



WHY QUERY OPTIMIZATION ???

Query Optimization is crucial in Snowflake (and any database system) for several important reasons:

Performance Improvement: Query optimization in Snowflake is all about making your queries run faster. When you're working with big datasets, complicated combinations of data, and lots of calculations, poorly written queries can be really slow. But when you optimize your queries, they speed up. This means you'll get the answers you're looking for super quickly, which makes your data analysis work much smoother and faster

Cost Efficiency: Snowflake, like many cloud-based data warehouses, charges based on the amount of data processed. Poorly optimized queries can consume more resources and therefore lead to higher costs. By optimizing your queries, you can reduce the amount of data that needs to be processed, ultimately saving on your operational expenses.

Reduce query run time: Some queries may have a certain SLA that they need to meet, not hitting those targets may result in delays in the delivery of data that could negatively impact the business. Query optimization can help improve the performance of queries by finding the optimal way to access and manipulate the data.

User Experience: If you're sharing data insights or reports with others, the speed at which they receive the results matters. Optimized queries mean quicker response times for your end-users, leading to a better overall experience and higher user satisfaction.

Scalability: As your data grows, the performance of unoptimized queries can degrade significantly. Query optimization ensures that your queries continue to run efficiently as your dataset expands, maintaining consistent performance levels.

Complex Analytics: Data analysts often deal with complex queries involving joins, aggregations, and filtering. Optimizing these intricate queries becomes essential to ensure that the results are accurate and generated in a timely manner.

Resource Utilization: Snowflake, being a cloud-based data warehouse, allocates resources dynamically to handle queries. Well-optimized queries utilize resources effectively, preventing unnecessary overloads and ensuring that the system remains responsive for all users.

Decision-Making: Timely access to accurate data is crucial for making informed decisions. Slow queries can delay decision-making processes, potentially impacting business operations. Optimized queries provide quick access to the required data for confident decision-making.

