

Mastering Snowflake Platform

Generate, fetch, and automate Snowflake data
as a skilled data practitioner



Pooja Kelgaonkar



Mastering Snowflake Platform

Generate, fetch, and automate Snowflake data
as a skilled data practitioner



Pooja Kelgaonkar

bpb

Mastering Snowflake Platform

*Generate, fetch, and automate
Snowflake data
as a skilled data practitioner*

Pooja Kelgaonkar



www.bpbonline.com

Copyright © 2024 BPB Online

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor BPB Online or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

BPB Online has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, BPB Online cannot guarantee the accuracy of this information.

First published: 2024

Published by BPB Online
WeWork
119 Marylebone Road
London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55519-764

www.bpbonline.com

Dedicated to

My beloved Parents,

Neelkrishna

&

My Daughter Anvi

About the Author

Pooja Kelgaonkar is a distinguished figure in the realm of data, standing as one of the Snowflake data superheroes, boasting nearly two decades of experience in the field. She has honed her expertise through diverse roles on various data platforms and distributed systems, emerging as a specialist in data modernization. She is currently serving as a senior data architect at Rackspace Technology in Toronto. As a key data resource, she leads the data implementations, ensuring seamless integration and optimal performance, ultimately contributing to the success and satisfaction of the clients.

Her commitment to knowledge extends beyond her professional role, as she is also the founder of an edutech platform. She is a strong advocate for knowledge sharing and community growth. Her commitment is evident in her contributions to the tech community through blogs on Snowflake and Google Cloud Platform (GCP). These insightful articles are regularly featured on Google and Snowflake community pages, solidifying her position as a thought leader.

Pooja's passion for community growth extends beyond the digital arena; she thrives as a public speaker, frequently participating in events, conferences, and tech webinars. Her involvement in these platforms allows her to share her wealth of knowledge and insights, contributing to the collective learning of the community. Pooja's multifaceted engagement with data, education, and community development showcases a profound commitment to advancing the field and empowering those within it.

About the Reviewer

Dipika Aher is a passionate data engineer with extensive professional experience in the domain of healthcare and manufacturing. With a skill set that includes Snowflake, DBT, Python, Unix, SQL, AWS, Informatica, Oracle, she is a holder of multiple industry certifications such as the AWS Solutions, Snowflake Certifications and more.

She is currently working as part of a development team and has been using cloud technologies to enhance her Snowflake related knowledge.

She believes that education is the key to empowering people and creating positive change in the world. She uses her expertise and enthusiasm to inspire and motivate others to pursue their passions and goals in the field of data engineering.

Acknowledgement

I extend my deepest gratitude to my parents and family, especially my daughter Anvi, for their unwavering support and encouragement throughout the writing of this book. Their constant belief in me has been my greatest motivation.

I am grateful to BPB Publications for presenting me the opportunity to author a book on mastering Snowflake. My heartfelt thanks go to the entire BPB team, whose collaboration in both editorial and technical aspects has made this journey an enriching experience.

Special acknowledgment goes to my mentor as well as the Snowflake community. They have consistently encouraged me to pursue my aspirations. It is an honor to be part of the Snowflake's elite group that is committed to continuous knowledge and learning.

To all the readers who have shown interest in my book and contributed to its realization, I express my sincere thanks. Your support and encouragement have been truly invaluable in bringing this project to life.

Preface

Data is continuously evolving and hence the world around it. Many have witnessed the transition from Mainframes to the cloud implementations. It is the need of the moment to have a robust, scalable, performance and cost efficient data platform that cater to the ever growing need of the data applications. In the rapidly evolving landscape of data management, Snowflake stands as a powerful platform, and offers unified experience to the customers. This book is your first step to unlocking its full potential.

In this book, you embark on a transformative exploration, blending theoretical insights with practical applications to empower you on your data mastery voyage. Whether you are a newcomer to delve into the world of Snowflake or an experienced practitioner seeking to refine your skills, this guide is crafted to meet you at your current level and propel you forward.

As the book unfolds, you will traverse the intricacies of Snowflake architecture, discover its key features, and navigate the nuances of working with different data types. You will continue the journey with a deep dive into advanced concepts such as data security, governance, and collaboration.

What sets this guide apart is its relation with real time use cases and references from traditional enterprise data architectures along with hands-on approach. Throughout the chapters, you will find practical exercises, real-time use cases, and reference architectures that bridge the gap between theory and application.

Get ready and fasten your seatbelt to embark on a transformative journey with Snowflake. Whether you are aiming to enhance your career, architect data solutions, or simply learn new skills, “Mastering Snowflake Platform” is your trusted companion to get you started.

Chapter 1: Getting Started with Snowflake – The first chapter is the foundation of the book, and it covers the history of the Snowflake, the need to implement Snowflake, and how readers can get started with Snowflake. This chapter guides to set up a trial or demo account to be used for various hands-on activities covered in subsequent chapters. This also covers the overview of

Snowflake certifications and various community events.

Chapter 2: Three Layered Architecture – This chapter helps readers to get started with Snowflake with a detailed view of architecture. This covers the various data platform architecture challenges and how Snowflake helps to overcome most of these challenges. This chapter defines 3 layers of architecture, their distinguishing features and setup.

Chapter 3: Data Types, Data Objects and SQL Commands – This chapter focuses on various data types, data objects and SQL commands supported in Snowflake. Snowflake supports ANSI SQL standard. However this chapter guides on DDL, DCL and DML supported in this platform. This chapter also covers unique database objects used and created in Snowflake.

Chapter 4: Data Loading and Unloading – Data loading and unloading is one the most important parts of data platform implementation. This chapter guides you to learn native utilities, commands supported by Snowflake. This covers various commands used to load batch and streaming datasets. This also covers data extraction and sharing with consumer groups.

Chapter 5: Understanding Streams and Tasks – Change data capture is the most critical part of the data platform implementation. This chapter guides you to streams – Snowflake native objects used for change data capture. This also covers native scheduling and orchestrating objects – Tasks.

Chapter 6: Understanding Snowpark – Spark is one of the fastest ways to implement ETL or ELT while implementing data pipelines. This chapter covers Snowpark – which is a native programming language like Spark. It is also a guide on how to understand and implement Snowpark.

Chapter 7: Access Control and Managing Users Roles – Access control is one of the key pillars of data governance. This chapter helps to understand various access control implementations in Snowflake. This also covers role and user management, how users are created, roles are created and maintained. This guides to set up the most important pillar of governance.

Chapter 8: Data Protection and Recovery – Data protection and recovery is the next pillar of data governance. This chapter guides you to understand data protection mechanisms with Snowflake. This also covers various data recovery options in Snowflake. This chapter helps to define appropriate policies, data masking and using the right data recovery technique when needed.

Chapter 9: Snowflake Performance Optimization – This chapter guides to

develop an understanding of performance measure, performance optimization techniques and need to implement optimization. This also covers understanding of Snowflake metadata objects that help to measure performance of the platform.

Chapter 10: Understanding Snowflake Costing and Utilizations – Cost is the most critical component of platform design, implementation, usage, and maintenance. This chapter guides to develop an understanding of cost measure, cost optimization techniques and need to implement optimization.

Chapter 11: Implementing Cost Optimizations – This chapter guides how to implement cost optimizations. This helps to develop an understanding of real time use cases and best practices followed for cost optimizations.

Chapter 12: Data Sharing – Data collaborations, data integrations and integrating with consumer applications is essential for business growth and requirements. This chapter guides how to implement data sharing, secure data sharing with Snowflake as well as non-Snowflake consumer groups. This also helps to understand various sharing options available in Snowflake.

Chapter 13: Data Cloning – Managing environments like DEV, QA, UAT and PROD becomes challenging when a user wants to develop, test their features with greater data volumes or production datasets. This chapter guides how to setup environments using data cloning features and helps to understand how easy it is to clone and maintain copies, data across environments.

Chapter 14: Understanding Snowsight – Earlier Snowflake offered two versions of web based user interfaces – Classic console and Snowsight. Now with latest sign ups, Snowsight is the only UI made available to the users and all earlier accounts migrated to Snowsight. This chapter helps to understand Snowsight features and distinguishing factors of the interfaces. This also covers the intuitive dashboarding feature of Snowsight.

Chapter 15: Programming Connectors and Drivers – Snowflake offers various connectors, drivers to work with a set of programming languages. This chapter helps to understand the various connectors and drivers. This also guides to setup sample drivers and Snowflake native command line interface – Snowsql.

Chapter 16: Workload Patterns with Snowflake – Snowflake is data on cloud and supports various data implantations. This chapter helps to understand Data Warehouse, Data Lake, Lake house requirements and implementing with Snowflake. This also covers a variety of real time use cases, data architectures for reference.

Chapter 17: Introduction to Snowflake's Advance Features – Snowflake continuously works on their features, releases updates and introduces new features with every release. This chapter helps to understand new features introduced by Snowflake. This also covers a variety of real time use cases for reference.

Code Bundle and Coloured Images

Please follow the link to download the **Code Bundle** and the **Coloured Images** of the book:

<https://rebrand.ly/xar4q49>

The code bundle for the book is also hosted on GitHub at **<https://github.com/bpbpublications/Mastering-Snowflake-Platform>**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **<https://github.com/bpbpublications>**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePUB files available? You can upgrade to the eBook version at **www.bpbonline.com** and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at :

business@bpbonline.com for more details.

At **www.bpbonline.com**, you can also read a collection of free technical articles, sign up for a range of

free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at business@bpbonline.com with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit www.bpbonline.com. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit www.bpbonline.com.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord\(bpbonline\).com](https://discord(bpbonline).com)



Table of Contents

1. Getting Started with Snowflake

- Introduction
- Structure
- Objectives
- Why Snowflake?
- History of Snowflake
- Snowflake certifications
- Snowflake community
- Setting up a trial account with Snowflake
- Connecting to Snowflake
 - Snowflake Web User Interface*
 - Classic console*
 - Context setting*
 - Snowsight*
 - Command line interface*
 - Download and setup SnowSQL*
 - Connecting to Snowflake*
- Conclusion
- Points to remember
- Practical
- Multiple choice questions
- Answers

2. Three Layered Architecture

- Introduction

Structure

Objectives

Traditional data warehouse challenges

Legacy data warehouse challenges

Typical data lake challenges

Hadoop ecosystem

Extended support to NoSQL

Snowflake architecture

Cloud services layer

Authentication and access control

Optimizer

Metadata storage and management

Result cache

Security

Availability of cloud services layer

Uses of cloud services layer

Virtual warehouse layer

Virtual warehouses: Single and multi-cluster

Single cluster warehouses

Multi-cluster warehouses

Virtual warehouse properties

Virtual warehouse sizes

Warehouse scaling

Scaling up

Scaling out

Scaling modes

Scaling policies

Virtual warehouses: Creation and modifications

Warehouse setup using console

Warehouse: Using SQL commands

Creating warehouse

Other warehouse commands

Virtual warehouses: Billing and usage

Storage layer

Time-travel

Failsafe

Data cloning

Storage usage and billing

Cache in Snowflake

Metadata cache

Query result cache

Warehouse cache

Conclusion

Points to remember

Multiple choice questions

Answers

Questions

3. Data Types, Data Objects and SQL Commands

Introduction

Structure

Objectives

Understanding data types

Numeric data types

String and binary data types

Binary data types

Logical data types

Date and time data types

Date and time intervals and constants

Supported date and time arithmetic operations

Semi-structured data types

VARIANT

OBJECT

OBJECT constant

OBJECT elements

OBJECT considerations

ARRAY

ARRAY elements

ARRAY considerations

Geospatial data types

Geography

Geometry

Using SQL commands

Data definition language

Account and session DDL

User and security DDL

Warehouse and resource monitor DDL

Database, schema and share DDL

Table, view and sequence DDL

Data loading / unloading DDL

User-defined functions, external functions, and stored procedures DDL

Data pipeline DDL

Data manipulation language

Standard DML

Data loading/unloading DML

File staging commands

Creating database objects

Creating users and roles

Creating roles using SQL commands

Creating users using SQL commands

Creating roles using Web Console

Creating users using Web Console

Creating warehouses

Creating warehouse using SQL command

Creating warehouse using Web Console

Creating databases and schemas

Creating database using SQL command

Creating database using web console
Creating schemas using SQL command

Setting up context

Setup context using SQL command

Creating tables and views

Creating table using SQL command

Implementing extended SQL objects

UDF example (JavaScript)

UDF example (SQL)

Tabular functions

Sample function

Creating function

Using function

SQL UDFT function

Creating function

Using function

External functions

Conclusion

Points to remember

Practical: Create Snowflake objects

Multiple choice questions

Answers

Questions

4. Data Loading and Unloading

Introduction

Structure

Objectives

Understanding data load needs

Creating Snowflake load objects

Source files

Loading from file locations

Loading files from local system

Loading data from Web UI

Using COPY INTO to load data

Bulk load using COPY INTO

COPY syntax

Transformations supported in COPY INTO

COPY with transformation syntax

VALIDATE command

COPY optional parameters

COPY INTO options

Options

COPY INTO using for bulk loads

COPY for local system

PUT command

Understanding streaming loads

Implementing Snowpipe

Automate the load using cloud messaging

Using Snowpipe REST endpoints

Snowpipe versus bulk load

Snowpipe features and recommendations

Snowpipe commands

SQL commands

CREATE PIPE

ALTER PIPE

DROP PIPE

DESCRIBE PIPE

SHOW PIPE

Loading real time data

Snowpipe

Snowflake connector for Kafka

Snowpipe streaming

Snowpipe streaming versus Snowpipe Streaming cost

Loading near real time data

Implementing Snowpipe

Understanding data unloading

Database feed

File feed

Using COPY to unload data

COPY INTO

Exported feeds, formats and locations

Compression supported

Encryption supported

Bulk unloading from a table or query

Bulk unloading to one or multiple files

File naming in case of multiple file exports

Bulk unloading using PARTITIONs

COPY INTO options

Exporting data to named internal stages

GET command

Exporting data to named external stages

Steps to unload data to named external stages

Practical: Data loading and unloading

Creating stages

Creating file formats

Creating storage integrations

Creating LOAD commands and load using COPY

Load from local using COPY

Unload from local using COPY

Conclusion

Points to remember

Questions

Multiple choice questions

Answers

5. Understanding Streams and Tasks

Introduction

Structure

Objectives

Understanding Change Data Capture

Understanding Snowflake Streams

Types of Streams

Standard Streams

Append-only Streams

Insert-only Streams

Validity of change logs

Billing of Streams

Implementing data capture with Streams

CREATE STREAM

ALTER STREAM

DESCRIBE STREAM

DROP STREAM

SHOW STREAMS

Stream examples

Understanding tasks

Serverless tasks

User-managed tasks

Scheduling tasks

Implementing scheduling with Tasks

TASK commands

CREATE TASK

ALTER TASK

DROP TASK

EXECUTE TASK

SHOW TASKS

Practical: Create Snowflake Tasks and Streams

Exercise 1

Exercise 2

Exercise 3

Conclusion

Points to remember

Multiple choice questions

Answers

Questions

6. Understanding Snowpark

Introduction

Structure

Objectives

Why Snowpark?

Understanding Snowpark

Snowpark-optimized warehouse

Snowpark-optimized warehouse billing

Snowpark-ML

Snowpark-benefits over Spark Connector

Implementing Snowpark

Environment setup

Using Python and developing code

How does Python dataframe API work?

Using Snowpark for ML

Use cases to implement Snowpark

Data engineering use cases

Practical: Setting up Snowpark and use cases

Snowpark engineering use cases

Conclusion

Points to remember

Questions

Multiple choice questions

Answers

7. Access Control and Managing Users Roles

Introduction

Structure

Objectives

Snowflake access control overview

Snowflake role and users

Types of roles

Snowflake securable objects

Understanding RBAC

Understanding default roles in Snowflake

Implementing RBAC with Snowflake

Step 1: Creating Snowflake resources

Create environments

Create DEV databases and schemas

Create QA databases and schemas

Create BI databases and schemas

Create ML databases and schemas

Create warehouses for users and use cases

Step 2: Create custom roles

Create DEV custom role

Step 3: Granting access to the custom roles

Grant access of DATABASES to custom role

Grant access to WAREHOUSES to the custom role

Grant access to QA custom role

Grant access to DEV custom role in QA environment

Step 4: Create users (user onboarding)

Create users

Step 5: Assign custom roles to the users

Assign custom roles to the users

Step 6: Using the custom roles

Using custom roles created

Testing and validating access of the custom roles created

Step 7: Validating the access privileges

Using custom roles created

Understanding access hierarchy

Managing users and roles

User commands

CREATE command

DROP command

ALTER command

DESCRIBE command

SHOW command

Practical

Conclusion

Points to remember

Questions

Multiple choice questions

Answers

8. Data Protection and Recovery

Introduction

Structure

Objectives

What is data protection?

Implementing dynamic masking

Implementation commands

Create masking policy

Normal masking policy

Conditional masking policy

Alter masking policy

Drop masking policy

Show masking policy

Describe masking policy

Using dynamic masking

Step 1: Create custom role

Step 2: Grant custom role to a user

Step 3: Create masking policy

Step 4: Apply masking policy to the database objects

Step 5: Query data and validate masking policies

Understanding data recovery options

Understanding time travel

Time travel SQL extensions

Time travel data retention

Time travel setup

Time travel storage cost

Time travel storage for temporary and transient tables

Understanding Failsafe

Failsafe storage cost

Implementing time travel

Setup time travel at create table

Access historical data

Cloning objects

Commands to access dropped and restored objects

List dropped objects

Restore dropped objects

Reference use case

Implementing data replication

Database replication

Share replication

Replication group

Replication schedule

Business continuity

Practical: Create Snowflake policies, tagging objects

Exercises

Conclusion

Points to remember

Questions

Multiple choice questions

Answers

9. Snowflake Performance Optimization

Introduction

Structure

Objectives

Understanding Snowflake performance

Query performance

Understanding Snowflake metadata objects

INFORMATION_SCHEMA

Using *INFORMATION_SCHEMA*

ACCOUNT_USAGE

DATA_SHARING_USAGE

ORGANIZATION_USAGE

Introduction to *ACCOUNT_USAGE*

Dropped object information

Latency of the details

Data retention

Account usage views

Account usage functions

Reader account usage views

Roles and permissions

Introduction to *INFORMATION_SCHEMA*

Information schema views

Information schema functions

Calculating and understanding performance measures

Understanding QUERY_HISTORY

Introduction to Query Acceleration Service

SYSTEM\$ESTIMATE_QUERY_ACCELERATION function

QUERY_ACCELERATION_ELIGIBLE view

Enable query acceleration service

Query acceleration usage

Query acceleration billing

Implementing performance optimization

Query performance optimization

View historical performance in Snowsight

View query profile Snowsight

Listing queries for performance optimization

Warehouse performance optimization

Reducing queue time

Resolving memory spillage

Changing warehouse size

Optimizing cache

Limiting concurrent queries

Using query acceleration service

Storage performance optimization

Automatic clustering

Materialized views

Search optimization service

Choosing the right strategy

Performance considerations

Conclusion

Points to remember

Questions

Multiple choice questions

Answers

10. Understanding Snowflake Costing and Utilizations

Introduction

Structure

Objectives

Understanding Snowflake costing

Visibility

Control

Optimization

Component costing in Snowflake

Cloud services cost

Compute cost

Storage cost

Serverless cost

Data transfer cost

Using Snowflake metadata objects

View compute usage

View Cloud services usage

View serverless usage

Monitoring Snowflake costs

Conclusion

Points to remember

Questions

Multiple choice questions

Answers

11. Implementing Cost Optimizations

Introduction

Structure

Objectives

Components costing in Snowflake

Compute or virtual warehouse layer

Cloud services layer
Storage layer
Serverless services
Sample use case

Implementing cost optimization

Compute optimization

Access setup to warehouse

Setup the right size of the warehouse

Setup query time limit

Setup query queue time limit

Setup auto suspend and auto resume

Enforce spend limits

Storage optimization

Store only required data

Cloud services optimization

Implementing cost dashboards

Conclusion

Points to remember

Questions

12. Data Sharing

Introduction

Structure

Objectives

Data sharing needs

Implementing data sharing

How does data sharing work?

Implementing sharing with Snowflake consumers

Data sharing considerations

Data sharing commands

Data sharing pre-requisites

Creating a data share

Creating a data share with multiple databases

Using a data share

Implementing sharing with non-Snowflake consumers

Creating reader account

Understanding data sharing options

Listing

Marketplace listing

Private listing

Personalized listing

Data exchange

Providers tasks

Consumer tasks

Practical: Create Snowflake shares

Scenario

Technical requirement

Account setup

Data sharing

Conclusion

Points to remember

Questions

13. Data Cloning

Introduction

Structure

Objectives

Data cloning needs

Zero copy cloning

Implementing data cloning

Cloning considerations

Cloning implementation

Syntax of CREATE

Reference use cases and examples

Understanding the usage of cloned objects

Practical: Create Snowflake clones

Conclusion

Points to remember

Questions

14. Understanding Snowsight

Introduction

Structure

Objectives

Understanding Snowsight

Snowsight interface

Worksheet pages

Data pages

Dashboard page

Data page

Marketplace page

Activity page

Admin page

Implementing dashboards with Snowsight

Practical: Create Snowflake dashboards

Use case

Initial set up

Key asks

Conclusion

Points to remember

Questions

15. Programming Connectors and Drivers

Introduction

Structure

Objectives

Understanding programming connectors

Snowflake connector for Python

Snowflake connector for Kafka

Snowflake connector for Spark

Understanding drivers

JDBC driver and ODBC driver

GO driver

Understanding Snowsql: CLI

Installing Snowsql

Connecting to Snowsql

Using password to connect via Snowsql

Conclusion

Annexure

Points to remember

Questions

16. Workload Patterns with Snowflake

Introduction

Structure

Objectives

Understanding platform needs

Extract, Transform, and Load

Extract, Load and Transform

Implementing data solutions with Snowflake

Snowflake's features

Data solutions

Reference use cases and architecture

Data warehouse reference use case

Use case

Technical requirements
Business requirements
Proposed solution design
Reference architecture
Pipeline design
Platform design

Data lake reference use case

Use case
Technical requirements
Business requirements
Proposed solution design
Reference architecture
Pipeline design
Platform design

Data mesh reference use case

Use case
Technical requirements
Business requirements
Proposed solution design
Reference architecture
Platform design

Snowflake recommendations

Recommendations

Warehouse recommendations
Storage recommendations
Load recommendations
Transformation recommendations
Usage recommendations

Conclusion

Points to remember

Questions

17. Introduction to Snowflake's Advance Features

Introduction

Structure

Objectives

Understanding new features and releases

Snowflake releases

New features

SQL extensions

Schema features

SQL features

Understanding new types of tables

Dynamic tables

Event tables

Hybrid tables

Iceberg tables

Introduction to new Snowsight features

Worksheets

Data governance

Native apps

Budgets

Data loading

Stage options

Introduction to new LLM capabilities

Conclusion

Points to remember

Index

CHAPTER 1

Getting Started with Snowflake

Introduction

The first chapter is the foundation of the book and covers the history of the Snowflake, the need to implement Snowflake, and getting started with Snowflake. This chapter guides to setting up a trial or demo account to be used for various hands-on activities covered in subsequent chapters. This also covers the overview of Snowflake certifications and various community events.

You will learn more about Snowflake data platform capabilities in upcoming chapters of this book. Every chapter consists of real-time use cases and references to explain each concept. We have also provided lab questions with every chapter to help you learn with examples. There are a set of questions to check your knowledge at the end of each chapter.

This book is uniquely designed and well-written to help you understand Snowflake's features easily. You will learn Snowflake seamlessly as you read through this book. With every concept and practical lab in this, you will surely want to plan for certification.

Structure

This chapter consists of the following topics:

- Why Snowflake?
- History of Snowflake
- Snowflake certifications

- Snowflake community
- Setting up a trial account with Snowflake
- Connecting to Snowflake

Objectives

By the end of this chapter, you will be able to understand the need for a Snowflake data platform over traditional or enterprise platforms. You will also be able to set up a trial account for yourself. This trial account will be used to perform various exercises throughout this book.

Why Snowflake?

In the earlier era setting up a data platform needed tedious efforts to design, derive the capacity of the system, and purchase hardware-software or appliances to set up an ecosystem to support the data needs. These ecosystems had their own limitations in terms of scaling – horizontal vs vertical, cost-efficiency, performance efficiency, operational cost, and huge maintenance cost associated-support teams, upgrades, patches, EOLs activities, and so on.

The cloud has broken down almost all these limitations and taken over the majority of legacy ecosystems to the cloud. The cloud offers scalability, efficiency, low operations, and no or low-cost maintenance. Cloud-native and SaaS services replaced the legacy ecosystems however, had their own limitations or dependencies or locking with vendors. For example, if you are using Google Cloud Bigquery – you are tied with using it only for Google Cloud; though this can be integrated with any other platforms available. There are also limitations associated with most cloud-native or managed services in terms of the type of workload they support. Data Lake, Data Warehouse, data analytics, and data science are often treated as separate workloads and designed differently integrating with each other. There are various data platforms available in the market to support data workloads as per business requirements.

Snowflake eliminated the need to design and define data workloads separately. With Snowflake, you can use the same data platform to cater all types of workload needs - Data Lake, Data Warehouse, data analytics, and data science needs. This also caters to a workload where you can combine analytical and transactional workloads: **Unistore**.

You need to learn about Snowflake and its unique offerings, as they are one of the leading data platforms in the market. You can learn Snowflake and get started with your data career journey or change your career path as well. It will be beneficial to learn Snowflake, considering the ample opportunities available in the market and Snowflake's adaptability.

History of Snowflake

Snowflake is a one-stop solution for a variety of workloads. Snowflake's all-in-one platform enables organizations to quickly set up a centralized data platform. This platform can be used to generate values from the data stored within, extending implementations to various applications with data protection, security, and compliance.

This is the journey that started in 2012 when the founders met for the first time with a vision of building a data warehouse for the cloud from scratch to unlock the potential of unlimited analytics from heterogeneous data. They had aimed to build a solution that is not only secure and impactful but also cost-effective and easier to manage.

Within three years, Snowflake's data warehouse – built from scratch on the cloud – was available in 2015. Snowflake's unique, cloud-agnostic architecture disrupted the data warehousing market. With Snowflake, data engineering also changed from technical to business-oriented implementations. This has made data analytics simpler, that helps users generate data stats which in turn helped organizations make data-driven decisions.

In another three years, in 2018, Snowflake introduced – data sharing. This is the most critical feature that is used to share data with internal or external stakeholders with appropriate access controls and security. Interestingly, you can also share data with Snowflake as well as non-snowflake users. You will learn more about this in [*Chapter 12: Data Sharing*](#).

In 2021, Snowflake announced its expansion to support the wider category of data engineering with Snowpark. This is a new development framework which is a unique combination that makes it simpler to design and develop data engineering workloads on Snowflake. This can also be used to extend support to data science capabilities. You will learn more about this in [*Chapter 6: Understanding Snowpark*](#).

Also, in 2021, Snowflake announced Snowflake organizations. With this, it is very easy to manage multiple accounts for the same customer. You can tag

various accounts that are active and required under an organization. You can also set up utilization and usage at the account as well as organization level for tracking.

In 2022, Snowflake added the Security data lake. This is a type of workload that enables the full visibility of security logs. Snowflake's newest workload: **Unistore**, is a very unique platform where you can combine the power of transactional and analytical operations.

Snowflake certifications

Snowflake offers basic and advanced-level certifications. Earning a certification badge definitely adds more weight to your profile.

Snowflake SnowPro Core is the first level of the foundation certification exam. Snowflake offers five advanced certifications based on your role. You can refer to *Figure 1.1* for basic and advanced certifications available:

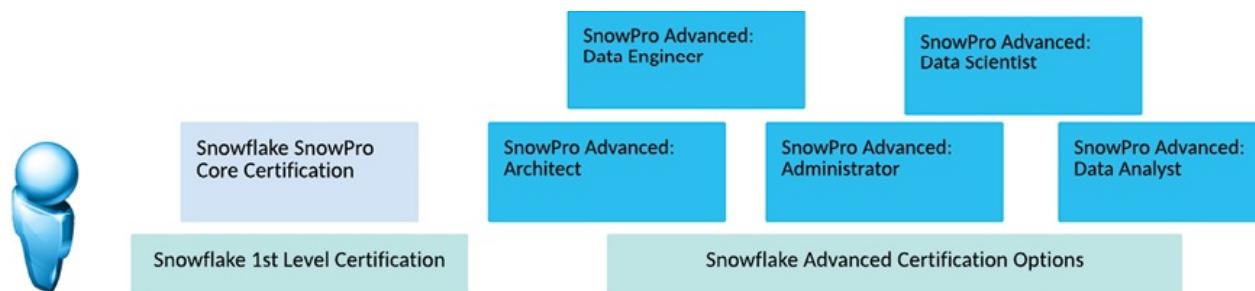


Figure 1.1: Snowflake certifications

You will need to complete the foundation - SnowPro Core Certification before you appear for the advanced certifications. Once you pass the certification, it is valid for two years, and you can appear for a re-certification exam to renew your certification for another two years. An advanced certification will automatically renew your SnowPro Core Certification for the next two years. Advanced certifications are available for various roles – Data Engineer, Architect, Administrator, Analyst, and Data Scientist. You can appear for the advanced one based on your role and experience. You can find more details about certifications here: <https://www.snowflake.com/certifications/>.

Snowflake community

Snowflake runs various community initiatives. This is one of the most active

communities where many users can contribute and connect with other community members.

There are Snowflake user groups that you can join if you are interested in connecting and joining. User groups held events in person as well as virtually across various regions. You can find more details here:

<https://usergroups.snowflake.com/>.

Snowflake runs the Data Superheroes community program every year. This program is for Snowflake experts who are highly active in the community. The active contributors are recognized as Data Superheroes, and Snowflake announces members of this elite group at the beginning of the year. You can learn more about this program here: <https://medium.com/snowflake/all-you-need-to-know-about-snowflake-data-superheroes-a36914e2e614>. Refer to the following figure:



Figure 1.2: Snowflake data superheroes

You can start contributing to the community as an individual user or through your partner account. There are various trainings available on the community portal as well as the partner network portal. This book is one guide to learning and getting started with Snowflake. Once you understand the concepts and complete practical labs from the book then you can refer to the quick starts to navigate through data engineering, data lakes, and other types of workload-

specific use cases.

You can also use Snowflake's documentation to get started with Snowflake. There are also quick labs available that enable users to perform hands-on labs based on the workloads and learn Snowflake. You can find use case-specific hands-on labs here: <https://quickstarts.snowflake.com/>.

Setting up a trial account with Snowflake

Snowflake offers a 30-day trial account that is worth \$400 to practice, perform hands-on labs, and learn easily. Snowflake offers three versions on all 3 public cloud platforms that users can choose while setting up a trial version.

You can follow these steps to set up a trial account:

1. Open the link: <https://signup.snowflake.com/>.
2. Fill in these details in the signup form specified as shown in *Figure 1.3*:
 - a. **First name:** Your first name
 - b. **Last name:** Your last name
 - c. **Email:** You can specify your office email or personal email to set up an account.
 - d. **Company:** Provide your company name
 - e. **Role:** Provide your role with your organization
 - f. **Country:** Specify your country of residence.

Start your 30-day free Snowflake trial which includes \$400 worth of free usage

First Name*

Last Name*

Email*

Company*

Role*

United States

By clicking the button below you understand that Snowflake will process your personal information in accordance with its [Privacy Notice](#)

CONTINUE

or sign in to an existing account

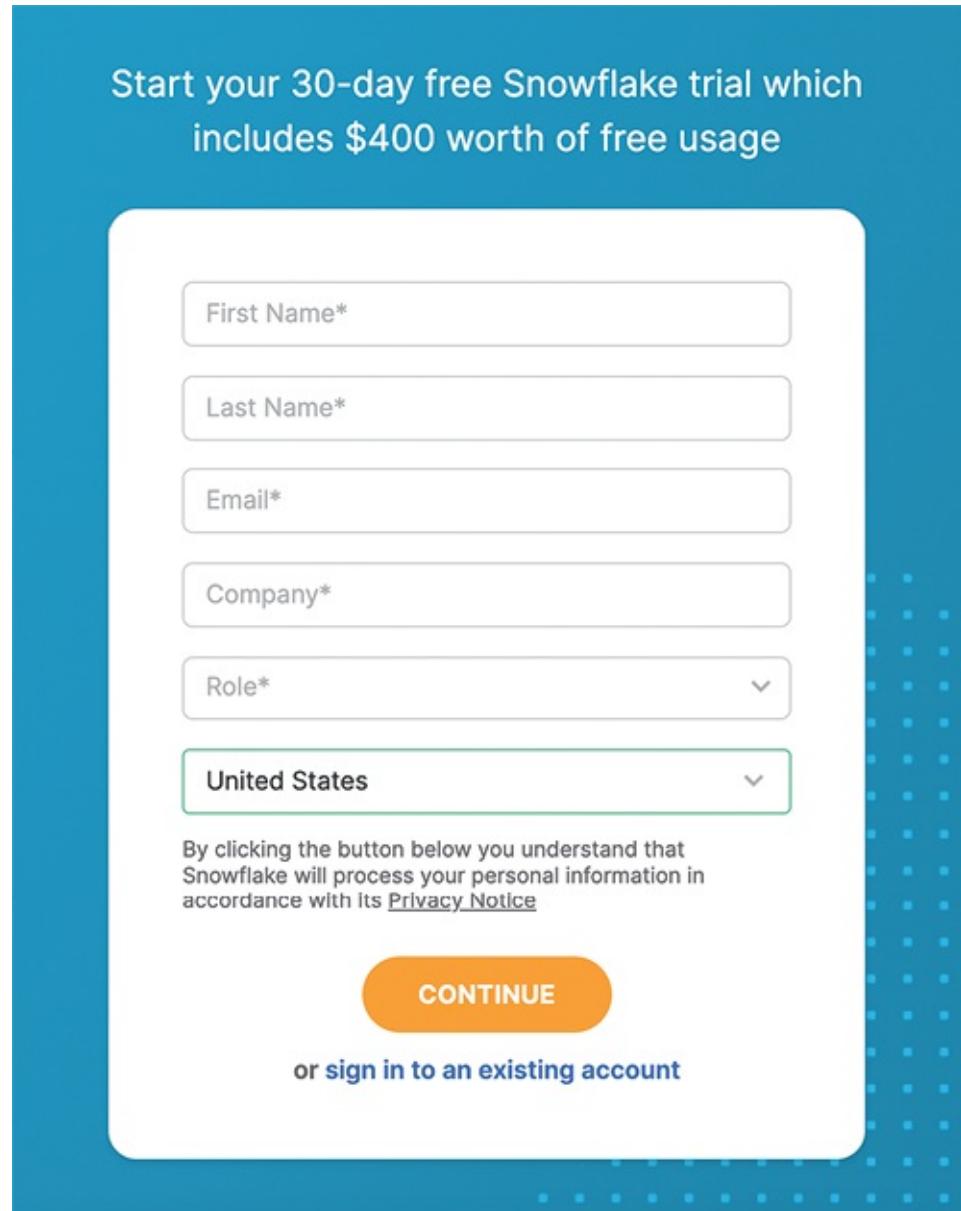
The image shows a screenshot of the first page of a Snowflake trial account sign-up form. The background is teal. At the top, it says "Start your 30-day free Snowflake trial which includes \$400 worth of free usage". Below that is a white rounded rectangular form. It contains five input fields: "First Name*", "Last Name*", "Email*", "Company*", and "Role*". The "Region" field, which has "United States" selected, is highlighted with a green border. Below the form is a paragraph about privacy and a link to the "Privacy Notice". At the bottom is an orange "CONTINUE" button and a link to "sign in to an existing account".

