Snowflake Learning Journey – Comprehensive Summary

Date: December 8, 2025
Learning Path: Data Engineering Foundation with Snowflake

Table of Contents

---

1. Snowflake Architecture

Overview

Snowflake is a cloud-native data platform built on three core layers:

- Storage Layer – Scalable cloud object storage (S3, Azure Blob, GCS) storing data in Parquet format
- Compute Layer – Virtual warehouses (clusters) that execute queries independently
- Cloud Services Layer – Metadata management, authentication, query optimization, and transaction control

Key Characteristics

- Shared-Nothing Architecture – Storage and compute are decoupled; scale independently
- Snowflake Metadata Store – Centralized metadata repository tracking objects, privileges, and versioning
- Multi-Cloud Support – AWS, Azure, GCP
- Zero-Copy Clone – Instantly copy databases/schemas without duplicating data

Advantages

- Pay only for compute used (auto-suspend, auto-resume)
- Instant elasticity – scale up/down on demand
- Time-travel and fail-safe for data recovery
- Secure data sharing without copying data

---

2. Virtual Warehouses

What is a Virtual Warehouse?

A virtual warehouse is a compute cluster that executes SQL queries. Each warehouse is independent and can run in parallel.

Warehouse Types

Standard Warehouse

- Traditional multi-purpose compute
- Suitable for general SQL queries, ETL, analytics
- Example: 1–16 credits per hour

Snowpark-Optimized Warehouse

- Optimized for Snowpark (Python/Java/Scala) workloads
- Higher memory per compute unit
- Best for ML and data science jobs with large in-memory operations

Multi-Cluster Warehouse

- Auto-scales based on query queue demand
- Multiple clusters share the warehouse name
- Ideal for high-concurrency environments (many concurrent users/jobs)
- Example: Min 2, Max 10 clusters

Creating & Monitoring Warehouses

Create a warehouse:

sql

```
USE ROLE SYSADMIN;


CREATE OR REPLACE WAREHOUSE MY_WH
  WAREHOUSE_SIZE = MEDIUM
  AUTO_SUSPEND = 60
  AUTO_RESUME = TRUE
  INITIALLY_SUSPENDED = FALSE;
```

Monitor warehouse:

sql

SELECT * **FROM** SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY;

SELECT * **FROM** SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY;

Key parameters:

- WAREHOUSE_SIZE – XS, S, M, L, XL, 2XL, 3XL, 4XL (credits per minute)
- AUTO_SUSPEND – Minutes before auto-pause (default 600)
- AUTO_RESUME – Automatically resume on query submission
- MAX_CLUSTER_COUNT – For multi-cluster warehouses

---

3. User & Role Management (RBAC)

Access Control Framework

Snowflake uses Discretionary Access Control (DAC), Role-Based Access Control (RBAC), and User-Based Access Control (UBAC):

- DAC – Object owner can grant privileges
- RBAC – Privileges assigned to roles; roles assigned to users (primary method)
- UBAC – Direct user privileges (exceptions only)

System-Defined Roles

Role	**Purpose**	**Can Do**
ACCOUNTADMIN	Top-level admin	All privileges; account management
SECURITYADMIN	Security & grants	Grant/revoke privileges; manage users/roles
USERADMIN	User/role admin	Create users and roles; user lifecycle
SYSADMIN	System admin	Create databases, warehouses, schemas, tables
PUBLIC	Default role	Auto-assigned to all users
GLOBALORGADMIN	Org-level admin	Manage organization across accounts

Creating Custom Roles

sql

USE ROLE SECURITYADMIN;

CREATE ROLE ROLE_APP_DB_OWNER **COMMENT** = 'Owns APP_DB and can grant privileges';

CREATE ROLE ROLE_APP_RW **COMMENT** = 'Read/write access to APP_DB';

CREATE ROLE ROLE_APP_RO **COMMENT** = 'Read-only access to APP_DB';

-- Attach into hierarchy for management

GRANT ROLE ROLE_APP_RW **TO** ROLE SYSADMIN;

GRANT ROLE ROLE_APP_RO **TO** ROLE SYSADMIN;

Creating Users & Assigning Roles

sql

USE ROLE USERADMIN;

CREATE USER APP_DE_USER

PASSWORD = 'TempP@ssw0rd'

