

Snowflake Reference

From first query to production-grade cloud data warehouse patterns.

Last reviewed 29 Sep 2026 ·

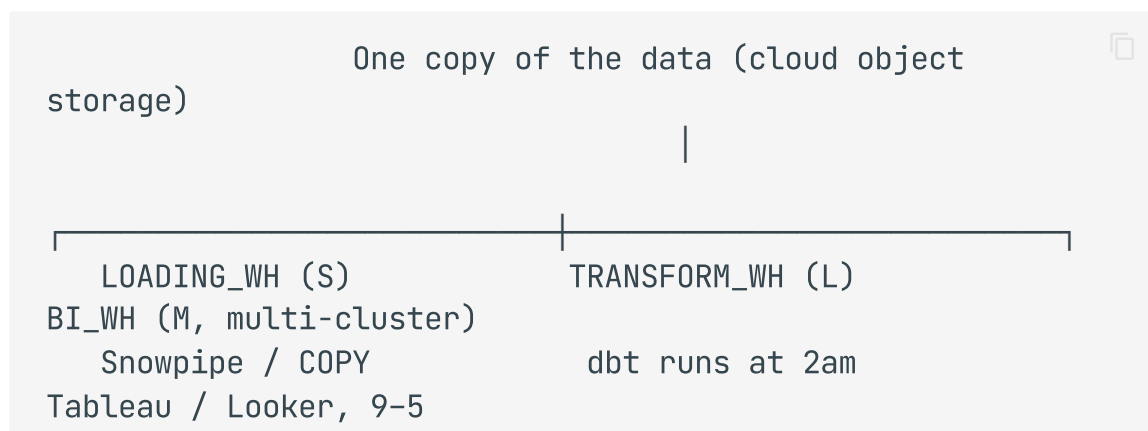
Prerequisites: [SQL](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/sql-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/sql-reference/)

Related: [dbt](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/dbt-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/dbt-reference/) · [Data Modeling](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/data-modeling/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/data-modeling/) · [Terraform](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/) · [BigQuery](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/bigquery-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/bigquery-reference/) · [Amazon Redshift](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/redshift-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/redshift-reference/) · [Glossary](https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/) (https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/)

Overview

Challenge: Traditional databases couple storage and compute on the same servers. Heavy workloads such as month-end reporting slow down every other user, and growing data volumes force larger servers whether or not more compute is needed.

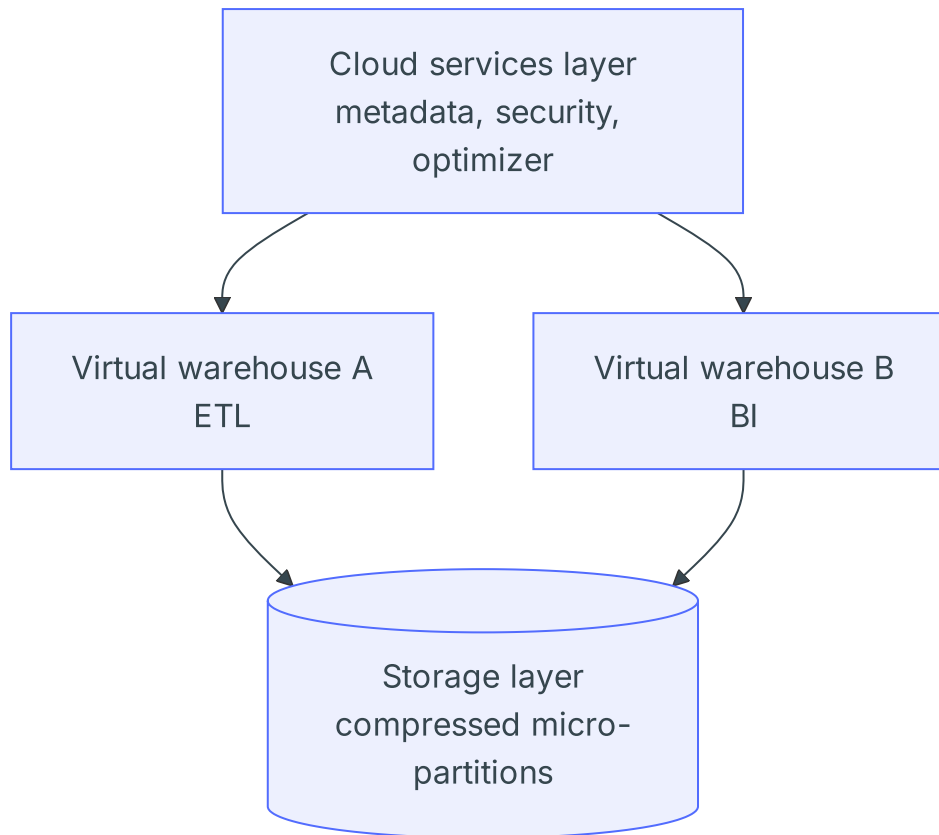
Solution: Snowflake is a cloud data warehouse that keeps data in a single central store and runs queries on independent compute clusters ("virtual warehouses"). Loading, transformation, BI, and data science workloads each receive their own compute, do not contend with one another, and suspend automatically when idle.



pay only while running
scales out when busy

pay only while running

Additional capabilities: standard SQL with useful extensions (`QUALIFY` , `FLATTEN`), native semi-structured data handling, time travel for recovering from mistakes, zero-copy cloning for development environments, and minimal tuning — no indexes or vacuuming.



