Introduction

What is Snowflake

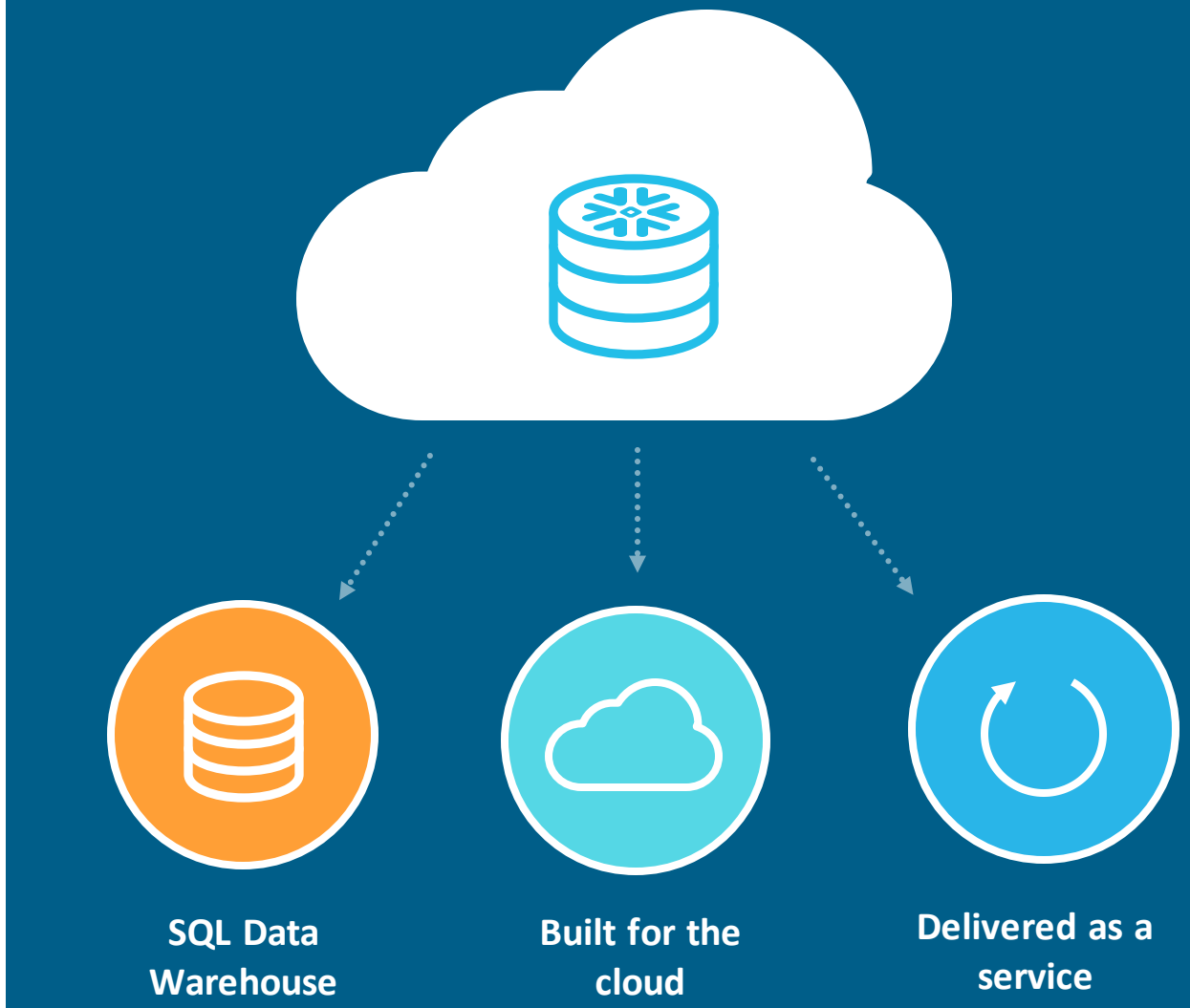
Snowflake is a fully relational ANSI SQL Columnar data warehouse build for the Cloud and Delivered as a service. Snowflake delivers the performance, concurrency, and simplicity needed to store and analyze all of an organization's data in one solution. Snowflake's Massively Parallel Processing clusters can Scale upto Petabyte size

Our Vision

Provide all users anytime, anywhere insights so they can make actionable decisions based on data

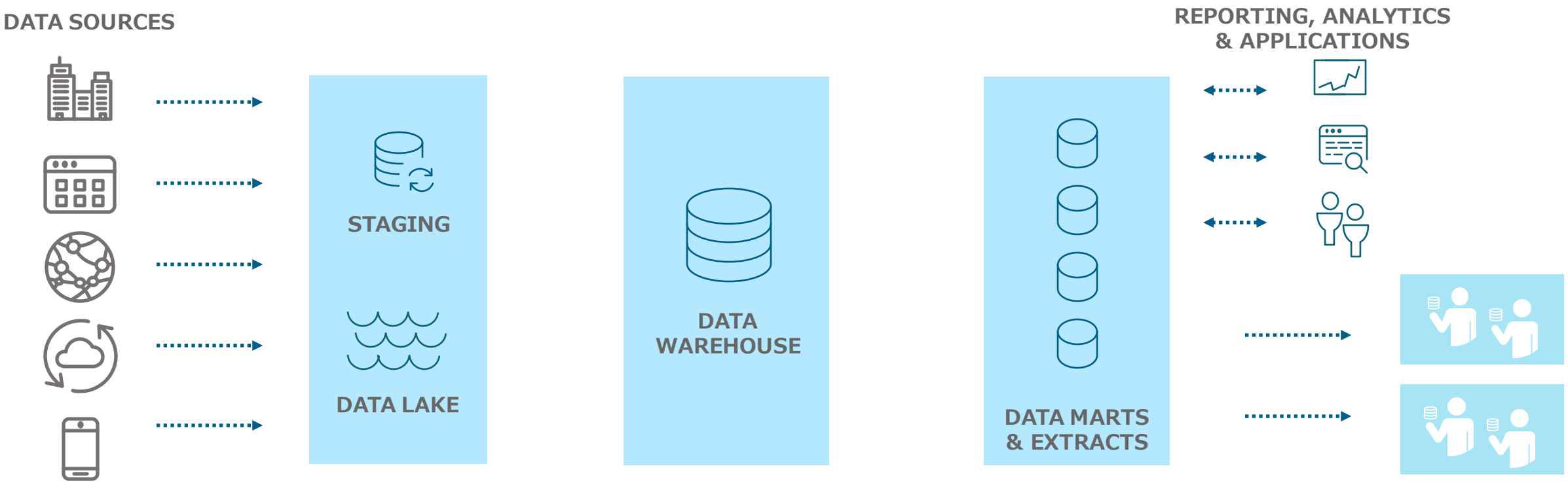
Our Solution

Next-generation data warehouse built from the ground up for the cloud and for today's data and analytics



Complete Cloud Datawarehouse

What customers are doing with Snowflake



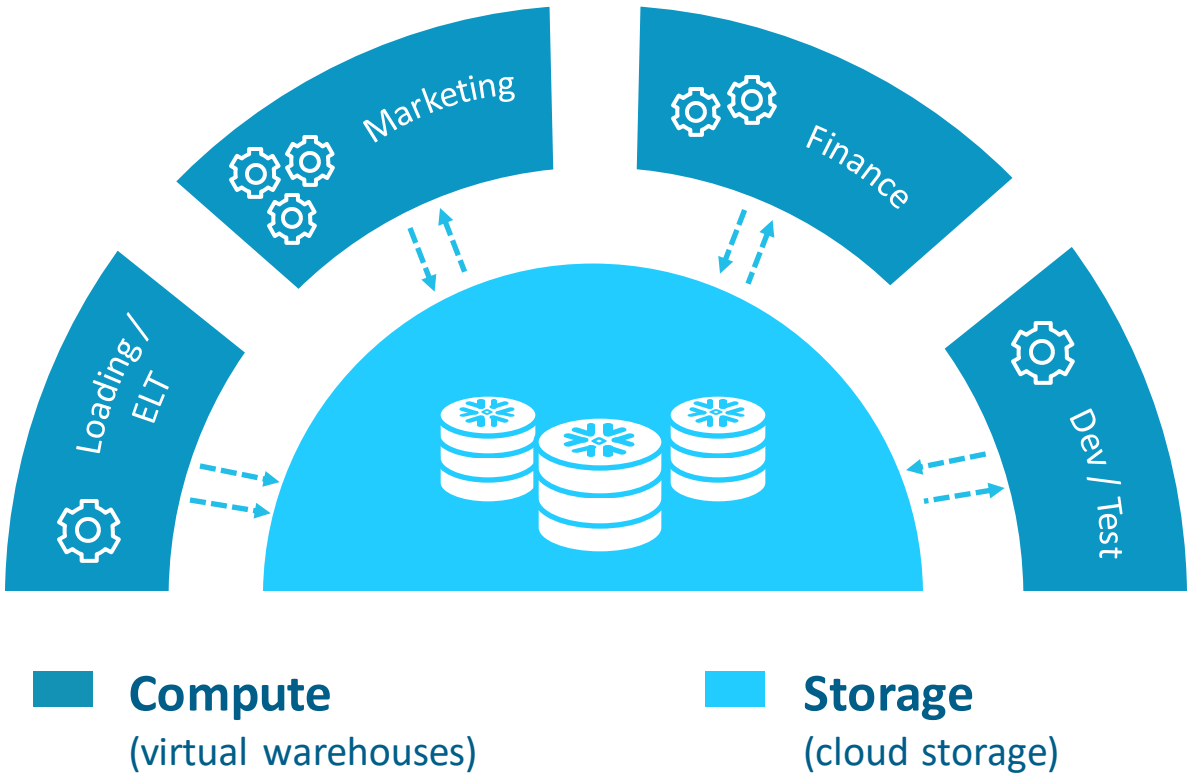
Gaming company replaced Hadoop + SQL database with Snowflake