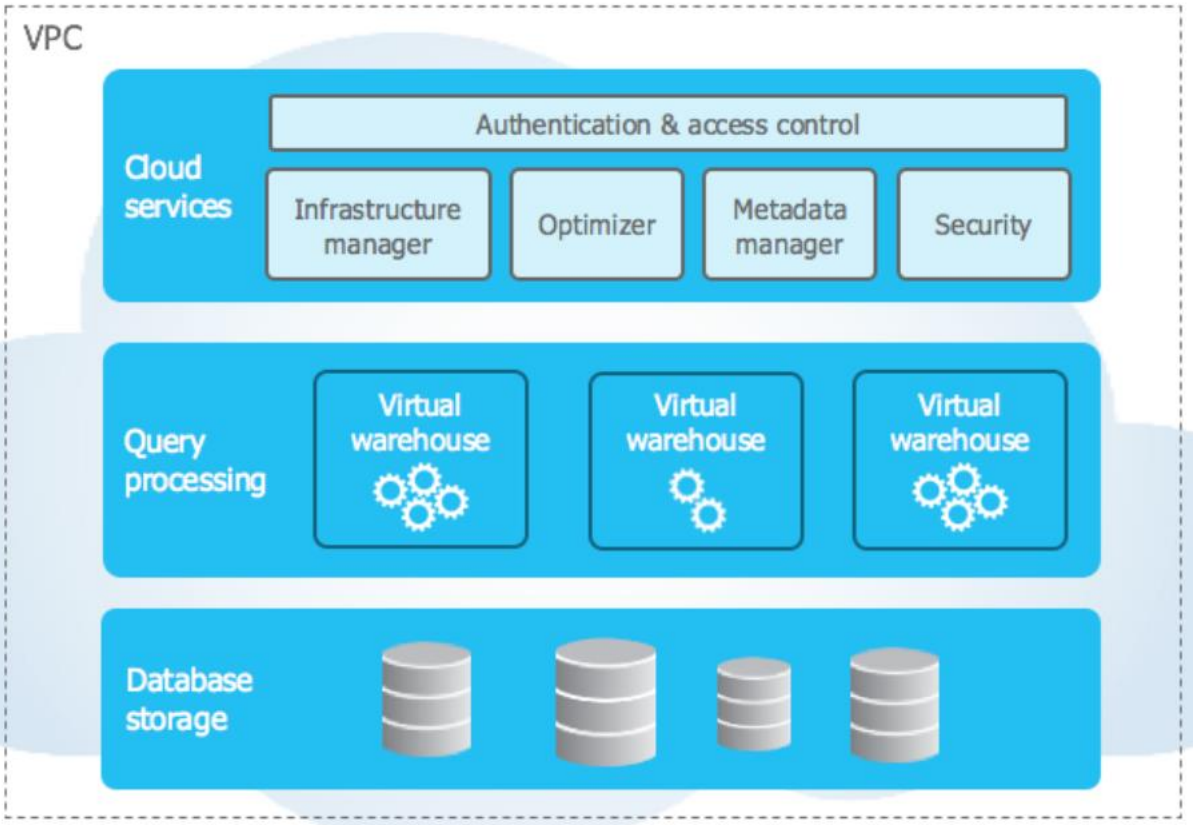
ARCHITECTURE OF SNOWFLAKE

Now before going deep into the technique to optimize query first understand the architecture of snowflake.

Snowflake's architecture is a hybrid of traditional shared-disk and shared-nothing database architectures. Similar to shared-disk architectures, Snowflake uses a central data repository for persisted data that is accessible from all compute nodes in the platform. But similar to shared-nothing architectures, Snowflake processes queries using MPP (massively parallel processing) compute clusters where each node in the cluster stores a portion of the entire data set locally. This approach offers the data management simplicity of a shared-disk architecture, but with the performance and scale-out benefits of a shared-nothing architecture.

In simple words:

- Snowflake is like a mix of a big shared storage room (shared-disk) and lots of individual workers with their own tools (shared-nothing).
- It keeps all the data in one main place that everyone can use easily, just like a library.
- But when you need to do things with the data, it spreads the work among many helpers, each with their own piece of data. This makes things quicker and more efficient, like each helper has their own tools handy.
- This combination makes Snowflake great at handling data in a simple way while also being really fast and good at handling lots of work at the same time.



It Consists of three layers:

- **Database Storage**
- **Query Processing**
- **Cloud Services**

Database Storage Layer:

This is where the raw data is stored.

Snowflake stores data in a structured and compressed format, using a combination of columnar storage and a unique indexing scheme that improves query performance.

The data is divided into micro-partitions, which are small, immutable units of data that allow for efficient storage and querying.

Query Processing / Compute Layer:

The compute layer is responsible for processing and querying the data stored in the data storage layer.

Snowflake employs a virtual warehouse concept, which are clusters of compute resources that can be scaled up or down as needed.

This decoupling of storage and compute allows for better resource utilization and the ability to independently scale processing power and storage capacity.

Cloud Services Layer:

This layer includes various services that manage the metadata, access control, and query optimization.

The services layer handles tasks such as query parsing, optimization, and execution.

It also manages user authentication, data access permissions, and other administrative functions.

Snowflake's services layer plays a critical role in ensuring the security, performance, and manageability of the system.

QUERY OPTIMIZATION TECHNIQUES IN SNOWFLAKE

1. SNOWFLAKE SERACH OPTIMIZATION SERVICE (S0S)

Search Optimization Service aims to enhance the performance of queries under specific conditions, such as when the table is frequently queried on columns other than the primary cluster key, and when analytical queries involve extensive filtering predicates.

Frequently Queried Coloumn : The columns being frequently queried are not the primary cluster key columns. In a database, the primary cluster key (often referred to as the primary key) is used for indexing and organizing data. When queries predominantly use other columns for filtering and searching, it can impact the efficiency of Query Execution.

Analytical Queries with Extensive Predicates: Analytical queries involve aggregations, complex calculations, and often require filtering based on various conditions. Extensive predicates refer to a large number of filtering conditions applied to these analytical queries.

Both of this are taken care by SOS.

REAL TIME USE CASE OF SEARCH OPTIMIZATION SERVICE

- Snowflake supports semi-structured data types like VARIANT and OBJECT, which can store JSON, XML, or other unstructured data. You can optimize queries that involve searching and filtering within these semi-structured fields. This could be useful for analyzing log data, event records, or any data containing free-form text.

- Business users need fast response times for critical dashboards with highly selective filters.

- Data scientists who are exploring large data volumes and looking for specific subsets of data.

- Substring and regular expression searches.

- When Snowflake instance is connected to real-time data sources, optimizing queries to search and retrieve recent data updates quickly is important for up-to-date analytics.

NOTE:

search optimization is most effective for queries that involve searching for text within a large number of rows. If your queries only involve a small number of rows, or if the columns being searched are not text-based, search optimization may not provide significant performance benefits.

