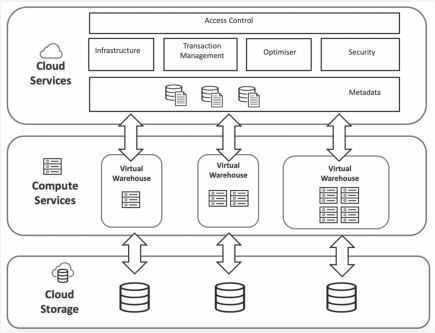
Snowflake Database Architecture



1. **Service Layer**: Accepts SQL requests from users, coordinates queries, managing transactions and results. It holds a cached copy of the results.
2. **Compute Layer**: Where the actual SQL is executed across the nodes of a VW. It holds a cache of data queried (referred as Local Disk I/O, SSD storage in reality). All data held is temporary -VW is active
3. **Storage Layer**: Provides long term storage of results (Remote Disk - implemented on either Amazon S3 or Microsoft Blob storage)

Virtual Warehouses

Overview

• resources provision for executing SQL SELECT statements and DML operations (DELETE, INSERT, UPDATE, COPY INTO)
• Can be started and stopped/resized at any time, running queries are not affected but only the new queries
• Two types: Standard and Snowpark-optimized

Credit Usage and Billing

• Snowflake utilizes per-second billing (with a 60-second minimum each time the warehouse starts) so warehouses are billed only for the credits they actually consume
• For a multi-cluster warehouse, the number of credits billed is calculated based on the warehouse size and the number of clusters that run within the time period

Data Loading

• Data loading performance doesn't necessarily improved by increasing the size of a warehouse. It is influenced more by the number of files being loaded (and the size of each file) than the size of the warehouse.
• Warehouse size can impact the amount of time required to execute queries especially for more complex queries. Larger warehouses have more compute resources available to process queries but not necessarily faster for small, basic queries.

Query Processing and Concurrency

Virtual Warehouses (cont)

• No. of queries that a warehouse can concurrently process is determined by the size and complexity of each query
• If queries are queuing more than desired, create another warehouse and redirect the queries to it manually. Resizing a warehouse can enable limited scaling for query concurrency but this is mainly intended for improving query performance.

Multi-cluster warehouses

• Multi-cluster warehouses are recommended to enable fully automated scaling for concurrency
• Only available for Enterprise Edition (or higher)
• Up to 10 clusters
• Support all same properties and actions as single warehouse

Mode:

• Maximized: same value for both max and min # clusters
- effective for statically controlling the available compute resources
• Auto-scale: different values for max and min # clusters
- dynamically manage the load based on scaling policy that determines when automatically starting or shutting down additional clusters.

> NOTE: Multi-cluster warehouses: best for scaling resources to improve concurrency for users/queries. Resizing the warehouse: best for improving the performance of slow-running queries or data loading

Scale Up vs. Scale Out

• Scale up: resizing a warehouse.
• Scale out: adding clusters to a multi-cluster warehouse (requires Snowflake Enterprise Edition or higher).

Warehouse Size

Warehouse Size	Credits / Hour	Credits / Second	Notes
X-Small	1	0.0003	Default size for warehouses created using CREATE WAREHOUSE.
Small	2	0.0006	
Medium	4	0.0011	
Large	8	0.0022	
X-Large	16	0.0044	Default for warehouses created in the web interface.
2X-Large	32	0.0089	
3X-Large	64	0.0178	
4X-Large	128	0.0356	
5X-Large	256	0.0711	Generally available in Amazon Web Services regions, and in preview in US Government and Azure regions.
6X-Large	512	0.1422	Generally available in Amazon Web Services regions, and in preview in US Government and Azure regions.

Larger warehouse sizes 5X-Large and 6X-Large are generally available in all Amazon Web Services (AWS) regions, and are in preview in the US Government regions (requires FIPS support on ARM) and Azure regions

Scaling Policy (Auto Mode)

Policy	Strategy	Starts…	Shuts Down..
Standard (default)	Favours starting additional clusters over conserving credits	First cluster will start immediately when one or more queries are detected in the queue. Each successive clusters waits to start 20 seconds after the prior one has started.	After 2-3 consecutive successful checks (performed at 1 minute intervals), which determine whether the load on the least-loaded cluster could be redistributed
Economy	Favours keeping running clusters fully--loaded rather than starting additional clusters	Only if the system estimates there's enough query load to keep the cluster busy for at least 6 minutes	After 5-6 consecutive successful checks (performed at 1 minute intervals), which determine whether the load on the least-loaded cluster could be redistributed

Snowpark-optimized Warehouses

Provide 16x memory per node compared to a standard Snowflake VW

• Recommended for workloads that have large memory requirements: ML training use cases using a stored procedure on a single VW node

• Benefits Snowpark workloads utilizing UDF or UDTF

• Snowpark-optimized warehouses are not supported on X-Small or SMALL warehouse sizes

• Snowpark-optimized warehouses are available in all regions across AWS, Azure, and Google Cloud.