On this page

Basics - [Architecture](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#architecture) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#architecture) - [Key Concepts](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#key-concepts) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#key-concepts) - [Databases, Schemas & Tables](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#databases-schemas--tables) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#databases-schemas--tables) - [Data Types](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#data-types) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#data-types) - [Querying Data](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#querying-data) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#querying-data)

Intermediate - [Loading Data](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#loading-data) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#loading-data) - [Virtual Warehouses](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#virtual-warehouses) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#virtual-warehouses) - [Semi-Structured Data](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#semi-structured-data) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#semi-structured-data) - [Time Travel](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#time-travel) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#time-travel)

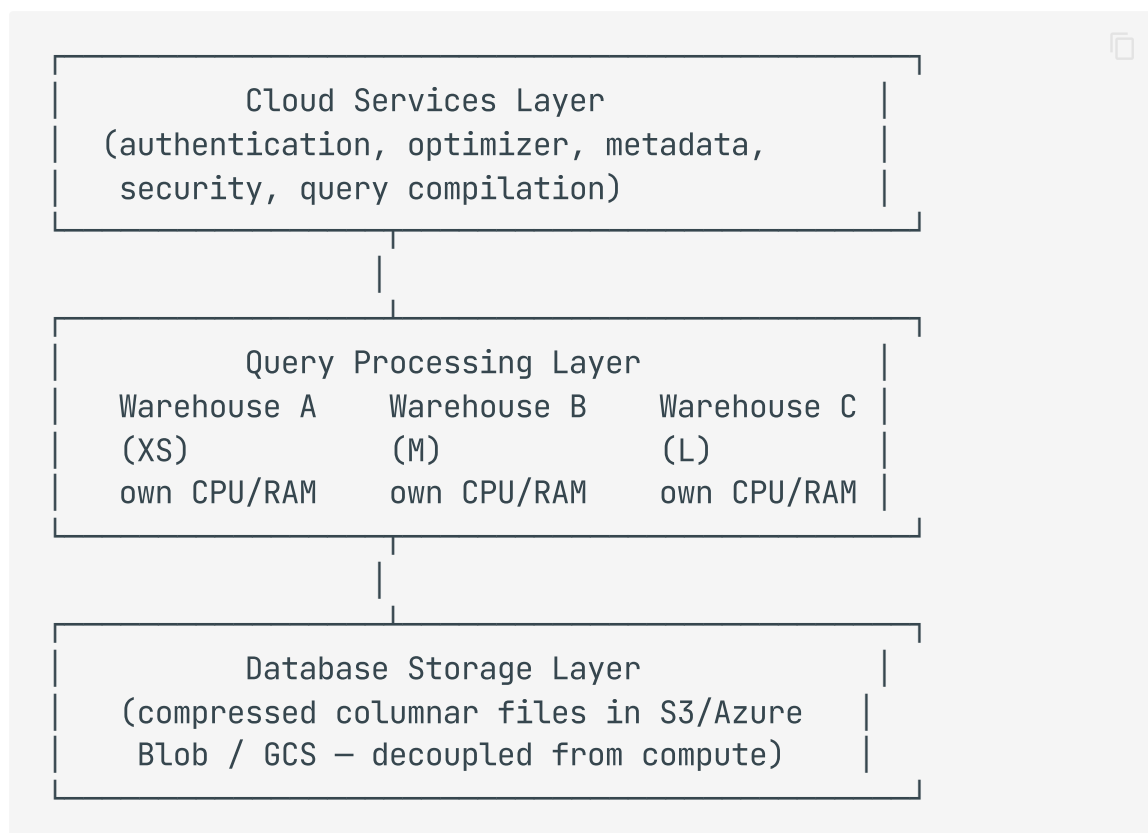
erence/#time-travel) - [Cloning](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#cloning) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#cloning) - [Streams & Tasks](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#streams--tasks) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#streams--tasks)

Advanced - [Query Performance](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#query-performance) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#query-performance) - [Clustering Keys](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#clustering-keys) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#clustering-keys) - [Cost Management](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#cost-management) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#cost-management) - [Access Control & RBAC](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#access-control--rbac) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#access-control--rbac) - [Snowflake-Specific SQL](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#snowflake-specific-sql) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#snowflake-specific-sql)

Reference - [Common Pitfalls](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#common-pitfalls) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#common-pitfalls) - [Cheat Sheet](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#cheat-sheet) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#cheat-sheet) - [Interview Questions](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#interview-questions) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#interview-questions) - [Further Reading](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#further-reading) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/#further-reading)

Architecture

Snowflake separates storage, compute, and cloud services into three independent layers.