HOW TO ENABLE SEARCH OPTIMIZATION??

To enable search optimization for your account you must enable this feature for specific columns or fields in columns or for the entire table

For example:

 # Adding search optimization for entire table.

alter table <table_name>

add search optimization;

Here, I have a table "demodatabase.db_source.script_src_date_day_lookup", will apply search optimization to it.

alter table demodatabase.db_source.script_src_date_day_lookup

add search optimization;

2. Minimize Data Movement

Minimizing data movement is a fundamental principle in optimizing query performance in Snowflake and other data warehousing systems. It involves reducing the amount of data that needs to be transferred and processed during query execution. This is crucial for improving query speed, resource utilization, and overall efficiency.

HOW TO ACHIEVE THIS ??

- **Using partition pruning:**

Partition pruning is a technique used in Snowflake to improve query performance by reducing the amount of data that needs to be scanned when querying large tables that are partitioned.

It involves dividing a table into smaller, more manageable parts called partitions**,** based on a specific column or set of columns.

When a query includes a WHERE clause that filters on a partition key column, Snowflake can use partition pruning to eliminate partitions that do not contain any relevant data, reducing the amount of data that needs to be scanned during query execution. This results in faster query processing and better resource utilization

suppose you have a large table of sales data that is partitioned by date, with each partition containing sales data for a specific date range. If you run a query that filters on a specific date range, Snowflake can use partition pruning to eliminate all the partitions that do not contain any sales data for that date range, reducing the amount of data that needs to be scanned.

HOW TO IMPLEMENT PARTITION PRUNING??

For example, if you have a partitioned table on the “date” column, you can use the following query to only scan the partitions for the month of January:

```
SELECT *

FROM  <TABLE_NAME>

WHERE year >= ' 2020-01-01 ' AND date < ' 2020-01-31 ';
```

- **Using appropriate sorting:**

Sorting queries are a common type of database query where the results are arranged in a specific order based on one or more columns.

When query requires sorting, you can use the ORDER BY clause to specify the sort order. This can help Snowflake minimize data movement by sorting the data on each node before sending it to the query execution node.

HOW TO IMPLEMENT SORTING ??

For example, if you need to sort a table by the “name” column, you can use the following query:

```
SELECT *

FROM <TABLE_NAME>

ORDER BY name;
```

3. USE APPROPRIATE DATA TYPES

The right data type for your database columns is a critical decision that affects storage efficiency, query performance, and data integrity. Each database system provides a range of data types designed to accommodate different types of data and optimize storage and processing.

4. USE MATERIALIZED VIEW

A materialized view in a database is a pre-computed and pre-stored result set that represents the outcome of a specific query, often involving aggregations, calculations, or joins. Materialized views are used to enhance query performance by reducing the need to perform complex calculations or aggregations every time a query is executed.

This can be achieved by using appropriate aggregation levels:

For example, if you have a sales table with millions of rows, you can create a materialized view that aggregates the data at the monthly level.

```
CREATE MATERIALIZED VIEW monthly AS
SELECT DATE_TRUNC('MONTH', date) AS month, SUM(sales_amount) AS total_sales
FROM sales_table
GROUP BY 1;
```

5. USE CLUSTERING KEYS

Clustering keys in a database determine the physical organization of data on disk. When a table is defined with a clustering key, the data is organized and stored on disk based on the values in the clustering key column(s). This physical organization affects how the data is accessed and retrieved during query execution.

- **Choosing the appropriate clustering key:** The clustering key should be chosen based on the most frequently used filter conditions in your queries. For example, if you frequently query a sales table by date range, you can use the date column as a clustering key.
- **Using multi-column clustering keys:** If your queries frequently filter on multiple columns, you can use a multi-column clustering key to group the data together more efficiently.

HOW TO IMPLEMENT THIS??