Caching

1. **Result Cache (Service Layer)**: Holds the results of every query executed in the past 24 hours. These are available across VWs, so query results returned to any user who executes the same query, provided the underlying data has not changed.

2. **Local Disk Cache (VMs)**: 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 (DB Storage)**: Holds the long term storage. This level is responsible for data resilience.

> NOTE: Results are retained for 24 hours, but 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.

System Performance Tuning Best Practice

• **Auto-Suspend**: Snowflake will auto-suspend a virtual warehouse with the SSD cache after 10 minutes of idle time by default. Best to leave this setting alone.

• **Scale up for large data volumes**: Have a sequence of large queries + massive (multi-terabyte) size data volumes - scale up to improve query performance

• **Scale down - but not too soon**: Tune the warehouse size dynamically (once large task has completed), but don't keep adjusting it, or you'll lose the benefit.

Databases, Tables & Views

Micro-partitions

• All data in Snowflake tables is automatically divided into micro-partitions which contains between 50 MB and 500 MB of uncompressed data

• Tables are partitioned using the ordering of the data as it is inserted/loaded.

Benefits of Mico-partitions

• Don't need to be explicitly defined beforehand/maintained by users (derived automatically)

• Small in size which enables efficient DML and fine-grained pruning for faster queries

• Can overlap in their range of values, which, combined with their uniformly small size, helps prevent skew.

• Columns are stored independently (columnar storage) which allows efficient scanning of individual columns

> NOTE: Snowflake does not prune micro-partitions based on a predicate with a subquery, even if the subquery results in a constant

Data Clustering

• Clustering metadata that is recorded for each micro-partition created, is then utilised to avoid unnecessary scanning of micro-partitions during querying

Actions performed for queries on the table

Databases, Tables & Views (cont)

1. First, prune micro-partitions that are not needed for the query.
2. Then, prune by column within the remaining micro-partitions.

Clustering Information Maintained

• The total # micro-partitions
• # micro-partitions containing values that overlap with each other (in a specified subset of table columns).
• The depth of the overlapping micro-partitions.

Clustering Key

As DML occurs on large tables, the data might no longer cluster optimally. Snowflake designates one/more table columns as a clustering key to improve clustering of the underlying table micro-partitions automatically

Use clustering key when:

• Require the fastest possible response times, regardless of cost.
• Improved query performance offsets the credits required to cluster and maintain the table.
• Queries on the table are running slower than expected
• The clustering depth for the table is large.

> NOTE: Clustering is generally most cost-effective for tables that are queried frequently and do not change frequently

Comparison of Table Types

Type	Persistence	Cloning (source type → target type)	Time Travel Retention Period (Days)	Fail-safe Period (Days)
Temporary	Remainder of session	Temporary → Temporary Temporary → Transient	0 or 1 (default is 1)	0
Transient	Until explicitly dropped	Transient → Temporary Transient → Transient	0 or 1 (default is 1)	0
Permanent (Standard Edition)	Until explicitly dropped	Permanent → Temporary Permanent → Transient Permanent → Permanent	0 or 1 (default is 1)	7
Permanent (Enterprise Edition and higher)	Until explicitly dropped	Permanent → Temporary Permanent → Transient Permanent → Permanent	0 to 90 (default is configurable)	7

• After creation, transient/temporary tables cannot be converted to any other table type.
• The Fail-safe period is not configurable for any table type.
• Transient and temporary tables have no Fail-safe period (no additional data storage charges beyond the Time Travel retention period)
• Transient tables: a good option for managing the cost of very large transitory data but the data in these tables cannot be recovered after the Time Travel retention period passes.

