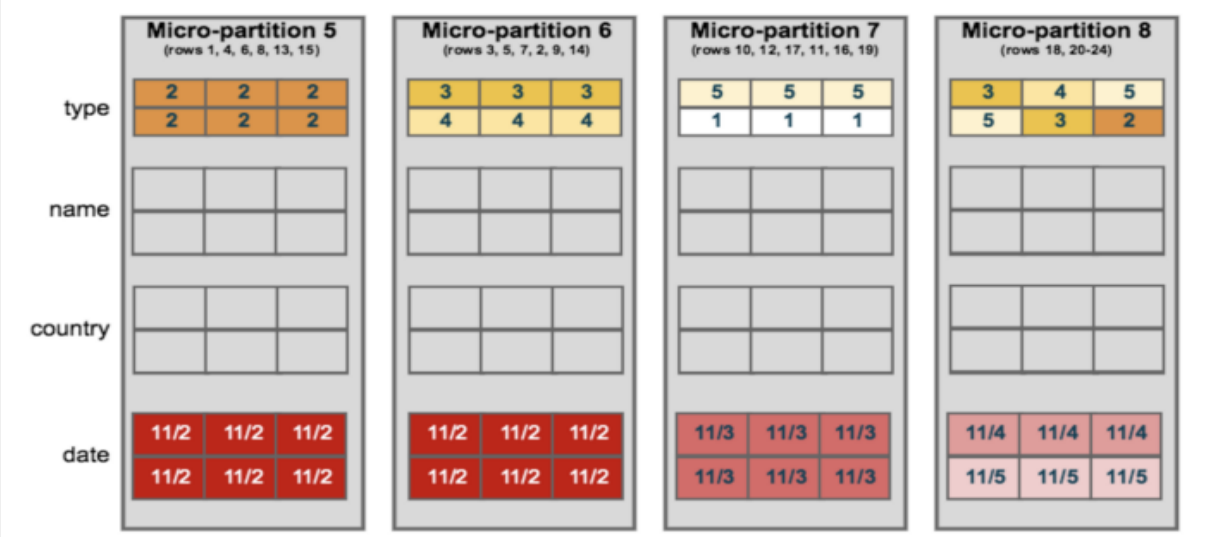
Chapter 5: Clustering

Fifth Chapter of the SnowFlake SnowPro Core Certification Complete Course.

In this chapter, we will look at one of the most important concepts for optimizing tables in SnowFlake, Clustering. We will discuss the following points.

DATA CLUSTERING

Typically, data stored in tables is sorted along natural dimensions, for example, by date. This process is called clustering, and data that is not sorted/clustered may hurt queries performance, particularly on huge tables, as SnowFlake will have to analyze more micro-partitions to give the result of a query. Let's look at the following example, where the micro-partitions are ordered by date. In this case, if we had to query for the date 11/2, SnowFlake wouldn't scan the last two micro-partitions, improving the performance of the query.



Clustering example (via docs.snowflake.com).

In SnowFlake, clustering metadata is collected and recorded for each micro-partition. SnowFlake then leverages this clustering information to avoid unnecessary scanning of micro-partitions during querying, significantly accelerating the performance of queries that reference these columns.

Clustering metadata that is collected for the micro-partitions in a table:

- The number of micro-partitions that comprise the table.
- The number of micro-partitions containing values that overlap with each other.
- The depth of the overlapping micro-partitions.

CLUSTERING DEPTH

The clustering depth measures the average depth of the overlapping micro-partitions for specified columns in a table (1 or greater). The smaller the cluster depth is, the better clustered the table is. A table without partitions would have a cluster depth or 0, although this

is not possible as a table will contain at least one micro-partition (but this can appear as an exam question). You can use the following commands to get the Cluster Depth:

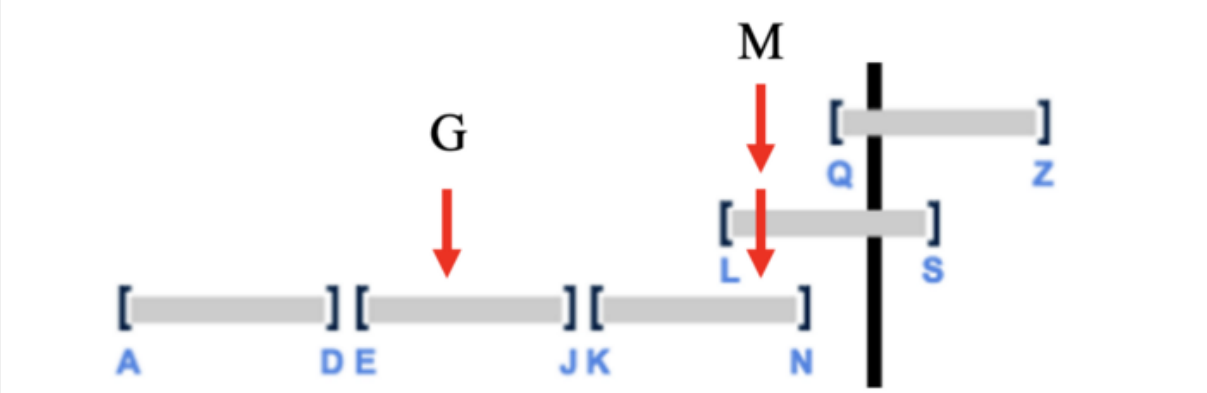
- SYSTEM$CLUSTERING_DEPTH
- SYSTEM$CLUSTERING_INFORMATION

Let's take a look at an example to understand the cluster keys. Let's imagine that we have the following table that is partitioned as follows:

Clustering Depth Example.

1. The first partition contains data from A to D.
2. The second partition contains data from E to J.
3. The third partition contains data from K to N.
4. The fourth partition contains data from L to S.
5. The fifth partition contains data from Q to Z.

It would be very simple to search for a value that starts with the letter G. SnowFlake would only have to go to the second partition and return the values, as there are no overlapping micro-partitions. But what would happen if we searched for a value that starts with the letter M? In this case, SnowFlake is accessing the third and fourth micro-partitions.



Clustering Depth looking for the G and the M letters.

Thus, we could say from this example:

- There are three overlapping micro-partitions.
- The clustering depth is 2. Why? Because if you look for data, it's going to check in two micro-partitions, as we saw before.

As we could see, a higher clustering depth would indicate that SnowFlakes checks a lot of micro-partitions, making the query slower.

CLUSTER KEYS

Clustering keys are a subset of columns or expressions on a table designated to co-locate the data in the same micro-partitions. This is useful for huge tables where the ordering was not ideal, or extensive insert operations have caused the table's natural clustering to degrade. Cluster Keys are placed on columns that are usually used in the WHERE / JOINS / ORDER BY... commands. They can also be a subset of expressions on a table, as we said before. Imagine you filter the table by month; you can use TO_MONTH to create a cluster key.

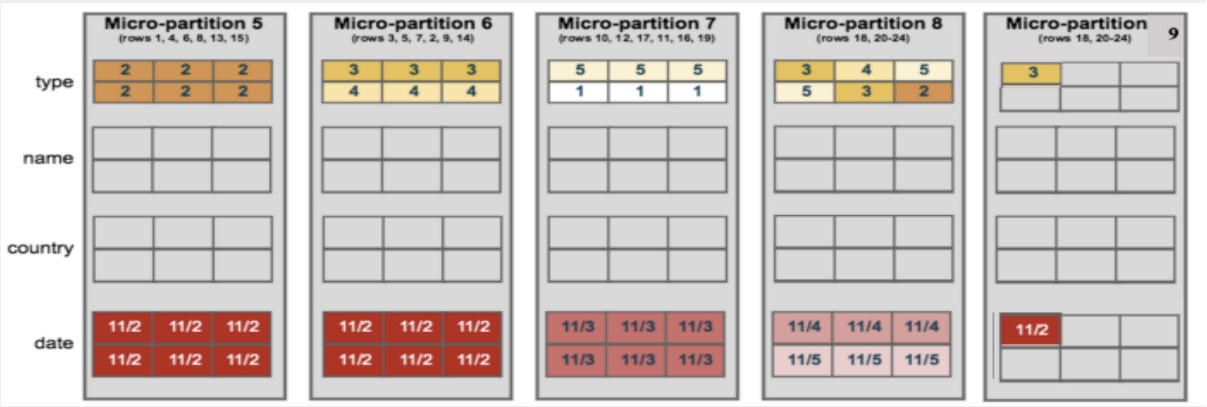
Some general indicators that can help determine whether to define a clustering key for a table include:

- Queries on the table are running slower than expected or have noticeably degraded over time.
- The clustering depth for the table is large.

