**What is snowflake data warehouse:**

Snowflake is a [**data warehouse**](http://www.stitchdata.com/resources/data-warehouse/) built on top of the AWS (Amazon Web Services) or Microsoft Azure or google cloud, cloud infrastructure. Snowflake is single platform for any workload, secure and governed access to all, unlimited performance and scalability.

There’s no hardware or software to install, configure, or manage, so it’s ideal for organizations that don’t want to dedicate resources for setup, maintenance, and support of in-house servers. And data can be moved easily into Snowflake using ETL solution.

Snowflake apart its architecture and data sharing capabilities. The Snowflake architecture allows storage and compute to scale independently, so we can use and pay for storage and computation separately. And the sharing functionality makes it easy for organizations to quickly share governed and secure data in real time.

Snowflake decouples the storage and compute functions. We can scale up or down as needed and pay for only the resources we use.

*Here: if I am not wrong, I believe our organization only use snowflake computation service, which is billed on a per-second basis. we use AWS for storage, so the storage bill we get from snowflake is a bypass from AWS.*

Snowflake is built specifically for the cloud, and it’s designed to address many of the problems found in older hardware-based data warehouses, such as limited scalability, data transformation issues, and delays or failures due to high query volumes

## **Snowflake Architecture**

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 fact, snowflake architecture consists of 3 layers, each of which is independently scalable: storage, compute, and service:

**Database storage**

The database storage layer holds all data loaded into Snowflake, including structured and semi-structured data. Snowflake automatically manages all aspects of how the data is stored: organization, file size, structure, compression, metadata, and statistics. This storage layer runs independently of compute resources.

we can combine structured and semi-structured data for analysis and load it into the cloud database without the need for conversion or transformation into a fixed relational schema first. Snowflake automatically optimizes how the data is stored and queried.

All data in Snowflake is stored in database tables, logically structured as collections of columns and rows. To best utilize Snowflake tables, particularly large tables, it is helpful to have an understanding of the physical structure behind the logical structure.

These topics describe *micro-partitions* and *data clustering*, two of the principal concepts utilized in Snowflake physical table structures.

**What are Micro-partitions?**

All data in Snowflake tables is automatically divided into micro-partitions, which are contiguous units of storage. Each micro-partition contains between 50 MB and 500 MB of uncompressed data (note that the actual size in Snowflake is smaller because data is always stored compressed). Groups of rows in tables are mapped into individual micro-partitions, organized in a columnar fashion. This size and structure allows for extremely granular pruning of very large tables, which can be comprised of millions, or even hundreds of millions, of micro-partitions.

Snowflake stores metadata about all rows stored in a micro-partition, including:

* The range of values for each of the columns in the micro-partition.
* The number of distinct values.
* Additional properties used for both optimization and efficient query processing.

The benefits of Snowflake’s approach to partitioning table data include:

* In contrast to traditional static partitioning, Snowflake micro-partitions are derived automatically; they don’t need to be explicitly defined up-front or maintained by users.
* micro-partitions are small in size (50 to 500 MB, before compression), which enables extremely efficient DML.
* Columns are stored independently within micro-partitions, 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.

## **Impact of Micro-partitions**

### Query Pruning

The micro-partition metadata maintained by Snowflake enables precise pruning of columns in micro-partitions at query run-time, including columns containing semi-structured data. In other words, a query that specifies a filter predicate on a range of values that accesses 10% of the values in the range should ideally only scan 10% of the micro-partitions.

For example, assume a large table contains one year of historical data with date and hour columns.our Call Center data for example. A query targeting a particular hour would ideally scan 1/8760th of the micro-partitions in the table and then only scan the portion of the micro-partitions that contain the data for the hour column; Snowflake uses columnar scanning of partitions so that an entire partition is not scanned if a query only filters by one column.

In other words, the closer the ratio of scanned micro-partitions and columnar data is to the ratio of actual data selected, the more efficient is the pruning performed on the table.

### Compute layer

The compute layer is made up of virtual warehouses that execute data processing tasks required for queries.

A virtual warehouse also called “warehouse’ is a cluster of compute resources in snowflake, a warehouse provides the required resources such as: CPU, memory, temp storage to perform operations like execute sql queries.

Each virtual warehouse (or cluster) can access all the data in the storage layer, then work independently, so the warehouses do not share, or compete for, compute resources.

The elastic nature of the cloud means if we want to load data faster, or run a high volume of queries, we can scale up your virtual warehouse to take advantage of extra compute resources. Afterward, we can scale down the virtual warehouse and pay for only the time we used.

Snowflake addresses concurrency issues with its unique multi-cluster architecture: Queries from one virtual warehouse never affect the queries from another, and each virtual warehouse can scale up or down as required. Data analysts and data scientists can get what they need, when they need it, without waiting for other loading and processing tasks to complete.

### Cloud services

The cloud services layer coordinates the entire system. It eliminates the need for manual data warehouse management and tuning. Services in this layer include:

Authentication, Infrastructure management, Metadata management, Query optimization, Access control.

# CACHING IN SNOWFLAKE DATA WAREHOUSE

In terms of performance tuning in Snowflake, there are very few options available. However, it is worth understanding how the Snowflake architecture includes various levels of caching to help speed queries.