Key insight: compute (virtual warehouses) and storage are completely decoupled. - Multiple warehouses can read the same data simultaneously - Warehouses can be paused without losing data - Storage costs and compute costs scale independently

Key Concepts

Concept	What it is
Virtual Warehouse	A cluster of compute nodes (CPU + RAM); runs queries; billed per second when active
Credit	Snowflake's billing unit; 1 credit = 1 hour of 1 XS warehouse
Micro-partition	Immutable ~50–500 MB columnar files; Snowflake's internal storage unit
Metadata Store	Stores min/max values, row counts per micro-partition — enables pruning
Result Cache	Query results cached 24h; identical query returns instantly at zero cost
Time Travel	Access historical data up to 90 days back
Fail-safe	7-day disaster recovery window (not queryable — Snowflake support only)
Stage	Named location for data files: internal (Snowflake-managed) or external (your S3/GCS/ADLS)
Snowpipe	Continuous, serverless ingestion from a stage — triggers on file arrival
Stream	<u>CDC</u> table — tracks inserts/updates/deletes since the last consumption
Task	Scheduled SQL or stored procedure; Snowflake's built-in scheduler
Share	Share live data with another Snowflake account without copying it

Databases, Schemas & Tables

```
-- Database
CREATE DATABASE analytics;
USE DATABASE analytics;
```



```

-- Schema
CREATE SCHEMA analytics.staging;
CREATE SCHEMA analytics.marts;
USE SCHEMA analytics.staging;

-- Fully qualified name: database.schema.table
SELECT * FROM analytics.marts.dim_customer;

-- Table
CREATE TABLE staging.orders (
    order_id    VARCHAR(36)    NOT NULL,
    customer_id INTEGER        NOT NULL,
    amount      NUMBER(12, 2),
    status      VARCHAR(20),
    created_at  TIMESTAMP_NTZ,      -- no timezone
    updated_at  TIMESTAMP_LTZ      -- local timezone
);

-- Create table from query
CREATE TABLE marts.order_summary AS
SELECT DATE(created_at) AS order_date, SUM(amount) AS
revenue
FROM    staging.orders
GROUP BY 1;

-- Transient table – no fail-safe (lower cost; use for
staging/temp data)
CREATE TRANSIENT TABLE staging.raw_orders (...);

-- Temporary table – session-scoped, auto-dropped
CREATE TEMPORARY TABLE temp_work AS SELECT ...;

-- External table – query files directly on S3 without
loading
CREATE EXTERNAL TABLE raw.orders_ext (
    order_id  VARCHAR AS (VALUE:order_id::VARCHAR),
    amount    FLOAT   AS (VALUE:amount::FLOAT)
)
WITH LOCATION = @my_s3_stage/orders/
FILE_FORMAT = (TYPE = PARQUET);

```

Data Types

Category	Type	Notes
Numeric	<code>NUMBER(p,s)</code> , <code>INTEGER</code> , <code>BIGINT</code> , <code>FLOAT</code> , <code>DOUBLE</code>	<code>NUMBER</code> is exact; use for money
String	<code>VARCHAR(n)</code> , <code>STRING</code> , <code>TEXT</code>	All stored as UTF-8; max 16 MB
Date/Time	<code>DATE</code> , <code>TIME</code> , <code>TIMESTAMP_NTZ</code> , <code>TIMESTAMP_LTZ</code> , <code>TIMESTAMP_TZ</code>	NTZ = no timezone, LTZ = local, TZ = with offset
Boolean	<code>BOOLEAN</code>	<code>TRUE</code> , <code>FALSE</code> , <code>NULL</code>
Semi-structured	<code>VARIANT</code> , <code>ARRAY</code> , <code>OBJECT</code>	Store any JSON/XML/Avro
Geospatial	<code>GEOGRAPHY</code> , <code>GEOMETRY</code>	For spatial queries

```
-- VARIANT – stores any JSON value
CREATE TABLE events (
  event_id INTEGER,
  payload  VARIANT    -- can hold any JSON structure
);

INSERT INTO events VALUES (1,
  PARSE_JSON('{\"type\":\"click\",\"page\":\"home\",\"user\":42}'));

-- Query VARIANT fields
SELECT payload:type::STRING      AS event_type,
       payload:user::INTEGER     AS user_id,
       payload:page::STRING      AS page
FROM   events;
```

Querying Data

Snowflake supports standard SQL plus many extensions. All SQL reference patterns apply here.

```
-- Standard SELECT (see sql-reference.md for full SQL coverage)
```

```

SELECT order_id, amount, status
FROM   staging.orders
WHERE  created_at >= DATEADD(day, -7, CURRENT_TIMESTAMP())
ORDER  BY created_at DESC
LIMIT  100;

-- Snowflake-specific: QUALIFY – filter window function
results inline
SELECT name, dept, salary,
       RANK() OVER (PARTITION BY dept ORDER BY salary DESC) AS
rnk
FROM   employees
QUALIFY rnk <= 3;    -- top 3 per department without a
subquery

-- PIVOT
SELECT *
FROM   monthly_revenue
PIVOT (SUM(revenue) FOR month IN ('Jan', 'Feb', 'Mar')) AS
p;

-- UNPIVOT
SELECT product, month, revenue
FROM   quarterly_data
UNPIVOT (revenue FOR month IN (jan, feb, mar));

-- FLATTEN – expand ARRAY or OBJECT into rows
SELECT f.value::STRING AS tag
FROM   products,
       LATERAL FLATTEN(INPUT => tags) f;

-- SAMPLE – random sample
SELECT * FROM orders SAMPLE (10);           -- 10% random
SELECT * FROM orders SAMPLE (1000 ROWS);    -- exactly
1000 rows