Search Optimization Service

• improve the performance of certain types of lookup and analytical queries that use an extensive set of predicates for filtering
• Features that available (Enterprise Ed./Higher)
- Column configuration
- Support for substring + REGEX searches
- Support for fields in VARIANT, OBJECT and ARRAY columns
- Support for geospatial functions with GEOGRAPHY objects
- Selective point lookup queries on tables (returns only small # of distinct rows)
• One of the ways to optimize query performance (others: clustering a table & create materialised views - cluster/unclustered)

>> **Works best to improve the performance of a query when the table is frequently queried on columns other than the primary cluster key**

Queries that benefited from SO

• runs for a few seconds or longer
• at least one of the columns accessed through query filter operation has at least 100k distinct values
• Equality or IN predicates; Substring & REGEX, etc.

Not supported by SO

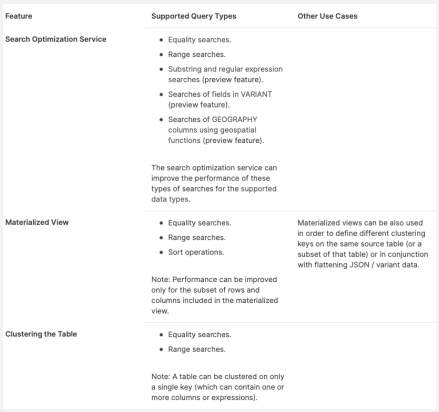
• External tables, materialised views, columns defined with a collate clause, col concatenation, analytical expressions, cast on table columns

Costs of SO service

• The service creates a search access path data structure that requires space
- storage cost depends on: the # of distinct values in the table (size is approx. 1/4 of the original table's size); Worst case scenario = same size as original table
• Adding the service + maintaining it consumes additional resources
- higher cost when there is high churn (large values of data in the table change) - proportional to the amount of data ingested

>> **The costs are proportional to the # of tables on which SO is enabled, the # of distinct values in those tables; the amount of data that changes in these tables**

Options for Optimising Query Performance



Feature	Supported Query Types	Other Use Cases
Search Optimization Service	• Equality searches. • Range searches. • Substring and regular expression searches (preview feature). • Searches of fields in VARIANT (preview feature). • Searches of GEOGRAPHY columns using geospatial functions (preview feature). The search optimization service can improve the performance of these types of searches for the supported data types.	
Materialized View	• Equality searches. • Range searches. • Sort operations. Note: Performance can be improved only for the subset of rows and columns included in the materialized view.	Materialized views can be also used in order to define different clustering keys on the same source table (or a subset of that table) or in conjunction with flattening JSON / variant data.
Clustering the Table	• Equality searches. • Range searches. Note: A table can be clustered on only a single key (which can contain one or more columns or expressions).	

Materialised Views

• Like a view that is frozen in place but when changes are detected, Snowflakes will refresh it automatically

• It's designed to improve query performance for workloads composed of common & repeated query patterns.

• It incurs additional costs.

When to use it?

• Query results contain a small # rows and/or columns relative to the base table

• Query results contain results that require significant processing

• Query is on an external table

• The view's base table does not change frequently

Advantages of Materialised Views

• improve performance of queries that use the same subquery results repeatedly

• Background service updates the materialised view automatically after changes are made to the base table

• Data accessed through materialised views is always current

>> We can't put a materialised view DIRECTLY on top of staged data.But, if we put an External Table in bt. them, we CAN put a Materialised view over staged data

>> INFORMATION_SCHEMA.VIEWS does not show materialized views. Materialized views are shown by INFORMATION_SCHEMA.TABLES

Loading Data

Best Practices in File Sizing

• No. of load operations that run in parallel **shouldn't be more** than no. of data files to be loaded. Aim to have data files **~100-250 MB (or larger)** in size compressed to optimise loading - same applies to Snowpipe

• Loading very large files (e.g. 100 GB or larger) is not recommended. Use ON_ERROR copy option value should you need to do so.

• Aborting could cause delays & credits wastage. Risking no portion of the file to be committed if data loading takes more than 24 hours.

• Smaller files, smaller processing overhead. Split large files by line to avoid records that span chunks

• VARIANT data type (Semi-structured data) has a 16 MB size limit on individual rows

• Snowpipe cost involves an overhead to manage files in the load queue (staging data) and resource consumption (loading data).