Consumer retailer modernizing DW by replacing legacy appliance with Snowflake

Market research company consolidated data marts to reduce costs and data silos

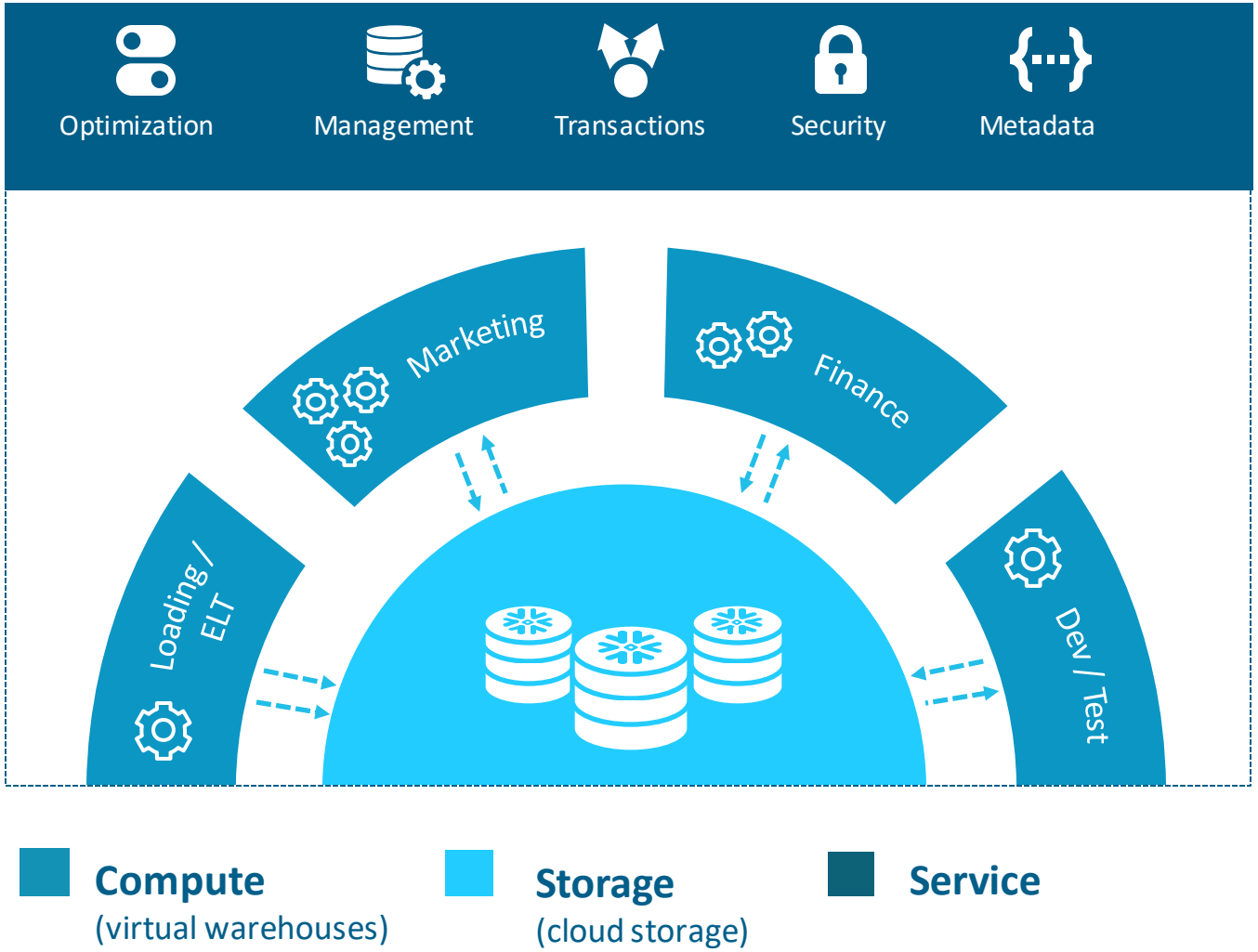
Mobile analytics company shares live data with clients

Snowflake architecture: Storage and compute



- Storage separated from compute
- Multiple compute clusters ("virtual warehouses")
- Virtual warehouses concurrently access data without contention

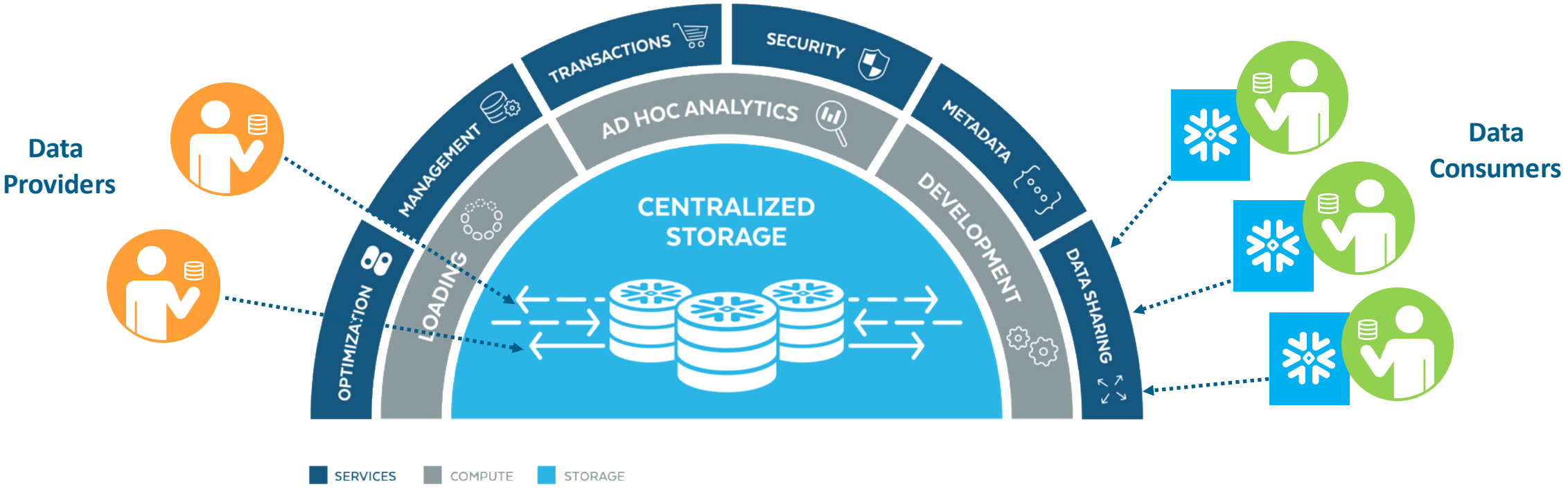
Snowflake architecture: Service



- Centralized management
- Metadata separate from storage and compute
- Full transactional consistency across entire system

Instant, live data sharing

- Architecture enabled
- Secure and integrated with RBAC
- Query and combine with existing data



Complete SQL database

Core SQL

- Simple DDL
- Support for Information Schema
- Data loading via bulk copy, INSERT
- Full support for update DML: MERGE, UPDATE, DELETE
- Multi-statement transactions

Analytic SQL

- Windowing functions (LEAD, LAG, RANK, NTILE, …)