```

Loading Data

Stages

```

-- Internal stage — Snowflake-managed storage
CREATE STAGE my_internal_stage
    FILE_FORMAT = (TYPE = CSV FIELD_OPTIONALLY_ENCLOSED_BY
= ''' SKIP_HEADER = 1);

-- External stage — your S3 bucket, authenticated via a
storage integration
-- (an IAM role Snowflake assumes — no access keys stored
in Snowflake)
CREATE STORAGE INTEGRATION s3_int
    TYPE = EXTERNAL_STAGE
    STORAGE_PROVIDER = 'S3'
    ENABLED = TRUE
    STORAGE_AWS_ROLE_ARN =
'arn:aws:iam::123456789012:role/snowflake-s3-read'
    STORAGE_ALLOWED_LOCATIONS = ('s3://my-bucket/data/');
-- DESC INTEGRATION s3_int; → copy the IAM user ARN +
external ID into the role's trust policy

CREATE STAGE my_s3_stage
    URL = 's3://my-bucket/data/'
    STORAGE_INTEGRATION = s3_int
    FILE_FORMAT = (TYPE = PARQUET);

-- List files in a stage
LIST @my_s3_stage;

-- Upload a local file to internal stage (SnowSQL CLI)
PUT file:///local/path/orders.csv @my_internal_stage;

```

COPY INTO — batch load

```

-- Load from stage into table
COPY INTO staging.orders
FROM @my_s3_stage/orders/2024/03/
FILE_FORMAT = (TYPE = PARQUET)
MATCH_BY_COLUMN_NAME = CASE_INSENSITIVE
ON_ERROR = 'SKIP_FILE';    -- ABORT_STATEMENT | CONTINUE |
SKIP_FILE

-- Load CSV with options

```

```

COPY INTO staging.customers
FROM @my_s3_stage/customers/
FILE_FORMAT = (
    TYPE = CSV
    SKIP_HEADER = 1
    FIELD_OPTIONALLY_ENCLOSED_BY = ''
    DATE_FORMAT = 'YYYY-MM-DD'
    TIMESTAMP_FORMAT = 'YYYY-MM-DD HH24:MI:SS'
)
PURGE = TRUE;      -- delete files from stage after
                    successful load

-- Validate before loading (dry run)
COPY INTO staging.orders
FROM @my_s3_stage/orders/
VALIDATION_MODE = 'RETURN_ERRORS';

-- Check load history
SELECT *
FROM   information_schema.load_history
WHERE  table_name = 'ORDERS'
ORDER  BY last_load_time DESC
LIMIT 10;

```

Snowpipe — continuous ingestion

```

-- Create a pipe pointing to a stage and target table
CREATE PIPE orders_pipe
    AUTO_INGEST = TRUE  -- triggered by S3 event
notifications
AS
COPY INTO staging.orders
FROM @my_s3_stage/orders/
FILE_FORMAT = (TYPE = PARQUET);

-- Check pipe status
SELECT SYSTEM$PIPE_STATUS('orders_pipe');

-- Manually trigger for specific files
ALTER PIPE orders_pipe REFRESH
    PREFIX = 'orders/2024/03/15/';

```

Virtual Warehouses

```
-- Create a warehouse
CREATE WAREHOUSE analytics_wh
    WAREHOUSE_SIZE = 'MEDIUM'           -- XS, S, M, L, XL,
2XL, 3XL, 4XL, 5XL, 6XL
    AUTO_SUSPEND = 120                  -- auto-suspend after
120 seconds of inactivity
    AUTO_RESUME = TRUE                  -- auto-resume when a
query arrives
    MAX_CLUSTER_COUNT = 3               -- multi-cluster:
scale out to 3 clusters
    MIN_CLUSTER_COUNT = 1               -- scale back to 1
when load drops
    SCALING_POLICY = 'STANDARD';        -- STANDARD | ECONOMY

-- Use a warehouse
USE WAREHOUSE analytics_wh;

-- Resize on the fly (takes effect immediately for new
queries)
ALTER WAREHOUSE analytics_wh SET WAREHOUSE_SIZE = 'LARGE';

-- Suspend / resume manually
ALTER WAREHOUSE analytics_wh SUSPEND;
ALTER WAREHOUSE analytics_wh RESUME;

-- Warehouse sizes and credits/hour
-- XS=1  S=2  M=4  L=8  XL=16  2XL=32  3XL=64  4XL=128
```

Multi-cluster warehouses

Multi-cluster allows a warehouse to spin up additional clusters when queued queries exceed a threshold. Solves concurrency bottlenecks without resizing.

```
CREATE WAREHOUSE reporting_wh
    WAREHOUSE_SIZE = 'MEDIUM'
    MAX_CLUSTER_COUNT = 5               -- up to 5 parallel clusters
    MIN_CLUSTER_COUNT = 1               -- always at least 1 running
    SCALING_POLICY = 'ECONOMY';        -- wait longer before
```

```
spinning up extra cluster
-- STANDARD scales
aggressively
```

Semi-Structured Data

Snowflake's `VARIANT` type stores any JSON, Avro, Parquet, or XML natively.

```
-- Dot notation to traverse JSON
SELECT
    payload:user_id::INTEGER      AS user_id,
    payload:event.type::STRING    AS event_type,
    payload:items[0].sku::STRING  AS first_sku, --
array index
    payload:metadata:source::STRING AS source
FROM events;

-- FLATTEN – turn array into rows
SELECT
    order_id,
    f.value:sku::STRING    AS sku,
    f.value:quantity::INT  AS qty
FROM orders,
    LATERAL FLATTEN(INPUT => line_items) f;

-- Check if a key exists
SELECT * FROM events WHERE payload:user_id IS NOT NULL;

-- Parse JSON string into VARIANT
SELECT PARSE_JSON('{ "key": "value" }'):key::STRING;

-- Convert VARIANT back to string
SELECT payload::STRING FROM events;

-- Build a VARIANT from columns
SELECT OBJECT_CONSTRUCT('id', id, 'name', name) AS json_row
FROM customers;
```