• Creating a new data file once per minute as best practice as it provides good balance between cost (i.e. resources spent on Snowpipe queue management and the actual load) and performance (i.e. load latency)

Best Practices in File Formats

Delimited Text Files

• UTF-8 is the default character set

• Fields that contain delimiter characters/carriage returns should be enclosed in quotes

• Consistent number of columns in each row

Semi-structured Data Files

• Extract semi-structured data elements containing “null” values into relational columns before loading them, else set STRIP_NULL_VALUES to TRUE if the "null" indicates only missing values

• Ensure each unique element stores values of a single native data type (string or number)

Numeric Data Files

• Avoid embedded characters, e.g. commas

• Number that includes a fractional component should be separated from the whole number portion by a decimal point

> NOTE: Snowflake checks temporal data values at load time. Invalid date, time, and timestamp values (e.g. 0000-00-00) produce an error

Best Practices in organising data by path

• Both internal and external stage references can include a path (or prefix in AWS terminology)

Loading Data (cont)

• It's recommended to partition the data into logical paths that include identifying details such as geographical location or other source identifiers
• Narrow the path to the most granular level that includes your data for improved data load performance.

Bulk Loading

Bulk Loading from a Local File System
COPY command is used to bulk load data from a local file system into tables using an internal (i.e. Snowflake-managed) stage using two steps:
1. Upload (i.e. stage) one or more data files to a Snowflake stage (named internal stage or table/user stage) using the PUT command.
2. Use the COPY INTO command to load the contents into table.
NOTE: Virtual Warehouse must be running to execute this step as it provides the compute resources to perform the actual insertion of rows into the table
By default, each user and table in Snowflake is automatically allocated an internal stage for staging data files to be loaded. In addition, you can create named internal stages
3 types of internal stages are supported: User, Table and Named
• User Stage
- reference using @~
- cannot be altered or dropped.
- do not support setting file format options, it must be specified as part of the COPY INTO command.
NOT suitable when:
1. Multiple users require access to the files.
2. The current user does not have INSERT privileges on the tables the data will be loaded into.
• Table Stage
- same name as the table; reference @%tablename
- cannot be altered or dropped
- do not support transforming data while loading it
- must be the table owner (have the role with the OWNERSHIP privilege on the table)
NOT suitable when:
1. you need to copy the data in the files into multiple tables
• Named Stage
- provide the greatest degree of flexibility for data loading
- users with the appropriate privileges on the stage can load data into any table

Bulk Loading (cont)

- the privileges can be granted or revoked from roles. In addition, ownership of the stage can be transferred to another role.
- recommended when you plan regular data loads that could involve multiple users and/or tables.

Snowpipe

Loads data from files according to the COPY statement defined in a referenced pipe (a named, first-class Snowflake object that contains a COPY statement) as soon as they are available in a stage.
• All data types are supported, including semi-structured data types such as JSON and Avro
• 2 mechanisms for detecting the staged files:
- Automating Snowpipe using cloud messaging
- Calling Snowpipe REST endpoints
• Generally loads older files first, but there is no guarantee that files are loaded in the same order they are staged
• Uses file loading metadata associated with each pipe object to prevent reloading the same files
• Difficult to estimate latency as many factors can affect Snowpipe loads, e.g. File formats and sizes, and the complexity of COPY statements (including SELECT statement used for transformations) can all impact the amount of time required
• Charges are calculated by per second/per core granularity
• Check usages/charges in 2 ways:
- Account -> Usage/Bill
- SQL -> Information Schema -> PIPE_USAGE_HISTORY

Bulk vs. Continuous Loading

Areas	Bulk Data Loading	Snowpipe
Mechanism	load batches of data in cloud storage/copy data files from local using COPY command	load small volume of data within minutes after files are staged to ensure near real time availability
Compute Resources	Requires a user-specified warehouse to execute COPY statements	Snowflake-provided resources (i.e. a serverless compute model)

Bulk vs. Continuous Loading (cont)

Authentication	Security options supported by the client for authenticating and initiating a user session	When calling the REST endpoints: Requires key pair authentication with JSON Web Token (JWT)
Load History	Stored in the metadata of the target table for 64 days. Available upon completion of the COPY statement as the statement output	Stored in the metadata of the pipe for 14 days. Must be requested from Snowflake via a REST endpoint, SQL table function, or ACCOUNT_USAGE view
Transactions	Always in a single transaction. Data is inserted into table alongside any other SQL statements submitted manually by users	Combined or split into a single or multiple transactions based on the number and size of the rows in each data file. Rows of partially loaded files (based on the ON_ERROR copy option setting) can also be combined or split into one or more transactions
Cost	The amount of time each virtual warehouse is active	According to the compute resources used in the Snowpipe warehouse while loading the files

Semi-structured Data

Data that does not conform to the standards of traditional structured data and have two main characteristics: nested data structures and lack of a fixed schema.