Figure 1.3: Snowflake Trial account sign-up form page 1

3. Click on **CONTINUE** to select the Snowflake version, cloud provider, **Region** and tick the checkbox to agree on terms and conditions before you click on **GET STARTED** as shown in the following screenshot:

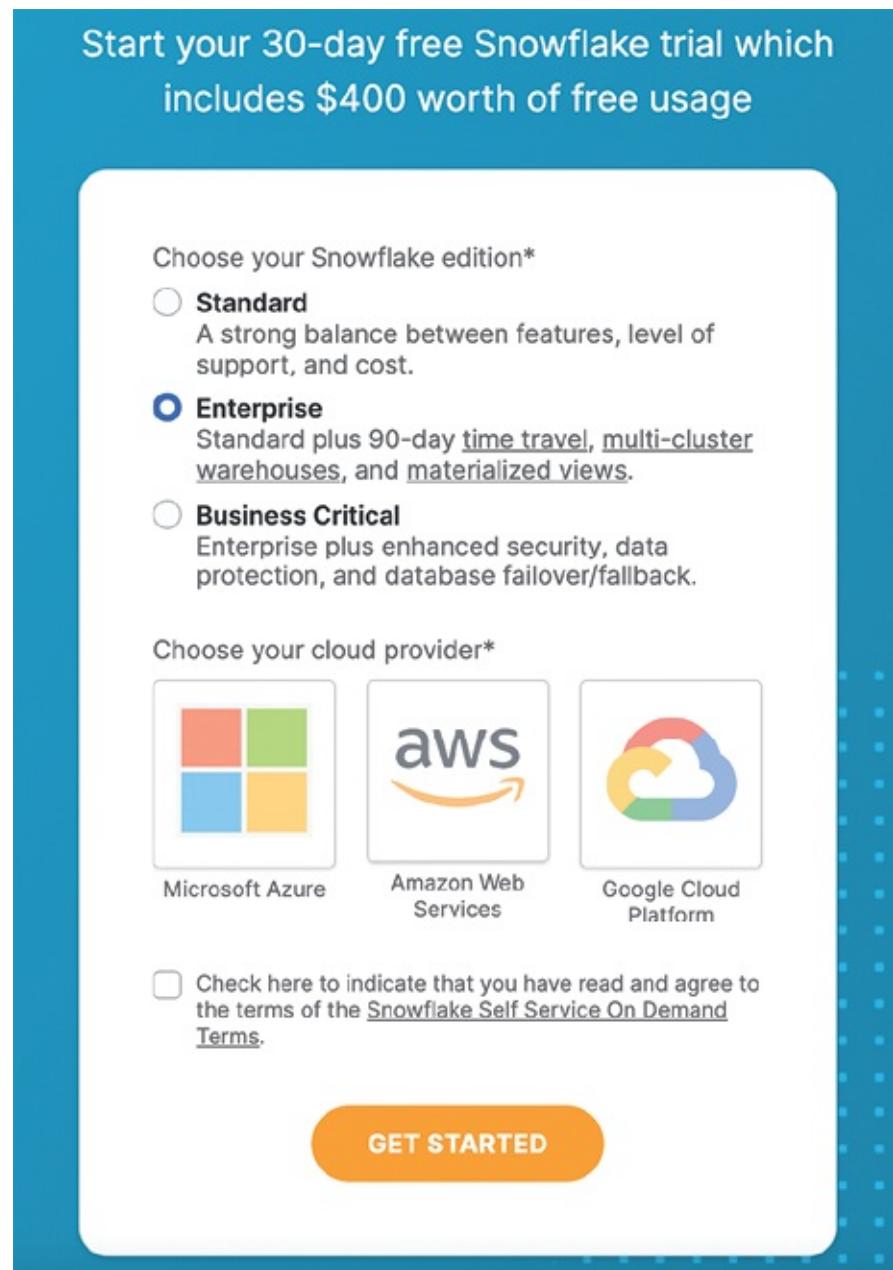


Figure 1.4: Snowflake Sign up form Page 2

- Once you click on **GET STARTED**, this creates an account link to connect to the trial account. Snowflake account is accessible via a URL. You have to save this URL and your credentials to connect to the trial account every time you want to use it.

Your Snowflake account link will look like this:

<https://<accountnumber>.snowflakecomputing.com/console/login#/> (this is for AWS-based accounts). For other cloud accounts, GCP and Azure will be mentioned along with Snowflake computing to distinguish the providers.

Connecting to Snowflake

Once the trial account is setup, you can copy the Snowflake account URL to connect to the instance. You can connect to Snowflake by using a CLI as well as Web UI. Snowflake offers two versions of Web UI – Classic Console and Snowsight. SnowSQL is the CLI used to connect to Snowflake via command prompt. You will learn more about connecting, installing, and setting up connection to Snowflake in upcoming sections.

Snowflake Web User Interface

There are two different web **User Interfaces (UI)** available: Classic Console and Snowsight. Currently, users can use both, or choose one of the UI as default.

Classic console

This is the classic web UI users used until Snowsight was launched. Classic UI has a bunch of features, attributes, and functionalities. The classic UI looks like *Figure 1.5* once you log in:

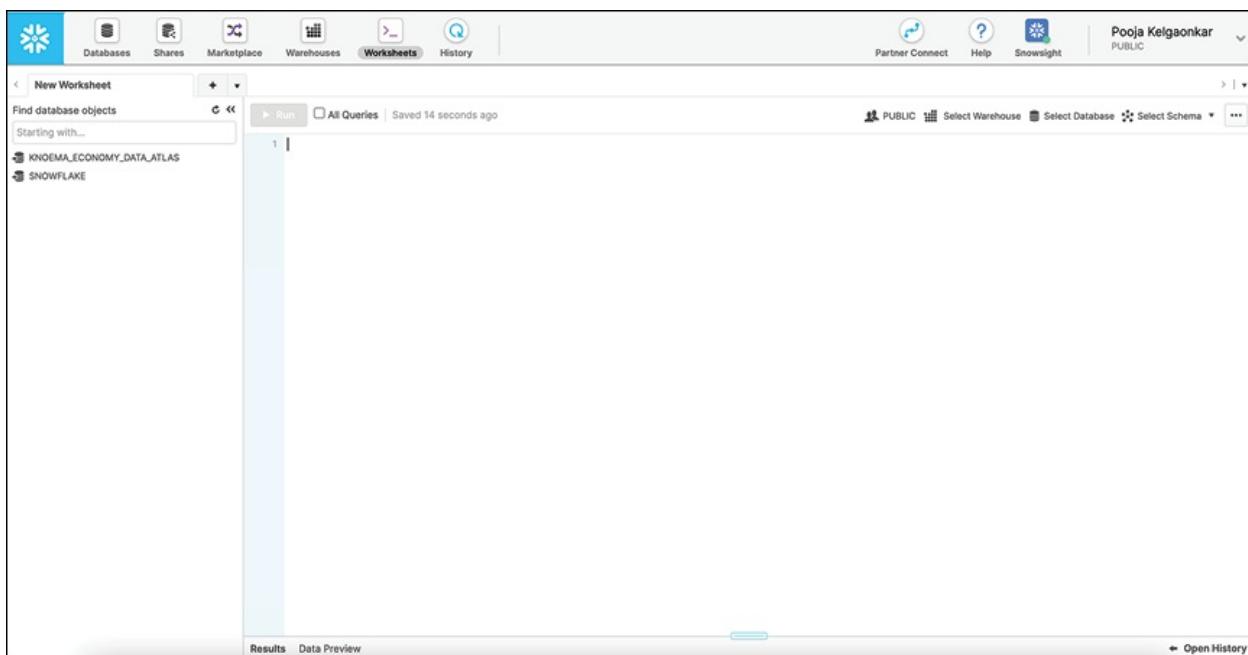


Figure 1.5: Classic Console

The top portion with options or buttons is referred as the ribbon as shown in the following screenshot:



Figure 1.6: Classic Console – Ribbon

The ribbon options vary based on the role selected for the session. Each worksheet is referred to as a session with Snowflake. Users can set the context separately for each session which can be different for each session. Context can be set up using console options as well as SQL commands. You will learn more about this in the upcoming chapters.

Context setting

You can set up context from the console using the left-hand side drop as shown in *Figure 1.7*:

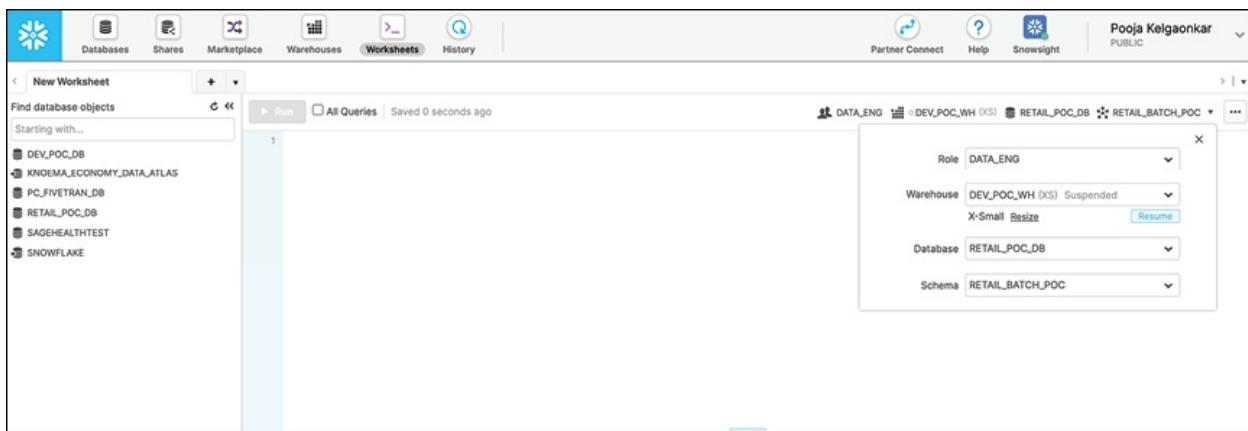


Figure 1.7: Context setting on Classic Console

You can also use SQL commands:

Use `role DATA_ENG;`

Use `warehouse DEV_POC_WH;`

Use `database RETAIL_POC_DB;`

Use `Schema RETAIL_BATCH_POC;`

Run them on the worksheet to set your session context. This is applicable to only the current active session, that is, the worksheet opens in this classic console as shown in *Figure 1.8*:

The screenshot shows the Snowflake Snowsight interface. At the top, there's a navigation bar with icons for Databases, Shares, Marketplace, Warehouses, Worksheets (which is selected), and History. On the right, it shows the user 'Pooja Kelgaonkar' and the role 'PUBLIC'. Below the navigation is a search bar and a sidebar titled 'New Worksheet' with a 'Find database objects' dropdown. The main area contains a query editor with the following SQL code:

```

1 Use role DATA_ENG;
2 Use warehouse DEV_POC_WH;
3 Use database RETAIL_POC_DB;
4 Use Schema RETAIL_BATCH_POC;

```

Below the query editor is a results panel with tabs for 'Results' and 'Data Preview'. The results show a single row with the status 'Statement executed successfully.' The results table has columns for 'Query_ID' (with a green checkmark), 'SQL' (with the query text), '61ms' (execution time), and '1 rows' (number of rows). There are also 'Filter result...', 'Copy', and 'Columns' buttons.

Figure 1.8: Context setting using SQL commands

Snowsight

Snowsight is the new web UI, introduced in 2021. This is also the default UI created for new accounts. This will soon become the default web UI and the Classic console will be deprecated.

Snowsight consists of all the features, and functionalities offered by Classic Console with many new features added. Snowsight has a separate navigation menu on the left-hand side as shown in [Figure 1.9](#):

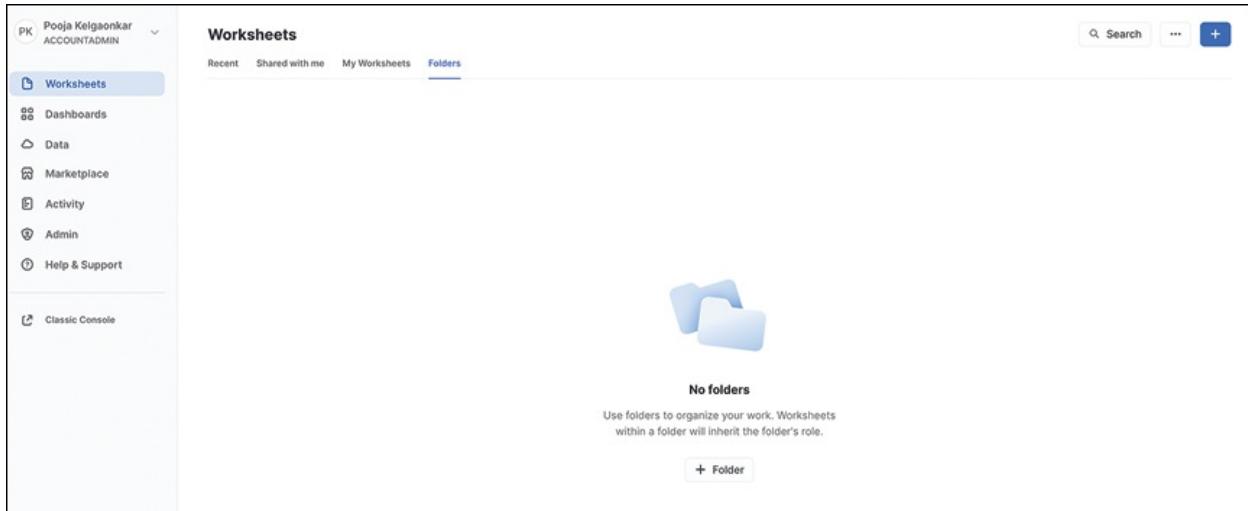


Figure 1.9: Snowsight Web UI

The navigation menu has the following options:

- **Worksheets:** This panel shows all the worksheets created, shared by users. You can also organize them in the folders. Classic does not have this feature to manage and share worksheets with other developers or users.
- **Dashboards:** This panel shows all the dashboards created and shared by you and other users. This is a new feature of Snowsight using which you can create dashboards leveraging data within Snowflake tables.
- **Data:** This panel lists down all the databases, tables, and objects created within your account based on your access role setup and use.
- **Marketplace:** This is the data marketplace, the same as the Classic console.
- **Activity:** This panel consists of **Query history**, **Copy history**, and **Task history** details that can be easily accessed from here.
- **Admin:** This panel is accessible to the account admin or with required privileges. This consists of warehouses, accounts, users, roles, usages, billing, and partner connect.

You can use the class console on the left panel to open the classic console.

Command line interface

Snowflake supports accessing and connecting using a **Command Line Interface (CLI)**. You can use Web UI as well as CLI to connect and perform required operations, develop, and test your data pipelines.

Web UI is configured by default and made accessible to users as soon as they setup an account. However, users need to download CLI and setup to connect to the Snowflake.

Download and setup SnowSQL

You can download and install SnowSQL on your local. You can follow these steps to setup CLI:

1. Download CLI from Web UI | **Help** | **Download**, as shown:

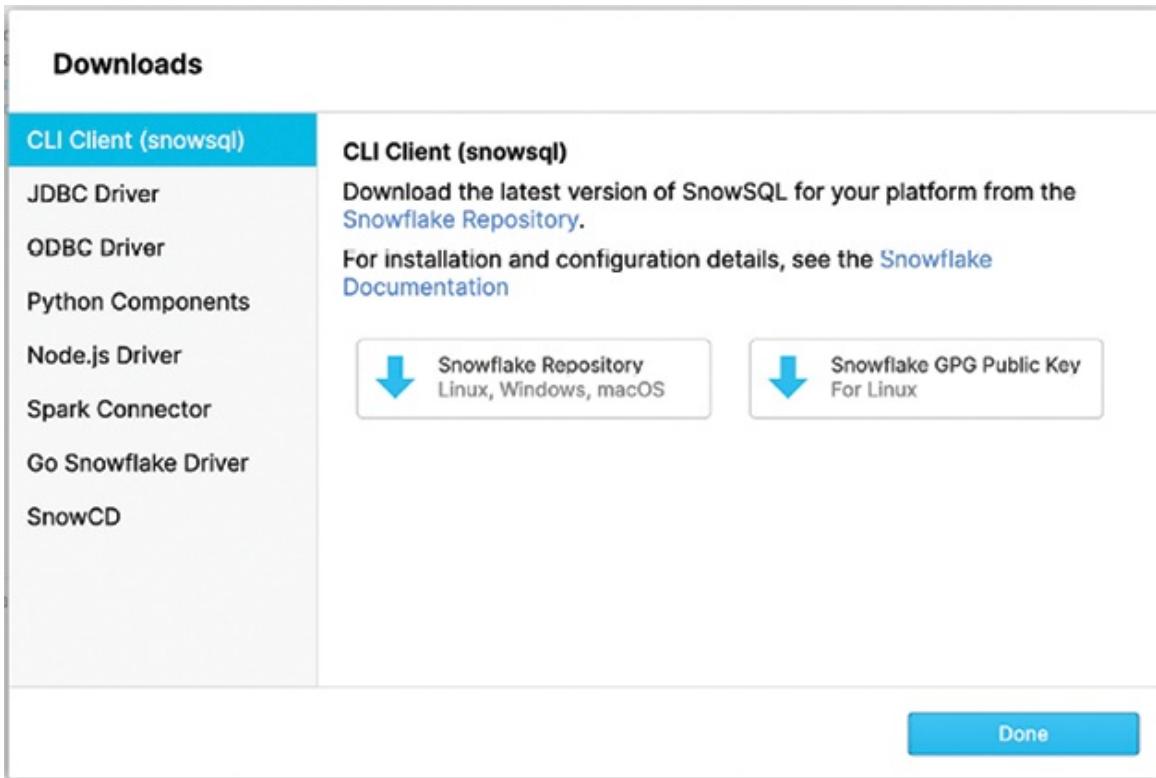


Figure 1.10: Download window

2. You can also download it from your Windows or Linux command line. Use the `curl` command from the prompt to download the packages and install them.

Connecting to Snowflake

You can use `snowsql` command to connect to Snowflake. Refer to the below details to know more about the command and parameters required to be set up while configuring the connection:

`snowsql <connection_parameters>`

Parameters and their description are showed in the following table:

Parameter	Description
<code>-a, --accountname TEXT</code>	Account identifier
<code>-u, --username TEXT</code>	Username to connect to Snowflake
<code>-d, --dbname TEXT</code>	Database to use for the session
<code>-s, --schemaname TEXT</code>	Schema in the database to use for the session

<code>-r, --rolename</code> TEXT	Role name to use for the session
<code>-w, --warehouse</code> TEXT	Warehouse to use for the session
<code>-o, --option</code> TEXT	Set SnowSQL options
<code>-f, --filename</code> PATH	File to execute – SQL commands in file and use file from command line to be executed
<code>-q, --query</code> TEXT	Query to execute – provide SQL to be executed
<code>--config</code> PATH	Path and name of the SnowSQL configuration file. By default, <code>~/.snowsql/config</code> .
<code>-P, --prompt</code>	Forces password to be provided interactively. This can be different from default config
<code>-c, --connection</code> TEXT	Named set of connection parameters to use
<code>--single-transaction</code>	Single session
<code>--private-key-path</code> PATH	Path to private key file

Table 1.1: SnowSQL connection parameters

Command to connect to Snowflake:

```
snowsql -a <account_identifier> -u <user> -d <testdb>
-s <testschema> -P
```

Once you run this command, it will prompt for a password and allow you to connect to Snowflake for an interactive session. As you know, specifying and storing passwords is critical. You can specify the **connection** profile as a part of the **config** and provide it while connecting to Snowflake. You can use the following mentioned connection **config** and setup:

```
[connections]

#accountname = <string>      # Account identifier to
connect to Snowflake.

#username = <string>          # Username in the account.
Optional.

#password = <string>           # User password. Optional.

#dbname = <string>            # Default database.
Optional.
```

```
#schemaname = <string>      # Default schema. Optional.  
#warehousename = <string> # Default warehouse.  
Optional.  
#rolename = <string>        # Default role. Optional.  
#authenticator = <string> # Authenticator:  
'snowflake', 'externalbrowser' (to use any IdP and a  
web browser), https://<okta_account_name>.okta.com  
(to use Okta natively), 'oauth' to authenticate using  
OAuth.
```

Note: Password stored in plain text in the **config** file. You should secure the file to have limited access. For example, in Linux or macOS, you can restrict the permissions by running **chmod**.

This is a sample example of a **connection** profile:

```
[connections.sample_test_connection]  
accountname = testorg-testaccount  
username = testuser  
password = xxxxxxxxxxxxxxxxxxxxxxxx  
rolename = DATA_ENG  
dbname = snowdbname  
schemaname = public  
warehousename = test_wh
```

You can save this connection profile and use this command to connect to Snowflake:

```
$ snowsql -c sample_test_connection
```

Conclusion

This chapter provided a summary of Snowflake's unique offering along with a brief history of its origin and fundamental concept used to build it from scratch on cloud. This introductory chapter helped you to get introduced to Snowflake certifications and community events. Most importantly, you learnt to setup a trial

account and configure connections to use this account for labs in this book. You will learn more about Snowflake as you go along with the next chapters in this book.

In the next chapter, you will learn about the traditional platforms and their challenges. You will also learn how Snowflake helps overcome traditional challenges with its unique three-layer architecture. This is the foundational chapter to getting started with the technical aspect of Snowflake by understanding three layers of architecture and services.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake is built from scratch on cloud.
- Snowflake is a data platform that offers designing various workloads like data warehouse, data lake, data analytics and many more.
- Snowflake offers variety of certifications depending on the role.
- Snowflake offers a trial account with \$400 credits to get started for free.
- Users can setup one or multiple accounts with their choice of cloud provider – AWS, GCP or Microsoft Azure.
- You can connect to Snowflake instance as a Web URL and use Classic console or Snowsight Web UI.
- You can also connect to Snowflake through programming interface over a CLI using SnowSQL.

Practical

Set up a Snowflake trial account with these options:

- Cloud – AWS
- Region – US/Canada/ASIA

Version: Enterprise

Multiple choice questions

1. What kind of workloads are supported by Snowflake?

- a. Data Lake
- b. Data Warehouse
- c. Both A and B
- d. Only analytical workloads

2. Select what best describes Snowflake:

- a. Cloud-agnostic
- b. Built from Scratch on the cloud
- c. MPP on Cloud
- d. Extending Cloud capabilities to the data platform

3. When was Snowflake founded?

- a. 2018
- b. 2015
- c. 2021
- d. 2012

4. How many advance certifications are available with Snowflake?

- a. 2
- b. 5
- c. 4
- d. 3

5. Select TRUE | FALSE

Snowflake SnowPro core is the first level certification required to enroll for advanced certification

- a. TRUE

b. FALSE

Answers

1	c
2	b
3	d
4	b
5	a

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 2

Three Layered Architecture

Introduction

This chapter helps the reader get started with Snowflake with a detailed view of the architecture. This covers the various data platform architecture challenges and how Snowflake helps to overcome most of these challenges. This chapter defines three layers of architecture, their distinguishing features, and their setup.

Structure

This chapter consists of the following topics:

- Traditional data warehouse challenges
- Legacy data warehouse challenges
- Typical data lake challenges
- Snowflake architecture
- Cloud services layer
- Virtual warehouse layer
- Storage layer
- Cache in Snowflake

Objectives

By the end of this chapter, you will be able to understand the Snowflake architecture and its three layers of architecture – storage, compute, and cloud services. Snowflake's unique architecture makes this data platform distinguishable from other cloud-based offerings. You will also be able to relate and understand common data challenges and how Snowflake is designed to overcome these challenges.

Traditional data warehouse challenges

The data warehouse is a repository of the data that is used for analysis, reporting, and dashboards. It is designed to be used for decision-making. Data residing in the warehouse can be used to generate statistics that help drive the business and make decisions based on the data available. There are various types of analysis and reporting available, such as prescriptive and predictive analytics.

This centralized data warehouse is developed over a period by sourcing, transforming, and loading data from various sources. This serves as a **Unified Data Platform (UDP)** or **Enterprise Data Warehouse (EDW)** for various stakeholders, internal as well as external. The data sourcing and consumption are dependent on the type of sources, frequency, accessibility, type of consumers, and so on. The data warehouse can be designed to store and organize data into various subject areas as per the data domain, such as sales, finance, marketing, and so on.

There are various vendors that offer data warehouse solutions like Teradata, IBM DB2, Netezza, and so on. These platforms offer support for setting up warehouses, and types of data, structured and semi-structured. This platform supports ANSI SQL standards and users can query data using SQL queries. Queries accessing, and processing data from the platform needs to be optimized all the time to ensure required performance and meeting SLAs.

The data warehouse solutions follow SMP or **Massively Parallel Processing (MPP)** implementations and have Shared Disk, Shared Nothing, and Shared Memory architecture. These are part of traditional legacy data warehouse systems which are different than Snowflake implementation. You can refer to traditional architectures in *Figure 2.1*:

Traditional Data Systems or Warehouse Architectures

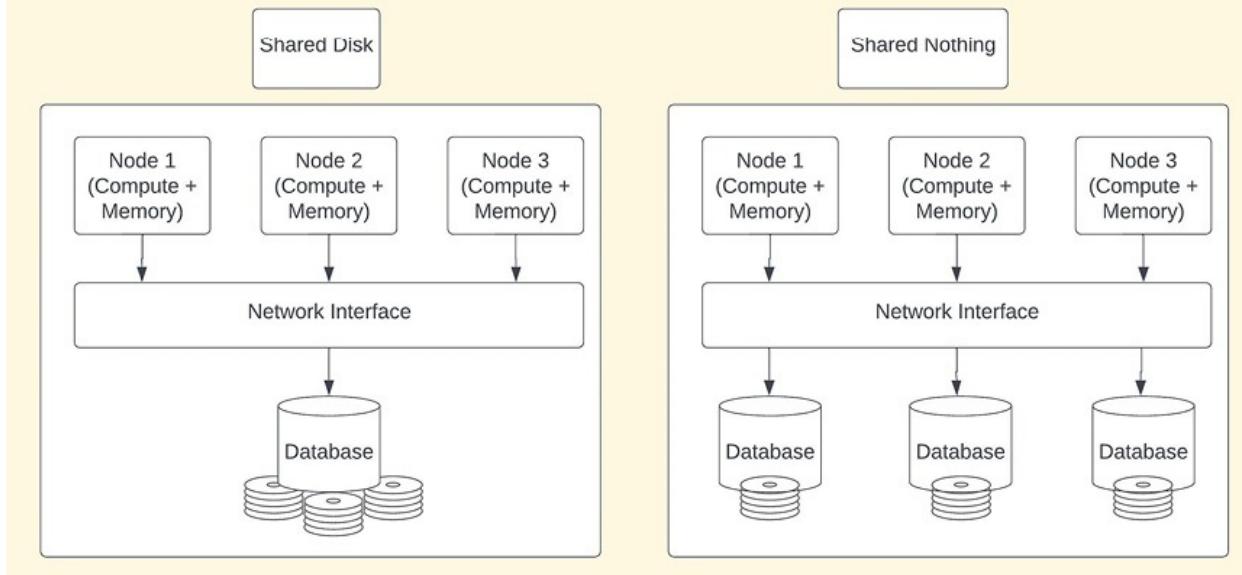


Figure 2.1: Shared Disk and Shared Nothing Architecture

Shared disks or database is an OLTP kind of implementation where the single disk is shared across various loads/nodes in the system. On the other hand, shared-nothing architecture is widely used implementation for Data Warehouse (OLAP) implementations, like Teradata, Netezza, and so on. Storage and compute are tightly coupled in these implementations. In case of data growth, you must purchase additional nodes to increase the size, capacity, and performance of the warehouse. You cannot increase the compute (CPU) and storage separately.

Legacy data warehouse challenges

The legacy data warehouse systems have typical challenges that include:

- **Infrastructure:** There is limited or provisioned or pre-purchased Compute and Storage capacity.
- **Data sharing:** Sharing data with consumer applications and setting up processes to share data in the form of file feed or data feed.
- **Consistency:** Most of the data warehouse systems have eventual consistency in case of consumer application data availability or data replication.
- **Data security:** Data classification, data masking, encryption, and security.

- **Operations and maintenance:** Greater maintenance cost and operational dependencies.
- **Technology limitations:** Requires special skills and knowledge of tools and technologies to develop products and features. Skilled resource dependencies.

Typical data lake challenges

Like data warehouse, you can also implement a data platform such as data lake. Data lake implementation has been common after Hadoop and Bigdata offerings. Many legacy warehouse systems migrated their workloads to data lakes or used data lakes to store archived data as well as monthly roll data.

With data lake implementation, processing, storing, and analysis have been efficient with commodity hardware used to set up the Hadoop ecosystem or clusters. If you compare warehouse appliance offerings with the Hadoop ecosystem using commodity systems, then you can easily understand the cost and scalability difference between these two. Implementing a data lake using Hadoop is MPP architecture where you can add and remove nodes for processing and the procurement may not be costly or time taking in comparison with appliance offerings. Refer to the following figure for a better understanding about the setup:

Data Lake - Edge Nodes & Data Nodes Setup

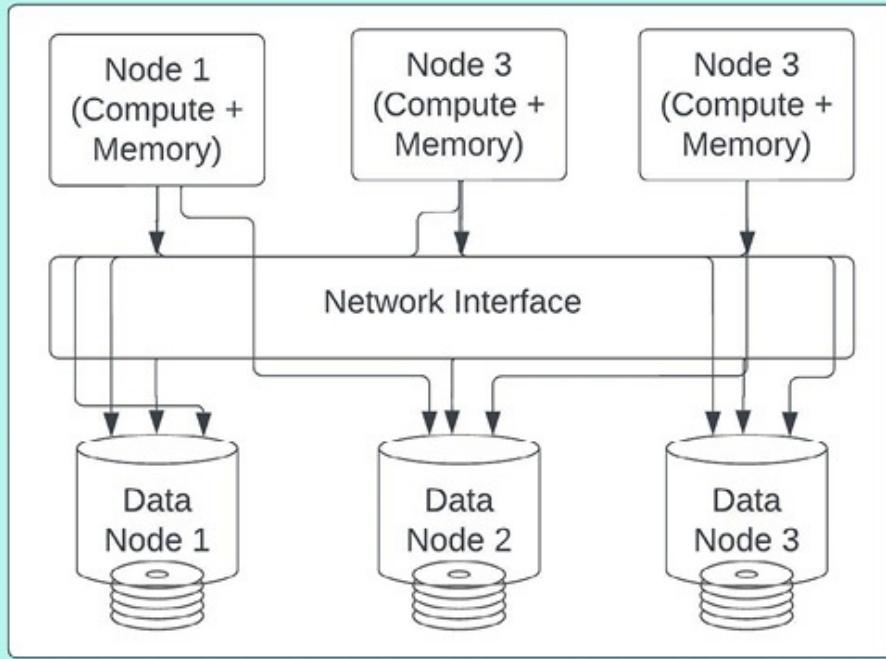


Figure 2.2: Data Lake – Hadoop Setup

In the case of legacy data lake implementation using commodity hardware, data nodes are connected to all edge nodes. Data can be accessed and shared between nodes.

Note: Data node represents an instance or machine which stores the data and have volume disks attached to it. Edge node represents an instance or machine that connects to front end applications or end users. These provide access to the processing and accessing data across data nodes.

Hadoop ecosystem

Hadoop is one of the implementations of data lake. Like data warehouse systems, data lake also brings up data from various sources and processes. A typical data lake architecture may not look different than the warehouse architecture. It has the following components:

- **Source layer:** Data sourced from various sources.
- **HDFS layer:** Files or storage layer.
- **Processing layer:** Transformations, processing and engineering pipelines.
- **Data marts:** Tables, data as target or consumer layer to be shared with

consumer applications.

- **Consumer layer:** Applications, end users, and business users who can access data from data marts and lakes using APIs or SQL queries for reporting, dashboarding, or end-user applications.

The following figure illustrates the Hadoop ecosystem:

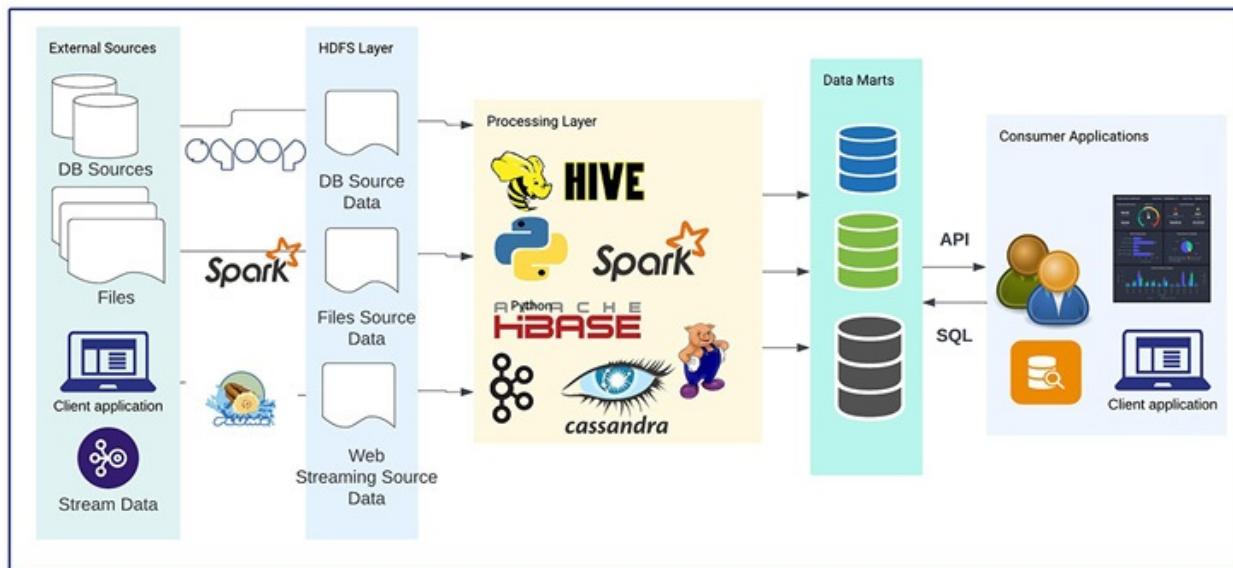


Figure 2.3: Hadoop Ecosystem

Though, the overall cost of the data lake allows users to add nodes to the cluster, it still has issues to maintain, operationalize, upgrade, and patching activities, technology dependency, and special skills to develop data engineering workloads – spark, map-reduce, Hive, Sqoop, and so on.

Data lake systems follow the shared-nothing architecture. Data nodes and edge nodes are defined to manage workloads. These systems also enabled users to bring in unstructured data – columnar, documents, images, videos, and so on. This platform has the capability to implement predictive analytics using machine learning models.

Extended support to NoSQL

Hadoop ecosystem also supports NoSQL solutions. NoSQL is where users can store nonrelational data and do not need specific schema. NoSQL is referred as Not Only SQL which is a great choice to store unstructured data like web or log data, social media data or images, videos or geospatial data.