Time Travel

Query, clone, or restore data from any point in the past (up to 90 days for Enterprise edition).

```
-- Query table as it was 1 hour ago
SELECT * FROM orders AT (OFFSET => -3600);    -- seconds

-- Query at a specific timestamp
SELECT * FROM orders AT (TIMESTAMP => '2024-03-14
09:00:00'::TIMESTAMP);

-- Query before a specific statement executed
SELECT * FROM orders BEFORE (STATEMENT => '01b48fd2-0001-
b7b8-...');

-- Restore a dropped table
UNDROP TABLE staging.orders;

-- Restore a table to a prior state (create from time
travel, then swap)
CREATE OR REPLACE TABLE orders_restored
    CLONE orders AT (TIMESTAMP => '2024-03-14
00:00:00'::TIMESTAMP);

-- Set retention period (default 1 day for Standard, up to
90 for Enterprise)
ALTER TABLE orders SET DATA_RETENTION_TIME_IN_DAYS = 7;
```

Cloning

Zero-copy cloning creates an independent copy of a table, schema, or database instantly — no data is physically copied until one side modifies it.

```
-- Clone a table (instant, zero storage cost initially)
CREATE TABLE orders_backup CLONE orders;

-- Clone a schema (clones all objects within it)
CREATE SCHEMA staging_backup CLONE staging;
```

```
-- Clone a database
CREATE DATABASE analytics_dev CLONE analytics;

-- Clone at a point in time (useful for dev/test
environments)
CREATE DATABASE analytics_dev CLONE analytics
    AT (TIMESTAMP => '2024-03-01 00:00:00'::TIMESTAMP);
```

Cloning is the standard way to create dev/test environments in Snowflake. Clone production, work in the clone, discard when done — no cost until data diverges.

Streams & Tasks

Streams — CDC on tables

A stream tracks all DML changes (INSERT, UPDATE, DELETE) to a table since it was last consumed.

```
-- Create a stream on a source table
CREATE STREAM orders_stream ON TABLE staging.raw_orders;

-- Query the stream — see what changed
SELECT *,
    METADATA$ACTION,          -- INSERT or DELETE
    METADATA$ISUPDATE,        -- true if part of an UPDATE
    (appears as DELETE + INSERT)
    METADATA$ROW_ID
FROM orders_stream;

-- Consume the stream in a merge (marks it as consumed)
MERGE INTO marts.orders AS target
USING (
    SELECT * FROM orders_stream WHERE METADATA$ACTION =
    'INSERT'
) AS source ON target.order_id = source.order_id
WHEN MATCHED THEN UPDATE SET target.amount = source.amount,
    target.status = source.status
WHEN NOT MATCHED THEN INSERT VALUES (source.order_id,
    source.amount, source.status);
-- After this executes, orders_stream resets — only shows
new changes going forward
```

```
-- Check if a stream has unconsumed data
SELECT SYSTEM$STREAM_HAS_DATA('orders_stream');
```

Tasks — scheduled SQL

```
-- Create a task that runs every hour
CREATE TASK refresh_summary
  WAREHOUSE = analytics_wh
  SCHEDULE = 'USING CRON 0 * * * * UTC'
AS
INSERT INTO marts.hourly_summary
SELECT DATE_TRUNC('hour', created_at) AS hour,
       COUNT(*) AS orders, SUM(amount) AS revenue
FROM   staging.orders
WHERE  created_at >= DATEADD(hour, -1, CURRENT_TIMESTAMP())
GROUP BY 1;

-- Tasks start suspended – resume to activate
ALTER TASK refresh_summary RESUME;

-- Task chain – run task B after task A completes
CREATE TASK transform_task
  AFTER extract_task      -- no SCHEDULE needed,
  triggered by parent
AS
  CALL transform_proc();

-- View task history
SELECT *
FROM   TABLE(information_schema.task_history())
ORDER BY scheduled_time DESC
LIMIT 20;
```

Query Performance

Query Profile

The most important performance tool. Open any query result → Query Profile to see: - Execution time per node - Bytes scanned vs pruned - Spill to disk (means warehouse is too small for the query) - The most expensive step

Micro-partition pruning

Snowflake stores metadata (min/max values, row count) for each micro-partition. Filters on those columns skip entire partitions without reading them.

```
-- This can skip most partitions if created_at has good min/max range
SELECT * FROM orders WHERE created_at >= '2024-03-01';

-- SHOW TABLES includes partition stats
SHOW TABLES LIKE 'orders';

-- Check how well a query prunes
SELECT * FROM orders WHERE order_id = 'abc123';
-- Check Query Profile: "Partitions scanned" vs "Partitions total"
-- Good: 1/10000 partitions scanned
-- Bad: 10000/10000 partitions scanned (full scan – consider clustering)
```

Result cache

```
-- Identical query returns instantly from cache (0 credits used)
SELECT COUNT(*) FROM orders;
-- Run again – "result reused" in Query Profile

-- Disable result cache for benchmarking
ALTER SESSION SET USE_CACHED_RESULT = FALSE;

-- Cache lives in the cloud services layer – reused across users and warehouses
-- when the query text is identical and the role can access the tables
```

```
-- Invalidated when underlying data changes (or after 24h without reuse)
```