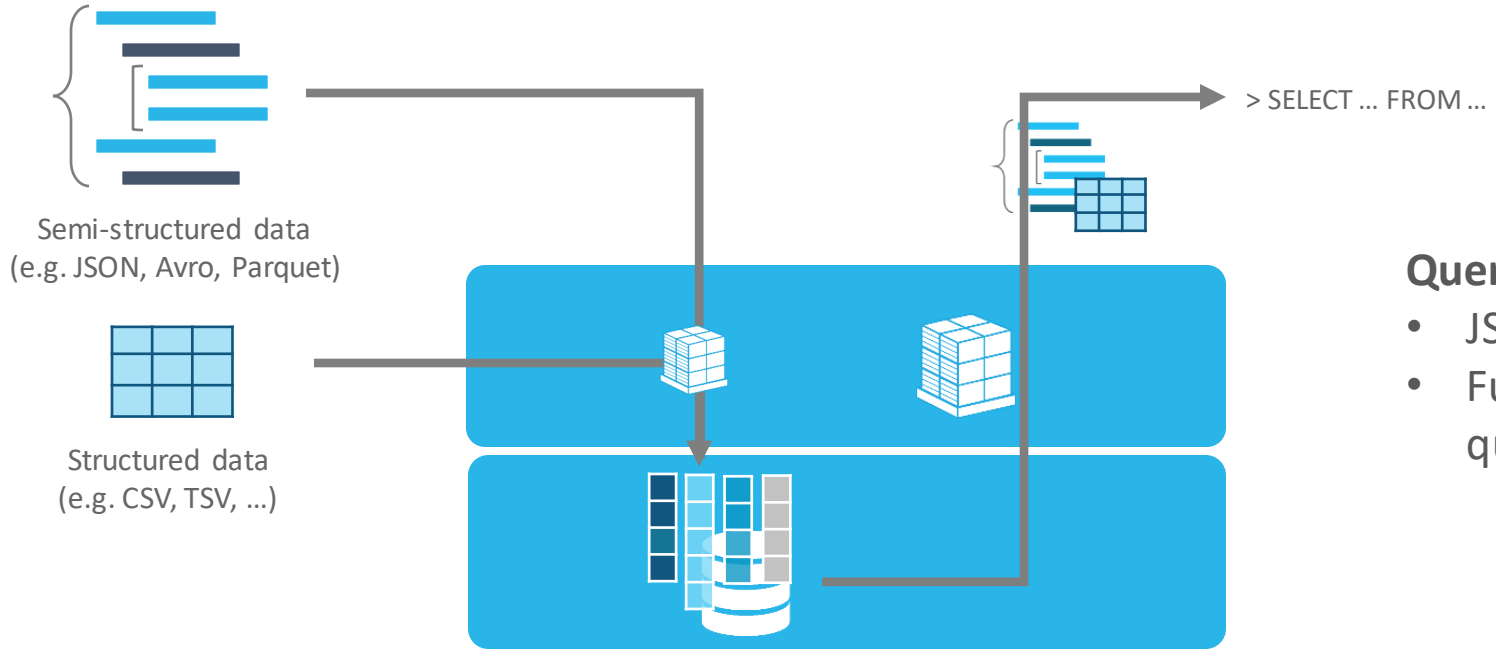
Extensibility

- SQL UDFs
- Javascript UDFs
- Session Variables

```
SHOW LOAD HISTORY

INSERT INTO …

BEGIN …
COMMIT;

SELECT foo, bar, binky, LEAD(bar) OVER …

CREATE OR REPLACE FUNCTION
get_countries_for_user ( id number )…
```

Sensitivity: Internal & Restricted

Native support for structured and semi-structured data



Query optimization

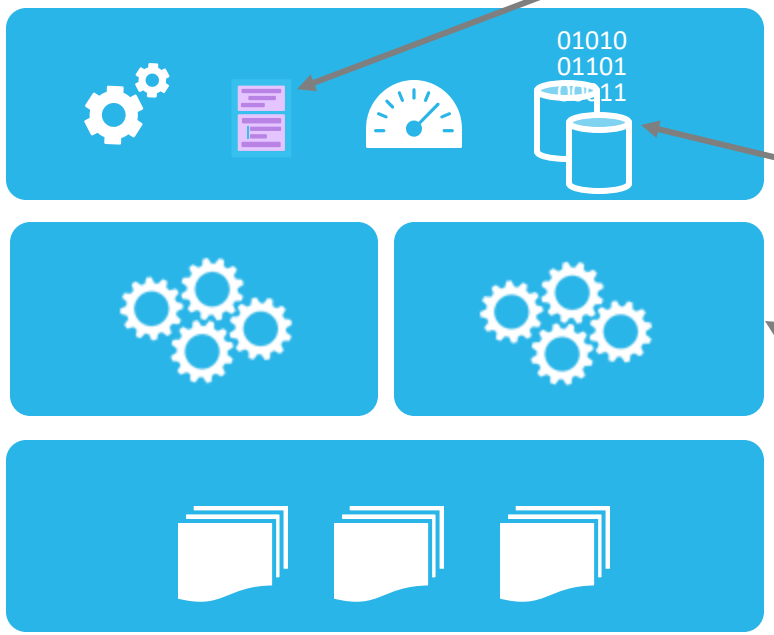
- JSON paths in SQL queries
- Full database optimization for queries on semi-structured data

Storage optimization

- Transparent discovery and storage optimization of repeated elements

Sensitivity: Internal & Restricted

Dynamic caching for optimal query performance



Result Cache

- All query results stored for 24 hours unless underlying data changes
- Identical queries are returned instantly without requiring compute

Metadata Cache

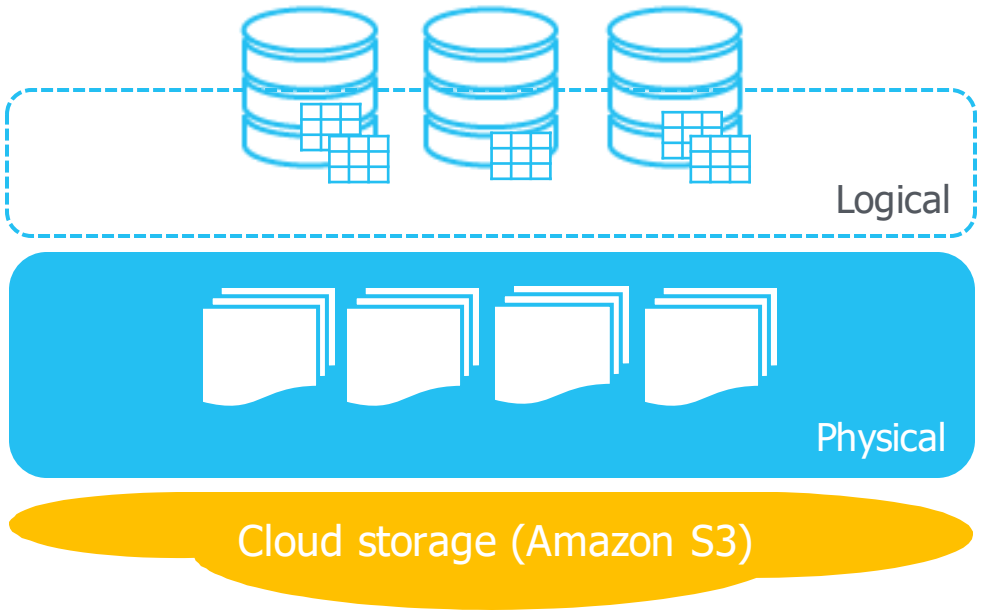
- Improves compile times for queries against commonly used tables

Virtual Warehouse Cache

- Data loaded into warehouses is stored in local SSD storage
- Cache entries are invalidated if underlying data changes