DEFAULT_ROLE = ROLE_APP_RW

MUST_CHANGE_PASSWORD = TRUE;

USE ROLE SECURITYADMIN;

GRANT ROLE ROLE_APP_RW **TO USER** APP_DE_USER;

Granting Privileges

Grant on database:

sql

USE ROLE ROLE_APP_DB_OWNER;

GRANT USAGE ON DATABASE APP_DB **TO** ROLE ROLE_APP_RW;

GRANT USAGE ON SCHEMA APP_DB.APP_SCHEMA **TO** ROLE ROLE_APP_RW;

GRANT SELECT, **INSERT**, **UPDATE**, **DELETE ON ALL TABLES** IN **SCHEMA** APP_DB.APP_SCHEMA **TO** ROLE ROLE_APP_RW;

Grant on future objects:

sql

GRANT SELECT ON FUTURE **TABLES** IN **SCHEMA** APP_DB.APP_SCHEMA **TO** ROLE ROLE_APP_RO;

Key Principles

- USAGE privilege required to *see/reference* database or schema
- SELECT/INSERT/UPDATE/DELETE required for table operations
- Object owner can grant privileges on their objects
- SECURITYADMIN (with MANAGE GRANTS) can grant any privilege globally
- No privilege = no access (deny by default)

---

4. Connecting to Snowflake

Snowsight UI

- Web-based IDE for writing and executing SQL
- Visual query results, charts, and dashboards
- Role switching and warehouse selection
- Built-in documentation
- Access: https://<account>.snowflakecomputing.com

SnowSQL

Command-line interface for Snowflake.

Install & connect:

bash

 # Install SnowSQL (macOS, Linux, Windows available)

snowsql -a <account_identifier> -u <username>

 # Example commands

snowsql -a xy12345.us-east-1 -u sandeep_reporter

SELECT * FROM my_table;

!exit

VS Code Extension

Snowflake SQL Tools for VS Code

- Write SQL in IDE
- Execute queries
- Browse databases/schemas/tables
- Connect with credentials or key-pair auth

Setup:

1. Install Snowflake SQL Tools extension
2. Set connection in settings or .vscode/settings.json
3. Select warehouse/database/schema
4. Write and execute SQL

Python Connector

Connect Python to Snowflake for ETL, data science, and automation.

python

```
from snowflake.connector import connect

conn = connect(
    user='sandeep_reporter',
    password='password',
    account='xy12345.us-east-1',
    warehouse='MY_WH',
    database='APP_DB',
    schema='APP_SCHEMA'
)

cursor = conn.cursor()
cursor.execute('SELECT * FROM CUSTOMERS LIMIT 10;')
results = cursor.fetchall()
for row in results:
    print(row)

cursor.close()
```

conn.close()

JDBC Connection

For Java applications.

java

```
import java.sql.*;

public class SnowflakeConnection {
  public static void main(String[] args) throws Exception {
    Class.forName("net.snowflake.client.jdbc.SnowflakeDriver");

    String connectionString = "jdbc:snowflake://<account>.snowflakecomputing.com/" +
      "?warehouse=MY_WH&database=APP_DB&schema=APP_SCHEMA";

    Connection conn = DriverManager.getConnection(
      connectionString,
      "sandeep_reporter",
      "password"
    );

    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM CUSTOMERS;");

    while (rs.next()) {
      System.out.println(rs.getString(1));
    }

    rs.close();
    stmt.close();
    conn.close();
  }
}
```

---

5. Data Types & Table Design

Data Type Categories

Snowflake supports diverse data types grouped as:

Numeric

- NUMBER(precision, scale) – Default (38, 0)
- INT, BIGINT, SMALLINT, TINYINT
- FLOAT, DOUBLE, REAL
- DECIMAL, NUMERIC (synonyms for NUMBER)

String & Binary

- VARCHAR(length) – Default 16777216 bytes
- STRING, TEXT (synonyms for VARCHAR)
- CHAR(length) – Fixed length
- BINARY, VARBINARY

Date & Time

- DATE – Date only
- TIME – Time only
- TIMESTAMP_NTZ – Timestamp without time zone
- TIMESTAMP_LTZ – Timestamp with local time zone (stored as UTC)
- TIMESTAMP_TZ – Timestamp with time zone (stored with TZ info)
- DATETIME (synonym for TIMESTAMP_NTZ)