Clustering Keys

When queries consistently filter on a high-cardinality column but partition pruning is poor, define a clustering key. Snowflake will physically reorganize micro-partitions so filtered ranges are co-located.

```
-- Add a clustering key
ALTER TABLE orders CLUSTER BY (DATE(created_at));

-- Multi-column clustering
ALTER TABLE events CLUSTER BY (user_id, DATE(event_time));

-- Check clustering depth (lower = better; >6 means recluster)
SELECT SYSTEM$CLUSTERING_INFORMATION('orders',
'(DATE(created_at))');

-- Reclustering is automatic (Automatic Clustering service, billed in credits)
-- Pause / resume it per table to control cost
ALTER TABLE orders SUSPEND RECLUSTER;
ALTER TABLE orders RESUME RECLUSTER;

-- Drop a clustering key
ALTER TABLE orders DROP CLUSTERING KEY;
```

Use clustering only when: - The table is very large (>1 TB) - Queries consistently filter on the same column(s) - Query Profile shows poor partition pruning

Cost Management

```
-- Credit usage by warehouse (last 30 days)
SELECT warehouse_name,
       SUM(credits_used) AS total_credits,
       ROUND(SUM(credits_used) * 3.0, 2) AS estimated_usd
```

```

-- illustrative multiplier; check
docs.snowflake.com/en/guides-overview-cost for current
rates
FROM    snowflake.account_usage.warehouse_metering_history
WHERE    start_time >= DATEADD(day, -30, CURRENT_TIMESTAMP())
GROUP BY 1
ORDER BY 2 DESC;

-- Query cost – bytes scanned and execution time
SELECT query_id, query_text, warehouse_name,
       execution_time / 1000 AS seconds,
       bytes_scanned / 1e9 AS gb_scanned,
       credits_used_cloud_services
FROM    snowflake.account_usage.query_history
WHERE    start_time >= DATEADD(day, -1, CURRENT_TIMESTAMP())
ORDER BY credits_used_cloud_services DESC
LIMIT 20;

-- Storage costs
SELECT TABLE_SCHEMA, TABLE_NAME,
       ROUND(ACTIVE_BYTES / 1e9, 2) AS active_gb,
       ROUND(TIME_TRAVEL_BYTES / 1e9, 2) AS
time_travel_gb,
       ROUND(FAILSAFE_BYTES / 1e9, 2) AS failsafe_gb
FROM    snowflake.account_usage.table_storage_metrics
ORDER BY active_gb DESC
LIMIT 20;

```

Cost reduction patterns

```

-- 1. Ensure warehouses auto-suspend
ALTER WAREHOUSE my_wh SET AUTO_SUSPEND = 60; -- 60 second
idle timeout

-- 2. Use transient tables for staging (no fail-safe cost)
CREATE TRANSIENT TABLE staging.raw_orders (...);

-- 3. Reduce time travel on large tables that don't need it
ALTER TABLE staging.raw_orders SET
DATA_RETENTION_TIME_IN_DAYS = 0;

```

```
-- 4. Query result cache is free – don't disable it in
production

-- 5. Use COPY INTO over INSERT for bulk loads (more
efficient)

-- 6. Right-size warehouses – profile queries before
scaling up
```

Access Control & RBAC

Snowflake uses role-based access control (RBAC). Everything is granted to a role, and users are assigned roles.

```
-- Create roles
CREATE ROLE analyst;
CREATE ROLE data_engineer;
CREATE ROLE loader;

-- Grant privileges to roles
GRANT USAGE ON DATABASE analytics TO ROLE analyst;
GRANT USAGE ON SCHEMA analytics.marts TO ROLE analyst;
GRANT SELECT ON ALL TABLES IN SCHEMA analytics.marts TO
ROLE analyst;
GRANT SELECT ON FUTURE TABLES IN SCHEMA analytics.marts TO
ROLE analyst;

GRANT ALL ON SCHEMA analytics.staging TO ROLE
data_engineer;
GRANT USAGE ON WAREHOUSE analytics_wh TO ROLE analyst;
GRANT USAGE ON WAREHOUSE transform_wh TO ROLE
data_engineer;

-- Create user and assign role
CREATE USER alice
  PASSWORD = 'strong_password'
  DEFAULT_ROLE = analyst
  DEFAULT_WAREHOUSE = analytics_wh;

GRANT ROLE analyst TO USER alice;
```

```

-- Role hierarchy – grant role to another role
GRANT ROLE analyst TO ROLE data_engineer; -- engineers
inherit analyst privileges

-- Show grants
SHOW GRANTS TO ROLE analyst;
SHOW GRANTS TO USER alice;
SHOW GRANTS ON TABLE orders;

```

Snowflake-Specific SQL

```

-- QUALIFY – filter window results without a subquery
SELECT *, RANK() OVER (PARTITION BY dept ORDER BY salary
DESC) AS rnk
FROM employees
QUALIFY rnk = 1; -- highest earner per department

-- IFF – shorthand CASE WHEN for single condition
SELECT IFF(amount > 1000, 'large', 'small') AS order_size
FROM orders;

-- ZEROIFNULL / NULLIFZERO
SELECT ZEROIFNULL(revenue) FROM report; -- NULL → 0
SELECT NULLIFZERO(quantity) FROM items; -- 0 → NULL

-- DATE_TRUNC
SELECT DATE_TRUNC('month', created_at) AS month, COUNT(*)
FROM orders GROUP BY 1;
SELECT DATE_TRUNC('week', event_time) AS week, COUNT(*)
FROM events GROUP BY 1;

-- DATEADD / DATEDIFF
SELECT DATEADD('day', -7, CURRENT_DATE());
SELECT DATEDIFF('day', hire_date, CURRENT_DATE()) AS
tenure_days FROM employees;

-- TO_CHAR – format a date as string
SELECT TO_CHAR(created_at, 'YYYY-MM-DD HH24:MI:SS') FROM
orders;