Sensitivity: Internal & Restricted

Data storage



Logical organization

Databases, schemas, tables

Physical storage

Files in Amazon S3

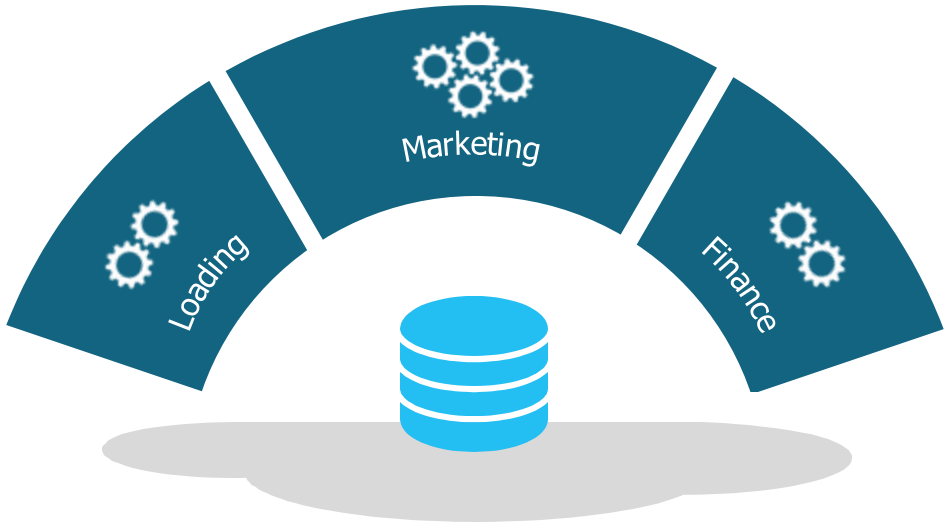
Block of data per file

Proprietary columnar, compressed format

Data never overwritten

Sensitivity: Inter

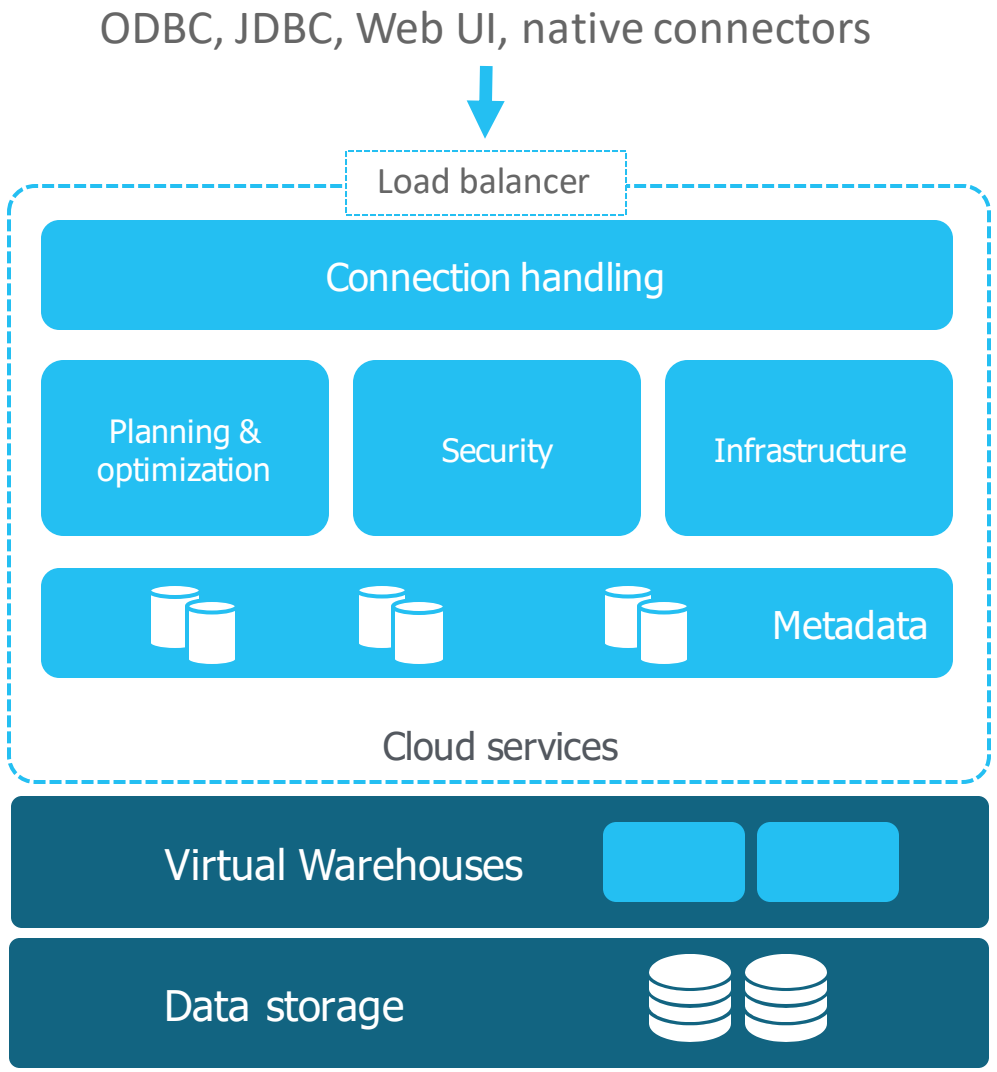
Data processing



“Virtual warehouse”

- MPP compute cluster
- Resizable on the fly, up and down
- Able to access data in any database
- Transparently caches data accessed by queries

Cloud services



Distributed services for database management

Connection management

Metadata storage & management

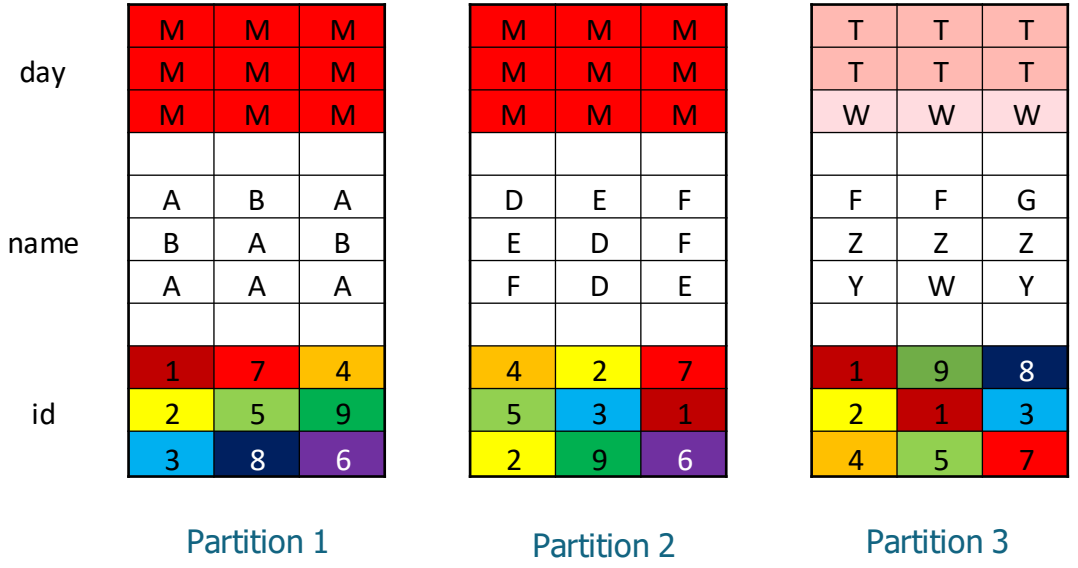
Infrastructure management

Query planning and optimization

Security management

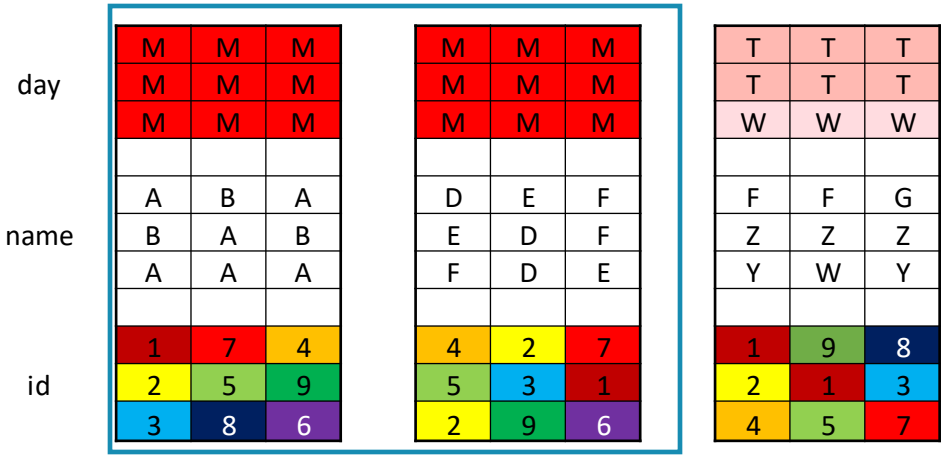
Snowflake data storage

- All database data stored in partitions
- True columnar storage within partitions
- Information about each partition stored in metadata
- Pruning and filtering applied at partition granularity



Explicit clustering optimization

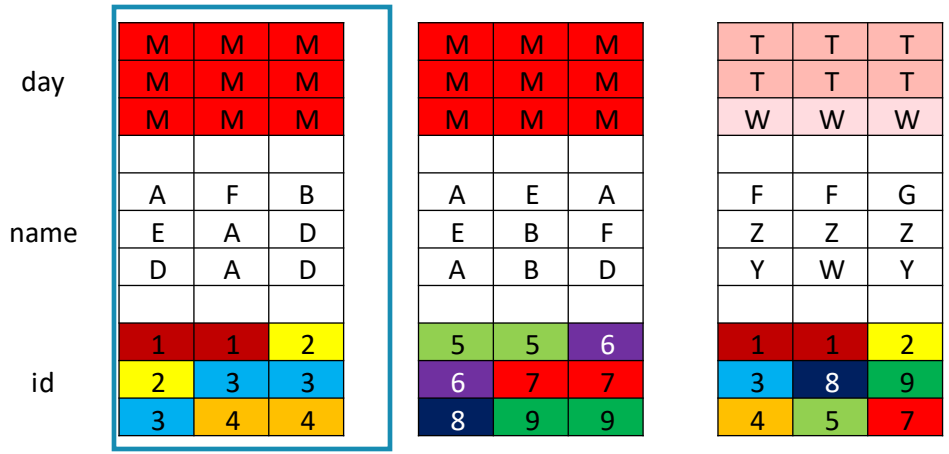
Default clustering



- By default, data clustered within partitions by arrival order at load
- Optimal in cases where data arrives ordered by column used in common query predicates
- Not optimal when columns in query predicates not ordered at load e.g. all files scanned for query:

```
select name where day = 'M' and id = 2
```