RECLUSTERING

From time to time, as DML operations (INSERT, UPDATE, DELETE, MERGE, COPY) are performed on a clustered table, the data in the table might become less clustered. Let's imagine we have the table of the previous example, and we want to add another row with the "11/2" date, but the micro-partitions 5 & 6 are full of data. It will go to the micro-partition 9 for example. Or what if we modify a value from the micro-partition 5 from "11/2" to "11/4"? It will make the table less clustered. If we wanted to look for the "11/2" value, we would have to access three micro-partitions instead of just two like before:

To solve that, SnowFlake provides periodic & automatic re-clustering to maintain optimal clustering. This re-clustering operation consumes both credits and storage, and for this reason, the more frequently a table changes, the more expensive it will be to keep it clustered. Therefore, clustering is generally most cost-effective for tables that are queried often and do not change frequently.



How micro-partitions can be less clustered.

In the following example, we can see how re-clustering works. Date and Type are defined as the clustering key. When the table is re-clustered, new micro-partitions (5–8) are created, and the old ones will be marked as deleted. We had to access three partitions if we needed to do a query with a filter with Type = 2 and Date = 11/2. After re-clustering, we would only need to access one micro-partition.

Re-clustering example (via docs.snowflake.com).

TYPICAL EXAM QUESTIONS ABOUT CLUSTERING

1. What techniques would you consider to improve the performance of a query that takes a lot of time to return any result?

1. Define partition keys
2. Create cluster keys & turn auto clustering on the table
3. Create an index on the search result

Solution: 2.

2. Which of the following clustering metadata for the micro-partitions is maintained by SnowFlake in a table?

1. The number of micro-partitions that comprise the table.
2. The number of micro-partitions containing values that overlap with each other.
3. The depth of the overlapping micro-partitions.
4. None of the above.

Solution: 1, 2, 3.

3. Which of the below will you consider while choosing a cluster key

1. Columns that are typically used in the selective filters.
2. Columns are frequently used in join predicates.
3. Columns with extremely high cardinality.
4. Columns with extremely low cardinality.

Solution: 1, 2. A column with very low cardinality (e.g., a column indicating only whether a person is male or female) might yield minimal pruning. At the other extreme, a column with very high cardinality (e.g., a column containing UUID or nanosecond timestamp values) is also typically not a good candidate to directly use as a clustering key.

4. Does re-clustering in SnowFlake require manual configuration?

1. True
2. False

Solution: 2.

5. Is re-clustering in SnowFlake only triggered if the table would benefit from the operation?

1. True
2. False

Solution: 1.

6. What can you easily check to see if a large table will benefit from explicitly defining a clustering key?

1. Clustering depth
2. Clustering ratio
3. Values in a table

Solution: 1.

7. Which system functions are available in SnowFlake to view/monitor the clustering metadata for a table?

1. SYSTEM$CLUSTERING_DEPTH
2. SYSTEM$CLUSTERING_INFORMATION
3. SYSTEM$CLUSTERING_METADATA

Solution: 1, 2.

8. Is clustering generally most cost-effective for tables that are queried frequently and do not change often?

1. True
2. False

Solution: 1. The more frequently a table changes, the more expensive it will be to keep it clustered.

9.Which of the below columns are usually a good choice for clustering keys?

1. UUID column from a Customer in a 10TB table.
2. Gender male/female in a 20TB table.
3. Timestamp in a 10TB table.
4. Store_id in a 2TB table.

Solution: 4. It cannot have extremely high and low cardinality. UUID and timestamp have extremely high cardinality, as there will be a lot of customers/timestamps. Gender low cardinality. Store_id looks like the most convenient option.

Thanks for Reading!

Let me know if this was helpful. If you have any concerns and suggestions please feel free to reach out. Whatsapp +91-7999498574

LinkedIn: https://www.linkedin.com/in/avinash-sharma-553378151/

Regards,

Avinash Sharma