Based on the type of data to be stored, NoSQL is categorized into four types – document store, columnar store, key-value store, and graph databases. Document store can be implemented using MongoDB where documents are stored in the form of JSONs. Key-value stores are the databases where information is stored in the form of key-value pairs. DynamoDB can be used to implement key-value stores. Columnar store is the type of database where data is stored in the form of columns and column families in place of rows and columns. There is no schema enforcement in case of column families. Each column family defined can have separate columns and store data accordingly. HBASE and Cassandra are examples of columnar implementations. Graph databases are where data is stored and related in one-to-many and many-to-many relations that can be represented in the form of graphs. Neo4j is used for graph database implementation.

NoSQL can be used based on the type of data and formats required for processing. However, they may face performance issues in case of any analytical processing required on top of these datasets. Users may not be able to use ANSI SQL standards or SQL queries to query data in NoSQL. There are specific skills required to access data.

Note: Most of the NoSQL can be used for data scientists to implement data science use cases. These are not replacement to data warehouse or data lake however these databases can be part of data lake or warehouse implementations.

Following are the benefits of data lake over data warehouse:

- **Reduced infrastructure cost:** Cost-effective cluster implementation.
- **Additional support to the data:** Unstructured data supported.
- **Extended support to predictive analytics:** Support to machine learning implementations.

Despite additional benefits over legacy data warehouse systems, data lake implementations using legacy ecosystem components have their own limitations and challenges. The legacy data lake systems have typical challenges are similar to data warehouse limitations that includes:

- **Infrastructure:** There is limited or provisioned or pre-purchased Compute and Storage capacity – fixed scalability up to available nodes.
- **Data sharing:** Sharing data with consumer applications and setting up processes to share data in the form of file feed or data feed.

- **ANSI SQL:** Hive supports most of the DDLs and DMLs. However there are limitations based on the ecosystem setup. Supporting OLTP and OLAP may not be as simple as it is with data warehouse or database systems.
- **Consistency:** Most of the data warehouse systems have eventual consistency in case of consumer application data availability or data replication.
- **Data security:** Data classification, data masking, encryption, and security.
- **Operations and maintenance:** Greater maintenance cost and operational dependencies.
- **Technological limitations:** Requires special skills and knowledge of tools and technologies to develop products and features. Skilled resource dependencies.

Most of the cloud-based solutions overcome these data platform challenges. However, they also impose their own limitations or support specific implementations. Snowflake is a data solution designed and developed from scratch on the cloud. Snowflake supports various workloads such as data warehouse, data lake, and lakehouse. Snowflake is a unique platform because of its architecture.

Snowflake architecture

Snowflake is a **Software as a Solution (SaaS)** offering that is designed and built from scratch for cloud platforms. Snowflake supports all 3 major cloud providers – AWS, GCP and Azure. Snowflake supports various data workloads like data warehousing, data lake, data applications, data exchange, and Unistore solutions on the cloud. You can set up the Snowflake data platform on your choice of hyperscalar, and it can be integrated with the rest of the two cloud platforms. The workloads are:

- **Data warehousing:** This workload represents data warehousing solution where you can setup Snowflake as an **Enterprise Data Warehouse (EDW)**.
- **Data lake:** This workload represents data lake solution implementation with Snowflake.
- **Data applications:** This represents implementations of data applications

leveraging Snowflake native features.

- **Data exchange:** This represents data sharing capabilities with Snowflake and non-Snowflake consumers.
- **Unistore:** This represents data processing and analytical capabilities. You can run transactional as well as analytical loads, and operations on defined objects.

Snowflake's processing engine is native SQL. You will learn more about these workloads, support to application needs in upcoming chapters. The following figure represents various workloads of the Snowflake data platform:

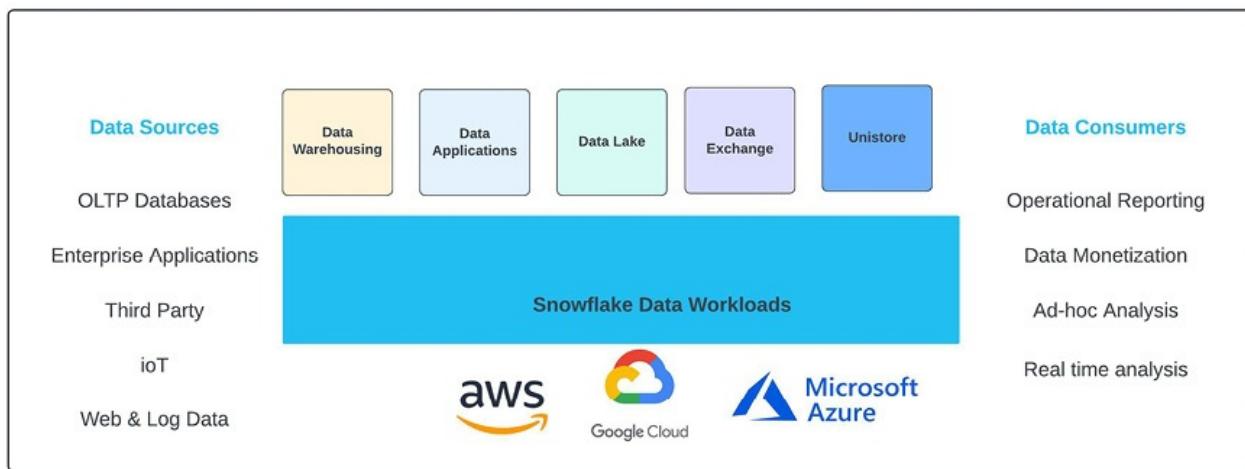


Figure 2.4: Snowflake Workloads

Multiple workloads can be implemented on one data platform easily using Snowflake. Snowflake's architecture has storage and compute separated, which makes it unique in comparison with any other cloud native offerings. Snowflake architecture has three layers: Cloud Services, Compute Layer and Storage Layer as shown in the following figure:

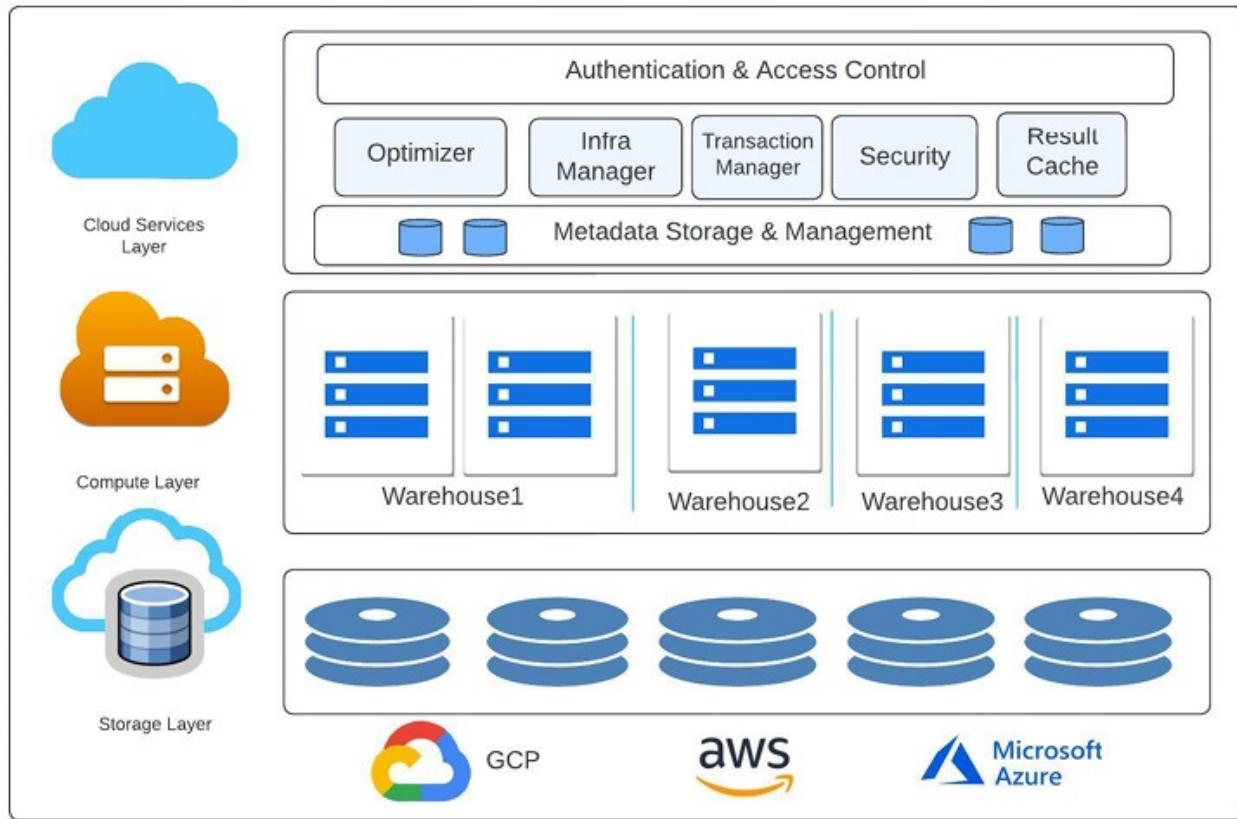


Figure 2.5: Snowflake's three layered architecture

- Snowflake's first layer is cloud services layer which is the topmost layer of its architecture. This layer takes care of authentication, authorization, metadata management, cache management, result cache, query plan generation, optimization of query based on the metadata captured, and so on. This layer is often referred to as the brain of Snowflake.
- The second layer is the warehouse layer, this is also called as compute layer. Snowflake supports various t-shirt sized warehouses to support a variety of operations. Users can spin up as well as auto scale warehouses on demand without any downtime. One user can have multiple warehouses spun up to support data needs in parallel.
- Snowflake's third layer is storage layer which is a common layer across all different warehouses or computes instantiated to support various operations like load, data processing, querying or analysis, and so on. All data stored within Snowflake native tables gets stored in storage layer. Users can query the same data, same table from same storage layer in different ways simultaneously. This is called a **shared data architecture**

where data is shared across and queried by concurrent users and processes. Storage and compute can be scaled independently of each other based on data needs.

Each layer maintains the caches that are used in relevant operations, loads running on Snowflake. You will learn more about these in the upcoming section.

Cloud services layer

This is the first layer of architecture that interacts with the end users, queries, engineering pipelines, and so on. This service layer consists of set of services that caters to activities like authentication, authorization, metadata management, cache management, result cache, query plan generation, optimization of query, and so on. This layer is also referred as Snowflake's brain as this uses various services and generate set of instructions or execution plan to handle all incoming requests. The query plan generated is optimized based on the metadata captured and cache maintained at service layer. Take a look at the following figure for a better understanding:

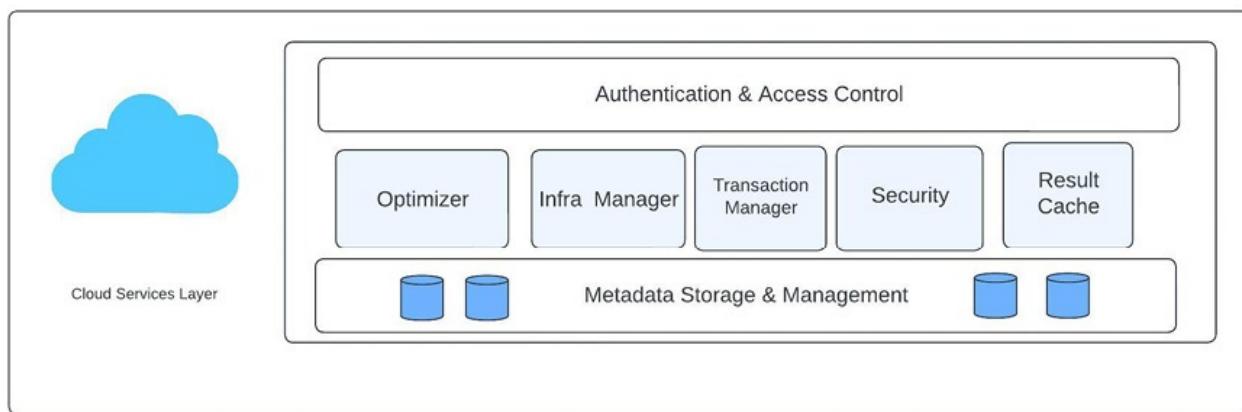


Figure 2.6: Cloud Services Layer

Authentication and access control

Service layer consists of set of services that helps to authenticate and authorize users requesting connection requests to Snowflake. For instance, a user logging in with a specific user ID and password is validated by this service and then allowed to connect to Snowflake.

Optimizer

As the name of the service suggests, the optimizer takes care of the execution plan. Whenever a user or application submits a Snowflake query, this is sent to cloud services layer optimizer service to verify the metadata available, evaluating available cache to generate the optimized query execution plan. This is the first step before queries are submitted to compute layer for processing.

Metadata storage and management

All SQL queries executed are captured and stored as a metadata in cloud services layer. For instance, all DDL and DML operations run against an object is recorded as metadata information. This also stores the variety of metadata in the form of information that is used by optimizer while creating an execution plan. Also, all DDL operations are metadata specific operations and may not need compute or processing layer to complete them. All type of metadata is stored in the cloud services layer that is required to generate optimized execution plans.

Result cache

Like metadata information of operations being run against an object in Snowflake, the result cache layer stores result set as cache. This cache is maintained for every type of DML operations run in Snowflake and maintained for 24 hours. This is used to generate result set in case users are running same queries for given time and there are no changes to the underlying datasets or data. For instance, user *A* has run aggregates on product table to calculate the availability of products and user *B* runs the same aggregates to report the availability. A result cache will be built as and when user *A* runs the query and stored for 24 hours. When user *B* runs the same query then the optimizer checks if there is any existing cache available and table data is maintained then the result generated will be from result cache. In such cases, query processing and compute may not be required if cache is valid.

Security

This service takes care of data security along with security required for data sharing. This is separate from access control where object security and using appropriate set of policies is the scope of service.

Availability of cloud services layer

Cloud services layer is a fully available service where it runs on multiple

availability zones of the cloud region. Result cache is maintained at each available zone where cloud services is available.

Note: This layer also supports scaling where it can scale independently. This is an automated process and doesn't need any direct manipulation by end users.

Uses of cloud services layer

Cloud services billing is already part of the overall Snowflake pricing. Generally, ten percent of the overall credit usage is considered as a threshold for cloud service layer's billing. If the customer usage is below the ten percent threshold, then the customer will not have any additional usage charges for service layer.

As you know, cloud services layer is used to manage, maintain metadata, result cache, optimizer, and access controls. These services are required to be used when end user starts interacting with Snowflake. Every workload or query executed on Snowflake uses portion of cloud services resources. The usage may vary based on the type of query or processing run on Snowflake.

The overall usage of services layer is explained in the following scenarios:

- Users running large, complex queries with multiple joins.
- Users running simple queries with session variables or using session information.
- Users using row level inserts or loads in place of bulk or batch loads.
- If users are using any third-party tools with JDBC or ODBC connectors to read or write data to Snowflake.
- If specific commands or queries being run against metadata views or schema: **INFORMATION_SCHEMA**.

The overall cloud services cost is up to 10% of total usage. However, you can also use some of these usage patterns or techniques to ensure your usage is not more than 10 percent:

- Using queries that uses result cache to generate results and avoid any additional processing.
- Identifying the query patterns and maintaining result cache for subsequent runs or executions where result cache can be used.
- Using result cache for large and complex queries and maintaining it in

case of processing like pipeline or reporting needs.

- Identifying and maintaining frequency of the queries being run to take advantage of result cache.

Note: Identifying and defining access patterns can help to take advantage of result cache as well as metadata maintained at cloud services layer. You can apply this to the reporting or analytical patterns to avoid additional computational as well as cloud service cost.

Virtual warehouse layer

This is the second layer of the Snowflake architecture. This layer is the processing engine of the Snowflake where each compute is referred to as a virtual warehouse. Virtual warehouse is a single or multiple instance or node cluster where each compute resource is the machine with CPU memory and temporary storage. Virtual warehouse can be created on demand by Snowflake users. Users can also define the policies to scale the clusters based on the workloads being run on the warehouse. You will learn more about warehouses, scaling and clusters in upcoming sections. Take a look at the following figure for an understanding of a virtual warehouse layer:

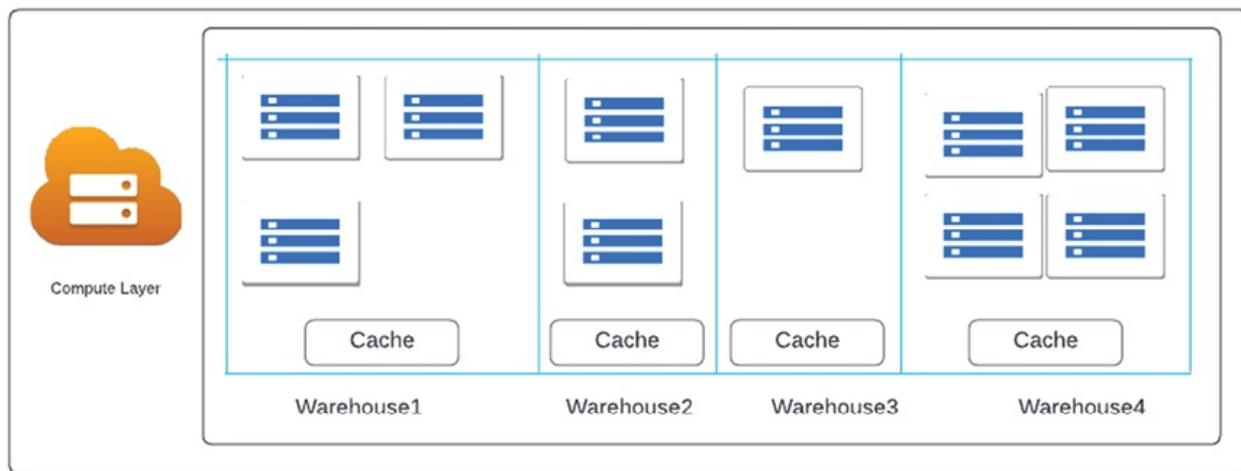


Figure 2.7: Virtual Warehouse Layer (Compute Layer)

Most of the loads running on Snowflake need a virtual warehouse. SQL queries, DML queries, loading and unloading data requires a running virtual warehouse. As you already know, cloud services layer services can be used to run some of the queries like DDLs where a virtual warehouse may not be required or the queries for which the result cache is available.

Snowflake architecture separates Compute and Storage layers. One can be scaled irrespective of another layer. When you have multiple warehouses running in parallel, they all can access the same underlying data from storage layer without any concurrency or contention and performance issues. Virtual warehouse is an independent compute that accesses data from storage and does not share any resources between them. You can refer to [Figure 2.9](#) where multiple warehouses are running in parallel without sharing any resources.

Virtual warehouse is billed for every second its active or in running state whether it is being used or not. These warehouses can be started and stopped any time based on the workload requirements. For example, you can use a warehouse for loading data and can suspend it as soon as loads are complete. You can use another warehouse for your data processing or queries depending on the complexities, concurrency of executions.

Snowflake supports scaling up and down, starting and stopping as well as resizing the warehouses on demand as well as on the fly along with the workloads. Users do not need any downtime to spin up additional resources to the warehouse clusters, loads or queries are managed automatically based on the time resize or additional warehouses requested.

Note: Snowflake warehouses come with a predefined size. Users can pick and choose from the available sizes depending on the queries or workloads. There are no custom sizes available to be defined or used.

Virtual warehouses: Single and multi-cluster

Snowflake supports two types of warehouses: single cluster and multi-cluster. These are classified based on the compute resources or clusters available in them. Users can use both warehouses in the same environments. You can also define multiple warehouses to cater to various needs of the teams.

Consider you are working on setting up a platform and onboarding various teams to read, write and process data on the platform. You can create separate compute or warehouses per team to control, monitor and track usage of the warehouse hence Snowflake credits. You can setup the warehouse based on their needs. You can take the following as a sample reference:

- **Application dev team:** This is development team that needs access to read, write and process data. They will be building pipelines to process data and need access to warehouse that can consider loading, queries, and validating loads. You can create a multi-cluster warehouse of small size

with a minimum of 1 and maximum of 2/3 clusters with auto-scale mode.

- **BI teams:** This is the analytics team that runs queries to publish reports and dashboards. There can be users running long or complex queries to develop and test BI needs. You can create small or medium single cluster warehouse for BI team.
- **Consumer application team:** This is the consumer team who accesses data from your platform and concurrent users can run queries to generate reports, data extracts for applications. You can create a multi-cluster warehouse with minimum 1 and maximum 2 clusters.

This is just a reference scenario; you can create warehouses based on the application needs and track their usage. Having separate warehouses per team or application can help to monitor and review billing of Snowflake credits.

Single cluster warehouses

These are the warehouses that are defined to use as a single instance. You can create them with available warehouse sizes. These are the type of warehouses typically used to run the batch loads or complex queries.

These are the properties supported by single cluster warehouses:

- **Warehouse size:** You can specify the size of the warehouses.
- **Resizing warehouses:** You can resize the warehouses anytime.
- **Auto-suspend:** Suspending the clusters when they are not in use.
- **Auto-resume:** Resuming the clusters automatically when queries submitted, and compute required to execute the query.

Multi-cluster warehouses

These are the warehouses that are defined to use as multiple instances in a cluster. You can create them with the available warehouse sizes. These are the type of warehouses typically used to run concurrent loads, queries, and user access requests.

This also supports same properties as single cluster warehouse. You can resize, resume, suspend and size of warehouse. You will observe this while learning the next section to create warehouses in this chapter.

Virtual warehouse properties

Virtual warehouses are billed based on their uptime and the size of the warehouse. Snowflake supports property that can be used to bring up and down the warehouse depending on the workloads. You can create warehouses based on the type of queries, loads and pipelines being run. Once warehouses are created, users can use them during processing. Imagine, leaving these warehouses up and running 24x7 whether the loads are running or not. You will have heavy billing and usage. Consider, you are using the warehouse only for a daily batch load that runs between 8am EST to 4pm EST and need the warehouse to be available or use it for 8 hours. You might end up paying for the rest of 16 hours too, when the warehouse is not being used.

Snowflake's auto suspend feature helps users to define automated suspends in case of the warehouse not being used. You can suspend these warehouses if they are not in use. Similarly, auto-resume can be used to resume the warehouse whenever needed to execute the load or queries. This is taken care of by Snowflake automatically and is therefore referred to as *auto-suspend* and *auto-resume*.

Virtual warehouse sizes

Snowflake warehouses are defined in the form of t-shirt sizes. Each size caters to specific compute and storage capabilities. Snowflake compute usage and billing is calculated based on the size of the warehouse and the uptime. Usage consumed is calculated based on the credits used for given warehouse size. You will learn more about billing in upcoming section. Snowflake supports these warehouses:

Warehouse Size	Credits / Hour
X-Small	1
Small	2
Medium	4
Large	8
X-Large	16
2X-Large	32
3X-Large	64
4X-Large	128
5X-Large	256
6X-Large	512

Figure 2.8: Snowflake Warehouse Size Chart

Warehouse scaling

Snowflake supports auto scaling and users can define policies to scale up and down. There are two types of scaling possible for warehouses. Users can resize the warehouse from smaller size to a larger size. This is referred as scaling up. This is usually used in case of single instance of warehouse. Users can also define and use warehouse clusters with more than one warehouse running to support workload. This is referred as a multi-cluster warehouse. Users can define policies to add and remove warehouses from the cluster depending on the workload and resources required to complete processing. This is referred to as scaling in and out.

Scaling up

Scaling up refers to the resizing of a warehouse and users can use this while dealing with large data volume loads or processing and running complex queries. Scaling can be applied to single cluster warehouses where single instance is being run and used. This cannot be applied to multi-cluster warehouses. You can also refer to scaling up as vertical scaling where you can resize the compute power.

Users can resize warehouse anytime using console or SQL queries. Users can also run SQL query to resize the warehouse while queries are in progress. All queries that are in running state will continue to run with current size and all the queries that are in queue will be run on the new resized warehouse.

Consider a scenario where a user is running a load with huge data volume, and this is a monthly load. The user has defined a small warehouse to load every day data using bulk loads and they run in each time. In case of a monthly load, it takes more time than usual with a small warehouse as the monthly volume is more in comparison to the daily volume. If you compare the load time it takes 4 hours to complete a monthly batch load using the small warehouse. In this case, the user can resize the cluster to M or L size to complete the monthly load. Let us assume it takes one hour to complete the load. This is almost the same usage as using 4 hours of small warehouse and one hour of large warehouse. The billing might not be this simple to compare, however you can consider the resize scenario to understand the performance benefits.

Note: Snowflake recommends experimenting with different types of queries along with different warehouse sizes to determine the best way to manage warehouses effectively and efficiently. Also, it is recommended to start small and increase in size with the experiment. This is the easiest way to identify an under-sized or underutilized warehouse.

Scaling out

Scaling out refers to adding and removing warehouses to the existing multi-cluster warehouse. A multi-cluster warehouse is the warehouse cluster with more than 1 compute resources. Multi-cluster can be defined along with scaling policy, minimum number of clusters and maximum number of clusters. Snowflake supports maximum up to 10 clusters in multi-cluster warehouses. You can relate scaling out to the horizontal scaling where additional computes are added to the processing. Users can create multi-cluster warehouses in two modes: maximized and auto-scaled. You will learn more about these in upcoming sections.

Scaling modes

Snowflake supports the following two modes while defining scaling policies with multi-cluster implementation:

- **Maximized:** This is the mode where minimum and maximum number of nodes or clusters are same. You can use this mode when you are implementing multi-cluster to support large number of concurrent users. This will provide a steady performance for the users.
- **Auto-scaled:** This is the mode where minimum and maximum clusters are different. Maximum clusters can be defined up to 10 clusters. Snowflake dynamically manages the start and stop of the clusters. You can use this mode when you do not have steady concurrent users and you want to scale

up and down based on the end user or concurrent user hits. If the users grow up, cluster scales up and adds cluster to the multi-cluster. Similarly, when user reduced cluster is decommissioned, cluster nodes are reduced.

Note: Snowflake recommends starting with auto-scale mode and with small number of clusters. You can change the maximum cluster count based on the warehouse loads until you determine the upper and lower number of clusters in multi-cluster warehouse.

Scaling policies

Snowflake supports two types of policies when defining auto-scaling for multi-cluster warehouses. This is applicable to the Auto-scaled mode where clusters can be added or removed based on the demand.

- **Standard policy:** The first cluster starts immediately as soon as a query is queued, or the system detects one more query than the running clusters can execute. Usually, every subsequent cluster waits to start 20 seconds after the earlier one started. For example, if your warehouse is configured with 10 max clusters, it can take a full 200+ seconds to start all 10 clusters. It prevents/minimizes queuing by favoring starting additional clusters over conserving credits.
- **Economy policy:** In this policy the additional cluster is not added until the system estimates if there is enough query load to have the cluster busy for a minimum of 6 minutes. It conserves credits by favoring keeping running clusters fully-loaded rather than starting additional clusters, which may result in queries being queued and taking longer to complete.

Virtual warehouses: Creation and modifications

You can create warehouses using Snowflake console as well as SQL queries. You can choose any one of these two ways to setup warehouses. Creating warehouse may not be a frequent activity, you can define and set it up along with the platform design. You may need to create new ones as and when you are onboarding teams to the platform.

Warehouse setup using console

You can create Snowflake warehouses using console options. You can follow below steps to create warehouse using console:

1. Go to Snowflake console using the account URL:

<https://<<accountidentifier>>.snowflakecomputing.com/console#/>

2. Choose the appropriate role to create warehouse. Use **SYSADMIN** or **Custom role** where you have permission to create a warehouse.
3. Choose warehouse option from the ribbon as shown in the following figure:

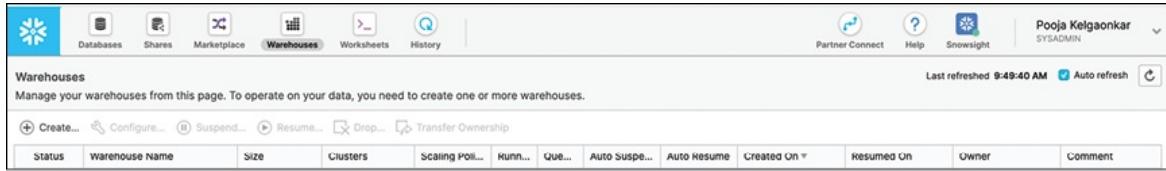


Figure 2.9: Snowflake Account Ribbon

4. Click on **Create Menu** option on the warehouse page and a pop-up window will open as shown in the figure. You need to provide these input details to create a warehouse, as shown in *Figure 2.10*:

- **Name:** Provide name of the warehouse. You can provide the name as per usage or requirements. For example, **DEV_WH** (for development team), **ML_WH** (for ML team) or **BI_WH** (for BI teams).
- **Size:** Size of the warehouse from XS to 6XL
- **Maximum clusters:** How many clusters needed for warehouse (1 is used for single cluster and more than 1 is used for multi-cluster warehouse)
- **Minimum clusters:** This is to be provided if you choose more than 1 cluster in multi-cluster warehouse setup.
- **Scaling policy:** Choose one from standard or economy.
- **Auto suspend:** Choose time when you want to suspend your warehouse automatically.
- **Auto resume:** This checkbox is to enable the auto resume in order to resume the warehouse automatically depending on the queries run.
- **Comment:** Short comment or description of the warehouse getting created. For example, **DEV** warehouse for application development or **BI** warehouse for analytics team.

Data you need to create one or more warehouses.

Create Warehouse

Name *

Size

Learn more about virtual warehouse sizes [here](#)

Maximum Clusters

Multi-cluster warehouses improve the query throughput for high concurrency workloads.

Minimum Clusters

The number of active clusters will vary between the specified minimum and maximum values, based on number of concurrent users/queries.

Scaling Policy

The policy used to automatically start up and shut down clusters.

Auto Suspend

The maximum idle time before the warehouse will be automatically suspended.

Auto Resume

Comment

[Show SQL](#)

Figure 2.10: Create warehouse

5. Click on **Finish** to create a warehouse.
6. Once the warehouse is created, you need to grant permission to the role to use it. Follow these steps to grant access to a role, as shown in *Figure 2.11*:
 - a. Click on **Grant** button to open a **Grant** window.
 - b. This asks for privileges to be granted on the warehouse selected. In this case, **TEST_DEMO_WH** is the warehouse name and access to this **WH** is to be granted to users.

Grant privileges on warehouse TEST_DEMO_WH

Privileges to grant	<input type="button" value="Select privileges"/> ▼ ⊖ ⊕
Grant privileges to	<input type="button" value="Select a role"/> ▼
<input type="checkbox"/> with Grant Option	

Figure 2.11: Grant Permissions to warehouse

- c. Click on the dropdown button to list down the privileges. You can grant any of these 4 permissions or select multiple permissions to be granted as shown:

Grant privileges on warehouse TEST_DEMO_WH

Privileges to grant	<input type="button" value="Select privileges"/> ▼ ⊖ ⊕
Grant privileges to	MODIFY MONITOR <input type="checkbox"/> with Grant Option OPERATE USAGE

Figure 2.12: Granting warehouse permissions

- d. Click on **Grant privileges to** and it opens a role drop down to choose an existing role from the platform as shown in the following figure:

Grant privileges on warehouse TEST_DEMO_WH

Privileges to grant	<input type="button" value="MODIFY, MONITOR, OPERATE, USAGE"/> ▼ ⊖ ⊕
Grant privileges to	<input type="button" value="DATA_ENG"/> ▼
<input type="checkbox"/> with Grant Option	

Figure 2.13: Grant Permissions warehouse

- e. The **with Grant Option** checkbox is used to grant privileges with grant option where the role granted will also be able to add or remove additional grants to other roles/users.
- f. Click on **Grant** and these permissions will be granted on the warehouse. You can see all granted permissions on the warehouse window as shown:

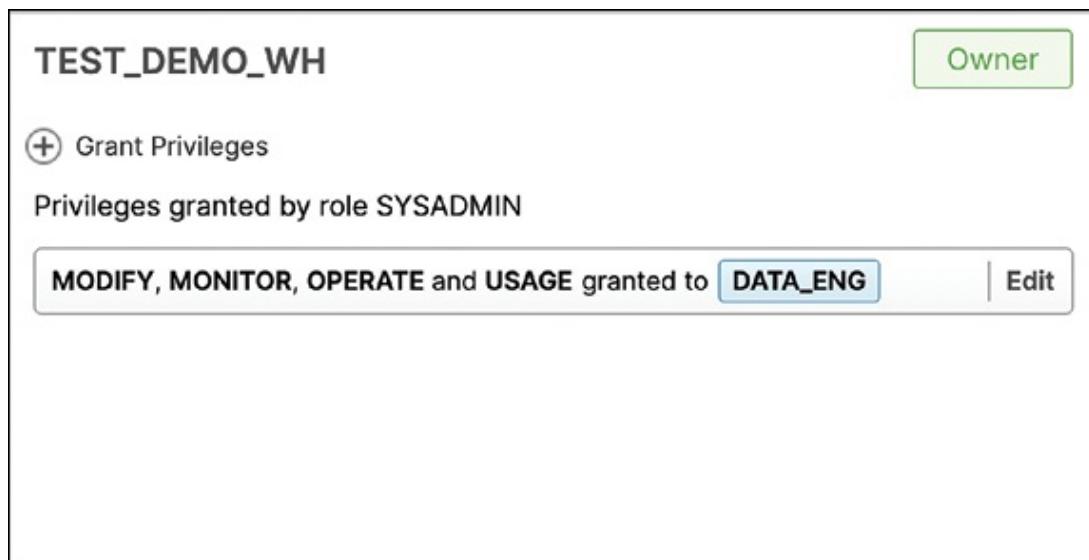


Figure 2.14: Warehouse permissions

Warehouse: Using SQL commands

Snowflake supports ANSI SQL standards and SQL is the native processing engine. This offers a set of SQL commands to **CREATE**, **ALTER**, **DESCRIBE**, **DROP** and list down warehouses in the system. You can use the following commands.