Explicit clustering



- Table explicitly clustered on 1+ columns
- Automatic incremental clustering on DML
- Reduces number of partitions scanned:

```
select name where day = 'M' and id = 2
```

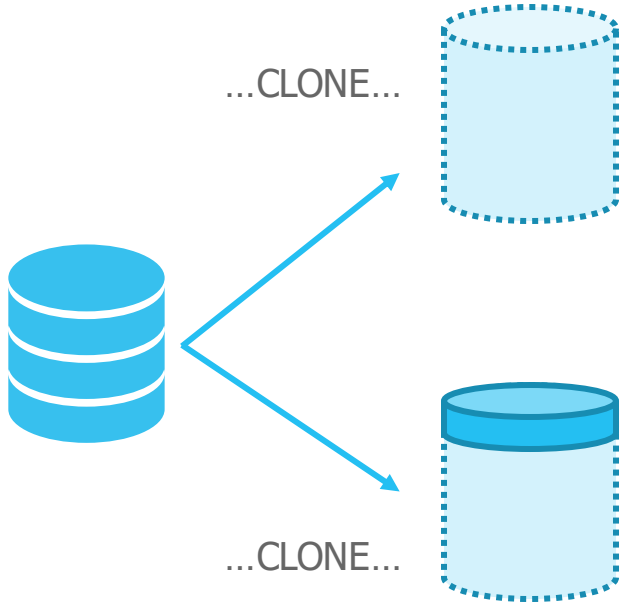
Zero-copy data cloning

Instant data cloning operations

- Databases and tables
- Metadata-only operation
- No data copying required

Modified data stored as new blocks

- Unmodified data stored only once



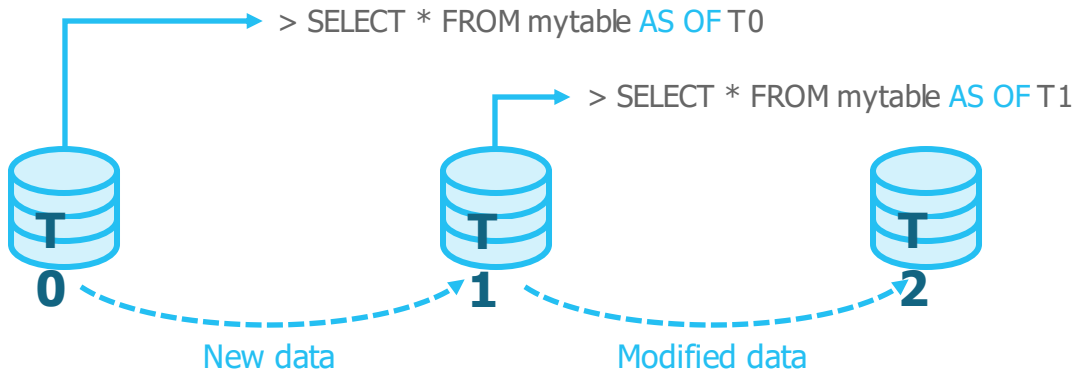
“Time travel” for data

Previous versions of data automatically retained

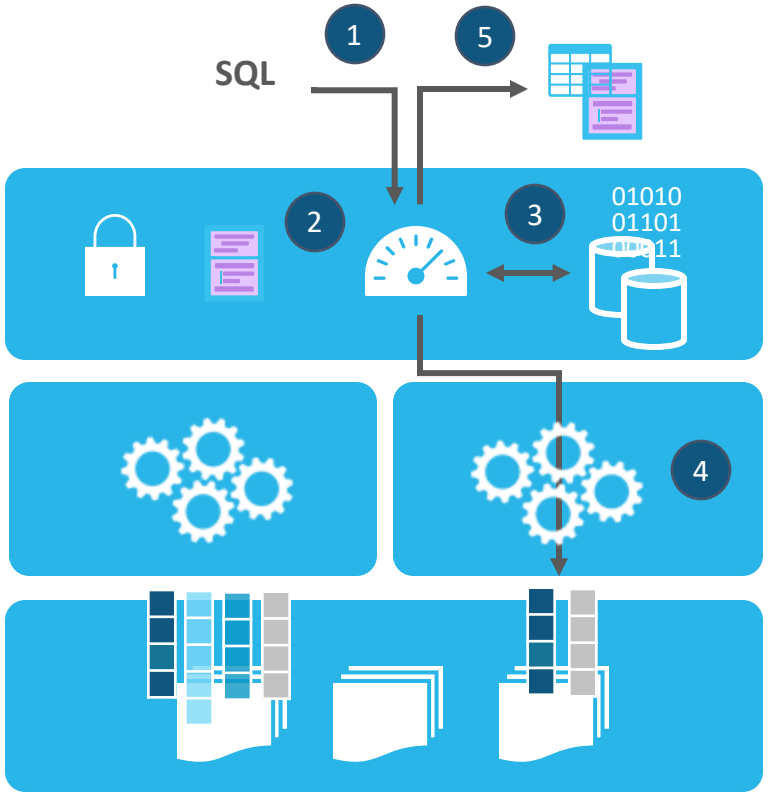
Retention period selected by customer

Accessed via SQL extensions

- AS OF for selection
- CLONE to recreate
- UNDROP recovers from accidental deletion



Query execution



1 **Query received by Snowflake**

Sent via standard ODBC, JDBC, or web UI interfaces

2 **Result cache lookup**

If the query matches an entry in the result cache then the result is returned immediately

3 **Planner and optimizer process query**

Prune and filter, then use metadata to identify exact data to be processed (or retrieved from result cache)

4 **Virtual warehouse processing**

Virtual warehouse scans only needed data from local SSD cache or Amazon S3, processes, and returns to cloud services

5 **Result set return**

Final result processed, stored in cache for future use, and returned to client

Data Loading & Unloading Techniques overview

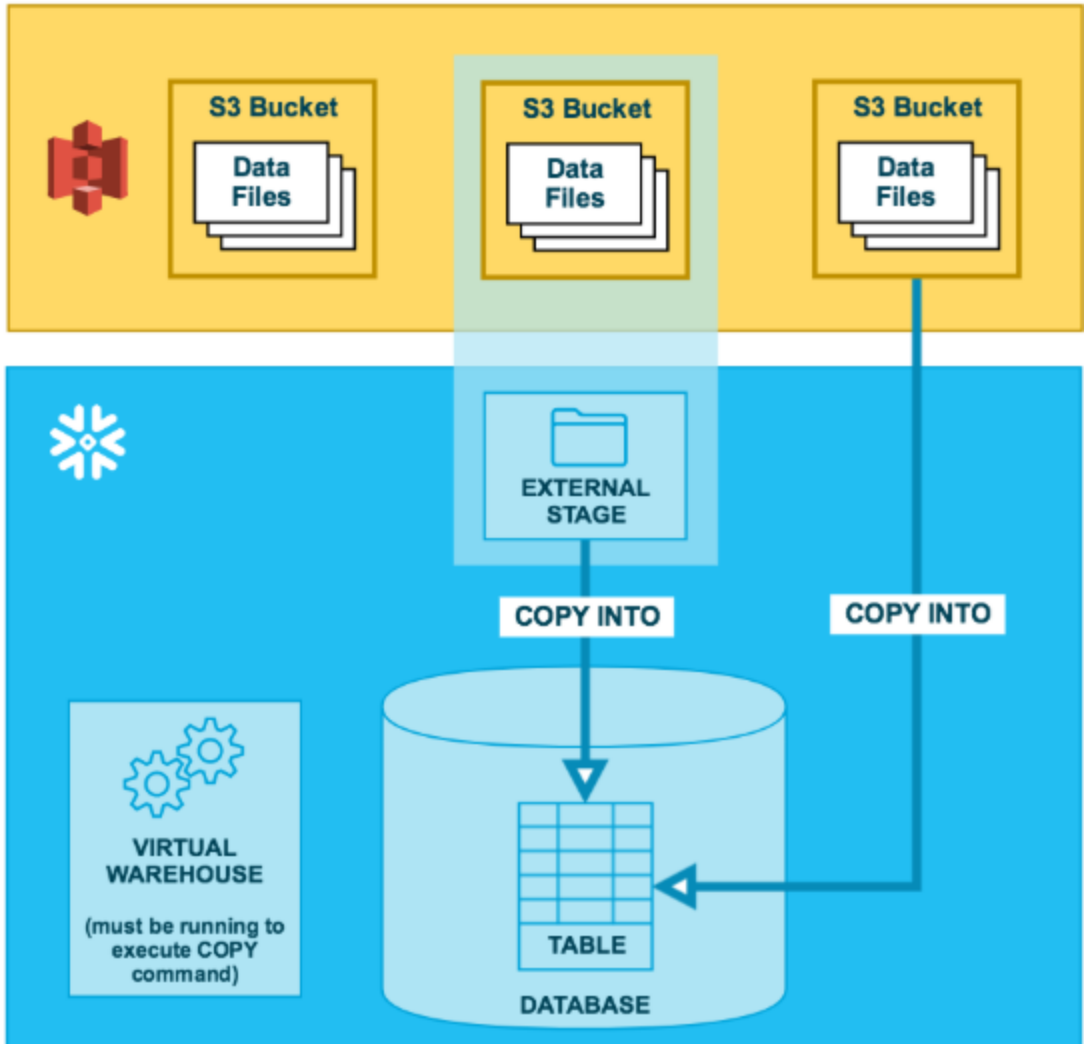
Snowflake supports bulk import (i.e. loading) of data from one or more files into a table in Snowflake databases using the COPY command. Snowflake also supports loading limited amounts of data through the web interface.