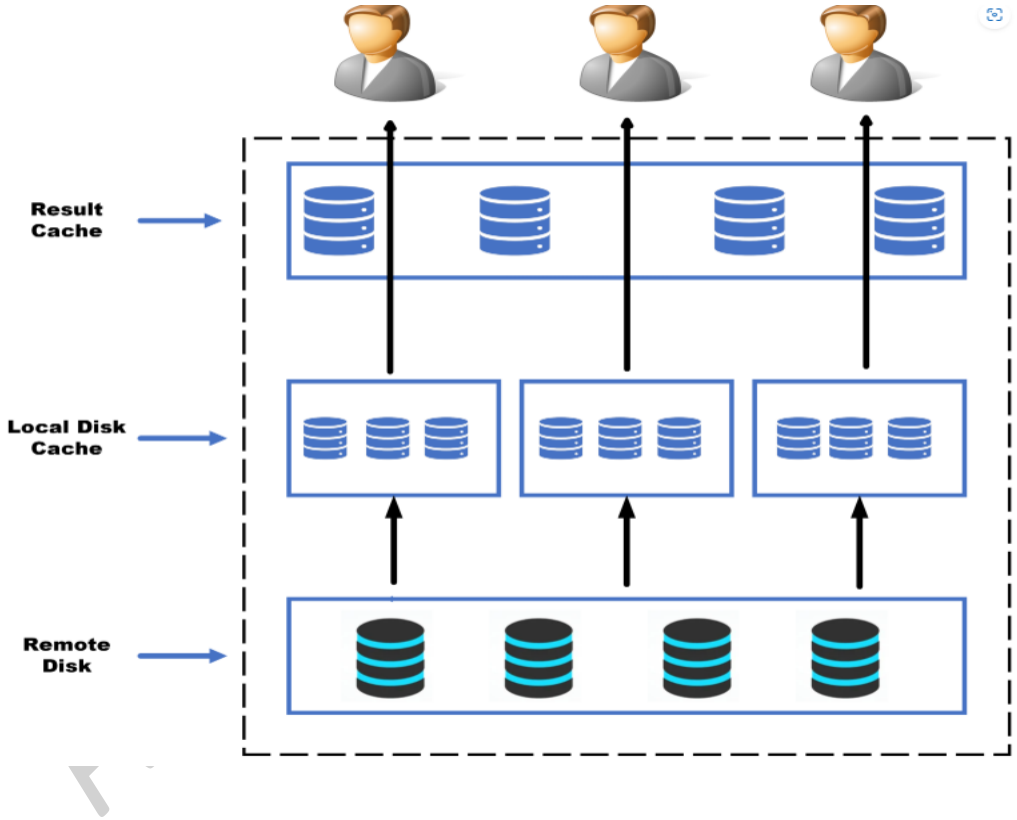
For example, if you frequently query a sales table by date range and product category, you can use the following clustering key:

#CREATING A TABLE

```
CREATE TABLE sales_table (
   date DATE,
   product_category VARCHAR,
   sales_amount FLOAT
)
#CLUSTERING TWO TABLE.
CLUSTER BY (date, product_category);
```

6. MAXIMIZE CACHE USAGE

Maximizing cache usage is a crucial strategy for optimizing query performance in Snowflake. The Snowflake cache is a feature that involves storing frequently accessed data in memory, allowing for faster retrieval compared to accessing data from disk. By taking advantage of caching, you can significantly improve query execution times and enhance overall system performance.



HOW TO IMPLEMENT CACHING??

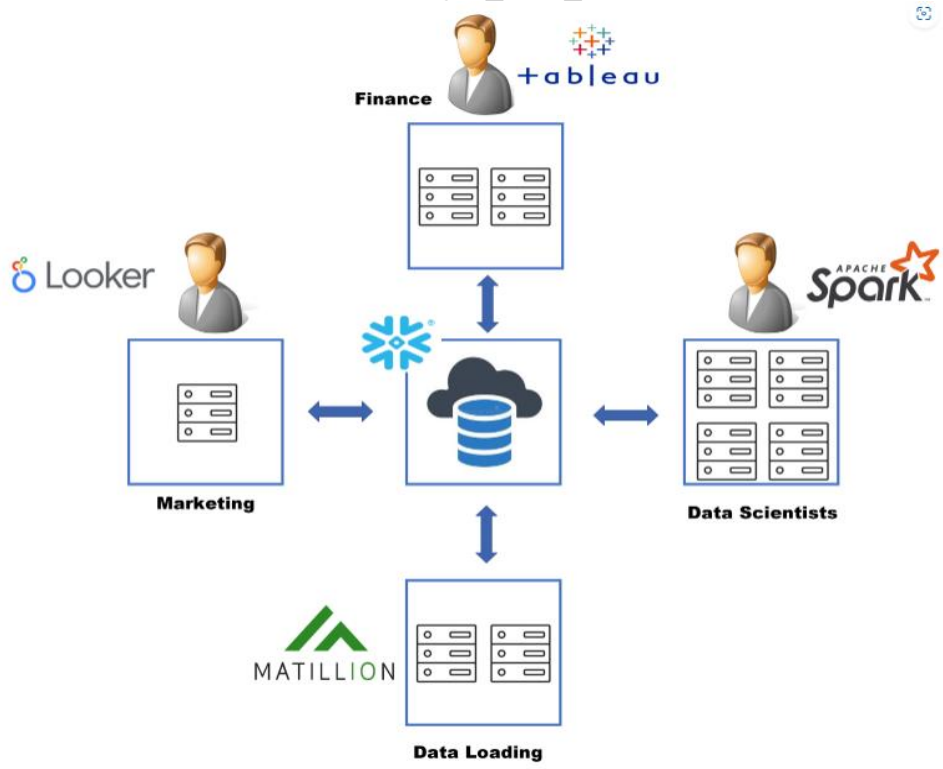
ALTER SESSION SET USE_CACHED_RESULT = TRUE