Creating warehouse

You can use this command to create single cluster warehouse:

```
CREATE OR REPLACE WAREHOUSE DEV_WH WITH
WAREHOUSE_SIZE='Small';
```

You can also use suspend and resume state as initial state as and when warehouse is created:

```
CREATE OR REPLACE WAREHOUSE DEV_WH
WAREHOUSE_SIZE='Small'
```

```
INITIALLY_SUSPENDED=TRUE;
```

You can use this command to create a multi-cluster warehouse:

```
CREATE OR REPLACE WAREHOUSE TEST_WH WITH
  WAREHOUSE_SIZE='Small'
  MAX_CLUSTER_COUNT = 01
  MIN_CLUSTER_COUNT = 02
  SCALING_POLICY = STANDARD
  AUTO_RESUME = TRUE
  INITIALLY_SUSPENDED = TRUE
  COMMENT = 'This is a test multi-cluster warehouse'
```

Other warehouse commands

Let us take a look at some other warehouse commands:

- **ALTER WAREHOUSE:** This is used to alter the size and state of the warehouse. You can change the state of the warehouse if auto-suspend and resume is not used as well as you can use to change the size of cluster:

```
ALTER WAREHOUSE DEV_WH RESUME;
```

```
ALTER WAREHOUSE DEV_WH SET warehouse_size=MEDIUM;
```

- **DESCRIBE WAREHOUSE:** This is used to describe the warehouse:

```
DESCRIBE WAREHOUSE DEV_WH;
```

```
+-----+-----+
| created_on | name
| kind       |
+-----+-----+
| 2023-05-18 02:05:00.000 -0700 | DEV_WH |
```

WAREHOUSE |

```
+-----+-----+
-----+-----+
```

- **DROP WAREHOUSE:** This is used to drop the warehouse:

```
DROP WAREHOUSE DEV_WH;
```

- **SHOW WAREHOUSES:** This lists down all the warehouses in the account:

```
SHOW WAREHOUSES;
```

```
SHOW WAREHOUSES LIKE 'dev%';
```

- **GRANT WAREHOUSE:** This command is used to grant permissions to the roles:

```
GRANT OPERATE ON WAREHOUSE DEV_WH TO ROLE
DATA_ENG;
```

```
GRANT OPERATE ON WAREHOUSE DEV_WH TO ROLE DATA_ENG
WITH GRANT OPTION;
```

Virtual warehouses: Billing and usage

Snowflake warehouse billing is calculated based on the size of warehouse, number of warehouses, and the total time clusters that are up and running. Snowflake bills every second and calculates the credits consumed. Warehouse bills for 1 minute initially followed by per second billing. Snowflake warehouse billing is calculated based on the size and in case of re-size, the billing is calculated based on the warehouse size earlier and later for the time its running.

Snowflake billing is often referred as usage and this is calculated in terms of compute usage: warehouse usage, storage usage and cloud services layer usage. Warehouse usage caters to only warehouse billing, and it is reported via account usage view. You need to have **ACCOUNTADMIN**, or appropriate permissions or role assigned to view the usage details. You can also view the usage from console using **Account** ribbon option.

Storage layer

This is the third layer of Snowflake architecture and referred to as database layer. This is the centralized place where entire data is stored including structured, and

semi-structured data. Data is stored in internal optimized, compressed, columnar format. Snowflake uses underlying cloud storage to store the data where it is hosted – AWS, GCP and Azure.

Snowflake manages all aspects of data storage: file size, structure, compression, metadata, statistics, and so on. Users cannot access data from Snowflake data layer directly. This data is accessible only via SQL queries.

Snowflake stores data in the form of snowflake objects and this follow a hierarchy. The hierarchy is: **Snowflake account | Databases | Schemas | Tables & other objects**. Table data is stored in the form of micro partitions and this help not only to store but also to improve performance of queries and accessing data from the storage layer. You will learn more about this in upcoming chapters. Refer to the following figure for a better understanding about storage layer:



Figure 2.15: Storage layer

Data stored is encrypted and compressed. Table data stored in the form of micro partitions that are stored as optimized, immutable, compressed columnar format that is encrypted using AES-256 encryption. Snowflake can store data up to petabytes with no impact on warehouse layer. These two layers can be scaled independent of each other. This can also be referred as unlimited scaling for storage and compute.

Snowflake storage layer supports two unique features: Zero copy cloning and Time-travel. These are the distinguishing features that make Snowflake different in comparison with other platforms. These features are explained in upcoming chapters.

Time-travel

Time-travel is the Snowflake **Continuous Data Protection (CDP)** feature where

you can access historical data at any point. The historical data can be data that has changed or deleted or dropped objects. You can retrieve data and objects as part of this feature. Default time-travel is of 1 day for all accounts. You can also extend this to 90 days for enterprise edition. You can query, access, and retrieve data for up to 90 days using time-travel. This feature support varies based on the type of tables and objects. Time-travel storage is also considered as a component to calculate storage cost. You will learn more about time-travel and its usage in [*Chapter 8, Data Protection and Recovery.*](#)

Failsafe

Failsafe is a Snowflake CDP feature where it ensures availability and maintenance of the data. This feature is used for disaster recovery of historical data. Unlike time-travel, failsafe data retrieval can be done only up to 7 days and only by the Snowflake support team. Users cannot access failsafe data using queries. This feature is standard to all accounts, no additional licensing is required to enable this feature. This needs additional storage, and this cost gets added as part of failsafe storage and overall storage usage and billing. You will learn more about failsafe in the upcoming [*Chapter 8: Data Protection and Recovery.*](#)

Data cloning

Data and database can be cloned to create snapshots of the data. Users can create snapshots and setup environment quickly for development or testing. Data cloning works at a metadata level and actual cloning does not take place. Each cloned object points to the same micro-partitions and underlying data as original table until data is changed, modified or deleted. For any new changes, a new storage is created and maintained for clones. You will learn more about cloning in [*Chapter 13: Data Cloning.*](#)

Storage usage and billing

Storage cost is computed on a monthly basis for storing data in Snowflake. The cost is calculated based on a flat rate per terabyte. The cost depends on the type of account (capacity or on demand) and the region (US or EU).

Storage billing is calculated based on these usage capabilities:

- Storage used for files stored for bulk data loading/unloading (stored compressed or uncompressed).

- Storage of database tables with historical data for time travel.
- Storage associated with fail-safe for database tables.
- Storage associated with database clones.

Some other costs are:

- **File cost:** Storage cost incurred by files staged for bulk loading and unloading. This is based on the file size.
- **Database cost:** This is the cost associated with data stored in the tables. Database cost also consists of historical data maintained as part of time-travel. Snowflake stores data in compressed format and is used to calculate database cost for account.
- **Time-travel and failsafe cost:** This cost is calculated for every 24 hours as and when the data changes. The historical data stored is for a given number of days based on the type of tables. The cost is calculated based on the percentage of the data changed – updated, deleted, and so on. The portion of data is maintained along with historical data for cost calculation.
- **Database clones cost:** This is the cost associated with database clones and additional storage maintained for changed data. This component makes storage computation a little trickier as every change creates a new storage along with its CDP capabilities to maintain time-travel or failsafe.

Cache in Snowflake

Snowflake maintains caching. Cache is maintained at three different layers of Snowflake. These are available at metadata, warehouse, and result level.

Whenever a user submits query to Snowflake, parser validates the query and optimizer checks if there is any cache available to serve the result to the user. If the query is run often then the same query shares result from the result cache. You will learn more about the different kinds of cache in the subsequent section.

Metadata cache

This is the cache completely maintained and managed by cloud services layer. Users do not have any control on the metadata cache. Snowflake generates, captures and manages details of the objects, micro-partitions, and clustering.

Metadata for tables can be stored in the form of row count, table total size, file references, and table versions. Whenever users run query that can be fulfilled by metadata cache, like getting count may not need a warehouse in active state. Snowflake optimizer generates a query plan based on the available caches, checks if the warehouse is needed to execute query and generate plan.

Snowflake metadata stores table definitions, micro partitions with these details for each micro-partitions of the Snowflake object:

- Range of values stored in micro-partitions with **MIN** and **MAX** values
- Count of **NULL** values
- Count of distinct values

The following figure represents metadata cache:

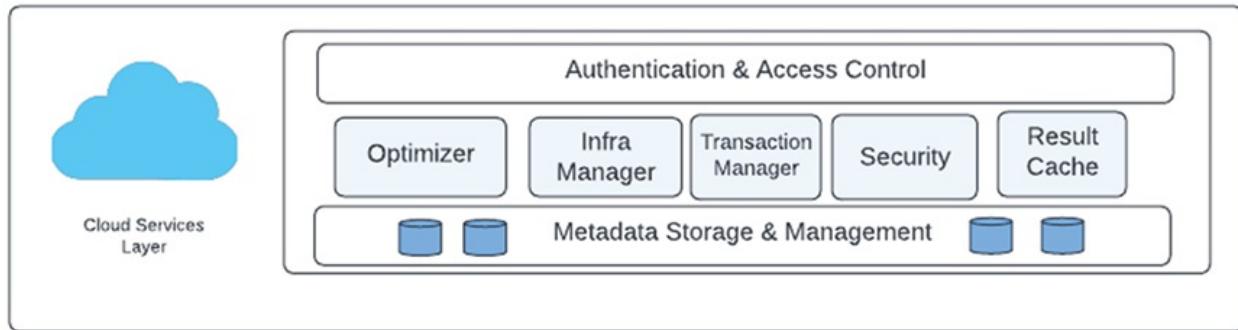


Figure 2.16: Metadata cache

Cache also stores total number of partitions and depth of overlapping partitions to store clustering details. This cache can be used to cater to set of queries as below:

- **MIN** and **MAX** of column values
- Count of the table

Query result cache

Query result cache is the fastest way to generate the result and access data from Snowflake. This is the cache is built from the queries run and persisted for 24 hours. This cache is different than warehouse cache and metadata cache. This cache is time based and purged every 24 hours. This cache is managed by cloud services layer and available across all warehouses, as shown:

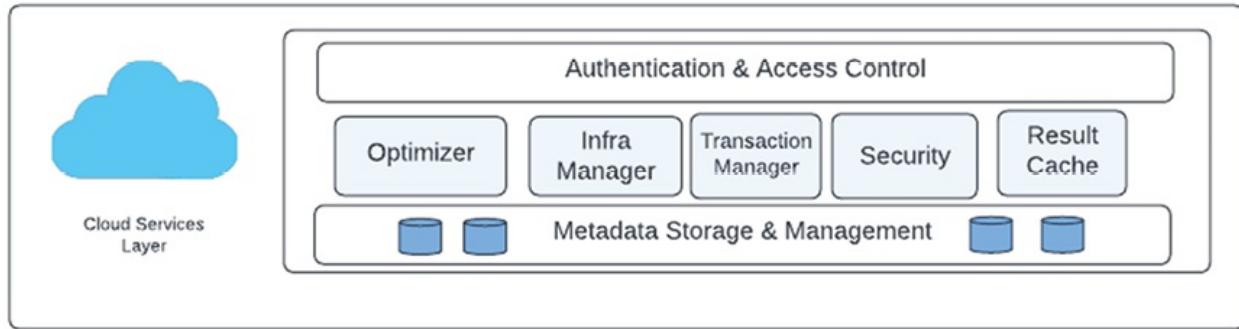


Figure 2.17: Result Cache

Query results are cached and accessible to all users. If one user runs a query, then the query result is stored in the cache and used to generate result for the same query executed by another user. Result cache is used to cater to the same query executed by multiple users with same permissions. This cache cannot be accessed directly by users however this can be used across users while generating optimized query plan. This is the only cache that can be controlled by user by running SQL query for specific session.

Warehouse cache

Virtual warehouse layer consists of cache along with the warehouses. This cache is being built, maintained and managed at warehouse level. Each warehouse is a compute with CPU and storage (Local SSDs). Whenever a user runs queries and they are processed on warehouses, it builds a local cache based on the type of queries being run and data accessed. The following figure explains warehouse cache:

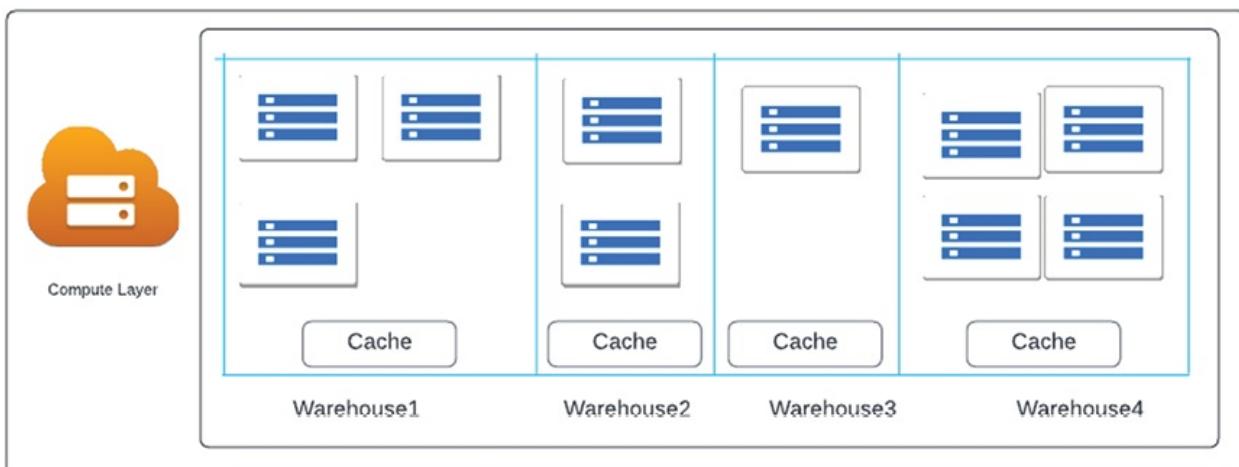


Figure 2.18: Warehouse Cache

This cache is specific and local to the warehouse. These cannot be accessed across warehouses. Whenever a query is submitted to be executed on the warehouse, it checks the data available in cache and checks if the query can be fulfilled by the cache available. If the result can be generated from cache, query result is generated and shared with end users. If the cache is not available, then it submits request to the data layer and gathers data to be maintained at cache.

Global services layer or cloud services layer maintains freshness of data. All warehouse cache is maintained at the warehouse layer and recorded as metadata information in the services layer. Optimizer checks data freshness and all available caches before producing a query plan. There are 3 types of cache available in Snowflake: metadata cache, warehouse cache and query result cache. Optimizer uses available cache and generate optimized plan.

A certain rule is followed to persist and refresh the cache. Cache is persisted until warehouse is running and active. Cache is discarded and freed whenever warehouse is suspended.

Conclusion

This chapter provided a summary of Snowflake's three-layered architecture. This chapter helped you to understand Snowflake architecture, distinguishing features in comparison with traditional architecture. This also helps you to learn Snowflake caching and its benefits to optimize the performance of Snowflake. You will learn more about various Snowflake features and offerings as you go along with the next chapters in this book.

In the next chapter, you will learn about Snowflake's data types, objects and SQL commands. You will learn more about support to ANSI SQL standards, Snowflake extended features, and native objects. This is the next chapter in the series to understand, learn, explore and gain hands on experience with Snowflake.

Points to remember

Following are the key take aways from this chapter:

- Snowflake architecture is three layered architectures.
- Snowflake architecture layers – Cloud Services Layer, Warehouse Layer, and Storage Layer.

- Cloud services layer takes care of metadata operations, parsing, optimizing etc.
- Warehouse Layer or Compute Layer takes care of processing or compute required to execute workloads on Snowflake.
- Storage Layer cater to the storage of Snowflake and stores data in compressed format.
- Snowflake maintains 3 types of caches – Result cache, Warehouse cache, and Metadata cache.

Multiple choice questions

1. **What are the layers of Snowflake architecture? (Select all that apply)**
 - a. Cloud services layer
 - b. Virtual warehouse layer
 - c. Storage layer
 - d. Global layer
2. **What are the different names of virtual warehouse layer?**
 - a. Virtual warehouse layer
 - b. Compute layer
 - c. Processing layer
 - d. Machine layer
3. **What are the types of cache available in Snowflake?**
 - a. Metadata cache
 - b. Result cache
 - c. Warehouse cache
 - d. Storage cache

Answers

1	a, b, c
2	a, b, c
3	a, b, c

Questions

1. How can Snowflake's compute and storage be scaled independent of each other?
2. How does Snowflake's query optimizer generate the query plan?
3. What are Snowflake layers and what are the services of each layer?
4. How can a virtual warehouse be created and granted to the users?
5. What are the types of caches available and used?
6. What is Snowflake usage and how is it computed?

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 3

Data Types, Data Objects and SQL Commands

Introduction

This chapter focuses on various data types, data objects and SQL commands supported in Snowflake. Snowflake supports ANSI SQL standard however this chapter guides on DDL, DCL and DML supported in this platform. This chapter also covers unique database objects used and created in Snowflake.

Structure

These are the topics covered in this chapter:

- Understanding data types
- Using SQL commands
- Creating database objects
- Creating warehouses
- Setting up context
- Creating tables and views
- Implementing extended SQL objects

Objectives

By the end of this chapter, you will be able to understand the Snowflake support to ANSI SQL standards. Snowflake has unique data types to handle and process semi-structured and unstructured data. You will also be able to learn more about Snowflake's extended support to the database objects. A quick checkpoint at the end of the chapter will help you to evaluate your understanding of the topics covered in this chapter.

Understanding data types

Snowflake supports the basic data types along with some of the extended data types that can be used to handle specific data requirements. These data types can be used while defining a table, used as a variable in data processing, these can also be used as parameters while defining the processing logic in engineering pipelines.

Snowflake also supports data conversions from one data type to another depending on the data and processing needs. Snowflake also supports automatic data type conversion for example, loading float number to an integer column converts automatically and this is called as implicit coercion. Automatic conversions are done automatically whenever needed and conversions can also be done using appropriate **CAST** options.

Snowflake supports all basic data types and additionally extended data types. These data types can be categorized into these categories:

- Numeric data types
- String and binary data types
- Logical data types
- Date and time data types
- Semi-structured data types
- Geospatial data types

Each of these data types are explained in subsequent sections of the chapter.

Numeric data types

This data type supports storing number data. This can be a fixed-point number or floating-point number depending on the type of data to be stored and processed. Each of this fixed and floating point supports a set of data types as defined as

follows:

- **Fixed point data types:** As the name describes, these data types are fixed values with no precisions. These are used for zero precision values:

Data type	Description
NUMBER	This allows number upto 38 digits
DECIMAL, NUMERIC	These are synonyms with NUMBER
Integer categories INT, BIGINT, SMALLINT, TINYINT, BYTEINT	These store integer values with 0 precisions. INT type can be defined depending on the range of values stored in it.

Table 3.1: Snowflake Numeric Data Types

- **Floating point data types:** On the contrary to the fixed point, floating point supports data stored with precision values. These are set of data types that define range of values and precisions to be stored in it.

Data type	Description
FLOAT, FLOAT4, FLOAT98	This allows 64-bit floating-point numbers
DOUBLE, DOUBLE PRECISION, REAL	These are synonyms of FLOAT

Table 3.2: Snowflake Float data types

Snowflake supports special values for **FLOAT** data types:

- **NaN** (Not A Number)
- **inf** (infinity)
- **-inf** (negative infinity)

String and binary data types

Snowflake supports data types for storing string values as well as binary values. There are separate set of data types for each category.

String data types:

Data type	Description
VARCHAR	This allows Unicode UTF-8 characters. If there is no length specified then the default is the maximum allowed length (16,777,216), that is, 16MB.

CHAR, CHARACTER, NCHAR	These are synonyms of VARCHAR. If there is no length specified with CHAR by default it is CHAR(01)
STRING, TEXT, NVARCHAR, NVARCHAR2, CHAR VARYING, NCHAR VARYING	This is synonym of VARCHAR.

Table 3.3: Snowflake string data types

In other systems where **CHAR** and **VARCHAR** stores ASCII data however **NCHAR** and **NVARCHAR** stores Unicode data. In the case of Snowflake, **VARCHAR** and rest of all other data types store data in the form of Unicode UTF-8 characters. There is no difference in handling of these data types.

You can consider the following criteria while defining size of the **VARCHAR** column:

- **Storage:** Storage is consumed based on the actual data stored in the column. If the column is defined to store 16MB data and you have only 1 character data stored, then it consumes only 1 character.
- **Performance:** There is no performance difference with **VARCHAR** and **NVARCHAR** with column length.
- **Tools for working data:** Based on the tools used for ETL or ELT operations or BI tools, these look out for data defined with columns with their length values. In this case, if you are aware of the size then you can add it with column definition.
- **Collation:** Snowflake supports collation, if you are defining a column with collation then the storage and value varies depending on the data stored in it.

Note: Collation allows users to specify alternative rules for comparing strings, which can be used to compare and sort data according to a particular language or other user-specified rules. for example, case sensitive, accent sensitive, punctuation, and so on.

Binary data types

Snowflake supports storing binary data into binary data types:

Data type	Description
BINARY	The maximum length is of 8MB.
VARBINARY	These are synonyms of BINARY

Table 3.4: Snowflake Binary data types

Logical data types

As the name describes, Snowflake supports a single logical data type: **BOOLEAN**.

This data type supports storing 2 values: **TRUE** and **FALSE**. This also allows to load Unknown value which is represented by **NULL**. Snowflake supports **BOOLEAN** conversions as shown below:

- **Explicit conversion to BOOLEAN:** Values can be converted to Boolean values from text string and numeric to Boolean by using the **TO_BOOLEAN** or **CAST** functions:

Data type	Description
String conversions	<ul style="list-style-type: none"> Values converted to TRUE: ‘true’, ‘t’, ‘yes’, ‘y’, ‘on’, ‘1’. Values converted to FALSE: ‘false’, ‘f’, ‘no’, ‘n’, ‘off’, ‘0’. Conversion is case-insensitive. All other text strings cannot be converted to Boolean values.
Numeric conversions	<ul style="list-style-type: none"> Zero (0) converted to FALSE. Any non-zero value converted to TRUE, that is, 1,-1

Table 3.5: Snowflake Logical data types

- **Implicit conversion to Boolean:** Unlike explicit, Boolean values can be implicitly converted from **string** to **Boolean**.

Data type	Description
String conversions	<ul style="list-style-type: none"> ‘true’ is converted to TRUE. ‘false’ is converted to FALSE. The string values are not case-sensitive.
Numeric conversions	<ul style="list-style-type: none"> Zero (0) converted to FALSE. Any non-zero value converted to TRUE. That is, 1,-1

Table 3.6: Snowflake Boolean conversion

Date and time data types

Snowflake supports managing dates, times, and timestamps. This section describes the supported formats for string constants used in manipulating dates, times, and timestamps:

Data type	Description

DATE	Allows DATE in format of YYYY-MM-DD, DD-MON-YYYY and more.
DATETIME	Allows date with time. This is also alias of TIMESTAMP_NTZ.
TIME	Allows time and can be stored in the form of hh:mm:ss Time also supports precision and default is of 9 precisions to represent nanoseconds.
TIMESTAMP	Allows date and time in the form of timestamp. This also stores timestamp in different time zones. Snowflake recommends using years between 1582 and 9999.
TIMESTAMP_LTZ	Timestamp stores in UTC time zone.
TIMESTAMP_NTZ	Wallclock values stored in precision.
TIMESTAMP_TZ	UTC time together with an associated time zone offset.

Table 3.7: Snowflake Date and Timestamp data types

Date and time formats:

Format type/element	Description
YYYY / YY	Year in 4 digits or 2-digit format.
MM/MON/MMMM	Month name in MM format, that is, month number or MON – month in short form or MMMM – month in full name, for example, MM=03, MON-MAR, MMMM=March
DD	Day of the month.
DY	Day of the week abbreviated (Mon, Tue).
HH24	24-hour format – represents hours of the time.
HH12	12-hour format with am and pm time of the hour.
AM, PM	Time element to represent time in AM or PM for HH12 hour format.
MI	Time in minutes.
SS	Time in seconds – 2-digit format.
FF[0-9]	Time in nano seconds upto 9 precisions
TZH:TZM , TZHTZM , TZH	Time zone of the hour and minute – offset from UTC – this can be represented by +/- sign, for example, GMT+5

Table 3.8: Snowflake Date and Timestamp formats

Date and Time examples:

- **Time in TIMESTAMP_TZ format:** Run a select given timestamp in TIMESTAMP_TZ format, as shown:

```
SELECT '2023-06-04 02:00:00'::TIMESTAMP_TZ as
```

```
time_tz;  
-----+  
time_tz  
-----+  
2023-06-04 02:00:00 -0800 |  
-----+
```

- Create a sample table with timestamp column to store data in **TIMESTAMP_LTZ** format. Insert sample records to test the date/time saved in the format given:

```
CREATE OR REPLACE TABLE demo_ltz(ts  
timestamp_ltz);
```

```
ALTER SESSION SET TIMEZONE =  
'America/Los_Angeles';
```

```
INSERT INTO demo_ltz values('2023-01-01  
16:00:00');
```

```
INSERT INTO demo_ltz values('2023-01-02 16:00:00  
+00:00');
```

```
-- here the time for January 2nd is 08:00 in Los  
Angeles (which is 16:00 in UTC)
```

```
SELECT ts, hour(ts) FROM demo_ltz;
```

```
+-----+-----+
```

TS	HOUR(TS)
Sun, 01 Jan 2023 16:00:00 -0800	16
Mon, 02 Jan 2023 08:00:00 -0800	8

- Create a sample table with timestamp column to store data in **TIMESTAMP_NTZ** format. Insert sample records to test the date/time saved in the format given:

```
CREATE OR REPLACE TABLE demo_ntz(ts
timestamp_ntz);
```

```
ALTER SESSION SET TIMEZONE =
'America/Los_Angeles';
```

```
INSERT INTO demo_ntz values('2023-01-01
16:00:00');
```

```
INSERT INTO demo_ntz values('2023-01-02 16:00:00
+00:00');
```

-- Here, If you notice that both times from different time zones are converted to the same "wallclock" time

```
SELECT ts, hour(ts) FROM demo_ntz;
```

TS	HOUR(TS)
Sun, 01 Jan 2023 16:00:00 -0800	16
Mon, 02 Jan 2023 08:00:00 -0800	8

Sun, 01 Jan 2023 16:00:00	16
Mon, 02 Jan 2023 16:00:00	16

Date and time intervals and constants

Let us take a look at the constants and intervals for date and time:

- **Constants:** Snowflake supports storing and processing fixed or constant values in the **Date** and **Timestamp** columns. These are also called as literals. These string values must be enclosed in the quotes.

For example:

```
date '2023-06-04'  
time '11:05:26'  
timestamp '2023-06-04 12:31:45'
```

- **Intervals:** **Date** and **Timestamp** column supports using interval constants to represent part of the date or timestamp in the form of day, hour, minute, month, and so on.

Example:

```
SELECT TO_DATE ('2023-06-04') + INTERVAL '1 day, 1 year';
```

```
+-----+  
| --+  
| TO_DATE ('2023-06-04') + INTERVAL '1 DAY, 1  
YEAR' |  
| -----+  
| -- |  
| 2024-06-05
```

```

|  

+-----  

--+

```

Supported date and time intervals

Interval constant supports below part of the date and time used as a parameter to extract the values from given field.

Part of date and time	Abbreviation
year	y, yy, yyy, yyyy, yr, years, yrs
quarter	q, qtr, qtrs, quarters
month	mm, mon, mons, months
week	w, wk, weekofyear, woy, wy, weeks
day	d, dd, days, dayofmonth
hour	h, hh, hr, hours, hrs
minute	m, mi, min, minutes, mins
second	s, sec, seconds, secs
millisecond	ms, msec, milliseconds
microsecond	us, usec, microseconds
nanosecond	ns, nsec, nanosec, nsecond, nanoseconds, nanosecs, nseconds

Table 3.9: Snowflake date and time intervals

Examples:

```
select to_time('05:05:19') + INTERVAL '3 hours, 18
minutes';
```

```

+-----  

--  

| TO_TIME('05:05:19') + INTERVAL '3 HOURS, 18 MINUTES'  

|  

| -----  

- |

```

```
| 08:23:19
```

```
|
```

```
+-----+  
--+
```

Supported date and time arithmetic operations

These can be achieved with arithmetic operations as well. Here are few examples for your reference:

```
select to_date('2023-05-05') + 1; ( adds a day to the given date)
```

```
+-----+  
| T0_DATE('2023-05-05') + 1 |  
| ----- |  
| 2023-05-06 |  
+-----+
```

```
select to_date('2023-05-10') - 5; ( Goes back to the 5 days from the given date)
```

```
+-----+  
| T0_DATE('2023-05-10') - 5 |  
| ----- |  
| 2023-05-05 |  
+-----+
```

Semi-structured data types

Snowflake supports semi-structure data processing and have three types of data

types that can be used to support semi-structured data. These are three data types that can be categorized as semi-structured:

- **VARIANT**
- **ARRAY**
- **OBJECT**

VARIANT is the snowflake supported data type which can be used to store any type of data in it. This stores 16 MB data in each field and can have values from JSON, Avro, ORC and Parquet format files. You can use one or combination of these data types to process these files specially to preserve the hierarchical data from these datasets. You will learn each of these data types in a subsequent section of this chapter.

VARIANT

This data type can store any type of data including **ARRAY** and **OBJECT**. The maximum length that can be stored is 16MB. This can be used to read semi-structure data and store as column value in table. **TO_VARIANT** is the function that can be used to convert the data to the **VARIANT** data type.

For example:

```
CREATE TABLE demo_var (f1 FLOAT, v1 VARIANT, f2  
FLOAT);  
  
INSERT INTO demo_var (f1, v1, f2) VALUES (3.20, NULL,  
4.25);
```

Now, to set the value of **VARIANT** column, use update statement and **TO_VARIANT** to convert the **FLOAT** to the **VARIANT** column:

```
UPDATE demo_var  
  
SET v1 = TO_VARIANT(f2); -- converts FROM a float TO  
a variant.
```

You can run **SELECT** to check all values from the table:

```
SELECT * FROM demo_var;
```

F1	V1	F2

3.20	4.25000000000000e+00	4.25
------	----------------------	------

VARIANT can also be used to JSON value as shown below:

```
SELECT TO_VARIANT(PARSE_JSON('{"key1": "value1",
"key2": "value2"}'));
```

You can consider these use cases to use **VARIANT** data type for processing or storing data in table or pipelines:

- Store hierarchical data with explicit definition of hierarchical data using **ARRAY** or **OBJECTS**.
- Loading JSON, Parquet, Avro and ORC file data to the column value directly without specifying explicit schema. Snowflake can convert these formats automatically to the **VARIANT**, **OBJECT** or **ARRAY**. This is an easier approach to have Snowflake read the data and define hierarchical format automatically or implicitly.

OBJECT

Snowflake support to the **OBJECT** data type is analogous to the JSON object. **OBJECT** stores data as key-value pairs and can be accessed as key-value pairs.

In Snowflake, **OBJECT** data type key is a **VARCHAR** and corresponding value is a **VARIANT** data type. The key-value in **OBJECT** cannot be **NULL**. The maximum value it can store is 16MB. This can be used to create hierarchical data and store semi-structured data.

OBJECT constant

Like date constant, you can define an **OBJECT** constant with a given set of values. You can define them in the form of key-value or use **OBJECT_CONSTRUCT** function to define the data as key-value pairs.

For example:

```
Select { 'Alberta': 'Edmonton' , 'Manitoba': 'Winnipeg' } as object_sample;
```

```
Select OBJECT_CONSTRUCT('Alberta', 'Edmonton',
'Manitoba', 'Winnipeg') as sample_object;
```

OBJECT elements

You can access the elements from **OBJECT** by specifying key in square brackets.

For example:

```
select sample_object['key1'] from demo_table;
```

OBJECT considerations

You can consider using **OBJECT** as data type whenever you have these use cases or requirements to store or process data:

- Multiple pieces of string data. You can consider setting up a lookup table to refer to reference values for example, setting up countries and currencies or provinces with their capitals as mentioned above.
- Data that has no natural order and can be saved in a way that can be accessed or queried using key columns.
- Data that varies or data which is not complete. You can have varied set of information for given key values. For example, using object to store details of the book catalog where some of the details may not be available due to old publications. In this case, you can still use object to store relevant values for given key.

ARRAY

This data type is like **ARRAY** in any other language. This can store pieces of information that can be stored and accessed based on the position of the data stored.

Following are the **ARRAY** characteristics:

- Each value in an **ARRAY** is **VARIANT**. Other data type values can be casted using **TO_VARIANT** to be stored in an **ARRAY**.
- Snowflake takes care of implicit conversion however you can also use **ARRAY_CONSTRUCT()** to convert values to **ARRAY**.
- You can store a variety of data with different data types in an **ARRAY** element as this stores **VARIANT** data type data. This is converted to **VARIANT**

implicitly.

- You can define **ARRAY** without any number specified or defined with it. Snowflake does not support fixed number of **ARRAY** as of now.
- Unlike **OBJECT**, **ARRAY** can contain **NULL** values.
- An **ARRAY** can be of 16MB considering the **VARIANT** size and limitations.

ARRAY constant

Similar to **OBJECT**, you can define constant value or set of values as an **ARRAY**. This can also be constructed using **ARRAY_CONSTRUCT**. Refer to the following examples:

```
Select [ 121, 212 ] as sample_array;
```

```
Select ARRAY_CONSTRUCT(121, 212) as sample_array;
```

ARRAY elements

ARRAY elements can be accessed with position values.

For example,

```
select sample_array[2] from demo_table;
```

```
select sample_array[2][0] from demo_table; -- for nested ARRAY records
```

ARRAY considerations

You can consider using **ARRAY** as data type if your data contains any of the data as mentioned below:

- Many pieces of data or information which is structured in the same way.
- Each piece of data should be processed the same way.
- Data has a natural order.

Geospatial data types

Snowflake extends support to geospatial data. These geospatial features can be in the form of points, lines and polygons on the Earth's surface. Snowflake

supports this data in the form of two data types:

- Geometry
- Geography

Geography

This data type stores data in the form of WGS 84 standard. Earth location or point is stored in the form of longitude and latitude. This does not support altitude at this moment. Snowflake also supports Geospatial functions to support operations or handling this data type.

Geometry

This data type stores and represents data in the form of planar in the form of Cartesian coordinate system. The coordinates are represented in the form of (x, y) which represents 2D object or shape. Snowflake also supports extended functions to operate on this data type.

You will learn to use these data types and choose the right fit for your data depending on the type and operations to be performed on them.

Using SQL commands

Snowflake supports ANSI SQL standards. These SQL commands are broadly classified into **Data Definition Language (DDL)**, **Data Manipulation Language (DML)** and **Data Control Language (DCL)**. Snowflake has its own set of commands that can be categorized into one of these categories as extension to support the Snowflake native objects.

Data definition language

DDL commands are used to define and modify objects in Snowflake. Snowflake supports various objects like databases, tables, schemas, views, functions, users, data warehouses, stored procedures, and so on. You will learn more about Snowflake objects and support extended ones in the next section of this chapter. DDL command can also be used to set environment variables or local parameters.

These are the DDL commands supported:

- **ALTER <object>**

- **COMMENT**
- **CREATE <object>**
- **DESCRIBE <object>**
- **DROP <object>**
- **SHOW <objects>**
- **USE <object>**

These DDLs are also categorized into groups as categories based on the type of operations, Snowflake objects and processing:

- Account and session DDL
- User and security DDL
- Warehouse and resource monitor DDL
- Database, schema, and share DDL
- Table, view, and sequence DDL
- Data loading / unloading DDL
- DDL for user-defined functions, external functions, and stored procedures
- Data pipeline DDL

Account and session DDL

These are the DDL commands used to set account and session level parameter. Along with setting and using parameters these can also be used to:

- Viewing parameters in the system at multiple levels.
- Account-level and session-level parameter setting.
- Statement to set a role, warehouse, database, or schema within a session.
- Supports multi-statement transactions within a session.
- Used to set and use SQL variables within a session.