```

```

-- TRY_CAST — safe cast, returns NULL on failure
SELECT TRY_CAST(raw_amount AS NUMBER(12,2)) FROM raw_data;

-- GENERATOR — generate a sequence of rows (useful for date spine)
SELECT DATEADD(day, SEQ4(), '2024-01-01'::DATE) AS cal_date
FROM TABLE(GENERATOR(ROWCOUNT => 365));

-- LISTAGG — concatenate values within a group
SELECT dept, LISTAGG(name, ', ') WITHIN GROUP (ORDER BY
name) AS employees
FROM employees
GROUP BY dept;

-- ARRAY_AGG and OBJECT_AGG
SELECT dept,
      ARRAY_AGG(name) AS name_array,
      OBJECT_AGG(name, salary) AS name_salary_map
FROM employees
GROUP BY dept;

```

Common Pitfalls

Pitfall	Symptom	Fix
Warehouses with long (or no) <code>AUTO_SUSPEND</code>	Credits burn all night with nothing running	<code>AUTO_SUSPEND = 60</code> for most warehouses; set resource monitors with credit quotas
One big shared warehouse for everything	ETL and dashboards slow each other down; you can't attribute cost	Separate warehouses per workload (load / transform / BI / ad hoc)
Scaling <i>up</i> to fix concurrency (or <i>out</i> to fix a slow query)	Cost rises but the problem doesn't go away	Slow single query → larger size; many queued queries → multi-cluster
Wrapping filtered columns in functions (<code>WHERE TO_DATE(ts) = ...</code>)	Poor pruning — scans every micro-partition	Filter on the raw column with a range; cluster on the expression if you must

Pitfall	Symptom	Fix
Clustering keys on small or rarely filtered tables	Automatic Clustering credits with no speed-up	Only for multi-TB tables with consistent filters and poor pruning in the Query Profile
<code>TIMESTAMP_LTZ</code> / <code>NTZ</code> mixed carelessly	Values shift by hours depending on the session timezone	Store UTC in <code>TIMESTAMP_NTZ</code> (or use <code>TIMESTAMP_TZ</code>), and set the account timezone explicitly
Access keys in stage definitions	Long-lived credentials inside Snowflake	Storage integrations (<u>IAM</u> role assumption)
Password-only service users	Blocked or flagged as Snowflake enforces <u>MFA</u> and phases out single-factor passwords	Key-pair auth (or OAuth) for service/pipeline users
Granting to users instead of roles; using <code>ACCOUNTADMIN</code> day to day	Permission sprawl, risky mistakes	<u>RBAC</u> hierarchy: access roles → functional roles → users; <code>SYSADMIN</code> for objects, <code>ACCOUNTADMIN</code> locked down
Forgetting <code>FUTURE</code> grants	New tables created by dbt are invisible to analysts	<code>GRANT SELECT ON FUTURE TABLES IN SCHEMA ...</code>
Consuming a stream in a task that fails midway	Assuming changes were lost (or processed twice)	A stream only advances when the DML that reads it commits; wrap consumption in a single transaction
Big <code>SELECT *</code> over wide VARIANT data	Slow and expensive	Flatten frequently used JSON paths into typed columns in Silver

Cheat Sheet

Task	SQL
Top N per group	<code>... QUALIFY ROW_NUMBER() OVER (PARTITION BY k ORDER BY ts DESC) = 1</code>
JSON path	<code>payload:user.id::NUMBER</code>
Explode array	<code>, LATERAL FLATTEN(INPUT => payload:items) f → f.value:sku::STRING</code>

Task	SQL
Safe cast	<code>TRY_CAST(x AS NUMBER(12,2)) · TRY_TO_DATE(s)</code>
Load files	<code>COPY INTO t FROM @stage/path FILE_FORMAT=(TYPE=PARQUET) MATCH_BY_COLUMN_NAME=CASE_INSENSITIVE</code>
Continuous load	<code>CREATE PIPE p AUTO_INGEST=TRUE AS COPY INTO ...</code>
Query the past	<code>SELECT ... FROM t AT(OFFSET => -3600) · BEFORE(STATEMENT => '<query_id>')</code>
Undo a drop	<code>UNDROP TABLE t</code>
Dev copy of prod	<code>CREATE DATABASE dev CLONE prod</code>
<u>CDC</u> on a table	<code>CREATE STREAM s ON TABLE t → read METADATA\$ACTION , METADATA\$ISUPDATE</code>
Scheduled SQL	<code>CREATE TASK ... SCHEDULE='USING CRON 0 * * * * UTC' AS ... → ALTER TASK ... RESUME</code>
Declarative pipelines	<code>CREATE DYNAMIC TABLE ... TARGET_LAG = '15 minutes' WAREHOUSE = wh AS SELECT ...</code>
Resize	<code>ALTER WAREHOUSE wh SET WAREHOUSE_SIZE = 'LARGE'</code>
Cap spend	<code>CREATE RESOURCE MONITOR rm WITH CREDIT_QUOTA = 100 TRIGGERS ON 100 PERCENT DO SUSPEND</code>
Who can see what	<code>SHOW GRANTS TO ROLE r · SHOW GRANTS ON TABLE t</code>
Last query's ID	<code>SELECT LAST_QUERY_ID()</code>

Warehouse sizing: each size step doubles credits/hour (XS=1, S=2, M=4, L=8, XL=16...) and roughly halves the runtime of queries that parallelize well — so a bigger warehouse can cost the same while finishing faster.