7. OPTIMIZE BULK DATA LOADING INTO TABLES USING COPY COMMANDS

Instead of doing Bulk Data Load try using COPY command.

By using the COPY command, you can load large datasets quickly and efficiently into Snowflake tables, enabling faster data processing and analysis.

8. USE SEPARATE WAREHOUSE FOR QUERIES, TASK LOADING AND LARGE FILES.



From this fig you can see that Finance, Data Scientist, Data Loading, and Marketing have separate warehouses which will help them with fast and responsive queries which will ultimately improve performance, scalability, and cost efficiency.

9. USE LIMIT ROWS OR TOP CLAUSE

Using a TOP or LIMIT clause can indeed improve the performance of SELECT queries, especially when dealing with large datasets. By limiting the number of rows returned by the query, you can reduce the amount of data that needs to be processed and transmitted, leading to faster query execution times and improved response times. This is particularly useful when you're interested in only a subset of the data for analysis or preview purposes.

The LIMIT or TOP clause also improves queries with an ORDER BY clause. For example, the first query below returns in just over two minutes on an X-SMALL warehouse, whereas the second query, which returns the top ten entries, is twelve times faster, taking just six seconds to complete.

This command takes 2m 5s for its execution.

```
select *
from SNOWFLAKE_SAMPLE_DATA.TPCH_SF100.ORDERS
order by o_totalprice desc;
```

Whereas when using top its execution time decreases to 6s.
Same thing goes when using limit too.

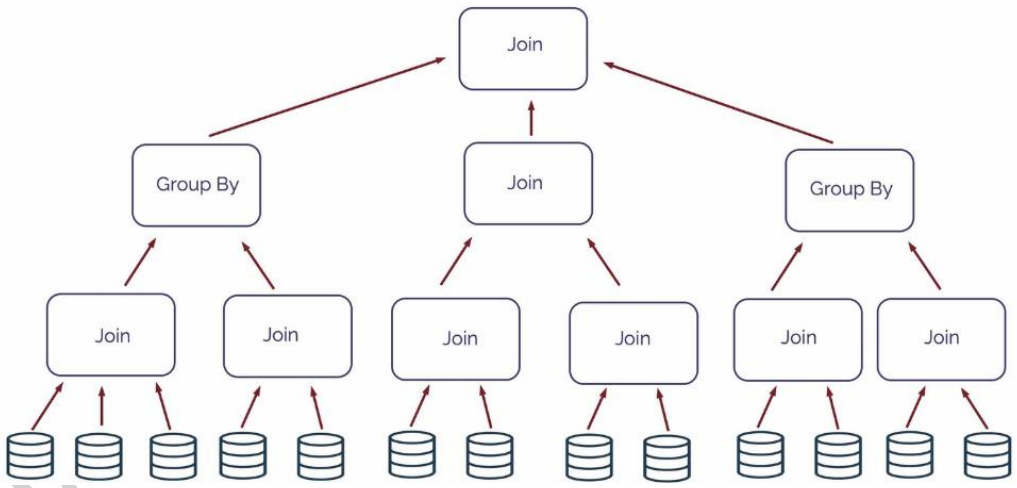
```
select top 10 *
from SNOWFLAKE_SAMPLE_DATA.TPCH_SF100.ORDERS
order by o_totalprice desc;
```

10.BREAKING-DOWN COMPLEX JOIN OPERATION

JOIN operations enable you to combine data from multiple related tables into a single result set. Instead of retrieving data from each table separately and then manually combining them, JOINs allow you to get the desired data in a single query.
This significantly reduces the amount of data transferred over the network and minimizes the need for additional processing on the client side.
Making querying more reliable.