Boolean

- BOOLEAN – TRUE/FALSE

Semi-Structured

- VARIANT – JSON-like data (any JSON value)
- OBJECT – Key-value pairs
- ARRAY – List of values

Geospatial & Vector

- GEOGRAPHY – Points, lines, polygons (earth coordinates)
- GEOMETRY – Geometric objects (2D/3D)
- VECTOR(type, dimension) – ML embeddings (e.g., VECTOR(FLOAT, 128))

Example: CREATE TABLE with Various Data Types

sql

```
CREATE OR REPLACE TABLE customer_orders (
  order_id       NUMBER(38,0)       PRIMARY KEY,
  customer_id    INT                NOT NULL,
  order_amount   NUMBER(10,2),
  discount_pct   FLOAT,
  email          VARCHAR(255),
  is_first_order BOOLEAN,
  order_date     DATE,
  order_time     TIME,
  created_at     TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
  updated_at     TIMESTAMP_LTZ,
  notes          STRING,
  receipt_file   BINARY
);
```

Example: Semi-Structured Data

sql

```
CREATE OR REPLACE TABLE click_events (
  event_id      NUMBER    PRIMARY KEY,
  user_id       VARCHAR,
  event_time    TIMESTAMP_TZ,
  event_data    VARIANT           -- Full JSON payload
);


INSERT INTO click_events VALUES (
  1,
  'user_123',
  CURRENT_TIMESTAMP(),
  PARSE_JSON('{"page": "home", "action": "click", "target": "signup_button"}')
);
```

-- *Query VARIANT fields*

SELECT event_data:page, event_data:**action FROM** click_events;

Example: Advanced Types

sql

CREATE OR **REPLACE TABLE** advanced_data (

id NUMBER **PRIMARY KEY**,

attributes OBJECT, -- *Key/value map*

tags ARRAY, -- *List*

location GEOGRAPHY, -- *Point/polygon*

embedding VECTOR(**FLOAT**, 128) -- *128-dim vector*

);

---

6. Hands-on Practices

Practice 1: Create Database & Role Structure

sql

USE ROLE SYSADMIN;

CREATE DATABASE TEST_DB;

CREATE SCHEMA TEST_DB.TEST_SCHEMA;

USE ROLE SECURITYADMIN;

CREATE ROLE ROLE_DB_OWNER;

GRANT ROLE ROLE_DB_OWNER **TO** ROLE SYSADMIN;

USE ROLE ROLE_DB_OWNER;

CREATE DATABASE APP_DB;

CREATE SCHEMA APP_DB.APP_SCHEMA;

Practice 2: Grant Privileges

sql

USE ROLE SECURITYADMIN;

-- Create read-only role

CREATE ROLE DATA_READ_ONLY;

-- Grant USAGE on database and schemas

GRANT USAGE ON DATABASE APP_DB **TO** ROLE DATA_READ_ONLY;

GRANT USAGE ON SCHEMA APP_DB.APP_SCHEMA **TO** ROLE DATA_READ_ONLY;

-- Grant SELECT on tables

GRANT SELECT ON ALL TABLES IN **SCHEMA** APP_DB.APP_SCHEMA **TO** ROLE DATA_READ_ONLY;

GRANT SELECT ON FUTURE **TABLES** IN **SCHEMA** APP_DB.APP_SCHEMA **TO** ROLE DATA_READ_ONLY;

Practice 3: Create User & Connect

sql

USE ROLE USERADMIN;

CREATE USER SANDEEP_REPORTER

PASSWORD = 'TempP@ssw0rd'

DEFAULT_ROLE = DATA_READ_ONLY

MUST_CHANGE_PASSWORD = TRUE;

USE ROLE SECURITYADMIN;

GRANT ROLE DATA_READ_ONLY **TO USER** SANDEEP_REPORTER;

Then connect via Snowsight, SnowSQL, VS Code, or Python as that user.