Table types: permanent (time travel + 7-day fail-safe) · transient (no fail-safe — staging) · temporary (session only) · external (query files in place) · Iceberg (open format in your own bucket)

Interview Questions

Q: Explain Snowflake's architecture. A: Three independent layers. Storage: data is kept as compressed, columnar, immutable micro-partitions in cloud object storage. Compute: virtual warehouses — independent clusters that read that storage, scale up (size) or out (multi-cluster), and suspend when idle, billed per second. Cloud services: authentication, metadata, the optimizer, transactions, and the result cache. Because storage and compute are separate, many workloads can query the same data concurrently without contention, and each scales and is billed independently.

Q: What are micro-partitions and how does pruning work? A: Snowflake automatically splits tables into immutable micro-partitions (~50–500 MB uncompressed) and records metadata for each — min/max values per column, distinct counts, and null counts. At query time the optimizer compares filter predicates against that metadata and skips partitions that can't match. Pruning works best when data is naturally ordered by the filter column (e.g. loaded by date); a clustering key can restore that order for large tables with a different access pattern.

Q: When would you scale a warehouse up versus out? A: Scale *up* (a larger size) when individual queries are slow or spill to disk — more memory and CPU per query. Scale *out* (multi-cluster) when many concurrent queries are queuing — more clusters serve more users, but no single query gets faster. The Query Profile shows spilling; warehouse load history shows queuing.

Q: What are Streams and Tasks, and how do they work together? A: A stream is a change-tracking object on a table: it records inserts, updates, and deletes since it was last consumed, using an offset rather than a copy of the data. A task runs SQL on a schedule or after another task. Together they form an in-Snowflake incremental pipeline: a task (optionally gated on `SYSTEM$STREAM_HAS_DATA`) `MERGE`s the stream's changes into a target, and the stream's offset advances when that transaction commits. Dynamic tables are the newer, declarative alternative for many of these cases.

Q: What is the difference between Time Travel and Fail-safe? A: Time Travel lets *you* query, clone, or restore data as it was at an earlier point — 1 day by default, up to 90 days on Enterprise — using `AT / BEFORE` and `UNDROP`. Fail-safe is a further 7-day window after Time Travel expires, recoverable only by Snowflake support, meant for disasters. Both add storage cost, which is why transient tables (no fail-safe) are used for staging.

Q: How does zero-copy cloning work and what is it used for? A: A clone copies metadata only — the new object points to the same micro-partitions as the source, so it's instant and initially free. Once either side changes data, only the changed micro-partitions are stored separately. It's used for dev/test environments cloned from production, backups before risky migrations, and CI databases for dbt PRs.

Q: How would you control Snowflake costs? A: Auto-suspend every warehouse (60 seconds is typical), right-size per workload and separate workloads for attribution, set resource monitors with quotas, use transient tables and shorter retention for staging, avoid unnecessary clustering, and let the result cache work. Then monitor `ACCOUNT_USAGE` views (`WAREHOUSE_METERING_HISTORY`, `QUERY_HISTORY`) for the most expensive queries and fix those — usually poor pruning, exploding joins, or full refreshes that should be incremental.

Q: How do you load data into Snowflake? A: Batch with `COPY INTO` from a stage (internal, or external S3/GCS/Azure using a storage integration). It tracks load metadata so files aren't loaded twice. For continuous loading, Snowpipe auto-ingests files as cloud event notifications arrive, and Snowpipe Streaming ingests rows directly (e.g. from the Kafka connector) without files. Managed tools (Fivetran, Airbyte) sit on top of these mechanisms.

Further Reading

- [Snowflake documentation](https://docs.snowflake.com/) (https://docs.snowflake.com/)
 - [Snowflake key concepts & architecture](https://docs.snowflake.com/en/user-guide/intro-key-concepts) (https://docs.snowflake.com/en/user-guide/intro-key-concepts)
 - [Micro-partitions & data clustering](https://docs.snowflake.com/en/user-guide/tables-clustering-micropartitions) (https://docs.snowflake.com/en/user-guide/tables-clustering-micropartitions)
 - [Dynamic tables](https://docs.snowflake.com/en/user-guide/dynamic-tables-about) (https://docs.snowflake.com/en/user-guide/dynamic-tables-about)
 - [Managing cost in Snowflake](https://docs.snowflake.com/en/guides-overview-cost) (https://docs.snowflake.com/en/guides-overview-cost)
 - [Snowflake Quickstarts](https://quickstarts.snowflake.com/) (https://quickstarts.snowflake.com/) — free hands-on tutorials
-

Previous: [Testing and CI/CD](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/testing-cicd/) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/testing-cicd/) • **Next:** [BigQuery](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/bigquery-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/bigquery-reference/) • **Back to:** [Index](https://sarangambekar1997.github.io/data-engineering-handbook/) (https://sarangambekar1997.github.io/data-engineering-handbook/)

 September 29, 2026