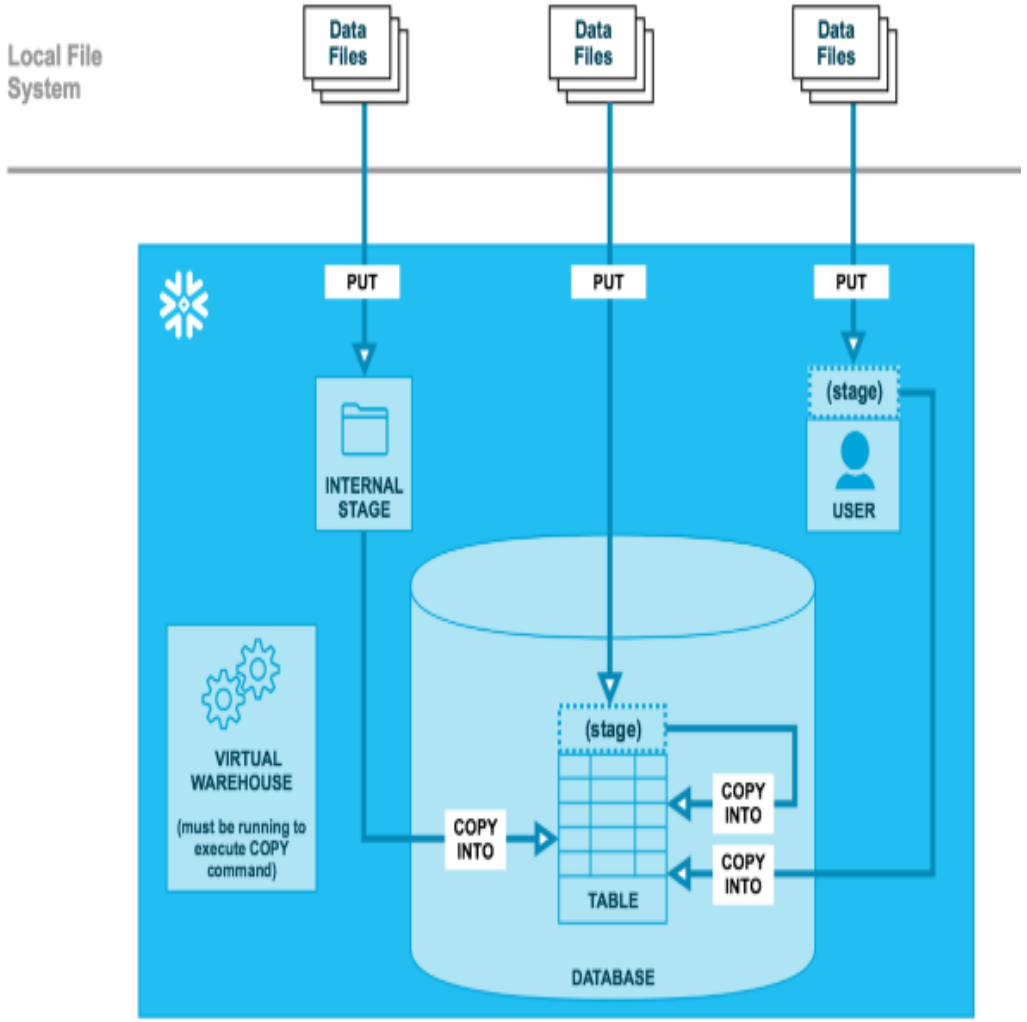
Snowflake supports the following file formats for data loading:

- Any flat, delimited plain text format (comma-separated values, tab-separated values, etc.).
- Semi-structured data in JSON, Avro, ORC, Parquet, or XML format (XML is currently supported as a preview feature).

As data is loaded, Snowflake converts the data into an optimized internal format for efficient storage, maintenance, and retrieval.

Data Loading & Unloading Techniques overview



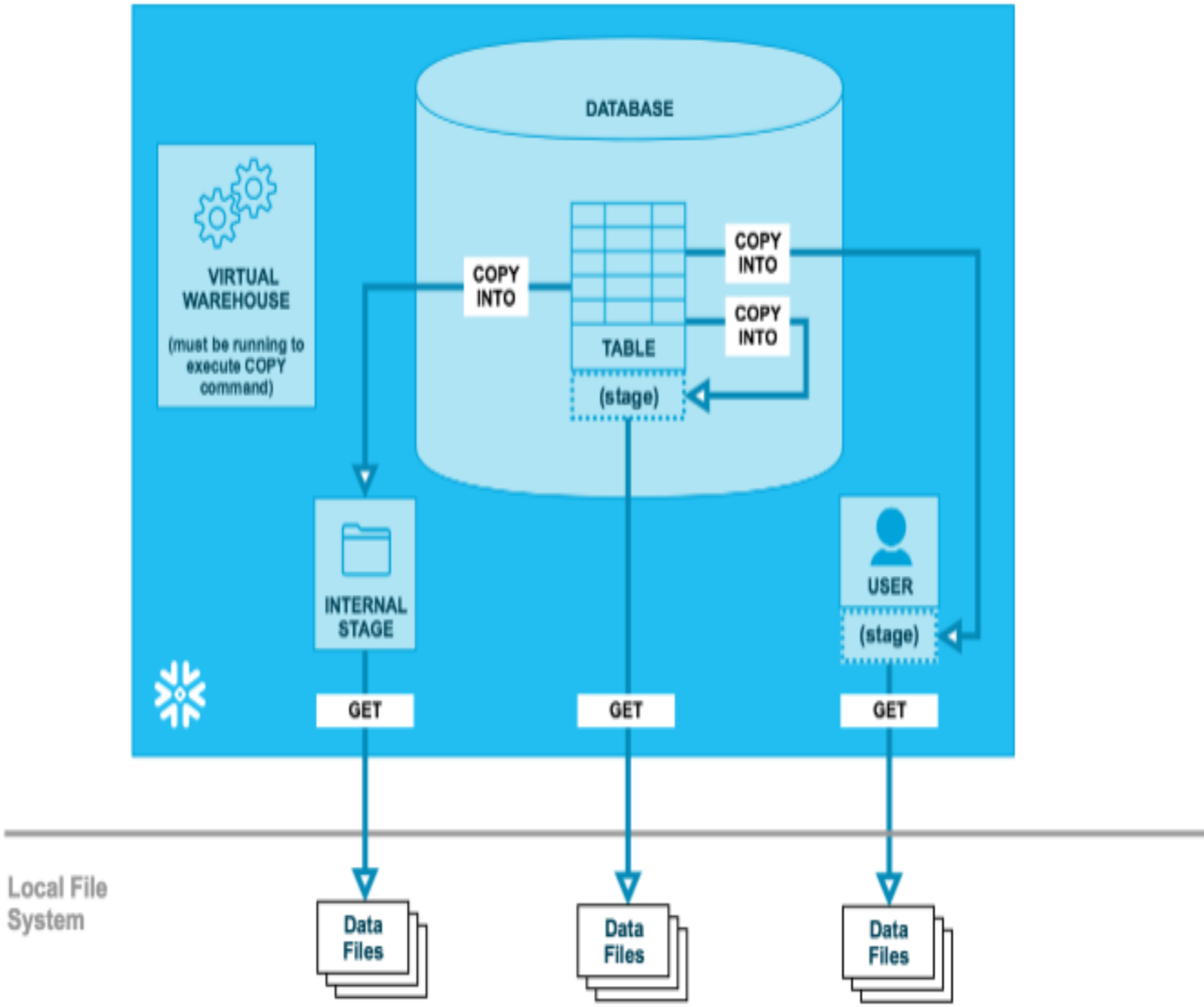


Data Loading & Unloading Techniques overview

Unloading Data
unloading data to a local file system is performed in two, separate steps:

Step 1:
Use the COPY INTO <location> command to copy the data from the Snowflake database table into one or more files in a Snowflake stage. In you the command, you specify the stage (named stage or table/user stage) where the files are written.
Regardless of the stage you use, this step requires a running, current virtual warehouse for the session. The warehouse provides the compute resources to write rows from the table.