DDL commands

Following are the DDL statements that are used to perform operations at account or session level operations. These DDL statements are categorized into type of

operations performed:

Type	Commands	Description
Account and functions	ALTER ACCOUNT	Used to set account parameters. Only users with ACCOUNTADMIN can use this command.
	SHOW FUNCTIONS	List down all system and user defined functions.
	SHOW PARAMETERS	List down all parameters listed for an account.
Managed accounts	CREATE MANAGED ACCOUNT	Used to create READER accounts used to share data with non-snowflake users. You will learn more about this in next chapter.
	DROP MANAGED ACCOUNT	Used to remove READER account
	SHOW MANAGED ACCOUNT	List down all READER accounts in account.
Session context	USE ROLE	Used to set the Role to be used in session.
	USE DATABASE	Used to set the database to be used in session.
	USE WAREHOUSE	Used to set the warehouse to be used in session.
	USE SCHEMA	Used to set the schema to be used in the session.
Session	BEGIN	This is used for multi-statement Begin transaction (BT) and End Transaction (ET) block.
	COMMIT	Commit the transactions
	ROLLBACK	Rollback transactions
	SHOW TRANSACTIONS	List down all running transactions
SQL variables	DESCRIBE TRANSACTIONS	Shows the state of transactions
	SET	Used to set a variable value or assign a value to the variable in the session.
	UNSET	Used to unset the value to the variable in the session
	SHOW VARIABLES	List down all used variables in the session

Table 3.10: Snowflake DDL Commands

User and security DDL

Snowflake supports a variety of commands to manage users and security. These commands can only be used by **ACCOUNTADMIN** or the role with **OWNERSHIP** permissions. These are the tasks users can perform:

- Password change
- Viewing user information
- Change role, virtual warehouse, namespace, and so on.
- Change session parameters

DDL commands

Following are the DDL statements that are used to manage users and security at account or session level operations. These DDL statements are categorized into type of operations performed:

Type	Commands	Description
User management	CREATE USER	Used to create users.
	ALTER USER	Used to alter user details.
	DROP USER	Used to drop users.
	DESCRIBE USER	Used to describe user.
	SHOW USERS	List down users.
Role management	CREATE ROLE	Used to create a role.
	ALTER ROLE	Used to alter details of a role.
	DROP ROLE	Used to drop a role.
	SHOW ROLES	List down all roles.
Object tagging management	CREATE TAG	Used to create object tag.
	ALTER TAG	Used to alter detail of object tag.
	SHOW TAGS	List down all tags.
	DROP TAG	Used to drop tags.
	UNDROP TAG	Used to recover dropped tag.
Access management	GRANT	Used to grant permissions of the role, and objects.
	REVOKE	Used to revoke permissions of the role, and objects.

	GRANT ROLE	Used to grant role to the user.
	GRANT OWNERSHIP	Used to grant ownership of the objects.
	REVOKE ROLE	Used to revoke role granted.
Third party integrations	CREATE/DROP/SHOW/DESCRIBE	These are DDL commands used to create and manage integrations. You will learn more about these integrations in upcoming chapters.

Table 3.11: Snowflake DDL commands for user management

Warehouse and resource monitor DDL

Snowflake's warehouse is compute resource. Warehouse is required to run DML operations of the workloads, queries. Warehouse DDLs can be used to create and manage warehouse resources. Resource monitors are the objects that can be used to monitor and control usage of warehouse and quota. You will learn more about resource monitors in the upcoming chapter.

DDL commands

Following are the DDL statements that are used to manage warehouse and resource monitors at account. These DDL statements are categorized into type of operations performed:

Type	Commands	Description
Virtual warehouses	CREATE WAREHOUSE	Used to create warehouses.
	DROP WAREHOUSE	Used to drop warehouses created.
	ALTER WAREHOUSE	Used to alter warehouse details.
	DESCRIBE WAREHOUSE	Used to describe warehouse.
	SHOW WAREHOUSES	List down warehouse created.
	USE WAREHOUSE	Used to set warehouse to be used for the given session.
Resource monitors	CREATE RESOURCE MONITOR	Used to create resource monitor.
	DROP RESOURCE MONITOR	Used to drop resource monitor.
	SHOW RESOURCE MONITOR	List down all resource monitors.
	ALTER RESOURCE MONITOR	Used to alter resource monitor created.

Table 3.12: Snowflake DDL commands for Warehouses

Database, schema and share DDL

Snowflake supports database and schema as part of their resource hierarchy. Databases and schema can be created and used to organize, manage objects. A database is a logical grouping of schemas. Schema is a logical group of database objects that is, tables, views, and so on. Snowflake supports DDL statements that can be used to create databases, schema, and database objects.

DDL commands

Following are the DDL statements that are used to create, manage databases and objects at account. These DDL statements are categorized into type of operations performed:

Type	Commands	Description
Database management	CREATE DATABASE	Used to create databases.
	DROP DATABASE	Used to drop databases created.
	ALTER DATABASE	Used to alter database details.
	DESCRIBE DATABASE	Used to describe database.
	SHOW DATABASES	List down databases created.
	USE DATABASE	Used to set database to be used for the given session.
	UNDROP DATABASE	Used to undrop database dropped.
Schema management	CREATE SCHEMA	Used to create schema.
	DROP SCHEMA	Used to drop schema.
	SHOW SCHEMAS	List down all schemas.
	ALTER SCHEMA	Used to alter schema created.
	USE SCHEMA	Used to set schema to be used for the given session.
	UNDROP SCHEMA	Used to undrop schema dropped.
Share management	CREATE SHARE	Create share.
	ALTER SHARE	Alter share created.
	DROP SHARE	Drop shares created.
	ALTER SHARE	Alter share created.
	DESCRIBE SHARE	Describe share created.
	SHOW SHARES	List down shares created.

Table 3.13: *Snowflake objects DDL commands*

Table, view and sequence DDL

Snowflake supports creation, managing and using database objects using DDL statements. Database objects consist of various objects like tables, view, stored procedures, functions etc. This section shares details of table, view, and sequence DDLs. Data is stored in the table in the form of rows and columns. View can be used to select rows and columns in one or more tables.

Snowflake also supports sequences, these are schema-level objects. These are used to generate sequence as unique numbers. Sequences can be used to maintain unique or primary key in the table.

DDL commands

Following are the DDL statements that are used to create and manage database objects at account. These DDL statements are categorized into type of operations performed:

Type	Commands	Description
Table management	CREATE TABLE	Used to create table.
	DROP TABLE	Used to drop table created.
	ALTER TABLE	Used to alter table details.
	DESCRIBE TABLE	Used to describe table.
	SHOW TABLES	List down tables created.
	SHOW COLUMNS	Used to list all columns of the table.
Standard view management	UNDROP TABLE	Used to undrop table dropped.
	CREATE VIEW	Used to create view
	DROP VIEW	Used to drop view
	SHOW VIEWS	List down all views
	ALTER VIEW	Used to alter view created
	DESCRIBE VIEW	Used to describe view
Sequence management	CREATE SEQUENCE	Create sequence
	ALTER SEQUENCE	Alter sequence created
	DROP SEQUENCE	Drop sequence created
	SHOW SEQUENCES	List down all sequences created
	DESCRIBE SEQUENCE	Describe sequence created

Table 3.14: Snowflake Data objects DDL commands

Data loading / unloading DDL

Snowflake supports data loading and unloading features. You can use COPY command to load/unload data to or from Snowflake. There are set of objects that need to be created and used to perform loading/unloading operations. You will learn more about Snowflake data loading/unloading in the upcoming chapter.

This section lists the various objects that can be created to be used for the operations.

DDL commands

Following are the DDL statements that are used to create, manage loading/unloading objects at account. These DDL statements are categorized into type of operations performed:

Type	Commands	Description
Stage management	CREATE STAGE	Used to create stage
	DROP STAGE	Used to drop stage created
	ALTER STAGE	Used to alter stage details
	DESCRIBE STAGE	Used to describe stage
	SHOW STAGES	List down stages created
	CREATE STAGE CLONE	Create stage that is cloned from other stage/database object
File format management	CREATE FILE FORMAT	Used to create file format
	DROP FILE FORMAT	Used to drop file format
	SHOW FILE FORMATS	List down all file formats
	ALTER FILE FORMAT	Used to alter file format created
	DESCRIBE FILE FORAT	Used to describe file format
Pipe management	CREATE PIPE	Create pipe
	ALTER PIPE	Alter pipe created
	DROP PIPE	Drop pipe created
	SHOW PIPES	List down all pipe created
	DESCRIBE PIPE	Describe pipe created

Table 3.15: Snowflake DDL commands for COPY

User-defined functions, external functions, and stored procedures DDL

Snowflake extends support to the user defined functions and stored procedures. You can use SQL standard DDL commands to create these database objects. Snowflake also extends support to the external functions, these can be created and invoke outside Snowflake. External functions are also categorized as user defined functions and can be created using function DDL commands.

DDL commands

Following are the DDL statements that are used to create, manage database objects like functions and stored procedure at account. These DDL statements are categorized into the type of operations performed:

Type	Commands	Description
UDF management	CREATE FUNCTION	Used to create function
	DROP FUNCTION	Used to drop function created
	ALTER FUNCTION	Used to alter function details
	DESCRIBE FUNCTION	Used to describe function
	SHOW USER FUNCTIONS	List down functions created
External Function Management	CREATE EXTERNAL FUNCTION	Used to create external function
	ALTER FUNCTION	Alter function details
	DROP FUNCTION	Drop function created
	SHOW EXTERNAL FUNCTIONS	List down all external functions in the account
Stored procedure management	DESCRIBE FUNCTION	Used to describe external function
	CREATE PROCEDURE	Create procedure
	ALTER PROCEDURE	Alter procedure created
	DROP PROCEDURE	Drop procedure created
	SHOW PROCEDURES	List down all procedures created
	DESCRIBE PROCEDURE	Describe procedure created

Table 3.16: Snowflake DDL commands for functions and procedures

Data pipeline DDL

Snowflake supports pipeline creation as well as scheduling pipelines. Snowflake native objects tasks and streams are used to create data pipeline and scheduling them to invoke the pipelines or objects (functions or stored procedures). You will learn more about streams and tasks in subsequent chapters in this book.

DDL commands

Following are the DDL statements that are used to create and manage Snowflake objects like tasks and streams at account. These DDL statements are categorized into type of operations performed:

Type	Commands	Description
Stream management	CREATE STREAM	Used to create stream
	DROP STREAM	Used to drop stream created
	ALTER STREAM	Used to alter stream details
	CREATE STREAM..CLONE	Used to create stream from cloned objects
	SHOW STREAMS	List downstream created
Task management	CREATE TASK	Used to create task
	ALTER TASK	Alter task details
	DROP TASK	Drop task created
	SHOW TASKS	List down all tasks in the account
	EXECUTE TASKS	Used to execute tasks created
	CREATE TASK CLONE	Create task from cloned objects

Table 3.17: Snowflake DDL commands for pipeline

Data manipulation language

Data manipulation language consists of set of commands that can be used to perform data operations, deals with managing, manipulating, transforming data. These DML commands are categorized into these categories based on the type of operations it supports:

- Standard DML
- Data loading/unloading DML
- File staging commands

Standard DML

This category consists of the standard commands used to load, transform, delete, update data based on the operations being performed.

DML commands

Command	Description
INSERT	Used to load data into a table
UPDATE	Used to update table fields based on the conditions specified.
DELETE	Used to delete data from a table based on the conditions specified if any
MERGE	Used to perform a combine operation to insert , update and delete data based on the multiple conditions. This can be used to perform multiple operations in a single command.
TRUNCATE TABLE	Delete all records from the table. Empty's table.

Table 3.18: Snowflake DML commands

Data loading/unloading DML

Like data loading/unloading DDL commands, set of DML commands are specified to be used along with the objects created. The DML statement **COPY** is used to load and unload data. You will learn more about **COPY** command and the operations in the upcoming chapter.

DML commands

Command	Description
COPY INTO <table>	Used to load data into a table
COPY INTO <location>	Used to unload data from a table to specified location.

Table 3.19: Snowflake COPY DML commands

File staging commands

Snowflake supports staging commands to manage staging objects created to support data load/unload operations. File stages can be created and accessed in Snowflake. These objects use the following commands to store, manage data in it.

DML commands

Command	Description

PUT	Used to put or upload data from local to the internal stage object
GET	Used to download data from internal stage to the local system or machine
LIST	Used to list down the objects from the stage
REMOVE	Used to delete object from the stage

Table 3.20: Snowflake File commands

As you go along with the chapters, you will use various DDL , DML and other SQL commands to perform various operations on data.

Creating database objects

Snowflake supports various database objects and extends support to the additional objects that are treated as database objects. Snowflake follows a resource hierarchy. A Snowflake account is the topmost resource in the hierarchy, one account can have multiple databases, each database can have multiple schemas, each schema can have multiple objects like tables, views, functions, and so on. Take a look at the flow chart below for a better understanding:

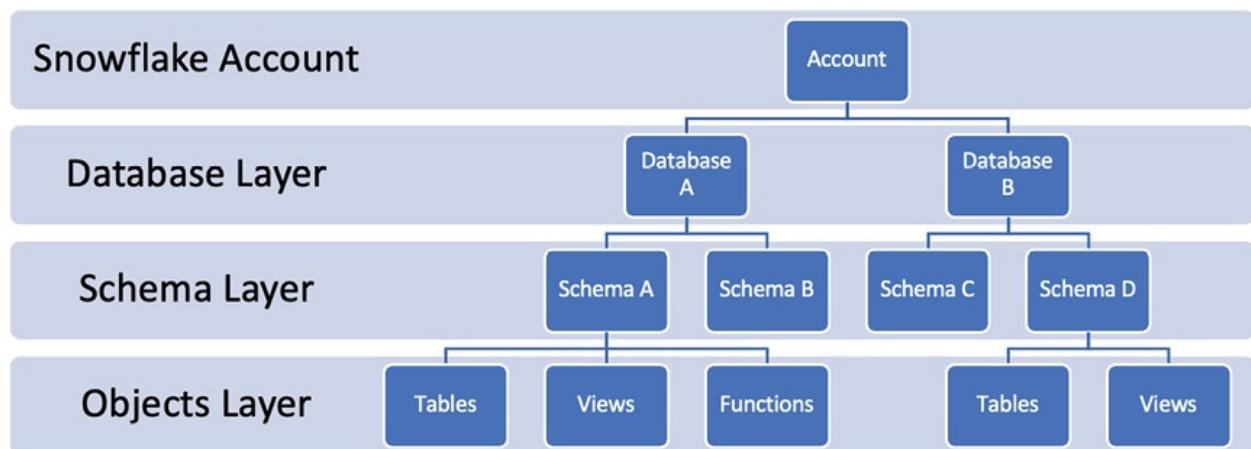


Figure 3.1: Snowflake objects hierarchy

You can use DDL commands to create objects in Snowflake. Consider a scenario where you are setting up an account for a customer **Proof of Concept (POC)** and you need to create resources – users, roles as well as objects, pipelines required to perform POC. You can follow the following steps to setup an account and objects:

1. Create users and roles
2. Create warehouses

3. Create databases and schemas
4. Setting up context
5. Create tables, view and other objects

You can consider the below use case details to setup objects for the given POC:

- **Domain:** Retail
- **POC:** Setup an account for retail use case
- **Sub-domains:** Sales, product, marketing
- **Users:** Data engineering, data analysts
- **Requirements:** Unified data platform that consists of all sub-domain data and various team can process, access data from the platform. All users from these sub-domain need access to the databases and objects that are relevant to the users. You need to set up the access, track usage, monitor performance for the sub-domain users.

Now, you can follow the steps mentioned to set up a POC account for retail domain customer.

Creating users and roles

Users need access to their domain data and each sub-domain is equivalent to a team that needs access to their datasets. You can onboard various users based on the requirements. You can setup the **Active Directory (AD)** integration to onboard users. You will learn more about this in the next chapter. For now, consider there are multiple users to be onboarded and each user needs corresponding access to the datasets.

Snowflake access is managed as **Role Based Access Control (RBAC)**. You can create roles as per the teams and their responsibilities. In the given scenario, you have 2 users: data engineering and analyst. You can create these 2 as roles to be granted to the users.

In Snowflake, you can create all objects using SQL commands or you can use Snowflake Web UI / Snowsight to create these objects using the ribbon options.

Creating roles using SQL commands

Create two roles as per the requirements for two teams. SYSADMIN is the

privilege or default role to be granted to the users to perform operations:

```
CREATE ROLE DATA_ENG COMMENT='This is Data engineering role'
```

```
GRANT ROLE DATA_ENG TO ROLE SYSADMIN;
```

```
CREATE ROLE DATA_ANALYST COMMENT='This is Data analyst role'
```

```
GRANT ROLE DATA_ANALYST TO ROLE SYSADMIN;
```

Creating users using SQL commands

Create user as per the requirements for two teams:

```
CREATE USER engineering_user1 PASSWORD='demo123'  
DEFAULT_ROLE = DATA_ENG MUST_CHANGE_PASSWORD = TRUE;
```

```
CREATE USER analyst_user1 PASSWORD='demo123'  
DEFAULT_ROLE = DATA_ANALYST MUST_CHANGE_PASSWORD =  
TRUE;
```

Creating roles using Web Console

You can create a Role using Web UI. You need **ACCOUNTADMIN** or **SYSADMIN** or associated privileges to create new roles. Go to **Account | Roles** in classic console. You can also do the same using Snowsight, go to **Admin | Users & roles**. Provide the same details as shown in *Figure 3.2*:

Create Role

Name *	DATA_ENG
Parent Role	SYSADMIN
Comment	This is Engineering Role

[Show SQL](#) [Cancel](#) [Finish](#)

Figure 3.2: Create Role using Console

Click on **Finish** to create the role. You can also click on **Show SQL** to get the SQL command and run it from the console.

Creating users using Web Console

You can create users using Web Console, as shown in the following figure:

Create User

General Advanced Preferences

Please select a new password that is 8 - 256 characters long and contains at least 1 digit(s), 1 lowercase letter(s), and 1 uppercase letter(s).

User Name * engineering_user1

New Password **

Confirm Password *

Comment This is Engineering user

Force Password Change [?](#)

Show SQL Cancel Next **Finish**

The screenshot shows a 'Create User' dialog box. At the top, there are three tabs: 'General' (highlighted in blue), 'Advanced', and 'Preferences'. A note below the tabs says: 'Please select a new password that is 8 - 256 characters long and contains at least 1 digit(s), 1 lowercase letter(s), and 1 uppercase letter(s.)'. The 'User Name' field contains 'engineering_user1'. The 'New Password' and 'Confirm Password' fields both show '.....*'. The 'Comment' field contains 'This is Engineering user'. A checkbox labeled 'Force Password Change' is checked. At the bottom, there are four buttons: 'Show SQL' (underlined), 'Cancel', 'Next', and a large blue 'Finish' button.

Figure 3.3: Create User using Console

Click on **Finish** to create the user. The **Force Password Change** checkbox enables users to change the password for the first-time login. You can also click on **Show SQL** to get the SQL command and run it from the console.

Creating warehouses

Once you have roles created and users onboarded to the account, you need to create a warehouse that supports DML operations to be performed by the user. As you know, a warehouse is a compute power required to perform operations, run queries. Warehouse billing is dependent on the usage. Considering the given use case and requirement to monitor performance and usage of warehouse, you can create separate warehouse for development and analysis work.

The data engineering team will use the engineering warehouse created, and the analyst team will use warehouse setup to run analytical queries. Engineering warehouse can be a cluster depending on the development workload and analytical one can be a standalone with small or medium size to run analytical

queries.

As you know, warehouse can be created using SQL command as well as console options.

Creating warehouse using SQL command

Run the following command to create an engineering warehouse, which is a cluster warehouse with maximum 2 nodes and minimum -1 with auto suspend after 5 minutes:

```
CREATE WAREHOUSE DEV_ENG_WH WITH WAREHOUSE_SIZE =  
'SMALL' WAREHOUSE_TYPE = 'STANDARD' AUTO_SUSPEND = 300  
AUTO_RESUME = TRUE MIN_CLUSTER_COUNT = 1  
MAX_CLUSTER_COUNT = 2 SCALING_POLICY = 'STANDARD'  
COMMENT = 'This is an Engineering Warehouse';
```

Run the following command to create an analytical warehouse , which is a warehouse with medium size and auto suspend after 5 minutes:

```
CREATE WAREHOUSE DEV_ANALYTICAL_WH WITH WAREHOUSE_SIZE  
= 'MEDIUM' WAREHOUSE_TYPE = 'STANDARD' AUTO_SUSPEND =  
300 AUTO_RESUME = TRUE MIN_CLUSTER_COUNT = 1  
MAX_CLUSTER_COUNT = 1 SCALING_POLICY = 'STANDARD'  
COMMENT = 'This is an Analytical Warehouse';
```

Creating warehouse using Web Console

You can use console options to setup a warehouse, as shown in the following figure:

Create Warehouse

Name *	DEV_ENG_WH
Size	Small (2 credits / hour)
Learn more about virtual warehouse sizes here	
Maximum Clusters	2
Multi-cluster warehouses improve the query throughput for high concurrency workloads.	
Minimum Clusters	1
The number of active clusters will vary between the specified minimum and maximum values, based on number of concurrent users/queries.	
Scaling Policy	Standard
The policy used to automatically start up and shut down clusters.	
Auto Suspend	5 minutes
The maximum idle time before the warehouse will be automatically suspended.	
<input checked="" type="checkbox"/> Auto Resume ?	
Comment	This is a Engineering Warehouse
Show SQL Cancel Finish	

Figure 3.4: Create Warehouse using Console

Creating databases and schemas

Once you have users onboarded, roles assigned, and the warehouse created, then you can set up the databases and schemas required. For the given use case, there are 3 sub domains. You can consider creating these three as schemas that can store corresponding tables. Database can represent the domain name.

Creating database using SQL command

You can use DDL command from the list to set up the database:

```
CREATE DATABASE RETAIL_DB COMMENT = 'This is Retail Database';
```

Creating database using web console

Go to **Account | Databases**. Click on + sign to create a database. The following pop-up box opens up:

The screenshot shows a 'Create Database' dialog box. At the top, it says 'Create Database'. Below that, there are two input fields: 'Name *' with the value 'RETAIL_DB' and 'Comment' with the value 'This is Retail Database'. At the bottom, there are three buttons: 'Show SQL' (underlined), 'Cancel', and a blue 'Finish' button.

Figure 3.5: Create Database using Console

Creating schemas using SQL command

There are 3 schemas to be setup for sub-domain: product, sales, marketing. The commands for the same are as follows:

```
CREATE SCHEMA PRODUCT COMMENT='this is a product schema';  
CREATE SCHEMA SALES COMMENT='this is a sales schema';  
CREATE SCHEMA MARKETING COMMENT='this is a marketing schema';
```

Setting up context

This is the most important session setting required to perform any operations on Snowflake. There are two ways users can set this up. You can use SQL command to set this up or use console options to set up these parameters.

Context setting refers to setting up session variables to use specific role, database, warehouse, and schema to perform operations specified in the session.

Setup context using SQL command

You can setup session parameters using the **USE** SQL command:

```
USE ROLE DATA_ENG;  
USE WAREHOUSE DEV_ENG_WH;  
USE DATABASE RETAIL_DB;  
USE SCHEMA PRODUCT;
```

Creating tables and views

Now, you can create database objects once the context is set. All operations run with this context will use given warehouse, role and objects like database and schema.

Creating table using SQL command

Create two sample tables in the **PRODUCT** schema, as shown:

```
create or replace TABLE ORDERS (  
    ORDER_ID VARCHAR(30),  
    STORE_ID NUMBER(38, 0),  
    ORDER_DATE DATE,  
    TEMPERATURE VARCHAR(50),  
    FUEL_PRICE FLOAT,  
    CPI VARCHAR(50),  
    UNEMPLOYMENT VARCHAR(50),  
    ISHOLIDAY VARCHAR(50),  
    CREATED_AT TIMESTAMP_NTZ(9),  
    UPDATED_AT TIMESTAMP_NTZ(9),  
    CDC_FLG VARCHAR(10),  
    FILE_NAME VARCHAR(100),
```

```
FILE_FORMAT VARCHAR(20),  
BATCH_LOAD_ID NUMBER(38, 0)  
);  
  
create or replace TABLE STORES (  
    STORE_ID NUMBER(38, 0),  
    STORE_TYPE VARCHAR(1),  
    STORE_SIZE NUMBER(38, 0),  
    CREATED_AT TIMESTAMP_NTZ(9),  
    UPDATED_AT TIMESTAMP_NTZ(9),  
    CDC_FLG VARCHAR(10),  
    FILE_NAME VARCHAR(100),  
    FILE_FORMAT VARCHAR(20),  
    BATCH_LOAD_ID NUMBER(38, 0)  
);
```

You can create the view using **CREATE VIEW DDL** statement on top of tables present in the databases.

These are some of the key steps while setting up an account for a use case to create all the required database objects – users, roles, warehouses, databases and schemas. Now, developers or analysts will start building their pipelines to load, process, and transform data to be available for analysts. Next, you will learn about extended support to the objects.

Implementing extended SQL objects

Snowflake users can create user defined functions to extend the features on top of system defined functions. **User Defined Functions (UDFs)** are used to modularize the code that is used recurrently to perform any specific operations. For instance, consider generating the unique number or batch id to represent the

batch of the data loaded to the table. You can create a generic logic and a function that can be used to create a batch ID. This can be called whenever you need to load batch ID.

Snowflake also offers support to the scalar as well as tabular functions. You can create functions using your choice of programming language and setup functions. Snowflake supports Java, JavaScript, Python, Scala and SQL to create user defined functions.

UDF example (JavaScript)

This is a sample JavaScript UDF which gives factors of the number:

```
CREATE OR REPLACE FUNCTION js_factor(p double)
RETURNS double
LANGUAGE JAVASCRIPT
STRICT
AS '
if (D <= 0) {
    return 1;
} else {
    var result = 1;
    for (var i = 2; i <= D; i++) {
        result = result * i;
    }
    return result;
}
';
```

UDF example (SQL)

This is a sample SQL UDF that generates the **batch_id** as unique identifier using

DATE elements:

```
CREATE OR REPLACE FUNCTION
RETAIL_POC_DB.RETAIL_BATCH_POC.GENERATE_BATCH_ID("EXC"
NUMBER(38,0))

RETURNS VARCHAR(12)

LANGUAGE SQL

AS '
    select concat(substr(current_date,1,4) ,
substr(current_date,6,2),
substr(current_date,9,2),exc)
';

'
```

These are samples for your reference. You can create and use your choice of programming language to create UDFs. Next, you will learn about the external functions.

Tabular functions

Snowflake supports UDFs that return multiple rows contrary to the scalar functions. These are referred to as **User Defined Table Functions (UDFT)**. This can be used and accessed from the **FROM** clause of the query.

Sample function

This function generates a static output with **Hello Word!**

Creating function

You can use the following SQL to create a UDFT:

```
/* create function */
```

```
CREATE FUNCTION demo_udft()
RETURNS TABLE(msgs VARCHAR)
AS
```

```
$$
    SELECT 'Hello'
    UNION
    SELECT 'World!'
$$;
```

Using function

UDFTs can be used only in **FROM** clause of the query:

```
SELECT msgs   FROM TABLE(demo_udft()) ORDER BY msgs;
+-----+
| MSGS   |
+-----|
| Hello  |
| World! |
+-----+
```

SQL UDFT function

This is another example where filtering out the data from another table based on the specified product ID.

Creating function

Let us take a look at the following example:

```
/* create table */
create or replace table orders_info (
    product_id varchar,
    quantity_sold numeric(11, 2)
);
```

```
/* load table */

insert into orders_info (product_id, quantity_sold)
values
    ('Recycled bags', 2500),
    ('Paper cups', 1100),
    ('travel bags', 2100);

/*create UDFT*/

create or replace function orders_of_products(Prod_ID
varchar)
    returns table (Prod_id varchar, Qty_Sold
numeric(11, 2))
as
$$
    select product_id, quantity_sold
        from orders_info
        where product_id = Prod_ID
$$
;
```

Using function

You can run the following SQL to get data from UDFT:

```
select Prod_id, Qty_sold
```

```

from table(orders_of_products('travel bags'))
order by prod_id;

+-----+-----+
| Prod_id      | Qty_sold    |
+-----+-----+
| travel bags |      5200.00 |
+-----+-----+

```

These are some of the examples where UDFTs can be used to derive required fields from huge tables with given filter.

External functions

Snowflake extends support to the external functions where the code executes outside of Snowflake. These are the types of UDFs where they do not have their own code. They execute the code that is stored and executed outside. Snowflake saves this as database object in database and schema.

The code that executes remotely can also be referred as remote service. The information is shared with remote service is using proxy service. Snowflake uses API integration to store security related information for these functions. There are various use cases where external functions can be used.

For example, consider a scenario where you need to push some metrics or data from Snowflake to another service like AWS cloud watch for logging or generating SNS alerts. You can create an AWS external function and integrate using API integration. This can be invoked using **CALL** statement which invokes the code and sends data to Cloudwatch. You will learn more about API integration in the subsequent chapters.

Conclusion

This chapter provided a summary of Snowflake's data objects and SQL commands supported to create, use them for data processing. This chapter helped you to understand various Snowflake commands used as well as Snowflake native or supported data objects like Pipe, Streams, External functions, and so on. You will learn more about using Snowflake commands for

loading and unloading, using streams and tasks as you go along with the next chapters in this book.

In the next chapter, you will learn about Snowflake's **COPY** command and its usage to load as well as unload data. You will also learn more about native features that can be used to support the data needs. This is the next chapter in the series to understand, learn, explore and gain hands on experience with Snowflake.

Points to remember

- Snowflake supports ANSI SQL standards.
- Snowflake supports all native data objects like tables, views, functions, stored procedures, and so on.
- There are some of the Snowflake specific objects that can be created using DDL commands and accessed using DML commands to develop, use data pipelines.
- Snowflake supports various use cases to load , transform and process data.

Practical: Create Snowflake objects

- **Exercise 1:** Account setup using console:
 - Once Snowflake trial account is setup, create sample users, roles, warehouses, databases and schemas.
 - Create 2 roles: Engineering and Data Scientists
 - Create 2 users: Engineer and Analyst
 - Create 2 warehouses: Engineering and ML
 - Create 2 databases for Retail domain: **RETAIL_DEV** and **RETAIL_ML**
 - Create schemas within databases:
 - **RETAIL_DEV** consist of **Product**, **Sales**, **Customer**
 - **RETAIL_ML** consist of **Customer_profiles**
 - Create tables within **RETAIL** schemas.

- **Exercise 2:** Account setup using SQL commands:
 - Once Snowflake trial account is setup, create sample users, roles, warehouses, databases and schemas.
 - Create 2 roles: Engineering and Data Scientists
 - Create 2 users: Engineer and Analyst
 - Create 2 warehouses: Engineering and ML
 - Create 2 databases for **Retail** domain – **RETAIL_DEV** and **RETAIL_ML**
 - Create schemas within databases:
 - **RETAIL_DEV** consist of **Product, Sales, Customer**
 - **RETAIL_ML** consist of **Customer_profiles**
 - Create tables within **RETAIL** schemas.

Multiple choice questions

1. **What are the data types used for semi-structure data processing? (Select all that apply)**
 - a. JSON
 - b. ARRAY
 - c. VARIANT
 - d. OBJECT
 - e. PARQUET
2. **What are the objects can be set as context in Snowflake?**
 - a. Role
 - b. Warehouse
 - c. Schema
 - d. Database

- e. All of the above

3. What are the types of functions supported in Snowflake?

- a. Scalar Function (UDF)
- b. UDFT
- c. External Function
- d. Python Function
- e. Only A, B, C

4. What command can be used to data load and unload?

- a. LOAD
- b. UNLOAD
- c. COPY
- d. COPY INTO

Answers

1	b, c, d
2	d
3	e
4	c

Questions

1. What are the various data types supported in Snowflake?
2. How does Snowflake support geospatial data with its data types?
3. What are the data types used to store and process hierarchical data?
4. How does Snowflake use **VARIANT** to store semi-structure data?
5. What are the Snowflake objects that can be created using DDL

statements?

6. How can Snowflake warehouse be created and used?
7. What is UDF in Snowflake?

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 4

Data Loading and Unloading

Introduction

Data loading and unloading is one of the most important parts of data platform implementation. This chapter guides the reader to learn native utilities and commands supported by Snowflake. This covers various commands used to load batch and streaming datasets. This also covers data extraction and sharing with consumer groups.

You will learn about Snowflake data loading and unloading capabilities in this chapter of this book. This chapter illustrates capabilities with real-time use cases and references to integrate source and target integrations. Sample examples and lab questions in this chapter will help you learn with hands-on labs. There are set of questions to check your knowledge on understanding concepts explained in this chapter.

Structure

This chapter consists of the following topics:

- Understanding data load needs
- Creating Snowflake load objects
- Using `COPY INTO` to load data
- Understanding streaming loads
- Implementing Snowpipe

- Understanding data unloading
- Using **COPY** to unload data
- Bulk unloading from a table or a query
- Bulk unloading to one or multiple files

Objectives

By the end of this chapter, you will be able to understand the data loading/unloading capabilities of Snowflake. This chapter helps you to create required objects to use data loading and unloading with **COPY** command. You will also be able to use Snowflake utility with a trial account setup in [Chapter 1, Getting Started with Snowflake](#).

Understanding data load needs

An enterprise data platform is integrated with heterogeneous sources. These sources bring in a variety of data. These sources can be other data systems, applications that generate data, source files as well as real-time data sources that send data in a real time. As part of enterprise data platform, you need to integrate a variety of sources to your data platform. Data can be processed, transformed and ready to be analyzed on the platform.

Typically, the data is loaded to landing layer from sources in the original form (same as source data). Once you have data staged to raw or landing layer, you can run data enrichments, transformations, and data quality checks on top of your data. Data loading is the initial and most critical part of the data platform. You can refer to the below reference architecture figure of a typical enterprise data warehouse:

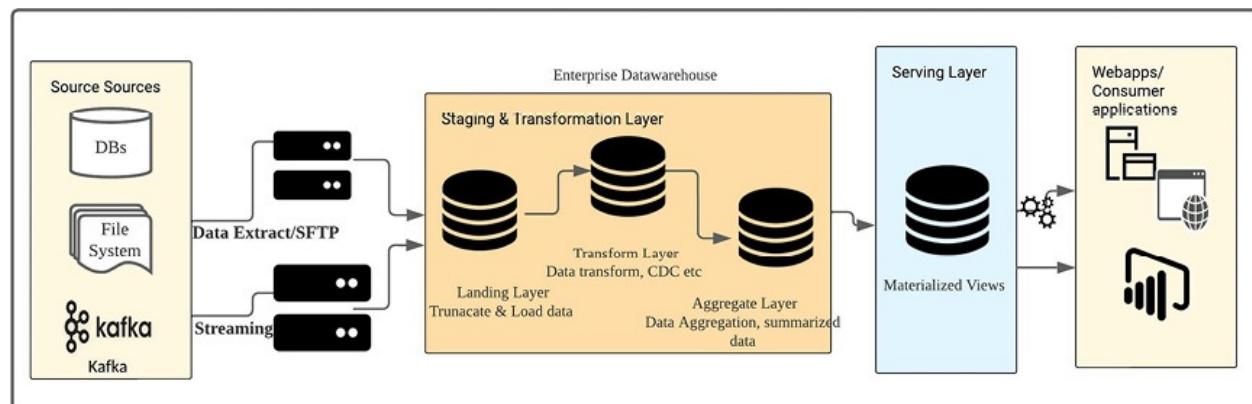


Figure 4.1: Enterprise Warehouse

In this enterprise architecture, you can observe that there are a variety of sources – file feeds, database sources and a streaming source (Kafka that brings in streaming data). Considering the heterogeneous sources to be integrated, there are various approaches to integrate data:

- **Files:** Files are shared by source systems over an SFTP. This SFTP can be a shared drive or File server or **Network Attached Storage (NAS)**. File load job can be used read files and load data to the raw layer.
- **DB sources:** DB sources can be integrated by running an extract script to read data from database sources and load to the target raw layer. This extract can be developed using scripts or ETL tools or DB native utilities to export data. This approach can become a time-consuming task considering the volume of data to be extracted and loaded to the data warehouse.
- **Real-time streaming:** In the given scenario, Kafka is used to get streaming data. Kafka connectors can be used to read data from streams and load to landing layer. You can load streaming data that can be a real time or near real time.