Practice 4: Create & Monitor Warehouse

sql

USE ROLE SYSADMIN;

CREATE WAREHOUSE ANALYTICS_WH

WAREHOUSE_SIZE = MEDIUM

AUTO_SUSPEND = 60

AUTO_RESUME = TRUE;

-- Monitor

```
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY
WHERE WAREHOUSE_NAME = 'ANALYTICS_WH'
ORDER BY START_TIME DESC;
```

Practice 5: Load Data (CSV → Snowflake)

```
sql
-- Create file format
CREATE FILE FORMAT FF_CSV
  TYPE = 'CSV'
  FIELD_DELIMITER = ','
  SKIP_HEADER = 1;

-- Create internal stage
CREATE STAGE STG_DATA FILE_FORMAT = FF_CSV;

-- Upload file (via SnowSQL or UI)
-- PUT file:///path/to/data.csv @STG_DATA;

-- Load into table
COPY INTO CUSTOMERS
  FROM @STG_DATA
  FILES = ('data.csv')
  ON_ERROR = 'ABORT_STATEMENT';
```

---

7. Next Steps & Learning Path

Immediate Next Topics

1. Data Loading & Unloading
    - COPY INTO from internal/external stages
    - Snowpipe for continuous loading
    - UNLOAD for exporting data
2. Transformations & Queries
    - Window functions (ROW_NUMBER, RANK, LAG, LEAD)

- CTEs and nested queries
- Aggregations and GROUP BY
- Joins (INNER, LEFT, RIGHT, FULL OUTER, CROSS)

3. Advanced Features
    - Snowpipe (streaming ingestion)
    - Streams (change data capture)
    - Tasks (scheduled SQL jobs)
    - Dynamic Data Masking (DDM) for security
4. Performance Optimization
    - Query profiling and EXPLAIN PLAN
    - Clustering keys
    - Micro-partitions and pruning
    - Query optimization best practices
5. ETL/ELT Pipelines
    - End-to-end pipeline design
    - Error handling & logging
    - Orchestration (Airflow, Prefect, dbt)
    - Testing & monitoring
6. Data Sharing & Governance
    - Secure Data Sharing
    - Iceberg tables and Delta format
    - Data catalog and lineage
    - Audit logging

Quick Reference: Common SQL Commands

Database & Schema Management

sql

CREATE DATABASE db_name;

CREATE SCHEMA db_name.schema_name;

SHOW DATABASES;

SHOW SCHEMAS IN **DATABASE** db_name;

USE DATABASE db_name;

USE SCHEMA schema_name;

Role & User Management

sql

CREATE ROLE role_name;

CREATE USER user_name PASSWORD = 'pwd';

GRANT ROLE role_name **TO USER** user_name;

GRANT USAGE ON DATABASE db_name **TO** ROLE role_name;

SHOW GRANTS **TO** ROLE role_name;

SHOW GRANTS **ON DATABASE** db_name;

Warehouse Management

sql

CREATE WAREHOUSE wh_name WAREHOUSE_SIZE = MEDIUM;

ALTER WAREHOUSE wh_name **SET** WAREHOUSE_SIZE = LARGE;

SUSPEND WAREHOUSE wh_name;

RESUME WAREHOUSE wh_name;

SHOW WAREHOUSES;

Table Operations

sql

CREATE TABLE table_name (col1 **TYPE**, col2 **TYPE**, ...);

INSERT INTO table_name **VALUES** (...);

SELECT * **FROM** table_name;

UPDATE table_name **SET** col1 = **value WHERE** condition;

DELETE FROM table_name **WHERE** condition;

DROP TABLE table_name;

Query Insights

sql

-- Query history

SELECT * **FROM** SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY;

-- Warehouse usage

```
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY;


-- Database usage
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.DATABASE_USAGE_HISTORY;
```

Resources

- Official Docs: **https://docs.snowflake.com**
- Snowflake University: **https://learn.snowflake.com**
- Community: **https://community.snowflake.com**
- GitHub: **https://github.com/Snowflake-Labs**

Learning Summary

✅ Covered:

- Snowflake architecture and multi-cloud deployment
- Virtual warehouses (types, scaling, monitoring)
- User/role management and RBAC best practices
- Multiple connection methods (UI, CLI, Python, JDBC)
- Data types and table design
- Hands-on practices with real SQL commands

🚀 Ready For:

- Building real ETL pipelines
- Loading data from external sources
- Designing role hierarchies for production
- Performance tuning and query optimization
- Advanced data engineering with Streams, Tasks, Snowpipe

Last Updated: December 8, 2025
Learning Status: Foundation Complete | Ready for Advanced Topics