Step 2:
Use the GET command to download the data files to your local file system.



Parsing Data in Snowflake

Using the PARSE_JSON Function

This function parses text as a JSON document, producing a VARIANT value. If the input is NULL, the output will also be NULL. If the input string is 'null', it is interpreted as a JSON null value, meaning the result is not a SQL NULL, but a valid VARIANT value containing null (the difference is apparent when printing this VARIANT value).

Semi-structured Data Functions

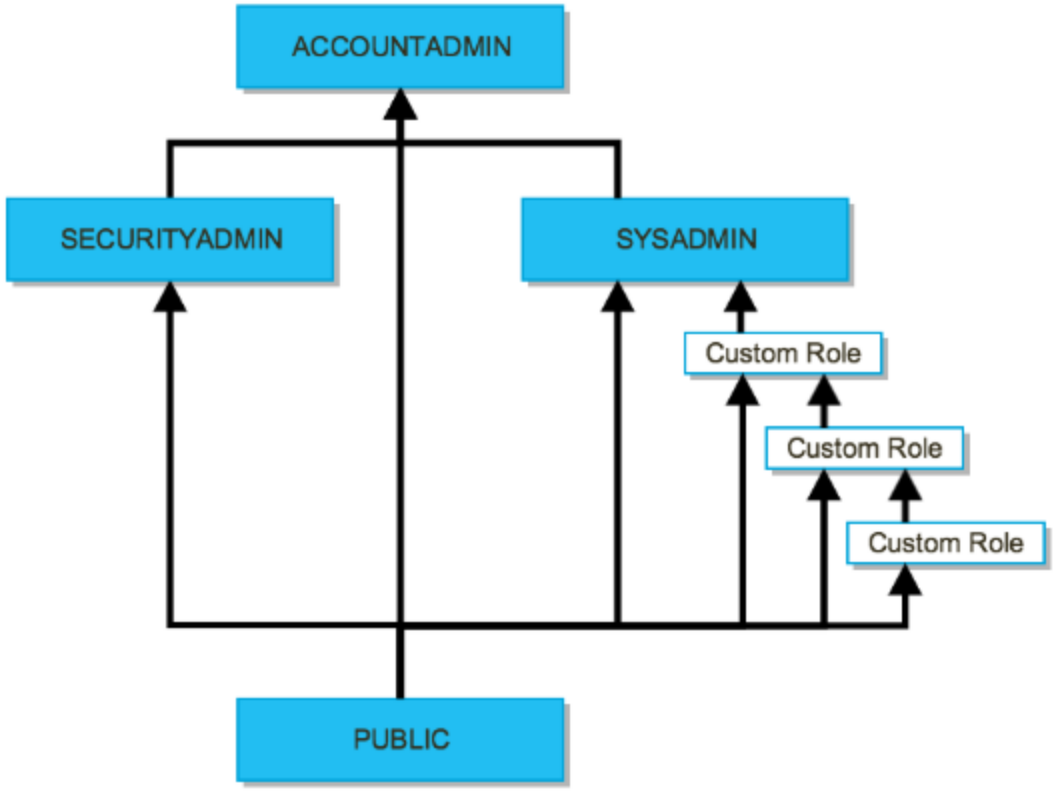
These functions are used with semi-structured data (JSON, Avro, XML), typically stored in Snowflake in VARIANT, OBJECT, or ARRAY columns.

Sub-category	Function	Notes
JSON and XML Parsing	CHECK_JSON	
	CHECK_XML	Preview feature.
	PARSE_JSON	
	PARSE_XML	Preview feature.
	STRIP_NULL_VALUE	
Array/Object Creation and Manipulation	ARRAY_AGG	See also Aggregate Functions.
	ARRAY_APPEND	
	ARRAY_CAT	
	ARRAY_COMPACT	
	ARRAY_CONSTRUCT	
	ARRAY_CONSTRUCT_COMPACT	
	ARRAY_CONTAINS	
	ARRAY_INSERT	
	ARRAY_POSITION	
	ARRAY_PREPEND	

Roles & User Prievelages

Role Hierarchy and Privilege Inheritance

The following diagram illustrates the hierarchy for the system-defined roles along with the recommended structure for additional, user-defined custom roles:



Overview of Snowpipe

Snowpipe is Snowflake's continuous data ingestion service. Snowpipe loads data within minutes after files are added to a stage and submitted for ingestion.

With Snowpipe's serverless compute model, Snowflake manages load capacity, ensuring optimal compute resources to meet demand. In short, Snowpipe provides a "pipeline" for loading fresh data in micro-batches as soon as it's available.

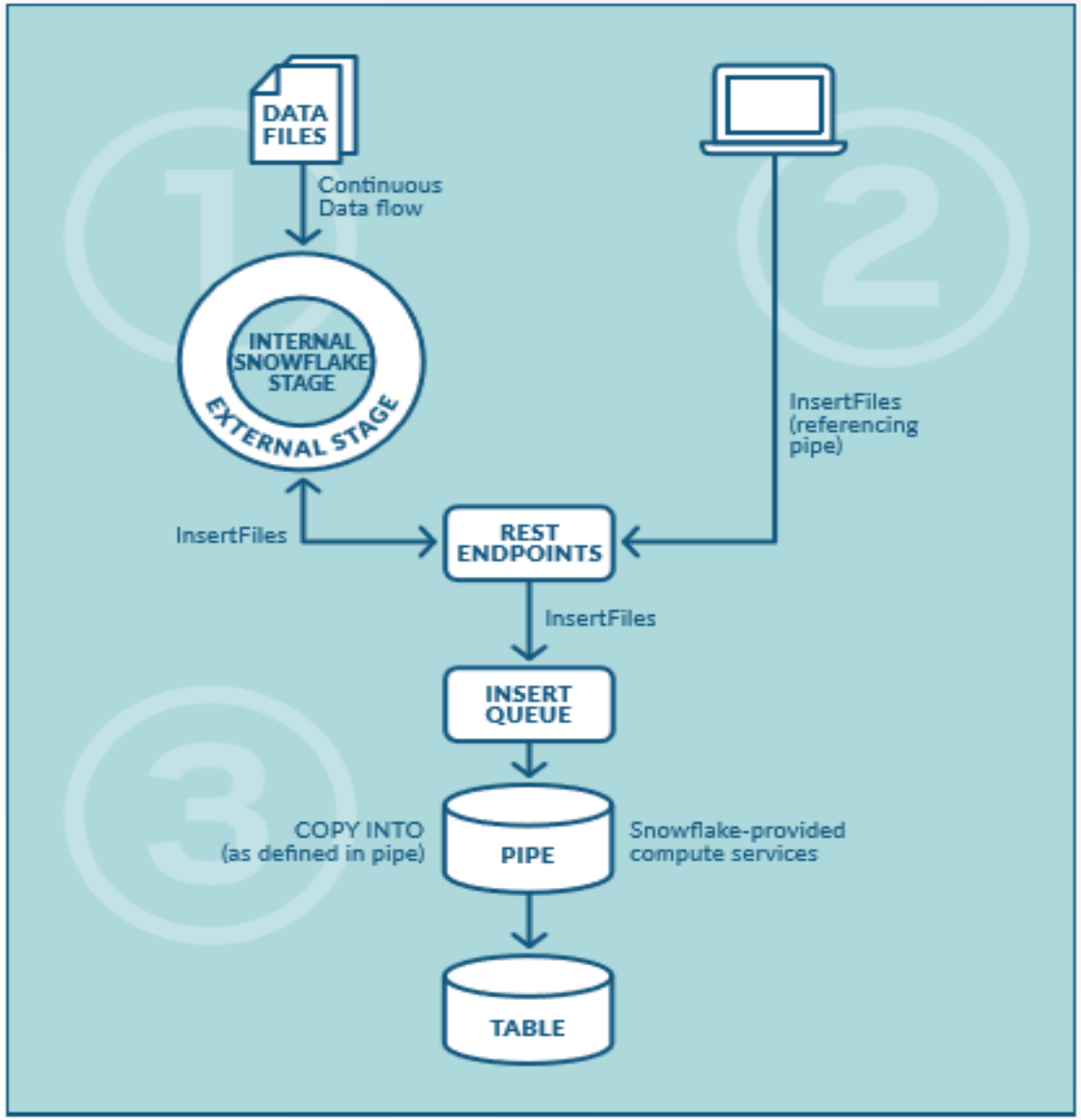
Snowpipe enables loading data from files as soon as they're available in a stage. This means you can load data from files in micro-batches, making it available to users within minutes, rather than manually executing COPY statements on a schedule to load larger batches.