Imagine the efforts and complexities involved in the integrating and processing data to the data warehouse. You can use native utilities, ETL tools, ELT tools to extract data from sources and load to the data platform.

Creating Snowflake load objects

Snowflake supports various ways to load data into Snowflake objects. As you have seen in the above example, *Figure 4.1*, enterprise warehouse data sources can vary, and ways to load data into Snowflake vary accordingly. You will need various services, objects, programs, utilities, or tools to bring in data to the Snowflake tables. You will learn more about some of the most used sources and ways to load data in this section.

Source files

This is a widely used and common type of source system where files are sent (pushed) or pulled from the sources. To load files into Snowflake, you can consider two scenarios – loading files from a file location and loading from local systems.

Loading from file locations

Snowflake can read files from a storage location like storage bucket on cloud. There are two types of objects required to be created while reading files from the storage buckets or locations.

If you are reading data from storage location managed by a business entity (outside the cloud where Snowflake is hosted) then you need to create an external stage object to read data from this location. If you are reading data from the storage location contained in Snowflake account, then you need to create an internal stage to read data from this location.

You can read data from storage location that can be part of a batch load or real-time loads. Real-time loads can be a near real time commonly referred as mini or micro batches to read data at a lower frequency from the storage locations.

When you are reading data from a file, you will need to know the file configurations. File configurations include details like type of file, delimiter of file in case delimited file, header details in case file consist of headers, skip records in case any records to be skipped etc. These configurations can be defined in a Snowflake object named – file format. These needs to be created to define file properties to be used while reading and loading file using **COPY** command from any stage locations.

file format, stages, and storage integrations are the Snowflake objects required to load source files. You will learn more about creating required Snowflake objects to load data from files in this section.

Internal stage

Stage is an object where data files are stored to be loaded into a table. There are 3 types of internal stages:

- **User stage:** This stage is created as and when a user is created in Snowflake. Users can use this stage to store files that can be used to load into various tables. This cannot be created, altered, or dropped using SQL command as this is the default stage allocated to every user. One user, one stage is used to load into multiple tables. You can list down the access user stage using the **LIST** command. User stages are usually referred as **@~[/path]**. You can also use this stage in **COPY** command to load data. You will learn more about **LIST** in the next section.
- **Table stage:** This stage is created as and when a table is created in

Snowflake. Users can use this stage to store files to be loaded to the corresponding table. This is used to stage files used by many users for the same table. This cannot be created, altered, and dropped using SQL command as this is created by default. One stage, many users, is to be loaded to only one table. You can access data files in the table stage using the **LIST** command. You can also use this stage in **COPY** command to load data. Table stages are referred to as **@%tablename**.

- **Named stage:** This stage is an object created with a SQL command. This stage can be used to store and load multiple files by multiple users and for multiple tables. Like any other database object, you can use SQL commands to create, modify and drop this stage. You can use **PUT** command to upload files to the internal stage from your local file systems. You will learn more about **PUT** and usage in loading from local system section. You will learn more about creating internal and external stages in upcoming sections. You can use **LIST** command to view all files in internal and external stages.

SQL commands

DDL commands can be used to create, alter, and drop stages. You can create named internal stages using SQL commands. User and table stages are default and created by default as and when these are created.

Create named internal stage: This stage is used to store the files for processing or loading. load files can be of different formats and need to define **FILE FORMAT** while using these files. You can specify the file format at stage level as well to store same format files. The stage with file format indicates that all files present in this stage follows same format type. The default format type is CSV.

Create a named internal stage to store load files which stores JSON format files.
Example:

```
CREATE STAGE BATCH_INT_STAGE FILE_FORMAT =
LOAD_JSON_FORMAT;
```

Note: As you notice here, you do not need to specify a keyword to specify the type of stage - internal or external stage while creating. This is an internal stage as this create statement is not referring to any cloud location. For external stage creation, you will use location and storage integration. You will learn more about the external stage in the next section.

Temporary stages are also supported in Snowflake. You can create a temporary stage as and when needed using this SQL command:

```
CREATE TEMPORARY STAGE LOAD_TEMP_STAGE FILE_FORMAT =  
LOAD_CSV_FORMAT;
```

Internal stages are used when you have a scenario to read files from the storage managed by Snowflake account. You can map required source files to the internal stage with appropriate roles and permissions to push or pull files.

Storage integrations

Storage integrations are also one of the type of objects in Snowflake. As you know, external stages are the objects that refers to the cloud location managed by business entity. These stages are on different location, cloud or other account and you need to grant permissions to integrate it with Snowflake.

Storage integration is used to store authentication and authorization to connect to these storage locations. These are considered as key objects to set up connectivity across clouds and locations. This can be created to integrate with any other provider – AWS, GCP, or Microsoft Azure.

With this object, you grant privileges on the storage buckets without providing any additional details to authenticate and authorize user accessing it. This is typically used while loading and unloading data from or to external stages.

You can create a single integration for a given cloud account, and the same integration can be used to create multiple external stages depending on the target bucket locations. The URL used in the external stage should be aligned with the storage location specified in storage integration.

Sample commands

Each integration command has cloud specific parameters as it creates integration with AWS, GCP and Microsoft Azure. You will learn the different parameters needed to create these integrations with below examples:

1. Create integration to read files from Amazon S3 files:

Here, **STORAGE_AWS_ROLE_ARN** specifies the **Amazon Resource Name (ARN)** of the AWS IAM role that grants privileges on the S3 bucket and data files.

```
CREATE STORAGE INTEGRATION AWS_STG_INT  
TYPE = EXTERNAL_STAGE  
STORAGE_PROVIDER = 'S3'
```

```
    STORAGE_AWS_ROLE_ARN =
'arn:aws:iam::000345621710:role/yourrole'

    ENABLED = TRUE

    STORAGE_ALLOWED_LOCATIONS =
('s3://source/path/', 's3://source /path_load/');
```

2. Create storage integration to read files from Google Cloud Storage:

Here, **STORAGE_PROVIDER** specifies the GCP storage as GCS.

```
CREATE STORAGE INTEGRATION GCP_STG_INT
```

```
    TYPE = EXTERNAL_STAGE
    STORAGE_PROVIDER = 'GCS'
    ENABLED = TRUE
    STORAGE_ALLOWED_LOCATIONS =
('gcs://gsource/load1/', 'gcs://gsource/load2/');
```

3. Create storage integration to read files from Microsoft Azure:

Here, **STORAGE_PROVIDER** and **AZURE_TENANT_ID** specifies the storage details and AZURE account where storage account belongs to.

```
CREATE STORAGE INTEGRATION AZURE_STG_INT
```

```
    TYPE = EXTERNAL_STAGE
    STORAGE_PROVIDER = 'AZURE'
    ENABLED = TRUE
    AZURE_TENANT_ID = '<tenant_id>'
    STORAGE_ALLOWED_LOCATIONS =
('azure://saccount.blob.core.windows.net/src_contai
'azure://saccount.blob.core.windows.net/src_contair
```

Like any other database objects, storage integrations can also be created, altered, dropped, and listed. You can use **SHOW** command to list down the existing integrations in your database/schema. List existing storage integrations:

```
SHOW STORAGE INTEGRATIONS;
```

List existing storage integrations based on a pattern:

```
SHOW STORAGE INTEGRATIONS LIKE 'gcp%';
```

You need to create storage integrations to set up connectivity with cloud locations to read source files or write files depending on load or unload requirements.

External stage

This is the type of named stage where you can create a stage referencing to any cloud storage locations. You can setup the access to external locations in various ways.

These are the access types used to connect to external location:

- **Storage integrations:** You can use storage integration access type to create an external stage. You can create storage integration as explained in the previous section.
- **IAM roles:** You can also create and use a role with all required permissions to be granted on cloud to access storage location files.

Sample examples

Create external stage to read data from AWS S3 bucket using storage integration:

```
CREATE STAGE LOAD_EXT_STG_1  
URL='s3://load_source/files/'  
STORAGE_INTEGRATION = AWS_STG_INT;
```

Create external stage to read data from AWS S3 bucket using IAM:

```
CREATE STAGE LOAD_EXT_STG_2  
URL='s3://load_source/files/'  
CREDENTIALS=(AWS_KEY_ID='567a1bb9a'  
AWS_SECRET_KEY='0dp2p0pd');
```

Create external stage to read data from GCP storage:

```
CREATE STAGE LOAD_EXT_GCP
```

```
URL= 'gcs://gsource/load1/'  
STORAGE_INTEGRATION = GCP_STG_INT;  
Create external stage to read data from Azure container:  
CREATE STAGE LOAD_EXT_AZURE  
URL='azure://saccount.blob.core.windows.net/src_contai  
STORAGE_INTEGRATION = AZURE_STG_INT;
```

You can use these SQL commands to create external stages. You can create external stages once you have storage integrations created and ready to be used. You can use **LIST** command to read and list data from all named stages. You will learn more about **LIST** in the upcoming sections of this chapter.

File formats

File formats are Snowflake objects and are needed while loading or unloading data. These file formats can also be used while creating a stage, or while reading a file from a stage in load/unload utility, that is the **COPY** command.

File formats vary based on the type of data and file used to store the data. These are the types of commonly used data:

- Structured data
- Semi-structured data
- Unstructured data

File formats can be created using DDL or web UI options. You can specify various parameters to specify the properties of files.

Sample use case

- Consider loading structured files from a stage to Snowflake. These are CSV files with the following properties:
 - Field delimiter is the pipe character (|).
 - Files consist of a header line that can be skipped.
 - Replace string **NULL** values with **NULL** values.
 - Replace empty strings as **NULL** values.
 - Files are compressed using **GZIP** compression.

Now, you can use these specified properties and **CREATE** command to create a file format. You can create a file format to be used for similar files. If you are getting 2-3 types of files, then you need to create 2-3 file formats as a one-time activity to be used during processing or loading or unloading data. Refer to the following SQL command to create a file format:

```
CREATE OR REPLACE FILE FORMAT LOAD_CSV_FLES  
TYPE = CSV  
FIELD_DELIMITER = '|'  
SKIP_HEADER = 1  
NULL_IF = ('NULL', 'null')  
EMPTY_FIELD_AS_NULL = true  
COMPRESSION = gzip;
```

- Consider another scenario to load semi-structured data in JSON file format. You need to create a JSON file format named **LOAD_JSON_FLS** that uses all the default JSON format options. Refer to the following **CREATE** command:

```
CREATE OR REPLACE FILE FORMAT LOAD_JSON_FLS TYPE =  
JSON;
```

- Consider loading widely used data formats of data lake systems – **PARQUET**, **ORC** or Avro. All types are supported by Snowflake. For instance, you need to create a **PARQUET** file format named **LOAD_PARQUET_FILES** that compresses using the Snappy compression:

```
CREATE OR REPLACE FILE FORMAT load_parquet_files  
TYPE = PARQUET  
COMPRESSION = SNAPPY;
```

File formats can be created using **CREATE**, altered using **ALTER**, and dropped using **DROP** command. These can be used while creating a stage or using **COPY** command to load or unload data. You will learn more about **COPY** command in the upcoming section of this chapter.

LIST command

This command is used to get a list of files from the stages. These can be used to get files from all types of named stages. **LIST** can also be used as **LS** while running command. The different stages are:

- Named internal stage
- Named external stage
- Stage for a specified table (table stage)
- Stage for the current user. (user stage)

Examples:

- List files in the stage for the **src_table** table:
`LIST @%src_table;`
- List files in the **src** path of the **load_stage** named stage:
`LIST @load_stage/src;`
- List files that match a pattern (that is, names with the string **dt_0**) in the table stage:
`LIST @%src_table PATTERN='.*dt_0.*';`

You can use this command while analyzing the data present in the stages before or after processing.

Users can use **COPY** command to load data in batch or near real-time to the Snowflake tables. You can read data from any type of named stage and load data. You can read data from your local system and load it to Snowflake. You can do it easily using Snowflake Web UI to read data from your local system and load it to the Snowflake table. You will learn more about local loads in the upcoming section.

Loading files from local system

By now, you know that stages can be used to host or store your source or target files. You can use the **COPY** command to load or unload data to these named stages. You need to create additional objects like stages, file formats, and storage integrations to create successful integration, as well as the required parameters to implement the load/unload process.

While you can use **COPY** to load files from the local system, you may need to run some additional steps to get data to the appropriate stage from the local system. However, loading from Web UI allows you to load data without loading it to any stages before loading to the table. This is a very simple process to bring in local data files to Snowflake.

Loading data from Web UI

You can use Snowflake classic console or Snowsight to load data. These are the steps used to load files from console:

1. Go to console, **Databases** | **Open database link** | **Tables** listed in database.
2. Click on the table you want to load and select load data, as shown:

Column Name	Ordinal	Type	Nullable	Default	Comment
STORE_ID	1	NUMBER(38,0)	true	NULL	
STORE_TYPE	2	VARCHAR(1)	true	NULL	
STORE_SIZE	3	NUMBER(38,0)	true	NULL	
CREATED_AT	4	TIMESTAMP_NTZ(9)	true	NULL	
FILE_NAME	5	VARCHAR(100)	false	NULL	
FILE_FORMAT	6	VARCHAR(20)	true	NULL	

Figure 4.2: Snowflake Classic Console

3. The **Load Data** wizard opens. Follow the steps to provide – **Warehouse**, **Source Files**, **File Format**, and **Load Options**:

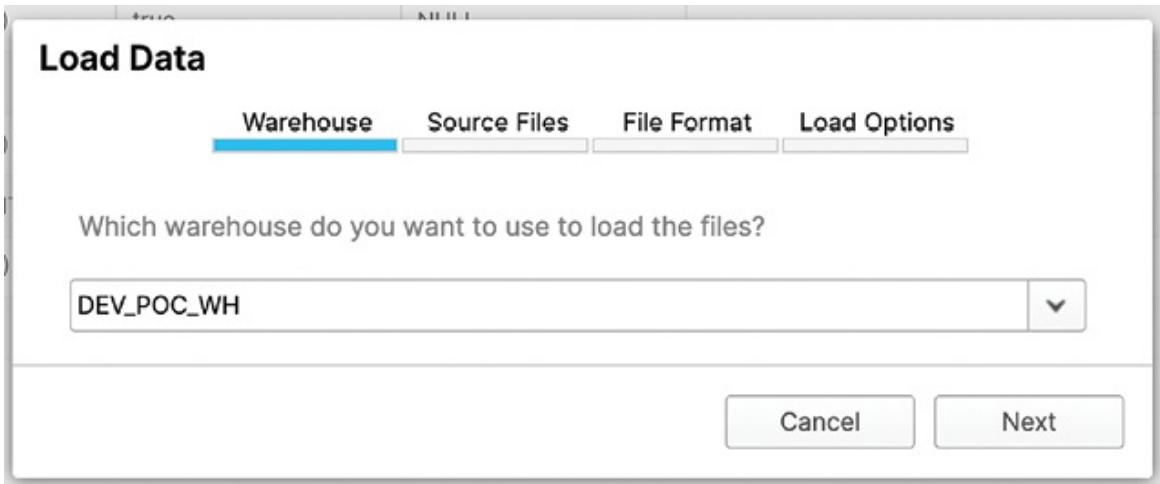


Figure 4.3: Load wizard

- Once you select files to be loaded from local system or external stage, select files and file format:

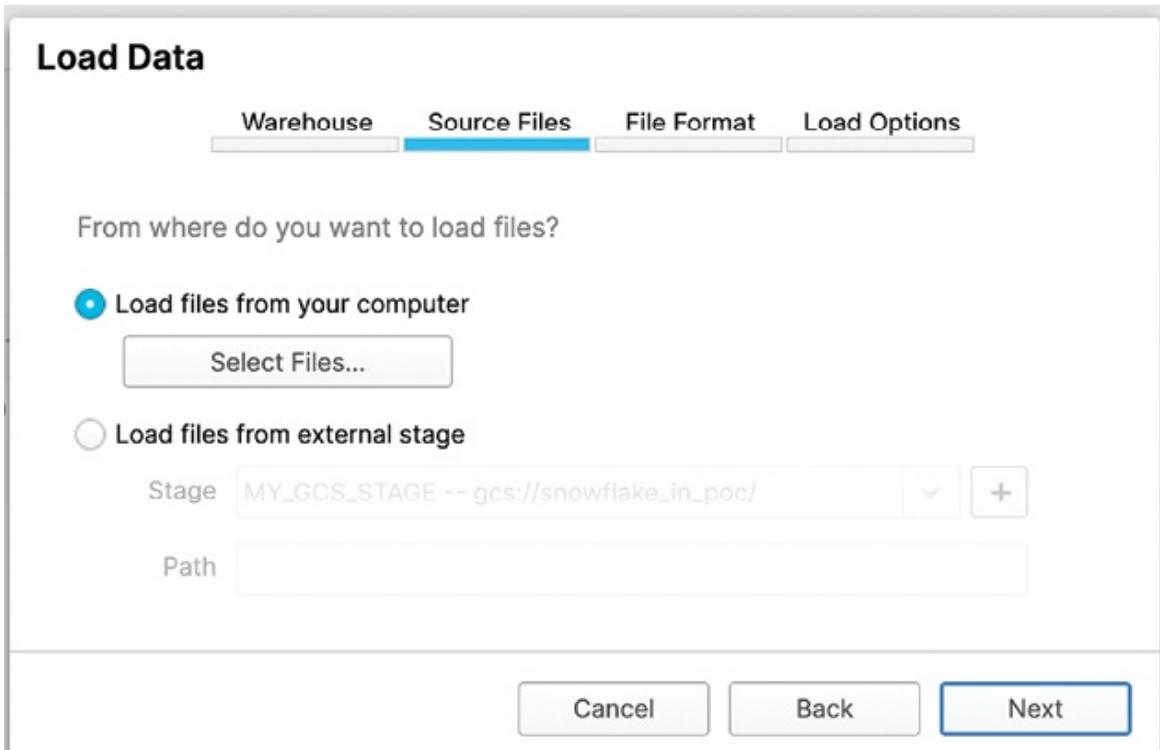


Figure 4.4: Load wizard – choosing files to be loaded

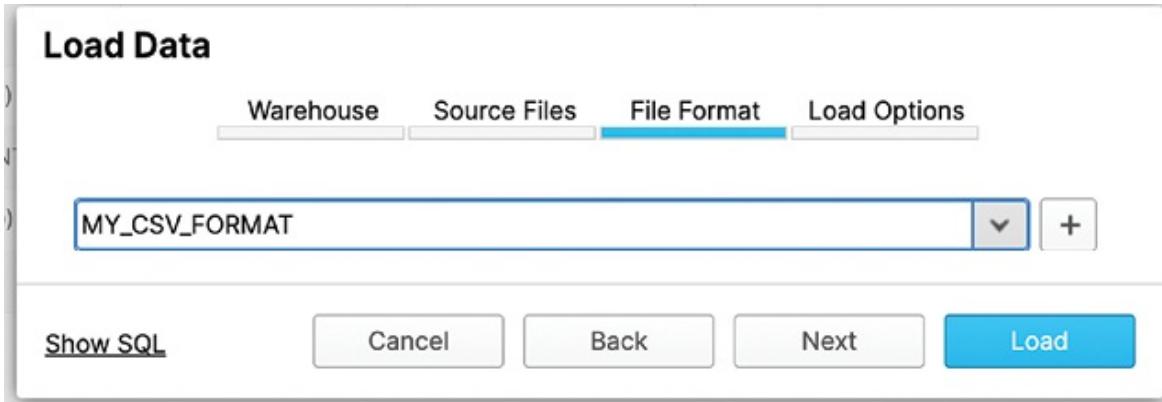


Figure 4.5: Load Wizard – Choosing File Format

5. Select the **Load Options** to load files to table:

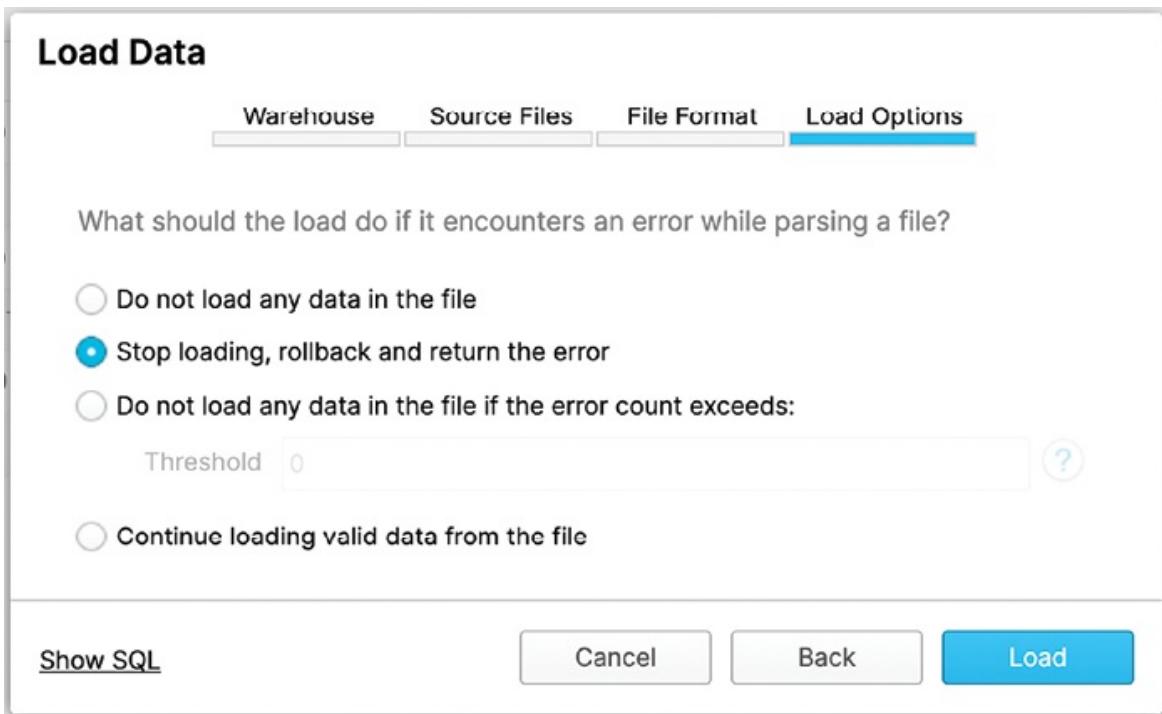


Figure 4.6: Choosing load options

Once you click on load, this instantiates load and loads the files. You can also read the data from the external stage, as shown in [Figure 4.3](#), and select one available from the Snowflake environment. Your load wizard will read data from the external stage selected and load files. You might have also noticed that the file format is also a drop down where it shows all available file formats in the system or Snowflake objects.

Note: Remember, this feature is limited and only recommended for use for smaller data loads. The load wizard is only intended for loading small numbers of files of limited size (up to 50 MB). This

file size limit is intended to ensure better performance because browser performance varies from computer to computer and between different browser versions. Also, the memory consumption required to encrypt larger files might cause a browser to run out of memory and crash.

You can also load data from your local systems using the **COPY** command. There are a set of additional steps required before you run the **COPY** command to upload or load data from local systems.

Using load wizard is not the recommended choice while you work on developing engineering pipelines for your applications. You need to use the bulk load utility for performance, better control, and automated loads. This can be done using the **COPY** command, and you are going to learn more about COPY in the upcoming section.

Using **COPY INTO** to load data

Snowflake's **COPY INTO** is used to load bulk and near real time data loads. **COPY** command can be used to read data from named stages to the Snowflake tables. For near real time data loads, **COPY** can be used and wrapped in an object called Snowpipe.

Bulk load using **COPY INTO**

Consider you have a requirement to load batches of data from source systems to target tables. Source files need to be staged at a location on the cloud in the named stage. As part of the load, the following file locations are supported:

- Named internal stages.
- Named external stages
- External locations

COPY syntax

Like any other load utility, **COPY** also has its set of parameters or options to be used along with the load. Refer to the below syntax to load data from the named stage:

```
/* Standard data load */  
COPY INTO <table_name>  
FROM { internalStage | externalStage |
```

```

externalLocation }

[ FILES = ( '<file_name>' [ , '<file_name>' ] [ , ...
] ) ]

[ PATTERN = '<regex_pattern>' ]

[ FILE_FORMAT = ( { FORMAT_NAME = '<file_format_name>'
|
[ TYPE = { CSV | JSON | AVRO | ORC | PARQUET | XML } [
formatTypeOptions ] } ) ]

[ copyOptions ]

[ VALIDATION_MODE = RETURN_<n>_ROWS | RETURN_ERRORS |
RETURN_ALL_ERRORS ]

```

Here,

- **<table_name>**: Target table name in Snowflake.
- **{ internalStage | externalStage | externalLocation }**: Stage location or external location where files are residing.
- **FILES**: File names to load data.
- **PATTERN**: Pattern of files to identify or read from a directory or stage location. For example: **daily*.csv**
- **FILE_FORMAT**: Format of file being loaded. This can be created as an object in Snowflake.
- **TYPE**: Type of files being loaded.
- **formatTypeOptions**: set of additional and optional options that can also be used along with format type.
- **copyOptions**: set of additional and optional options that can be used in the **COPY** command.
- **VALIDATION_MODE**: Validate the data being loaded and return errors if any.

Transformations supported in COPY INTO

Some basic transformations are supported in **COPY** command while loading data

from named stages. These transformations are basic transformations and does not involve complex transformations in **COPY**. Following are the transformations supported:

- Reordering the column
- Omit columns
- Casting columns
- Text truncating as per target column length

Note: **COPY INTO** does not need your columns to be in same order as of your target table. Also, the number of columns can be different when compared with target table.

COPY with transformation syntax

Following is the syntax to use basic transformations in **COPY**:

```
/* Load data with transformation */

COPY INTO <table_name> [ ( <col_name> [ , <col_name>
... ] ) ]

FROM ( SELECT [<alias>.]<file_col_num>[.<element>] [
, [<alias>.]<file_col_num>[.<element>] ... ]

          FROM { internalStage | externalStage } )

[ FILES = ( '<file_name>' [ , '<file_name>' ] [ , ...
] ) ]

[ PATTERN = '<regex_pattern>' ]

[ FILE_FORMAT = ( { FORMAT_NAME = '[<namespace>.]'
<file_format_name>' |
      TYPE = { CSV | JSON | AVRO | ORC | PARQUET | XML
} [ formatTypeOptions ] } ) ]

[ copyOptions ]
```

As you know, named stages can be table, user, internal and external stages referring to the cloud storage locations on AWS, GCP or Azure. Following are some of reference commands to load data without any transformations using **COPY** command:

```

-- S3 bucket
COPY INTO srctable1 FROM
's3://srcbucket/.../src1.csv';
-- Google Cloud Storage bucket
COPY INTO srctable2 FROM
'gcs://srcbucket/.../src2.csv';
-- Azure container
COPY INTO srctable3 FROM
'azure://account.blob.core.windows.net/ycontainer/...';

```

VALIDATE command

This command is used to validate the files loaded in past execution of the **COPY** command. This also returns all errors encountered during the load instead of just first error.

Note that this command does not return any result for COPY command that has this option ON_ERROR = ABORT_STATEMENT (default value).

You will learn more about **COPY** options in the subsequent section of this chapter.

COPY optional parameters

VALIDATION_MODE = RETURN_n_ROWS | RETURN_ERRORS | RETURN_ALL_ERRORS

Parameter	Description
RETURN_n_ROWS (for example, RETURN_10_ROWS)	This validates the specified number of rows, if no errors are encountered; otherwise, fails at the first error encountered in the rows.
RETURN_ERRORS	Returns all errors (parsing, conversion, and so on.) across all files specified in the COPY statement.
RETURN_ALL_ERRORS	Returns all errors across all files specified in the COPY statement, including files with errors that were partially loaded during an earlier load because the ON_ERROR copy option was set to CONTINUE during the load.

Table 4.1: COPY options

```
FILES = ( 'file_name' [ , 'file_name' ... ] )
```

This parameter specifies the list of files to be loaded. This is the fastest load of COPY INTO.

```
PATTERN = 'regex_pattern'
```

This parameter accepts a regular expression pattern string, enclosed in single quotes, that specifies the file names and/or paths to match.

```
FILE_FORMAT = ( FORMAT_NAME = 'file_format_name' ) or  
FILE_FORMAT = ( TYPE = CSV | JSON | AVRO | ORC |  
PARQUET | XML [ ... ] )
```

Note: Recommend avoiding patterns to filter on a large number of files, this helps in yielding better performance.

FORMAT_NAME = 'file_format_name' -> This specifies the existing file format.

TYPE = CSV | JSON | AVRO | ORC | PARQUET | XML [...] -> This specifies type of files being loaded to table.

Note: These two are mutually exclusive – **TYPE** and **FORMAT_NAME**.

```
TYPE = CSV
```

```
COMPRESSION = AUTO | GZIP | BZ2 | BROTLI | ZSTD |  
DEFLATE | RAW_DEFLATE | NONE
```

This is a string parameter which specifies the current compression of the load files.

```
RECORD_DELIMITER = 'character' | NONE
```

This is record delimiter that specifies character to separate records in a file. This represents end of record.

```
FIELD_DELIMITER = 'character' | NONE
```

This is field delimiter that specifies character to separate fields in a file.

```
SKIP_HEADER = <integer>
```

This skips the header or the number of rows to skip while loading data to the table.

```
SKIP_BLANK_LINES = TRUE | FALSE (Default: FALSE)
```

This is a Boolean parameter that is used to skip blank records in a file.

COPY INTO options

COPY supports a variety of options and parameters that can be used to add set of checkpoints incorporated within a single **COPY** command. You can use one or many from below list of options based on your requirements.

Options

```
ON_ERROR = CONTINUE | SKIP_FILE | SKIP_FILE_num |  
'SKIP_FILE_num%' | ABORT_STATEMENT (Default:  
ABORT_STATEMENT for bulk loading, SKIP_FILE for  
snowpipe)
```

The preceding string option is used to specify action to be taken as part of error handling. Each string input has different meaning:

- **CONTINUE**: This allows to continue the load file to the target table even if error occurs. The **COPY** statement returns error messages for each failed row.
- **SKIP_FILE**: This skips the file once error is found. This option is slower in performance as this buffer the entire file to verify for the errors and skips the file even if single error encountered. This may not be preferred for large files as this might result into slower performance and higher credit consumption. This has additional parameters:
 - **SKIP_FILE_num**: This is used to skip the file when the number of errors specified in the **num** are met or exceeds. For example, **skip_file_10** can be used to skip a file when it reaches 10 errors or exceeds more than 10. This can also be treated as error limit – where number of errors are allowed.
 - **'SKIP_FILE_num%'**: This skips a file when specified % of rows exceed. For example, **SKIP_FILE_10%**: A file is skipped when the error records reaches or exceeds 10% of overall record count.
- **ABORT_STATEEMT**: This option exit the load operation when error occurred. This is default load option.

```
SIZE_LIMIT = <num> (Default: null)
```

This is a numeric parameter which specifies the size of the data feed or files being loaded. The default size is null. There is no restriction of size while loading the data to target tables. Snowflake recommends loading data in smaller chunks if the load file is huge as it helps in running it in parallel loads and gain performance in bulk loading.

PURGE = TRUE | FALSE (Default: FALSE)

This is a **Boolean** option which specifies to remove the data files from stage upon successful load. This option of **COPY** command can delete files from stage locations once the load is completed successfully.

RETURN_FAILED_ONLY = TRUE | FALSE (Default: FALSE)

This is a **Boolean** option which specifies whether to share the load files which are failed to load in the statement result.

FORCE = TRUE | FALSE (Default: FALSE)

This is a **Boolean** option which specifies whether files to be loaded forcefully. Snowflake maintains metadata of load history of the target table. This history loads is used while loading data to the target table, if the file is matching with the history metadata it skips the file from loading. This feature helps to maintain data and avoid data duplication.

LOAD_UNCERTAIN_FILES = TRUE | FALSE (Default: FALSE)

This is a **Boolean** option that specifies action to be taken on files whose metadata is not available in the load history. **COPY** command skips such files by default.

COPY INTO using for bulk loads

You can setup a **COPY** job to read data from stage and load it to the target table. For example, you can consider some of these properties to setup load process:

- **COPY** job to load batch data from an external stage.
- Delimited file with | (Pipe) delimiter and a file format **load_delim_files**.
- This job reads single file of huge volume (>1GB) and starts loading to the target table.
- It may take 4-5mins to load this file to target table.
- A dedicated warehouse setup for batch loads and you are using the same warehouse.

- Warehouse used is of XS size.

Following is the **COPY** command used:

```
COPY INTO raw_table
FROM @src_ext_stage
FILE_FORMAT = (FORMAT_NAME = load_csv_files);
```

If you consider the specifications and the **COPY** command being run, can you say that it is the optimal performance or way or design to run **COPY** to load huge files? To answer this question, Snowflake recommends loading files in smaller sizes or batches. You can achieve this by making the following changes to the bulk load and get better performance:

- Split the large file into size of 200-300MB file each.
- Use the same **COPY** command to point to the storage location where these small files are saved. You can use pattern or regex to provide file names.
- You will be surprised to see that this job finishes in 2-2.5 mins.
- You can still use the same warehouse – No need to change or resize the warehouse to the larger size.

```
COPY INTO raw_table
FROM @src_ext_stage
PATTERN = 'source_data_split*.csv'
FILE_FORMAT = (FORMAT_NAME = load_csv_files);
```

The scenario is applicable to named stages – internal and external stages.

COPY for local system

You can use **COPY** command to load files from named stages as well as from local systems or client machine. You can achieve this along with a **PUT** command. You need to run **PUT** first to copy the files to named stage and run **COPY** to load data to the target table.

PUT command

This command is used to upload data files from local folder on a machine to

snowflake stages. You can upload data to the following stages:

- Named internal stage
- Internal stage for a specified table (Table stage)
- Internal stage for the current user (User stage)

Once you upload files to internal stages, these can be loaded into a table using **COPY INTO** command.

PUT syntax

```
PUT file://<path_to_file>/<filename> internalStage  
[ PARALLEL = <integer> ]  
[ AUTO_COMPRESS = TRUE | FALSE ]  
[ SOURCE_COMPRESSION = AUTO_DETECT | GZIP | BZ2 |  
BROTLI | ZSTD | DEFLATE | RAW_DEFLATE | NONE ]  
[ OVERWRITE = TRUE | FALSE ]
```

Examples

- **PUT file:///temp/data/loaddata.csv @load_int_stage;**: Load to a named internal stage from a Linux or MacOS machine.
- **PUT file:///C:/temp/data/loaddata.csv @~ AUTO_COMPRESS=TRUE;**: Load to current user stage from a windows machine.
- **PUT file:///temp/data/yorders_*1.csv @%orderdestiny_stg;**: Load to the table stage. **orderdestiny_stg** is the table name to be loaded. Wild char is supported to bring in all files of matching patterns.

Now, you know the **COPY** command and options that can be used to load batch/bulk data to the Snowflake tables. Do you know that the same **COPY** command can also be used to get streaming data to the Snowflake? You will learn more about getting streaming data to the Snowflake in next section of this chapter.