• Snowflake can import semi-structured data from JSON, Avro, ORC, Parquet, and XML formats and store it in data types such as VARIANT, ARRAY or OBJECT

- A VARIANT can contain any other data type, including an ARRAY or an OBJECT (used to build and store hierarchical data)
- An ARRAY or OBJECT can directly contain VARIANT (thus can indirectly contain any other data type, including itself)

> **NOTE: a Snowflake OBJECT corresponds to a "dictionary" or a "map". A Snowflake object is not an "object" in the sense of OOP. Curly braces indicate an OBJECT, which contains key-value pairs**

• Semi-structured data can be stored in a single column or split into multiple columns.

Loading Semi-structured Data

• Can explicitly specify all, some, or none of the structure when you load and store semi-structured data

• If your data is a set of key-value pairs, you can load it into a column of type OBJECT.

• If your data is an array, you can load it into a column of type ARRAY.

• If the data is complex or an **individual value requires more than about 16MB of storage space**, then you can, for instance, split the data into multiple columns, and some of those columns can contain an explicitly-specified hierarchy of data types.

> NOTE: When we split the data across multiple columns, we may use detect-and-retrieve feature but it is currently limited to Apache Parquet, Apache Avro, and ORC files.

Querying Semi-structured Data

• Operations that are supported:

- Accessing an element in an array.
- Retrieving a specified value from a key-value pair in an OBJECT.
- Traversing the levels of a hierarchy stored in a VARIANT.

• The query output is enclosed in double quotes because the query output is VARIANT, not VARCHAR. (Operators : and subsequent . and [] always return VARIANT values containing strings.)

• Flatten the array to retrieve all instances of a child element in a repeating array.

• Use GET/GET_PATH function to extract a value from a VARIANT column

Security Principles

Federated Authentication

User authentication is separated from user access through the use of one or more external entities that provide independent authentication of user credentials (e.g. Single Sign-On (SSO))

• consists 2 components: Service provider (SP) and Identity provider (IdP)

- Snowflake servers as the SP; IdP provides these services to Snowflake: Creating and maintaining user credentials and other profile information + Authenticating users for SSO access to the SP.

• 2 Types of IdP: The native Snowflake support provided by Okta and ADFS, Snowflake supports using most SAML 2.0-compliant vendors as an IdP: Google G Suite, Microsoft Azure Active Directory, OneLogin & Ping Identity PingOne

>> For a web-based IdP (e.g. Okta), closing the browser tab/window does not necessarily end the IdP session. If a user's IdP session is still active, they can still access Snowflake until the IdP session times out.

Multi-Factor Authentication (MFA)

• MFA is enabled on a per-user basis; To use MFA, users must enroll themselves

• All users with the ACCOUNTADMIN role be required to use MFA is recommended.

• Duo Push authentication mechanism is used when a user is enrolled in MFA

>> MFA token caching can be combined with connection caching in federated single sign-on: ensure that the ALLOW_ID_TOKEN parameter is set to true in tandem with the ALLOW_CLIENT_MFA_CACHING parameter.

Network Policy

• Allow restricting access to your account based on user IP address. It enables you to create an IP allowed list, as well as an IP blocked list, if desired.

• A network policy is not enabled until it is activated at the account or individual user level.

>> Only security administrators (i.e. users with the SECURITYADMIN role) or higher or a role with the global CREATE NETWORK POLICY privilege can create network policies. Ownership of a network policy can be transferred to another role.

• If a network policy is activated for an individual user, the user-level network policy takes precedence.

Snowflake Session

• A session is independent of an identity provider (i.e. IdP) session

• After the idle session timeout (with a max 4 hours of inactivity), the user must authenticate to Snowflake again.

• A session policy can modify the idle session timeout period

Security Principles (cont)

• Unset the session policy first and then set the new session policy to replace a session policy that is already set for an account or user

• The classic web interface tracks user activity and sends a heartbeat to an internal Snowflake monitor. This heartbeat recording is sent every three minutes and is not configurable.

Access Control

• Access control privileges determine who can access and perform operations on specific objects in Snowflake

• Discretionary Access Control (DAC): Each object has an owner -> grant access to that object.

• Role-based Access Control (RBAC): Access privileges are assigned to roles, which are in turn assigned to users.

>> A role owner (i.e. the role that has the OWNERSHIP privilege on the role) does not inherit the privileges of the owned role. Privilege inheritance is only possible within a role hierarchy.

• System-defined roles are created with privileges related to account-management. As a best practice, it is not recommended to mix account-management privileges and entity-specific privileges in the same role.