Overview of Snowpipe

The following diagram shows the Snowpipe process flow:

Snowpipe enables loading data from files as soon as they're available in a stage. This means you can load data from files in microbatches, making it available to users within minutes instead of manually executing COPY statements on a schedule to load larger batches.

- A pipe is a named, first-class Snowflake object that contains a COPY statement used by the Snowpipe REST service.
- The COPY statement identifies the source location of the data files (i.e., a named stage) and a target table.
- All data types are supported, including semi-structured data types such as JSON and Avro



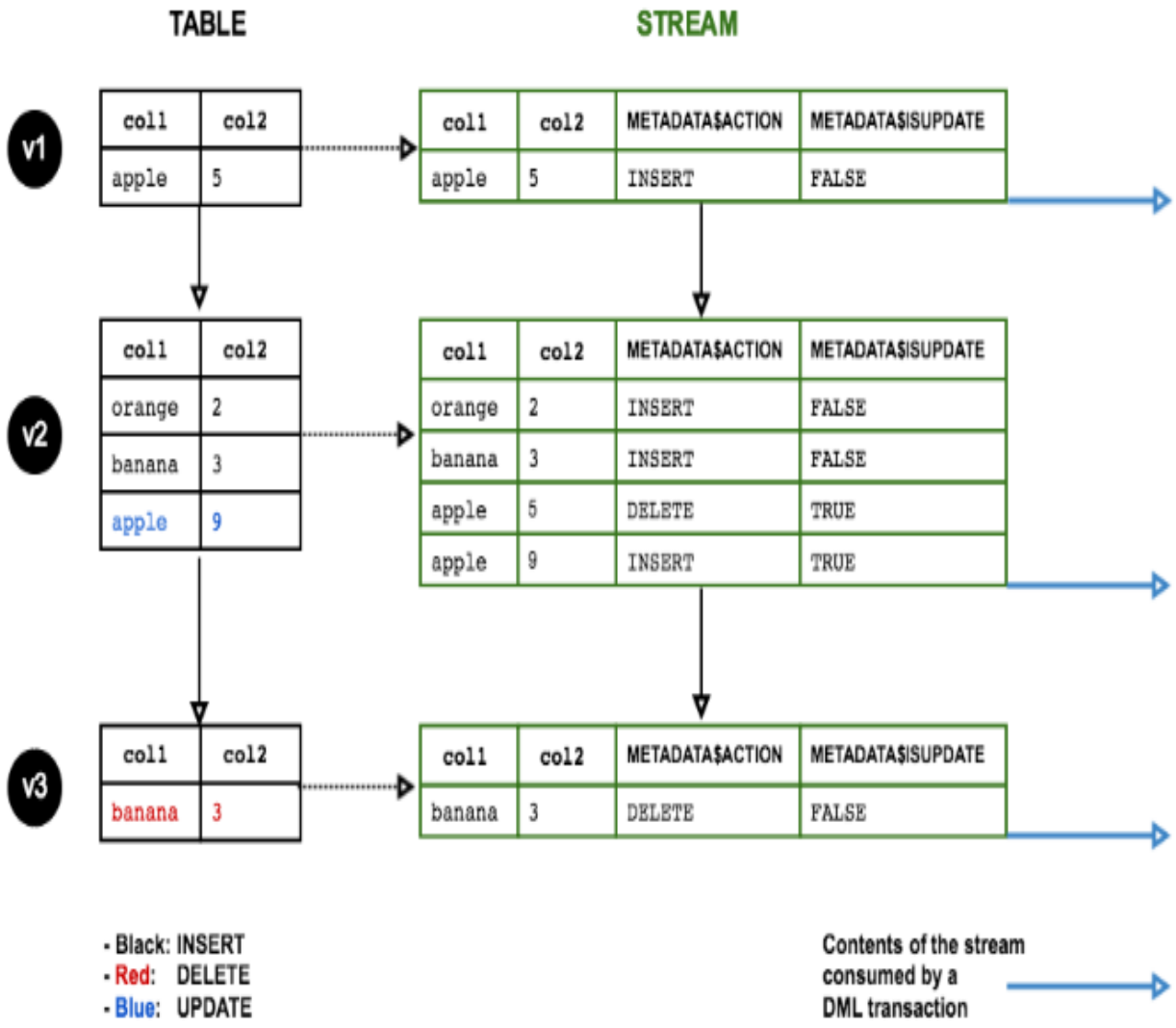
Change Tracking Using Table Streams

A stream object records data manipulation language (DML) changes made to tables, including inserts, updates, and deletes, as well as metadata about each change, so that actions can be taken using the changed data. This process is referred to as change data capture (CDC)

An individual table stream tracks the changes made to rows in a *source table*

A table stream (also referred to as simply a “stream”) makes a “change table” available of what changed, at the row level, between two transactional points of time in a table. This allows querying and consuming a sequence of change records in a transactional fashion.

A stream stores the offset for the source table and returns CDC records by leveraging the versioning history for the source table. When the first stream for a table is created, a pair of hidden columns are added to the source table and begin storing change tracking metadata. These columns consume a small amount of storage.



Stream Columns

A stream stores data in the same shape as the source table (i.e. the same column names and ordering) with the following additional columns:

METADATA$ACTION:

Indicates the DML operation (INSERT, DELETE) recorded.

METADATA$ISUPDATE:

Indicates whether the operation was part of an UPDATE statement. Updates to rows in the source table are represented as a pair of DELETE and INSERT records in the stream with a metadata column METADATA$ISUPDATE values set to TRUE.

Note that streams record the differences between two offsets. If a row is added and then updated in the current offset, the delta change is a new row. The METADATA$ISUPDATE row records a FALSE value.

METADATA$ROW_ID:

Specifies the unique and immutable ID for the row, which can be used to track changes to specific rows over time.

Working with Temporary and Transient Tables

In addition to permanent tables, which are the default for creating tables, Snowflake supports defining tables as either temporary or transient. These types of tables are especially useful for storing data that does not need to be maintained for extended periods of time (i.e. transitory data).

Data Storage Usage for Temporary Tables
For the duration of the existence of a temporary table, the data stored in the table contributes to the overall storage charges that Snowflake bills your account. To prevent any unexpected storage changes, particularly if you create large temporary tables in sessions that you maintain for periods longer than 24 hours, Snowflake recommends explicitly dropping these tables once they are no longer needed.

Transient Tables:
Snowflake supports creating transient tables that persist until explicitly dropped and are available to all users with the appropriate privileges. Transient tables are similar to permanent tables with the key difference that they do not have a Fail-safe period.

Data Storage Usage for Transient Tables
Similar to permanent tables, transient tables contribute to the overall storage charges that Snowflake bills your account; however, because transient tables do not utilize Fail-safe, there are no Fail-safe costs (i.e. the costs associated with maintaining the data required for Fail-safe disaster recovery).

Comparison of Table Types

The following table summarizes the differences between the three table types, particularly with regard to their impact on Time Travel and Fail-safe:

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

Best Practices in Snowflake

Snowflake data warehouse charges for the Storage and Compute separately. Make it as a standard default to suspend cluster idle for 5mins to save cost

Your account will be charged for all the data stored in schemas, tables, and databases created in your Snowflake architecture. This means that you pay for the data storage irrespective of whether it is in Active, Time-travel or Fail-safe State.

Snowflake automatically does the job of clustering on the tables, and this natural clustering process of Snowflake is good enough for most cases and gives good performance even for big tables.

Clustering keys can be useful only for very large tables, re-clustering a table on Snowflake costs additional credits.

The VARIANT data type has a 16 MB (compressed) size limit on the individual rows for Semi-Structured Data. There are data size limitations of Parquet files, it is recommended to split parquet files that are greater than 3GB in size into smaller files of 1GB or lesser for smooth loading. This will ensure that the loading does not timeout. Preparing Delimited Text Files

Subtracting a date from another date has to be replaced with the DATEDIFF function in Snowflake

Other Cloud Product Comparison

	Snowflake Elastic Data Warehouse	Amazon Redshift	Microsoft Azure Data Warehouse	Google BigQuery
✓	Broad performance across of range of relational queries	✓	✓	Low
✓	Concurrent throughput	✓	✓	Low
✓	High performance for JSON scans	Only w/Spectrum	No	✓
✓	Concurrent throughput for JSON	No	No	Moderate
✓	Scalable to handle large concurrency mix (single DW)	✓	✓	No
✓	Scale up, down, or off, quickly without delay	Low	✓	✓ (w/Limits)
✓	Multi-warehouse concurrency against same data	No	No	No

Thank You!

Sensitivity: Internal & Restricted