Understanding streaming loads

As you have learnt in the first section of this chapter, an enterprise data platform needs to bring in heterogeneous data from sources to the platform for processing,

transformations, data massaging. Once data is transformed and ready to be consumed, there are set of downstream applications, reporting and BI applications, data analytics as well as data science applications running on top of the target databases.

As you observed in [*Figure 4.1*](#), there are sources which share batch or bulk data as well as real time streaming data sourced from Kafka. Streaming data is the low volume, high frequency data sets that can be processed at a shorter interval to make data available to the consumer applications. Data is processed in a real time and near real time fashion.

- **Real time:** Data received in the form of streams, low volume high frequency and read it real time to be processed. In this case, time to market is minimal in comparison with bulk loads.
- **Near real time:** Data received in the form of streams or files with low volume, low/medium frequency or at a regular interval of few mins. In this case, time to market is more in comparison with real time and less in comparison with bulk loads.

Snowflake supports both the types of streaming loads. There are different approaches that can be used to bring in the real time and near real time data to the Snowflake tables.

Implementing Snowpipe

Snowpipe is a Snowflake object that can be used to load data to the table as soon as the file is available at the stage location. This enables users to load data in micro-batches where data can be made available to the users in a very short duration.

Snowpipe uses **COPY** to load data from stage locations to the target table. Pipe is the Snowflake object that stores the **COPY** statement used by Snowpipe. All features of **COPY** are supported along with semi-structured support to load JSON and Avro files.

Snowpipe can be invoked as soon as new files are available at stage location. This can be done in two ways:

- Automate the load using cloud messaging.
- Using Snowpipe REST endpoints

Automate the load using cloud messaging

This approach leverages the event notifications for cloud storage from cloud providers. This is used to inform Snowpipe that new files are available to be processed. Snowpipe then copies the files to the queue to be loaded to the target table. Snowpipe then loads data to the target table as specified in the **COPY** command in a serverless fashion.

These are the cloud storage services supported for the Snowflake accounts hosted on any cloud provider – AWS, GCP or Microsoft Azure:

Cloud Provider	Storage Type
AWS	S3
GCP	Google Cloud Storage (GCS)
Microsoft Azure	ADLS Gen2, Blob storage, General Purpose V2

Table 4.2: Supported Cloud Storage Services

Using Snowpipe REST endpoints

Client application calls public REST endpoint with the name of a pipe object and a list of data filenames. If matching files are discovered, they are queued for loading. Snowflake provides resources load data from queue to the target table based on the **COPY** specified in Pipe.

These are the cloud storage services supported for the Snowflake accounts hosted on any cloud provider – AWS, GCP or Microsoft Azure:

Cloud provider	Storage type
AWS	S3
GCP	Google Cloud Storage (GCS)
Microsoft Azure	ADLS Gen2, Blob storage, General Purpose V2

Table 4.3: Supported REST endpoints

Snowpipe versus bulk load

If you have noticed, Pipe uses **COPY** command and **COPY** can also be used to load bulk load data. You can use same **COPY** command to load bulk as well as real time data in micro or mini batches using it in Pipe.

Following are some of the key differences between bulk and Pipe using **COPY**:

Category	Snowpipe	COPY Bulk
Authentication	REST endpoint call needs key-pair authentication with JSON Web Token (JWT)	Uses security configurations specified in client applications.
Load history	Load metadata stored for 14 days	Load metadata stored for 64 days
Transactions	Loads are done based on the number of records in each file.	Loads are done in a single transaction
Compute resources	Snowpipe is serverless – this uses Snowflake provided resources	This uses client specified or user specified resources
Cost	Billed as per resource consumption while using Snowpipe	Billed for the time when Warehouse is active

Table 4.4: Comparison between Pipe and COPY

Snowpipe features and recommendations

Let us take a look at the features of Snowpipe:

- **Load file size:** Files roughly 100-250 MB (or larger) in size compressed and staging files per minute. This approach helps to maintain a balance between cost and performance.
- **Order of load files:** Each Snowpipe has its corresponding queue to sequence files that are pending to be loaded. As and when new files are arrived these are appended to the existing queue. Snowpipe usually loads files in the order in which they are received. However, there is no guarantee that these files are loaded in the same order they staged.
- **Duplicate data:** Snowpipe maintains file load metadata for each pipe object. This is maintained for 14 days, and this prevents loading the same files again or reloading the same files.

Snowpipe commands

Like any other Snowflake objects, pipe can be created, alerted, and dropped using SQL commands.

SQL commands

You can use DDL commands for the pipe. The DDL commands include **CREATE**, **ALTER**, **DROP**, **SHOW**, and **DESCRIBE**. The following section shares details of commands and syntax to use these DDL commands for pipe.

CREATE PIPE

Creates a new pipe to define the **COPY INTO <table>** statement to be used by Snowpipe to load data from an ingestion queue into tables:

```
CREATE [ OR REPLACE ] PIPE [ IF NOT EXISTS ] <name>
[ AUTO_INGEST = [ TRUE | FALSE ] ]
[ ERROR_INTEGRATION = <integration_name> ]
[ AWS_SNS_TOPIC = '<string>' ]
[ INTEGRATION = '<string>' ]
[ COMMENT = '<string_literal>' ]
AS <copy_statement>
```

Here, AUTO_INGEST - TRUE|FALSE

This indicator indicates whether auto ingestion is enabled to load data automatically.

Examples

- Create a pipe that loads data from stage into the target table:

```
create pipe loadpipe as copy into tgt_tbl from
@tgtstage;
```

- Along with **COPY**, you can add transformation to select only few columns from stage files:

```
create pipe loadpipe as copy into tgt_tbl (Col1,
Col2) from (select $5, $4 from
@tgtstage);
```

ALTER PIPE

This command is used to modify a limited set of properties for an existing pipe. This command also supports the following operations:

- Pausing the pipe.
- Refreshing a pipe.

- Adding/overwriting/removing a comment for a pipe.
- Setting/unsetting a tag on a pipe.

```

ALTER PIPE [ IF EXISTS ] <name>
SET { [ objectProperties ][ COMMENT =
'<string_literal>' ] }

ALTER PIPE <name> SET TAG <tag_name> =
'<tag_value>' [ , <tag_name> = '<tag_value>' ... ]

ALTER PIPE <name> UNSET TAG <tag_name> [ ,
<tag_name> ... ]

ALTER PIPE [ IF EXISTS ] <name> UNSET {
<property_name> | COMMENT } [ , ... ]

ALTER PIPE [ IF EXISTS ] <name> REFRESH { [ PREFIX
= '<path>' ] [ MODIFIED_AFTER = <start_time> ] }
```

Examples

```
alter pipe mypipe SET PIPE_EXECUTION_PAUSED = true;
```

Here,

```
PIPE_EXECUTION_PAUSED = TRUE | FALSE
```

This specifies whether to pause a running pipe or not.

DROP PIPE

This removes specified pipe from the database. The schema is as follows:

```
DROP PIPE [ IF EXISTS ] <name>
```

Example

```
DROP PIPE load_pipe;
```

DESCRIBE PIPE

This command is used to describe properties specified for a pipe:

```
DESC[RIBE] PIPE <name>
```

Example

```
DESC PIPE load_pipe;
```

SHOW PIPE

This command list down all types where you have access:

```
SHOW PIPES [ LIKE '<pattern>' ]
```

```
[ IN
```

```
{
```

```
ACCOUNT |
```

```
DATABASE |
```

```
DATABASE <database_name> |
```

```
SCHEMA |
```

```
SCHEMA <schema_name> |
```

```
<schema_name>
```

```
}
```

```
]
```

Example:

```
Use tgt_dbase;
```

```
Show pipes;
```

Loading real time data

Real time streaming data can be loaded into Snowflake using the following options:

- Snowpipe
- Snowflake Connector for Kafka
- Snowpipe streaming

Snowpipe

Snowpipe can also be used to load real time data to Snowflake. To use Snowpipe to load real time data, you need to have all streaming data to cloud storage buckets or locations. You can use Snowpipe to load data from storage in micro-batches or mini-batches. You can implement Snowpipe with an event trigger to auto ingest streaming data as and when it arrives in the storage bucket. You have already learned Snowpipe in the above section; however, you will learn more about implementing Snowpipe in a subsequent section of this chapter.

Snowflake connector for Kafka

Kafka uses a publish and subscribe model to read or write streams. Kafka allows users to read and write messages asynchronously. You will need a Kafka Topic to be subscribed to get streaming data from Kafka. You also need to add a subscriber to read data from topics and write to Snowflake tables.

The Kafka connector follows the given process to subscribe to Kafka topics and create Snowflake objects:

1. As you need a subscriber to a topic, Kafka connector subscribes to one or more topics based on the configurations provided.
2. Kafka connector creates these objects per topic:
 - a. One internal stage to store data files temporarily per topic.
 - b. One pipe to ingest the data files for each topic partition.
 - c. One table for each topic.

The following figure illustrates the data ingestion flow for Kafka with the Kafka connector:

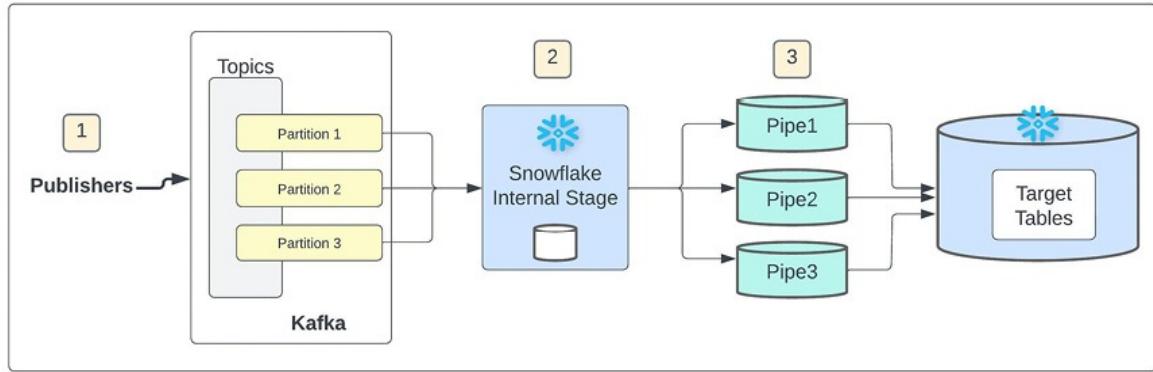


Figure 4.7: Kafka Connector Data Ingestion Flow

3. Set of applications publish JSON or Avro records as part of Kafka streaming.
4. The Kafka connector buffers messages from the Kafka topics. The connector triggers Snowpipe to ingest the temporary file.
5. A Snowflake-provided virtual warehouse loads data from the staged file into the target table.
6. The connector monitors Snowpipe and removes each file in the internal stage after confirming that the file data was loaded into the table.
7. If a failure prevents the data from loading, the connector moves the file into the table stage and produces an error message.
8. The connector repeats steps 2-4.

You can implement Kafka connector to bring in streaming data to Snowflake tables. Snowflake streaming is new feature introduced in public preview. You will learn more about this in next section.

Snowpipe streaming

Snowpipe streaming API is a low latency load that loads streaming data rows using Snowflake ingest SDK and application code. This streaming API writes data to the Snowflake target tables, unlike bulk loads or Snowpipe, which loads data to the staged files. This is a low latency, low-cost solution that makes this a powerful tool to load streaming data.

Snowflake streaming API is not a replacement for Snowpipe. This API complements Snowpipe. Snowpipe streaming API is used in streaming scenarios where data is streamed in the form of rows. This API removes the need to create a file or stage to store files at buckets to load data into Snowflake. This directly

reads from Kafka topics and loads to the Snowflake tables, as shown:

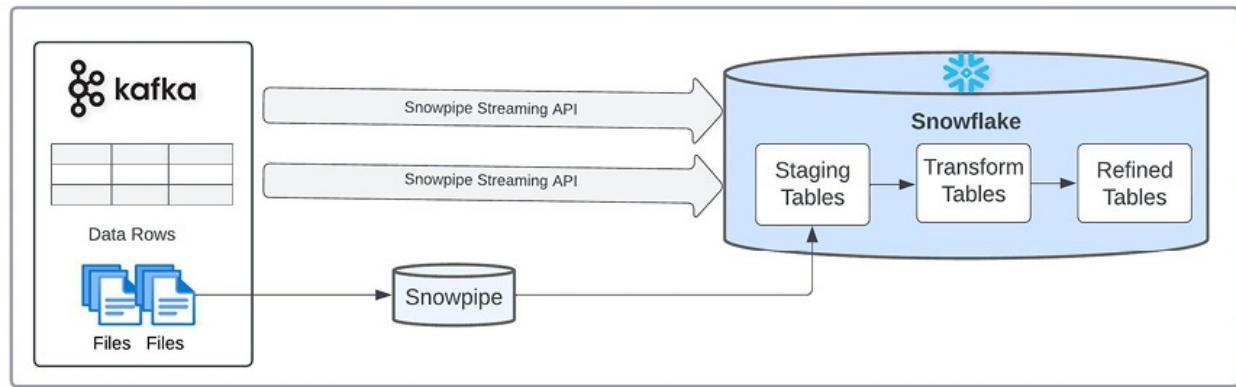


Figure 4.8: Snowflake Snowpipe Streaming API

Snowpipe streaming versus Snowpipe

The following table lists down the difference between Snowpipe and Snowpipe streaming API:

Category	Snowpipe streaming	Snowpipe
Form of data to load	Data in the form of rows	Data files
Third party software	Custom Java code wrapper with Snowflake SDK	None
Data ordering	Ordered insertion	Not supported
Load history	Recorded in streaming history view	Load history recorded in <code>load_history</code> view
Pipe object	Does not require any pipe object	Requires pipe object

Table 4.5: Comparison of Pipe and Streaming

Streaming cost

Snowpipe streaming is a serverless model where Snowflake manages all required resources. The accounts are charged based on the compute cost.

Note: Snowflake also has Snowpipe streaming feature in public preview.

Loading near real time data

Near real-time data caters to data availability with a low latency. This data load

contributes to mini and micro-batches. Snowpipe is used to get, and process near real time data to the Snowflake. You will learn more about implementing Snowpipe in the next section.

Implementing Snowpipe

Snowpipe is a Snowflake object that is used to bring in data in real time, near real time. As you know by now, Snowpipe needs data to be available on cloud storage to be loaded to the Snowflake tables. Snowpipe can be created using the **CREATE** statement. Snowpipe supports auto ingestion that can be enabled along with cloud event notifications.

In this section, you are going to learn steps to be followed to set an auto ingest in integration with AWS. Following are the use case details:

- External stage hosted on Amazon S3
- SQS event notification to identify new files arrived and ingest into load queue.
- Pipe reads data from the ingest queue.
- Load data into Snowflake target table

To implement this scenario, you need to follow the given steps:

1. Configure Access Permissions for the S3 Bucket.
 - a. AWS Access Control Requirements
 - b. Creating an IAM Policy
2. Create the IAM Role in AWS.
3. Create a Cloud Storage Integration in Snowflake.
4. Retrieve the AWS IAM User for your Snowflake Account.
5. Grant the IAM User Permissions to Access Bucket Objects.

You can refer to *Figure 4.5* to understand the auto ingestion flow for AWS S3. As you know, to automate ingestion you can use cloud native event services in integration with storage buckets. Refer to <https://docs.snowflake.com/en/user-guide/data-load-snowpipe-auto-s3> and follow step by step instructions to setup ingestion:

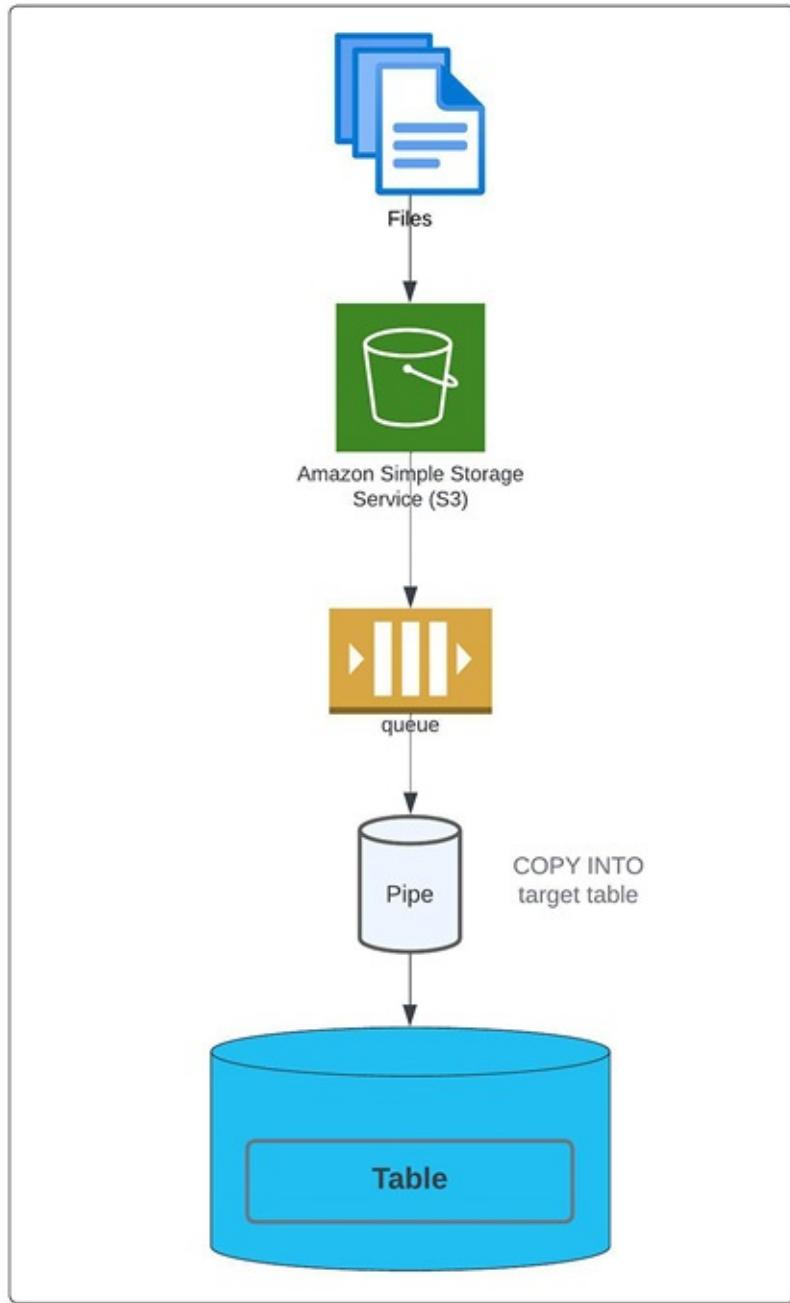


Figure 4.9: Auto Ingestion flow Snowpipe for AWS

Do you know, **COPY** can also be used to unload data from Snowflake? Unloading is also one of the use cases along with loading in data platform integrations. You will learn more about unloading use cases and using **COPY INTO** to unload data in upcoming sections.

Understanding data unloading

Data consumption is another important aspect of data platform integrations. Consumer integrations are where you have a set of consumer applications that consume data from the platform. Consumer applications like **Building Reports (BI)**, analytics, running AI/ML models or consuming data as part of upstream application. Refer to [Figure 4.1](#) – the Webapps or consumer applications there represent the consumer side of the platform design. Based on these use cases or consumption options, there are a set of ways you can use to make data available to the consumer applications:

- Go to **File feed | Generating data in the form of File feed**
- **Database Feed | Generating data and sharing** as part of target tables or views ready to be consumed by the application. This needs consumer applications to be onboarded or grant permission to read data from tables/views.

You can refer to the consumer integration portion highlighted in the enterprise reference architecture. These set of applications are referred as consumers as they consume data from the data platform.

Database feed

These are consumer feeds where consumer applications granted access to read data from the platform directly. Snowflake offers secure data sharing to share data with consumer applications. This is the most secure and recommended approach to share data with consumers. Data sharing allows users to share data with consumers via Snowflake as well as non-snowflake consumer accounts. You will learn more about data sharing in [Chapter 12: Data Sharing](#).

In a typical scenario where you may need to share data with internal stakeholders or internal applications, you can create a serving layer or target layer. You can create a consumer database and schema to store all consumer data. You can have secure views created to grant data access to the teams or applications. In this scenario, you need to onboard users to the Snowflake platform and grant appropriate Roles to access – read/write depending on business requirements. Also, allocate the warehouse to the consumer teams to manage, and track warehouse consumption for these users or applications.

This is also applicable to all BI, reporting, and data analytics tools where applications connect to Snowflake and read data from database objects. The user onboarding process, role, and warehouse allocation remain the same.

File feed

These are the consumer feeds where data is extracted from the data platform in the form of files and shared with consumer applications. Data can be shared over a shared drive or **Network Attached Storage (NAS)** or SFTPed to the target locations.

Snowflake data sharing makes it easy to share data with consumer applications; however, for any of the traditional systems where data is expected to be in the format of data files, data can be exported using the Snowflake **COPY** command.

You can consider the unloading, which is the file feed requirement to understand **COPY** command usage to export data. **COPY** can export data to the internal stage as well as the external stage. If you can export data to the external stage, then you can share cloud storage locations with consumer applications to read data from buckets. You can even download files from the internal stage to local systems or applications to SFTP files to the file server. These choices are dependent on the business and application requirements.

You can use the **COPY** command to unload data to the external stage, internal stage, as well as to the local systems.

Using **COPY** to unload data

As you know by now, the **COPY** command can be used to load data as well as unload data to the named stages. Like load properties, unload also has to support specific file formats, compressions, and locations. **COPY** options are also available for the unload scenario.

COPY INTO

Following are the file formats, and compressions supported with **COPY** unload.

Exported feeds, formats and locations

Based on the Snowflake architecture and integration with cross cloud using named external stages, you can export data into various locations and file formats:

Export feature	Locations	Notes and usage
Location	Local files	Exported files can be unloaded to the Snowflake Named internal

of files		location, then downloaded locally using GET command.
	Files in Amazon S3	Exported files can be unloaded to any provided bucket in S3, then downloaded locally using AWS utilities.
	Files in Google Cloud Storage	Exported files can be unloaded to any user-supplied bucket in Cloud Storage, then downloaded locally using Cloud Storage utilities.
	Files in Microsoft Azure	Exported files can be unloaded directly to any user-supplied container in Azure, then can be downloaded locally using Azure utilities.
File formats	Delimited files (CSV, TSV, and so on.)	Any valid delimiter is supported; default is comma (that is, CSV).
	JSON	
	Parquet	

Table 4.6: Supported export formats

Note: **COPY unload** supports only 3 types of file formats – Delimited, JSON and Parquet.

Compression supported

These exported files also support compressions, and you can use these compression methods to compress exported file feeds.

Location	Compression	Notes and usage
Internal or external location	Gzip	By default, gzip is used to compress files. Compression is not enabled by default, you can specify compression explicitly with supported ways.
	bzip2	
	Brotli	
	Zstandard	

Table 4.7: Supported compression formats

Encryption supported

Snowflake supports 128-bit or 256-bit encryption by default for any data that is stored within Snowflake storage and named internal stages – table stage, user stage, and internal stages.

Users can also use customized keys to encrypt data on named external stages.

Bulk unloading from a table or query

Like bulk loading, Snowflake supports bulk exports. The same **COPY** command can be used to export data in bulk. There are two steps required to download data

to local or user machines:

1. Run the **COPY INTO <location>** command to extract the data from the Snowflake database table into one or more files in a Snowflake or external stage.
2. Download the file from the stage:
 - a. Named internal stages - use the **GET** command to download the data file(s).
 - b. Named External stages AWS/Azure/GCP - Use the interfaces/tools by cloud providers to download the data file(s).

You can export the data as a bulk using a complete dump or provide specific query or conditions to export only required data feed:

COPY to export entire table dump:

```
COPY INTO @%salesistory/output/data_
FROM sales FILE_FORMAT = (FORMAT_NAME ='csv_format'
COMPRESSION='GZIP');
```

1. **COPY with query:** Snowflake supports **SELECT** query in **COPY** command in place of table name. You can specify joins, filters, transformations in the **SELECT** statement in **COPY** to extract specific use case or requirement data:

```
COPY INTO @external_aws_stage/output/data_ FROM
(SELECT order_date, order_id, product_id,
sum(order_amount) FROM order a join product b on
a.product_id = b.product_id where a.order_date <
current_date group by order_date,order_id,
product_id)

file_format=(format_name='csv_format'
compression='gzip');
```

Bulk unloading to one or multiple files

COPY unload supports exporting data to multiple files in place of one single file. **COPY** command supports set of parameters to export data into files. You can

export data to the single file or multiple files (part files).

- **Export to single file:** `COPY INTO <location>` command supports copy option `-SINGLE` for unloading data into a single file. Use `SINGLE = TRUE` to unload entire data into one file.
- **Export to multiple files:** `COPY INTO <location>` command supports copy option `SINGLE` for unloading data into a multiple files. Use `SINGLE = FALSE` to unload entire data into one file. Default value is `SINGLE = FALSE` (that is unload into multiple files).

Note: If user does not use this `COPY` option, then the default behavior is to export data into multiple files.

File naming in case of multiple file exports

Snowflake assigns unique name to each file. All exported files will be saved at a single location or bucket, all file names will have prefix of file name followed by a suffix. In case user doesn't specify the file name then Snowflake exports data in files starts with `data_ filenames`. by default, Snowflake adds a suffix to ensure unique file name across parallel execution threads, for example: `data_stats_0_1_0`.

Users can specify each file size using `MAX_FILE_SIZE` copy option while creating multiple files extract or exports.

Bulk unloading using PARTITIONS

As you know, Snowflake supports partition keys and partition data based on the type of data and business use cases. You can also use this column to export data for a given partition.

You can setup `SELECT` query to fetch data based on the date however if you can create `PARTITION` on date column then you can access the partition directly and fetch data based on the `PARTITION`.

The `COPY INTO <location>` command supports `PARTITION BY` option for partitioned unloading of data to Named stages. The `PARTITION BY` copy option needs an expression that supports unload operation partitions table rows into separate files unloaded to the specified stage.

`COPY INTO` is a native command used to load and unload data to and from Snowflake data objects. The same `COPY INTO` offers variety of features and

options to support the need of dataset handling, sanity checks, error handling, and so on. You can specify one or more of these options in a single **COPY INTO** command based on your business need.

COPY INTO options

OVERWRITE = TRUE | FALSE (Default: FALSE)

This is a **Boolean** option used while unloading data. This specifies if the target file can be overwritten in case of same file names at the target locations if any. Note that this option does not remove any of the files that do not match the target file names. The default value is **FALSE**.

Snowflake recommends using **INCLUDE_QUERY_ID = TRUE** option instead of **OVERWRITE = TRUE** and cleaning or truncating the target folder before every unload job to avoid data duplication.

SINGLE = TRUE | FALSE (Default: FALSE)

This is a **Boolean** option that specifies whether single file or multiple files to be generated in case of unloading data.

MAX_FILE_SIZE = num (Default: 16777216 (16 MB))

This option specifies the maximum size of each file generated while running it in parallel. Snowflake uses parallel execution to enhance the performance. Total number of threads cannot be modified manually. Maximum 5GB data can be unloaded from a database object to any named stage.

Exporting data to named internal stages

You can download data to the named internal stages. Once data is available on internal stage, you can run **GET** command to download or get data on your local machine or system.

You can consider a Snowflake account setup on AWS region, and you need to export data to S3 buckets. AWS S3 buckets are created as part of Named internal stages. You can follow these steps to export data to stage and local system:

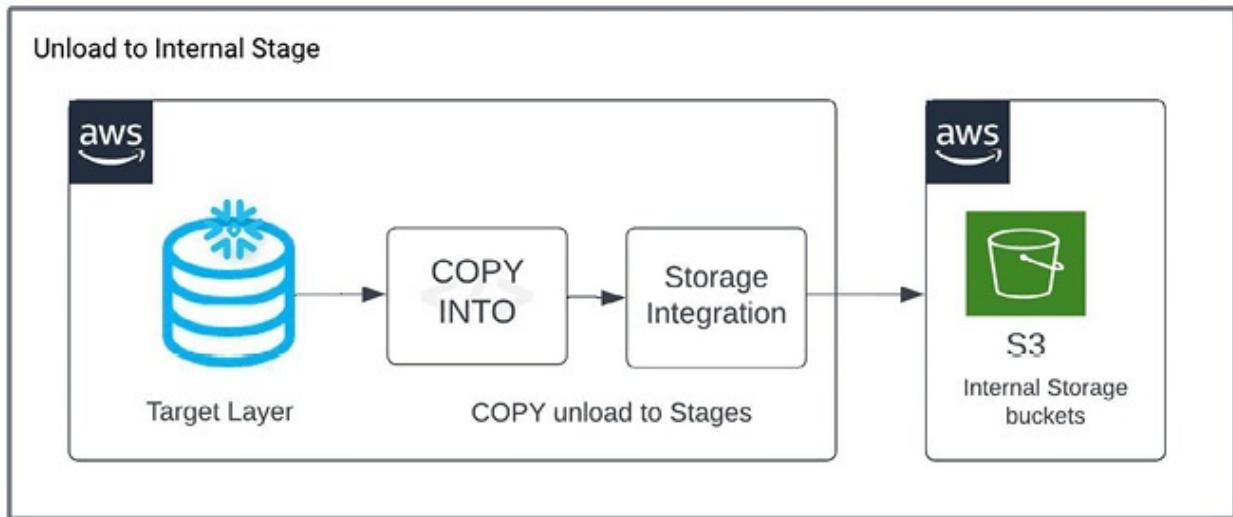


Figure 4.10: Unload to Internal Stage

1. Create internal stage:

```
CREATE OR REPLACE STAGE unload_hstr_stage
FILE_FORMAT = csv_unload_format;
```

2. Run **COPY** command to download data to internal stage:

```
COPY INTO @unload_hstr_stage/unload/ from
sales_history;
```

You can view the files exported using **LIST** command:

```
LIST @internal_unload_stage;
```

name	size	md5
last_modified		
unload_hstr_stage /unload/data_0_0_0.csv.gz	221	6f76dfba002a616bdzg4eae10de5bed3
	Mon, 26	June 2023 16:33:07 GMT

```
+-----+-----+-----+
-----+-----+
-----+
```

3. Download files to your local machine.

You can use **GET** command to download from internal stage.

4. To download on Linux/Mac OS:

```
GET @unload_hstr_stage/unload/data_0_0_0.csv.gz
file:///order_data/unload;
```

5. To download on Windows:

```
GET @unload_hstr_stage/unload/data_0_0_0.csv.gz
file://C:\data\unload;
```

GET command

Like **PUT**, **GET** is also used from SnowSQL command line, and this is used to download data from Named internal stages – table, user and internal stage. **PUT** is used to upload files from local to stage however **GET** is used to download files from stage to local.

Exporting data to named external stages

You can consider a scenario where you are working on cross cloud implementations and one of your consumer application runs on GCP and your Snowflake runs on AWS. Consider this as a data feed use case and you need to export data to GCS buckets as Named external stage for consumer applications. Refer to the following figure that displays the unloading process:

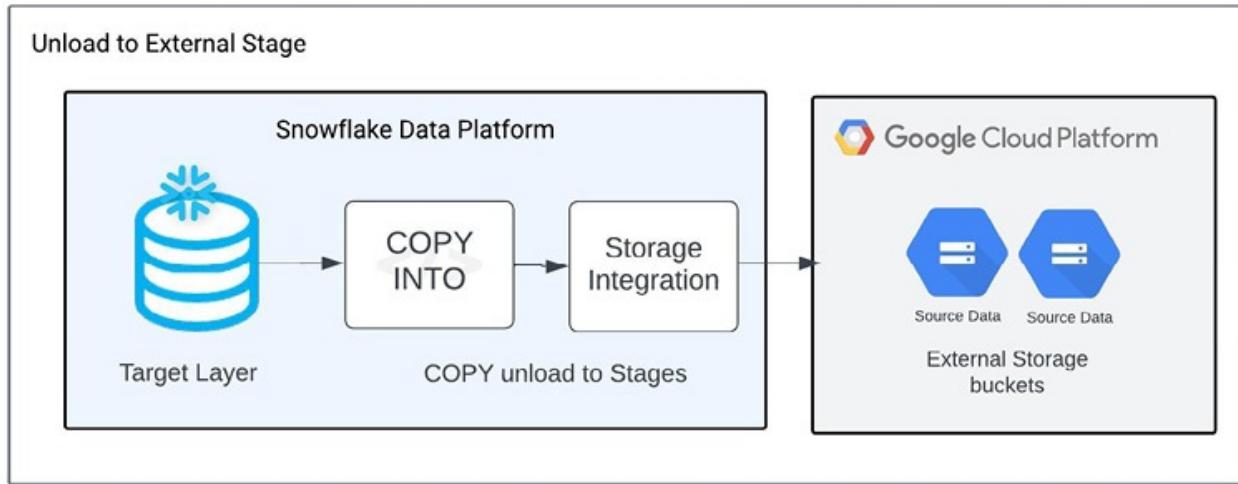


Figure 4.11: Unload to External Stage

Steps to unload data to named external stages

1. **Create storage integration:** You need to have GCP storage integration created before you use external stage exports or create external stages. You can follow the steps shared in storage integration section to create integrations.
2. **Create external stage:**

```
CREATE OR REPLACE STAGE downstream_unload_stage
URL='gcs://downstream/input'
STORAGE_INTEGRATION = gcs_int
FILE_FORMAT = csv_unload_format;
```

3. **Unload data to external stage:**

```
COPY INTO @downstream_unload_stage/date FROM
order_history;
```

4. **Unloading data directly to the storage bucket:**

```
COPY INTO 'gcs://order_history/unload/'
FROM order_history
STORAGE_INTEGRATION = gcs_int;
```

You have learnt about data loading, unloading use cases and using Snowflake

COPY command to load bulk as well as streaming data. The next section is a list of exercises to help you perform hands on and learn more.

Practical: Data loading and unloading

Creating stages

Exercise 1:

- Create a Named internal stage to store files in CSV format. Use the **PUT** command to load data from your local to the internal stage.
- List files present in the stage created in the above step using the **LIST** command.

Exercise 2:

- Create a table and locate its stage
- Use the **PUT** command to load data from your local to the table stage.
- List files present in the table stage

Exercise 3:

- Create a user and locate its stage
- Use the **PUT** command to load data from your local to the user stage
- List files present in the table stage created

Creating file formats

Exercise 4:

- Create a file format with the following properties:
 - Fixed width file format delimited by ~ (tilde)
 - Contains header record
 - Replace string nulls with **NULL**
- Create a file format with the following properties:
 - File is Parquet format file
 - Files are Snappy compressed

- List the file formats present using **SHOW FILE FORMATS**
- Create the same file formats with different names using Web UI

Creating storage integrations

Exercise 5:

- Create a AWS storage integration and use to load files:
 - Setup a demo account on AWS if you don't have any
 - Create a storage integration using **CREATE** command
 - List the integration created
- Load source files from storage integrations to target table
 - Create a sample table at your snowflake database and schema
 - Set the corresponding source files to AWS location
 - Create a storage integration to read files from AWS (or use the one created in above task)
 - Use **COPY INTO** to load files from storage integration

Creating LOAD commands and load using COPY

Exercise 6:

- Load files into the target table with these load conditions:
 - Load fails even if a single error occurred.
 - Loads a delimited CSV file with pipe delimiter (|).
 - Ignore the null records, if any.
- Load files into the target table with these load conditions:
 - Load fails when 10% of total records are errored out.
 - Loads a delimited CSV file with comma delimiter (,).
 - Skip the file metadata check.