This diagram illustrates the overall architecture which consists of three layers:

**Service Layer:**  Which accepts SQL requests from users, coordinates queries, managing transactions and results.  Logically, this can be assumed to hold the *result cache*– a cached copy of the results of every query executed.

**Compute Layer:**  Which does the heavy lifting.  This is where the actual SQL is executed across the nodes of a *Virtual Data Warehouse*.  This layer holds a cache of data queried and is often referred to as *Local Disk I/O* although in reality this is implemented using SSD storage.  All data in the compute layer is temporary, and only held as long as the virtual warehouse is active.

**Storage Layer:**  Which provides long term storage of results.  This is often referred to as *Remote Disk* and is currently implemented on either Amazon S3 or Microsoft Blob storage.

## Snowflake Cache Layers

The diagram illustrates the levels at which data and results are cached for subsequent use. These are:-

1. **Result Cache:**  Which holds the results of every query executed in the past 24 hours. These are available across virtual warehouses, so query results returned to one user is available to any other user on the system who executes the same query, provided the underlying data has not changed.
2. **Local Disk Cache:**  Which is used to cache data used by SQL queries.  Whenever data is needed for a given query it's retrieved from the Remote Disk storage, and cached in SSD and memory.
3. **Remote Disk:**  Which holds the long term storage.  This level is responsible for data resilience, which in the case of Amazon Web Services, means 99.999999999% durability.  Even in the event of an entire data centre failure.

### Clear the Snowflake Virtual Warehouse Cache

Unlike many other databases, you cannot directly control the virtual warehouse cache. However, you can determine its size, as (for example), an X-Small virtual warehouse (which has one database server) is 128 times smaller than an X4-Large. Each increase in virtual warehouse size effectively doubles the cache size, and this can be an effective way of improving snowflake query performance, especially for very large volume queries.

You can also clear the virtual warehouse cache by suspending the warehouse and the SQL statement below shows the command.

**The first query profile overview:**

This query returned in around 20 seconds, and demonstrates it scanned around 12Gb of compressed data, with 0% from the local disk cache. This means it had no benefit from disk caching.

he bar chart above demonstrates around 50% of the time was spent on local or remote disk I/O, and only 2% on actually processing the data. Clearly any design changes we can do to reduce the disk I/O will help this query.

The results also demonstrate the queries were unable to perform any *partition pruning* which might improve query performance. We’ll cover the effect of partition pruning and clustering in the next article.

**The second query profile overview:**

This query was executed immediately after, but with the result cache disabled, and it completed in 1.2 seconds – around 16 times faster. In this case, the Local Disk cache (which is actually SSD on Amazon Web Services) was used to return results.

the disk I/O has been reduced to around 11% of the total elapsed time, and 99% of the data came from the (local disk) cache. While querying 1.5 billion rows, this is clearly an excellent result.

**The third query profile overview:**

The third profile indicates the entire query was served directly from the result cache (taking around 2 milliseconds). many dashboard applications involve repeatedly refreshing a series of screens and dashboards by re-executing the SQL. In these cases, the results are returned in milliseconds.

results are normally retained for 24 hours, although the clock is reset every time the query is re-executed, up to a limit of 30 days, after which results query the remote disk.

unlike Oracle where additional care and effort must be made to ensure correct partitioning, indexing, stats gathering and data compression, Snowflake caching is entirely automatic, and available by default. Absolutely no effort was made to tune either the queries or the underlying design,

Clearly data caching makes a massive difference to Snowflake query performance, but what can you do to ensure maintain the performance when you cannot change the cache?

* **Auto-Suspend:**By default, Snowflake will auto-suspend a virtual warehouse (the compute resources with the SSD cache after 10 minutes of idle time.  Best practice?  Leave this alone.  Keep in mind, you should be trying to balance the cost of providing compute resources with fast query performance.  To illustrate the point, consider these two extremes:
  1. **Suspend after 60 seconds:**When the warehouse is re-started, it will (most likely) start with a clean cache, and will take a few queries to hold the relevant cached data in memory.  (Note:  Snowflake will try to restore the same cluster, with the cache intact, but this is not guaranteed).
  2. **Suspend Never:**  And your cache will always be *warm*, but you will pay for compute resources, even if nobody is running any queries. However, provided you set up a script to shut down the server when  not being used, it may make sense.

Scaling the Virtual Warehouse Cache

* **Scale up for large data volumes:**   If you have a sequence of large queries to perform against massive (multi-terabyte) size data volumes, you can improve workload performance by scaling up.  Simple execute a SQL statement to increase the virtual warehouse size, and new queries will start on the larger (faster) cluster.  While this will start with a clean (empty) cache, you should normally find performance doubles at each size, and this extra performance boost will more than out-weigh the cost of refreshing the cache.  However, if you scale up (or down) the data cache is cleared.
* **Scale down - but not too soon:**   Once your large task has completed, you could reduce costs by scaling down or even suspending the virtual warehouse.

**SNOWFLAKE STRUCTURE**

Databases and schemas are used to organize data stored in Snowflake:

* A database is a logical grouping of schemas. Each database belongs to a single Snowflake account.
* A schema is a logical grouping of database objects (tables, views, etc.). Each schema belongs to a single database.