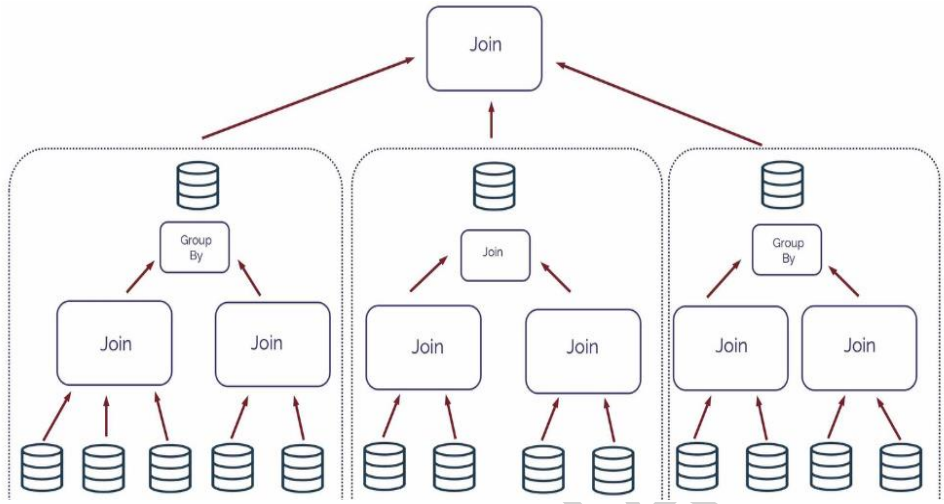
But what if we have complex join operation??

Refer the below figure:



Here we can see that the process is too complex. And it will directly impact on query optimization.

So instead of proceeding with this approach, we should break-down the process and write the data in one table.



Breaking down to complex query into simple one.

With the help of this approach, instead of executing the statement as a single query, break the problem down into multiple queries than write results to transient tables.

It will eliminate the high execution time.

11.USE QUERY PROFILING

Query profiling, also known as query performance profiling, is a process of analyzing and understanding the execution of a database query in order to identify performance bottlenecks, optimize query execution times, and improve overall database performance.

It involves capturing detailed information about how a query is processed by the database management system, including the resources consumed and the steps taken during execution.

Query profiling helps database administrators, developers, and analysts identify areas where queries can be optimized for better performance.

HOW TO DO QUERY PROFILING???

Once you run a query click the query profile view and you should identify the following red flags:

1. Analyze the query profile to identify the slowest stages of the query plan. Look for stages with high "elapsed time" or "execution time" values. These stages are likely to be the bottlenecks in your query.

2. Once you have identified the slowest stages, you can try to optimize them. Here are some tips:
    - If a stage involves a large number of rows, consider adding a filter to reduce the number of rows processed.
    - If a stage involves a large amount of data movement, consider using a more efficient join strategy or partitioning the data.
    - If a stage involves a complex computation, consider simplifying the computation or using a more efficient algorithm.

- Look for long-running or high-cost operations in the query plan. These are likely areas where the query is spending a lot of time or scanning a large amount of data.
- Examine the number of rows processed by each step in the query plan. This can help you identify areas where the query is performing unnecessary processing.
- Look for opportunities to minimize data movement by using appropriate sorting and partitioning.
- Review the query code to identify areas for optimization, such as using appropriate data types, and minimizing the use of subqueries and joins.

3. After you have made changes to your query, rerun it to see if the changes have improved the query performance.

12. AVOID USING SELECT *

Avoid using select * when you just require data from few coloumn.
Select * will unusually execute over the entire table.
So when we just require the data from few column, better I nstead of doing select * , specify coloumn name.

Example : Need sales_profit year wise from sales table.

While using below code,whole sales table will be considered.

```
Select * from sales;
```

While using this code only the specified coloumn will be taken care. It is more optimized.

```
Select sales_year, sales_profit
From sales;
```

CONCLUSION:

In conclusion, search optimization techniques in Snowflake empower you to design and structure your data, queries, and indexing strategies to efficiently handle search-like operations.

By choosing the right strategies and aligning them with your specific use cases, you can achieve substantial improvements in query performance, leading to enhanced data analysis and insights.

Keep Learning 😊