Load from local using COPY

Exercise 7:

- Load files from local into the target table with these load conditions:
 - Load fails even if single error occurred.
 - Loads a delimited CSV file with pipe delimiter (|).
- Load files from local using **PUT** and **COPY** command into the target table with these load conditions:
 - Upload sample files from local to internal stage.
 - Load files from internal stage to target table.

Unload from local using COPY

Exercise 8:

- Unload Snowflake table into Named internal stage with these unload conditions:
 - Unload into SINGLE file
 - Unloads a delimited CSV file with pipe delimiter (|).
- Unload Snowflake table into Named internal stage and download using **GET** command with these unload conditions:
 - Unload into multiple files.
 - Unloads a delimited CSV file with pipe delimiter (|).

Conclusion

This chapter provided a summary of Snowflake's data loading and unloading capabilities. This chapter helped you to understand various Snowflake database objects required to perform the loading and unloading of data. This also helps you to learn **COPY INTO**, along with **GET** and **PUT** commands to set up data loads to and from Snowflake. Importantly, you learned these concepts with practical examples with **COPY INTO**. You will learn more about other Snowflake features and offerings as you go along with the next chapters in this book.

In the next chapter, you will learn about Snowflake's native offerings to set up automated change data capture using Streams and features to setup and orchestrate commands and operations using Tasks. You will also learn how these

Snowflake features serve to the use case requirements and are integrated with applications.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake offers its native **COPY** command to perform all types of data loads.
- Snowflake's Snowpipe is used to load streaming and near real time data to the tables.
- Snowflake offers variety of solutions to get streaming data to the target tables.
- Snowflake's Snowpipe streaming and Snowpipe are serverless offerings, the resources required for operations are managed by Snowflake.
- Users can use **COPY** to load, unload data to all types of named stages.
- Users can also upload or unload data from Snowflake web UI however this is not the recommended way for larger loads as well as production loads. This can be used to perform test or POCs with smaller datasets in manual testing.
- Snowflake offers variety of objects to support **COPY INTO** like file formats, stages, storage integrations, and so on.
- Snowflake storage integrations allows platform to be integrated with any other cloud providers.
- Snowflake external stages allows users to integrate other cloud storages to read or write data outside Snowflake platform.

Questions

1. What will be the performance between two loads – one load that loads a smaller file < 1 GB using a small or XS size warehouse and the other load using files > 1GB but stored as part files on an external stage on the same size of warehouse. You are running these tests and want to choose the performance-efficient approach.
2. Can you use data load using Web UI – Classic or Snowsight? Are there

any limitations while using load wizard? What are the scenarios where you can use this approach?

3. How do Snowpipe, Snowpipe streaming, and Snowpipe Kafka connector differ?
4. How does Snowflake handle semi-structured data and formats supported to load data?
5. How can data loading and unloading be done using the **COPY** command?

Multiple choice questions

1. **How bulk loading can be implemented in Snowflake? (Select One)**
 - a. **COPY TO**
 - b. **LOAD**
 - c. **COPY INTO**
 - d. **COPY**
2. **Which object is used to load streaming data in Snowflake? (Select one)**
 - a. View
 - b. Table
 - c. Snowpipe
 - d. Task
3. **Choose correct statements: select two**
 - a. High volume, low frequency data load is called as Bulk load
 - b. Low Volume, high frequency data load is called as Real time or Streaming loads
 - c. Low frequency, low volume data load is called Streaming loads
 - d. High volume, high frequency data load is called as Bulk load

- 4. Which of these transformations are supported in `COPY INTO`? (Select two)**
- a. Truncate columns
 - b. Casting columns
 - c. Column Reordering
 - d. Pivoting
 - e. Aggregate
- 5. Which command is used to upload files from local to stages.**
- a. `COPY INTO`
 - b. `COPY`
 - c. `PUT`
 - d. `MOVE`
- 6. Which of this can be created or altered using DDL commands?**
- a. User stage
 - b. Table Stage
 - c. External stages
 - d. Internal stages
 - e. Named stage
- 7. When can Snowpipe be used?**
- a. Bulk loads
 - b. Batch loads
 - c. Real time loads
 - d. Event loads

8. Which statement is true about Snowpipe?

- a. Snowflake provides resources to run Snowpipe
- b. User can configure and setup automatic ingestions for Snowpipe
- c. Snowpipe can be triggered based on event notification and REST endpoints
- d. All of the above

9. What is true about – `COPY UNLOAD` (select all that applies)

- a. This is the only supported unload option
- b. `COPY` is used only to load files
- c. `COPY` is used to load and unload files
- d. This is also used from Snowflake classic console

10. What all formats are supported in `COPY UNLOAD`?

- a. XML
- b. JSON
- c. Avro
- d. Parquet
- e. ORC

Answers

1	c
2	c
3	a, b
4	a, b
5	c

6	c, d
7	c
8	d
9	c, d
10	b, d

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 5

Understanding Streams and Tasks

Introduction

Change data capture is the most critical part of data processing and platform implementation. This chapter helps to learn native utilities and commands supported by Snowflake to capture the change from the source systems. This chapter covers scenarios to help you understand the need for change data capture and the approach to implementing using Streams. This also covers tasks, which is a Snowflake native feature to schedule jobs, actions, and so on.

You will learn about Snowflake **Change Data Capture (CDC)** and scheduling, and orchestrating capabilities in this chapter. This chapter illustrates capabilities with real-time scenarios and references to integrate source changes using streams. Sample examples and lab questions in this chapter will help you learn with labs. There are a set of questions to check your knowledge of understanding concepts explained in this chapter.

Structure

This chapter consists of the following topics:

- Understanding Change Data Capture
- Understanding Snowflake Streams
- Implementing data capture with Streams
- Understanding tasks

- Implementing scheduling with tasks

Objectives

By the end of this chapter, you will be able to understand the streams and tasks of Snowflake. This chapter helps you to create the required objects to implement CDC using Streams. You will also be able to implement them using the trial account setup in [Chapter 1: Getting Started With Snowflake](#).

Understanding Change Data Capture

Change Data Capture (CDC) is one of the most important features of DWBI and data platform implementations. As the name says, CDC is used to identify, capture, and apply changes to the target data. Change data capture is the process of capturing the data changes in the source systems and flowing these changes to the downstream processes or systems. Downstream applications or systems can take action on the change identified and passed. The most common use case is to reflect the changes in different target systems so that the data in the systems stay in sync all the time. The following figure shows CDC in an enterprise data warehouse:

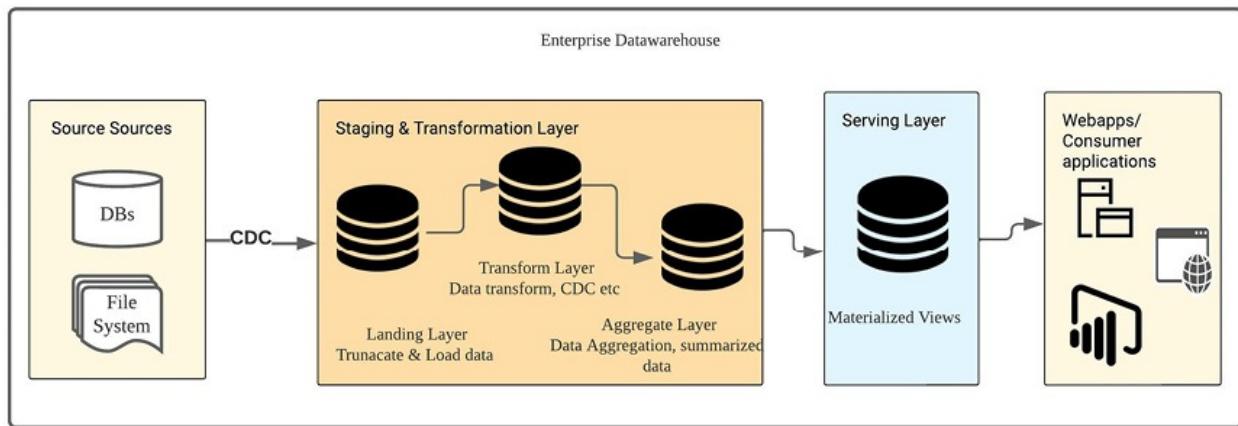


Figure 5.1: CDC in EDW

The CDC process consists of operations like inserting, updating, and deleting records from the target table. The changed data is identified based on the key columns and applied to the target table.

There are different types of CDCs - Type I, Type II, Type III, and more. In the case of Type I implementations, there is no history maintained. This type of

implementation will always have the latest and updated record without any trace to the previous changes or record values. This runs an update to the target row like an in-place update and does not maintain any rows of historical values.

Type II implementation is when you need to maintain the history of the record values changed over a period of time. This consists of old records as well as the latest record in the same target table. This track changes as version control with the current flag and dates. SCD Type II is the most common implementation of data changes where you maintain previous versions of the data with a flag to indicate whether the record is the latest or not. Consider a record type of customer details, where you maintain details of the customer when they join and track a record as and when the customer updates their profile details like email, phone or address, and so on.

Type III implementation is like history versioning, preserving versions of data. Track changes to the selected attributes and add an additional column to store the previous column value, which is modified later as changes are processed. In this case, you add an additional column to maintain values, whereas in Type II, you maintain the entire record with a flag.

In a typical implementation of SCD Type II, the target table needs to have a few extra columns to maintain the metadata or versioning of the rows getting modified. Additional columns like active indicator, start date, end date, and so on. are to be added to the target as well as intermediate tables processing these changes. Here, the active indicator column represents the state of the record if this is the latest and active or inactive or historical value. For any new or updated records, the indicator column is set to **TRUE** (**yes** or **Y**), and for any old records, it is set to **FALSE** (**N** or **No**). The start date is the date when the row is loaded into the table. The end date represents the date or time when the record gets updated for historical records. For all new/active records inserted, the default value of the end date is set to **NULL**.

You will need the following operations to implement changes to the target data:

- **Insert:** Insert new records received as part of delta changes.
- **Update:** Update the old record value flags to set them as a history record
- **Delete:** Delete old record (this will not be used in Type II)
- **UPSERT:** This is the operation that is referred to as **Update + Insert** where you update the existing record and insert the new value record received in delta changes. This is used widely to implement Type II.

Following are some of the most common steps followed to identify changes and apply them to the target table as part of Type II implementation:

- **Delta data capture:** Receive source data on a scheduled basis or pull source data on a regular basis.
- **Identify delta data:** Compare received delta data from the source against the target data to locate changes. Identify records/rows to be updated from the target table and new rows to be loaded.
- **Apply delta changes:** You need to run updates as well as inserts to apply delta changes to the target data. Apply updates and modify metadata columns to represent the change applied. Also, apply all new rows to the table via insert operations. Remember, for all new records or rows, populate metadata columns to represent latest record.
- Some of the DW or Data Lake implementations use **Batch_ID** or **LOAD_ID** to identify unique loads for the given run date. This column can be used to identify the delta data processed in the batches and use this column to process and access any records or rows for that specific run.

Now, you know the change data capture and its types. In the upcoming section, you will learn more about Snowflake streams and how these can be used to implement CDC.

Understanding Snowflake Streams

Stream is a Snowflake object that can be created, altered, or deleted as any other Snowflake object. Streams can be created on any type of table. This supports tables like, standard, directory, external tables as well as view with tables used in view DDL. Stream created on top of tables to track changes coming as part of **Slowly Changing Dimensions (SCD)** or CDC.

Once Streams are created on any table, it can track any type of DML changes on it. This object can keep track of any DML operations like inserts, updates, or deletes implemented on the source table. Streams identify and list these down operations. This is like a point-in-time snapshot of the source table. Every time, it compares the changes with the help of a snapshot captured and lists down the changes made to the source table to be logged into Stream.

Stream works at a transactional level where it captures every minute changes done to the source data. Stream continuously works on comparing, tracking, and

identifying changes to the source data. All operational changes are captured and logged into Streams created on top of these tables. These are maintained in the form of metadata columns recording action to represent whether this is an addition, deletion, or modification of the data.

Remember, stream never store or hold any actual data. However, this stores the log data used for CDC. Following are the metadata-related columns:

- **METADATA\$ACTION**: This refers to the DML operations (**INSERT**, **DELETE**) recorded.
- **METADATA\$ISUPDATE**: This refers to the operations that were part of an **UPDATE** statement. Updates are represented by the **DELETE** and **INSERT** operations.
- **METADATA\$ROW_ID**: This represents a unique and immutable (not changeable) ID of the record, and this can be used to track changes over time.

Types of Streams

There are three types of Streams:

- Standard
- Append-only
- Insert-only

Standard Streams

These types of streams are supported for standard tables, directory tables, or views. A standard stream tracks all types of DML changes to the source object. These changes include inserts, updates, and deletes, as well as table truncates. This stream performs a join on inserted and deleted rows in the change set to derive the row level delta.

Note: Standard streams cannot be used to retrieve change data for Geospatial data. Snowflake recommends creating append-only streams for Geospatial data.

Append-only Streams

These streams can be used for standard, directory tables, and views. This stream tracks rows that are inserted. As the name says, this does not capture any

changes due to updating, deleting, and truncating tables.

These stream returns rows that are inserted only and to be appended. Hence these are performance efficient than standard streams. For instance, the source data can be truncated as soon as the data from the append-only stream is consumed. The table truncation or records deletion does not contribute to any overhead of tracking as standard streams.

Insert-only Streams

These types of streams are supported only on external tables. This stream tracks only rows that are inserted. This does not track any delete operations. For example, when you create an external table from a cloud location, it contains source files. When you remove one of the source files, such as **File1**, and add a new file called **File2**, then the external table returns data from file 2 only.

Any file that get overwritten or appended are treated as new files. The old files are removed from the cloud storage, and new files are maintained. This insert-only stream contains all the new data from the new file. This stream does not capture any type of difference between the old and new versions of files.

Validity of change logs

You can define data retention on a table, and a stream defined on the table stores log data. This log data can be stale depending on the data retention. This does not apply to the external tables as these tables cannot have data retention defined on them. The stream validity or retention is equivalent to the table retention. For example, if the source table has 14 days of data retention, then streams can also hold data for 14 days, and if this data is not consumed within 14 days, then you might lose the changes as log data is not retained beyond 14 days.

You can create the stream based on your needs and business requirements to capture the changes. In the next section, you will learn more about creating and using these streams to capture the delta changes.

Billing of Streams

The major portion of the stream cost is associated with its processing time. This processing time is the time used by a warehouse to process the log data from streams. Along with computing, this also contributes to the storage cost depending on the data retention. Based on your streams and data retention policies, storage costs will be added to your monthly expenses. These usage

charges are added to daily as well as monthly billing and usage in the form of credits.

Implementing data capture with Streams

You can implement streams using DDL statements. You can refer to the following commands as DDL commands to create, alter, and describe streams.

CREATE STREAM

This command is used to create or replace a stream. As you all know, stream captures DML changes on a table or view:

```
CREATE STREAM source_stream ON TABLE source_tbl;
```

Also, you can create a stream using the time travel feature. You will learn more about time travel in *Chapter 8: Data Protection and Recovery*. You can use the following command to create a stream on a table with time travel and given time as specified here:

```
CREATE STREAM source_stream ON TABLE src_tbl BEFORE
(TIMESTAMP => TO_TIMESTAMP(40*30*86400));
```

You can also create a stream on a view:

```
CREATE STREAM view_stream ON VIEW tgt_view;
```

Insert-only streams can be created on an external table. This stream captures changes in data. You can create an external table first by referring to the following example:

```
CREATE EXTERNAL TABLE tgt_table_ext (
    dt_part date as to_date(substr(metadata$filename, 1,
10), 'YYYY/MM/DD'),
    tm_ts timestamp AS (value:ts_time::timestamp),
    usr_id varchar AS (value:user_id::varchar),
    clr varchar AS (value:color::varchar)
) PARTITION BY (dt_part)
LOCATION=@ext_tgt_stg
```

```
AUTO_REFRESH = false  
FILE_FORMAT=(TYPE=JSON);
```

Once the table is created, create a stream on the table:

```
CREATE STREAM tgt_tbl strm_ext ON EXTERNAL TABLE  
tgt_ext_table INSERT_ONLY = TRUE;
```

You can use the **SHOW** command to list down all streams:

```
SHOW STREAMS;
```

ALTER STREAM

This command is used to alter the details of the stream created:

```
ALTER STREAM tgt_stream SET COMMENT = 'New comment for  
stream';
```

DESCRIBE STREAM

This command is used to describe the streams created:

```
DESC STREAM tgt_stream;
```

DROP STREAM

This command is used to drop the streams created.

```
SHOW STREAMS LIKE 'tgt%';
```

```
DROP STREAM tgt_stream;
```

```
SHOW STREAMS LIKE 'tgt%';
```

SHOW STREAMS

This command is used to list the streams. You can list down the streams to which you have access:

```
SHOW STREAMS LIKE 'tgt%' IN demo_poc.public;
```

Stream examples

Consider a use case for a retail domain and you need to maintain details of customer profiles. You need to create a stream to capture changes from the source and apply them to the target table. Follow the following steps to create the tables and streams:

1. Create a table to store the customer names and profile details:

```
CREATE OR REPLACE TABLE customer_profiles (
    cust_id number(8) NOT NULL,
    cust_name varchar(255) default NULL,
    membership number(8) default NULL
);
```

2. Create a stream on top of this customer table:

```
CREATE OR REPLACE STREAM customer_check ON TABLE
customers;
```

3. Create a table to store details of user onboarding, and customer details:

```
CREATE OR REPLACE TABLE Cust_joined
(
    cust_id number(8),
    cust_join_dt DATE
);
```

4. Load data into the customers and customer onboarded table:

```
INSERT INTO customers (cust_id,cust_name)
VALUES
(100,'Jones'),
(201,'Jenniee'),
(302,'James'),
(403,'John');
```

```

INSERT INTO Cust_joined
VALUES
(100, '2023-07-01'),
(201, '2023-07-02'),
(302, '2023-07-04');

```

5. Apply the changes to the tables:

```

MERGE INTO customer_profiles c
USING (
    SELECT cust_id, cust_join_dt
    FROM Cust_joined s
    WHERE DATEDIFF(day, '2023-07-05'::date,
s.cust_join_dt::DATE) < -30) s
ON c.cust_id = s.cust_id
WHEN MATCHED THEN
UPDATE SET c.membership = 150;

```

6. Check the stream data captured:

```
SELECT * FROM customer_check;
```

ID	NAME	MEMBERSHIP	METADATA\$ACTION	METADATA\$ISUPDATE	
100	Jones	150	INSERT	False	5a6
201	Jenniee	150	INSERT	False	9b2
302	James	150	INSERT	False	752
403	John	0	INSERT	False	957

Table 5.1: Result dataset from customer_check

As you can see in the result preceding table, this captures the details of the records and status with metadata columns. Now, you know streams and how streams can be used. In the next section, you will learn more about Snowflake Tasks.

Understanding tasks

Imagine a scenario where you are running jobs and pipelines, or need to run some jobs, or code snippets based on the activities or triggers. You cannot run these pipelines or activities manually. You will need to set them up either using a scheduler or database object-like triggers (earlier data era). You can set them up using Snowflake tasks.

Snowflake tasks can execute the following type of code:

- SQL commands/statements
- Calling a **Stored Procedure (SP)**
- Using procedural logic using Snowflake scripting

Tasks can be set up to trigger at a given interval or based on any event. These can also be combined with Snowflake streams. Tasks will invoke streams to apply delta changes to the target table.

Tasks need compute resources to run the code provided – SQL code or SP or SQL Script. You can select compute models based on your tasks and application requirements. You can have two types of computing resources allocated to tasks:

- User-managed compute (using virtual warehouse)
- Snowflake managed (serverless compute model)

Serverless tasks

These are the type of tasks where compute is managed by Snowflake. Hence, these are also referred to as serverless tasks, as you do not have to manage any compute required for the task's execution. The compute resources required for tasks are allocated, scaled up, down, and resized based on the task workload automatically. Snowflake determines the ideal size of the warehouse based on the dynamic analysis of the stats of the most recent run. The maximum size it can allocate is 2XL. If you have multiple tasks defined, then these tasks can use the same warehouse or compute allocated to serverless tasks.

You can create serverless tasks while defining them in **CREATE** statement. If you are creating serverless tasks, you do not need to define the warehouse parameter, as it is managed by Snowflake. Users who have **CREATE** privilege should also have **EXECUTE MANAGED TASK** privilege. Serverless task billing is different from standard compute billing. You can view the billing on the Snowflake Web UI **Warehouse** tab to view usage.

Note: Serverless tasks have few limitations and cannot invoke below objects or functions:

- User defined functions developed using Java or Python code.
- Stored procedures written in Scala (using Snowpark).
- Stored procedures that call UDFs with Java or Python code.

User-managed tasks

As you know, these are the tasks whose compute resources are managed by users. You can specify the warehouse details while creating the task. As these are user-managed compute resources, you need to choose a warehouse that is sufficient enough to carry out operations.

Scheduling tasks

You can set up a task to run it on schedule or as a root task. This schedule can be created while creating a task or using the **ALTER** command. Only one instance of the task with the schedule. For example, if you are running a task and the next instance is kicked off, then the task execution is skipped.

You can also create a **Directed Acyclic Graph (DAG)** of the tasks with dependencies. DAG runs in a given sequence; each task can have multiple predecessors as well as multiple successors. A task is kicked off only when all predecessor tasks are complete.

As you see in the following DAG, task **A** is the root task, and tasks **B** and **C** are the successors of task **A**. They run in parallel once **A** is complete. Task **D** is dependent on tasks **B** and **C**, and this will not be triggered until tasks **B** and **C** are complete. Task **E** is dependent on task **D**. As and when **D** is complete, **E** will be triggered, as shown:

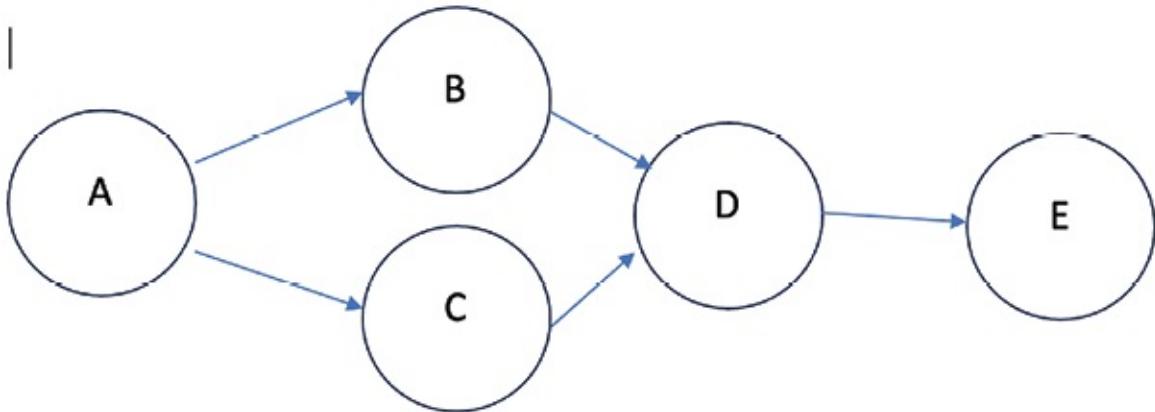


Figure 5.2: Tasks in DAG

You will learn more about implementing tasks and scheduling them in the next section.

Implementing scheduling with Tasks

Tasks can be implemented as Snowflake objects. You can use DDL commands to **CREATE**, **ALTER**, **DROP** commands to create, change, or delete tasks. In this section, you will learn more about **TASK** commands and their usage. You will also learn about a use case implementation that will help you understand **TASK** implementation.

TASK commands

Tasks can be created using the **CREATE** command, altered or modified using the **ALTER** command, and dropped using the **DROP** command. You can use the **SHOW** command to list down all the tasks present. **DESCRIBE** command can be used to get details or properties of the tasks created. You can use the **EXECUTE** command to run the existing task. As you know, tasks can be setup using scheduling time or a trigger. These tasks get executed based on their setup, you can also run them manually using the **EXECUTE** command.

CREATE TASK

This command is used to create a new task or replace an existing task. Like **CREATE** command, this command also follows syntax and given set of properties:

Syntax:

```
CREATE [ OR REPLACE ] TASK [ IF NOT EXISTS ] <name>
```

```
[ { WAREHOUSE = <string> } | {  
USER_TASK_MANAGED_INITIAL_WAREHOUSE_SIZE = <string> }  
]  
  
[ SCHEDULE = '{ <num> MINUTE | USING CRON <expr>  
<time_zone> }' ]  
  
[ ALLOW_OVERLAPPING_EXECUTION = TRUE | FALSE ]  
  
[ SUSPEND_TASK_AFTER_NUM_FAILURES = <num> ]  
  
[ ERROR_INTEGRATION = <integration_name> ]  
  
[ COMMENT = '<string_literal>' ]  
  
[ AFTER <string> [ , <string> , ... ] ]  
  
[ WHEN <boolean_expr> ]
```

AS

<sql>

Where:

- **WAREHOUSE**: Required for the managed tasks. For serverless tasks you do not need to provide the warehouse name.
- **SCHEDULE**: Provide details of task scheduling
- **OVERLAPPING_EXECUTION**: Use this parameter if you want to set up overlapping or concurrent executions of DAG instances. Remember, this parameter can be used at a root task for concurrent execution of DAGs.
- **SUSPEND_TASK_AFTER_NUM_FAILURES**: Specifies the number of failures **TASK** can accept. If this is reached the **TASK** gets suspended automatically.

These are the optional parameters that can be set at task creation. **Task Name** and **SQL** are the required parameters or minimal parameters required to create a task.

ERROR_INTEGRATION: Used to specify integration required to send alerts in case of failures. Only cloud integrations are supported, for example, API integrations. Notification integration used to send mail is not supported as **ERROR_INTEGRATION**.

Whenever you create a **TASK**, it gets created in the **SUSPENDED** task. You need to run the **ALTER** command to **RESUME** the **TASK** for executions. Each account can

have 10,000 resumed tasks or active tasks that are in a running state. There can be more than 10,000 in total, along with a suspended state.

Examples:

- Consider a scenario where you need to create a user-managed **TASK** that runs between 8AM to 4PM on Saturday:

```
CREATE TASK Demo_Task_hour  
  WAREHOUSE = demo_wh  
  SCHEDULE = 'USING CRON 0 8-16 * * SAT'  
AS  
  SELECT CURRENT_TIMESTAMP;
```

- You can also set up a task on top of the stream to insert the change data to the target table. Consider a use case where you need to insert records received from a stream that are newly inserted into the table and captured as **INSERT** metadata action. This task checks for updates every 10 minutes:

```
CREATE TASK cdc_task  
  WAREHOUSE = demo_wh  
  SCHEDULE = '10 minute'  
WHEN  
  SYSTEM$STREAM_HAS_DATA('CDC_STREAM')  
AS  
  INSERT INTO tgt_table (cust_id,cust_name) SELECT  
    cust_id, cust_name FROM cdc_stream WHERE  
    METADATA$ACTION = 'INSERT';
```

- Consider you want to create a task to run set of SQL statements as part of BT/ET- **Begin** and **End Transactions**:

```
CREATE TASK demo_logng  
  USER_TASK_MANAGED_INITIAL_WAREHOUSE_SIZE =
```

```

'XSMALL'

SCHEDULE = 'USING CRON 0 9-17 * * *

AS

BEGIN

    ALTER SESSION SET TIMESTAMP_OUTPUT_FORMAT =
'YYYY-MM-DD HH24:MI:SS.FF';

    SELECT CURRENT_TIMESTAMP;

END;/

!set sqlDelimiter=";"
```

ALTER TASK

This is used to alter task configurations and parameters. This command is used to resume the task as well as suspend it whenever needed. This is also used to change the pre and post dependencies of the tasks in DAG. This command is used to **SET** and **UNSET** values of **warehouse**, **schedule**, **config**, **comment**, **tag**, and so on.

Syntax:

ALTER TASK [IF EXISTS] <name> RESUME | SUSPEND -To resume and suspend task

ALTER TASK [IF EXISTS] <name> REMOVE AFTER <string> [, <string> , ...] | ADD AFTER <string> [, <string> , ...] -- change the predecessor or successor of task in DAG

ALTER TASK [IF EXISTS] <name> SET OR UNSET [PARAMETERS USED IN CREATE] -parameters used to set and unset

```
ALTER TASK [ IF EXISTS ] <name> SET OR UNSET TAG  
<tag_name> = '<tag_value>' - used to set tag
```

Examples:

- Command to resume or suspend task:

```
ALTER TASK demo_task RESUME;
```

```
ALTER TASK demo_task RESUME;
```

- Command to unset warehouse from user-managed task to serverless task:

```
ALTER TASK demo_task UNSET WAREHOUSE;
```

```
ALTER TASK demo_task SET  
USER_TASK_MANAGED_INITIAL_WAREHOUSE_SIZE =  
'SMALL';
```

- Change the schedule of a task:

```
ALTER TASK demo_task REMOVE AFTER prev_task_1,  
prev_task_2;
```

```
ALTER TASK demo_task ADD AFTER prev_task_3;
```

DROP TASK

This command is used to drop tasks.

Syntax:

```
DROP TASK [ IF EXISTS ] <name>
```

Examples

- List all available tasks:

```
SHOW TASKS LIKE 'demo%';
```

- Drop task:

```
DROP TASK demo_task;
```

- List all available tasks:

```
SHOW TASKS LIKE 'demo%';
```

EXECUTE TASK

This command is used to execute the tasks manually. This manually triggers the task, and it executes immediately. Task can be an independent task or asynchronous DAG. A successful invocation of root task triggers subsequent or child tasks in the DAG.

Syntax:

```
EXECUTE TASK <name>;
```

Example:

```
EXECUTE TASK demo_task;
```

SHOW TASKS

This task is used to list all tasks where users have privileges. This lists all tasks that are in any state across all accounts.

Syntax:

```
SHOW TASKS <name>;
```

Example:

```
SHOW TASKS LIKE 'demo%' IN demo_db.public;
```

These are the commands you can use to create, alter, describe and list tasks. You can use this feature to schedule operations in the Snowflake in the form of independent or asynchronous DAGs.

Practical: Create Snowflake Tasks and Streams

Exercise 1

1. **Create Stream:** Implement a scenario to create table, reference data, integrate changes and create streams on top of it to capture changes.
 - a. Create table to capture Retail order, purchase or sale data:

- i. **ORDER** table
 - ii. **SALES** table
- b. Create sample data inserts:
- i. Load initial data to the sample tables created in above step.
 - ii. Load changed data to the table to capture changes in stream (Run this after the step c).
- c. Create stream to capture changes on top of table.
- d. Run *b(ii)* to load change data to the table.
- e. Run query on Stream to check the changes captured.
- f. Apply the changes to the target table using metadata action columns.

Exercise 2

1. **Create Task:** Create a task to read data from the stream created in exercise 1 and process the changes on fixed schedule of every 10 minutes.
 - a. Create a task that runs on a schedule of every 10 minutes, for all days:
 - i. Schedule: 10 days
 - ii. Serverless task
 - iii. Command to capture changes from stream and apply to the target table.
 - iv. Create task
 - v. Change the status of the task to **RESUME**.
- vi. Monitor the task while applying changes to the table created in Exercise 1. introduce changes to the data manually to see changes captured.

Exercise 3

1. Convert the task created in exercise 2 to user-managed task.
 - a. Convert the task and capture the usage of serverless vs user-managed tasks.

Conclusion

This chapter provided a summary of Snowflake's Streams and Tasks. This chapter helped you to understand the scenario and stream object required to perform Change Data Capture. This also helps you to learn scheduling using Tasks and native scheduling using Snowflake. You will learn more about other Snowflake features and offerings as you go along with the next chapters in this book.

In the next chapter, you will learn about Snowflake's Snowpark. Snowpark is the programming offering that supports developing pipelines using Python, Java, and Scala. You will also learn Snowpark features, use cases, and implementation with Data Lake.

Points to remember

Following are the key takeaways from this chapter:

- Change Data Capture is the critical part of data platform implementation.
- Snowflake offers its native feature to support Change Data Capture.
- Snowflake's Streams are used to implement Slowly Changing Dimensions.
- Snowflake offers Tasks that can be set up to implement scheduling.
- You can use Streams and Tasks to automate change data capture.

Multiple choice questions

1. **When can Streams be used?**
 - a. Capture Data Changes
 - b. Capture DDL changes

- c. Capture Operational changes of table
- d. All of the above

2. Where can Tasks be used?

- a. Schedule operations
- b. Schedule SQL statements
- c. Schedule SQL scripting
- d. All of the above

Answers

1	a
2	d

Questions

1. Explore how you can implement different types of SCDs using Streams
2. How can operations be set up using Tasks to create asynchronous DAGs?
3. What are the use cases and scenarios you can implement using Snowflake Tasks?
4. Explore how you can compare the performance and usage of serverless vs user-managed tasks.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 6

Understanding Snowpark

Introduction

Data processing is critical to any data platform design and implementation. You might have observed maintaining separate environments to build data pipelines using various programming languages or services to serve business needs. One data platform can have multiple applications integrated and needs data movement between these applications. This might impact the data governance and lose control over maintaining governance policies and securities.

This chapter covers scenarios to help you understand the need for implementing and using Snowpark to serve various application and programming needs within the same data platform without compromising on data governance and with zero data movement. This also covers Snowpark implementation, architecture, and warehouses that support Snowpark operations.

You will learn about Snowpark programming as well as **Machine Learning (ML)** capabilities in this chapter. This chapter illustrates capabilities with real-time scenarios and references to implement Snowpark. Sample examples and lab questions in this chapter will help you learn with labs. There are a set of questions to check your knowledge of understanding concepts explained in this chapter.

Structure

This chapter consists of the following topics:

- Why Snowpark?

- Understanding Snowpark
- Implementing Snowpark
- Use cases to implement Snowpark

Objectives

By the end of this chapter, you will be able to understand Snowpark and its offering to support applications as well as ML needs. This chapter helps you to create the required objects to implement Snowpark. You will be able to implement them using the trial account setup in [*Chapter 1, Getting Started with Snowflake.*](#)

Why Snowpark?

Consider a scenario where you are building a data platform to integrate various applications and data processes to serve your business needs. Often, these applications are built using a preferred choice of languages and features. You might have observed this in earlier platform implementations where some of the applications use ETL tools, some of them use Hadoop, and spark implementations, and some of them might be using data platform native features and SQL to support application needs. In this scenario, you need to maintain various environments, software licenses, data pipelines, and programs.

Following are some of the challenges and complexities with traditional implementation:

- **Heterogeneous environments:** Customers often have separate environments for processing separate workloads built using different languages.
- **Capacity and resource sizing:** This is a common challenge and complex implementation to meet the capacity and resourcing needs of various applications on the same platform.
- **Data consistency:** Data needs to be shared and moved between/across these applications. This might cause data silos and consistency issues due to movement of the data.
- **Data governance and security:** There might be an impact in maintaining data control and strict implementation of policies across. You might

observe some of the issues due to a lose control over security and policies.

You can refer to *Figure 6.1* to illustrate the traditional approach and relate with the challenges mentioned above:

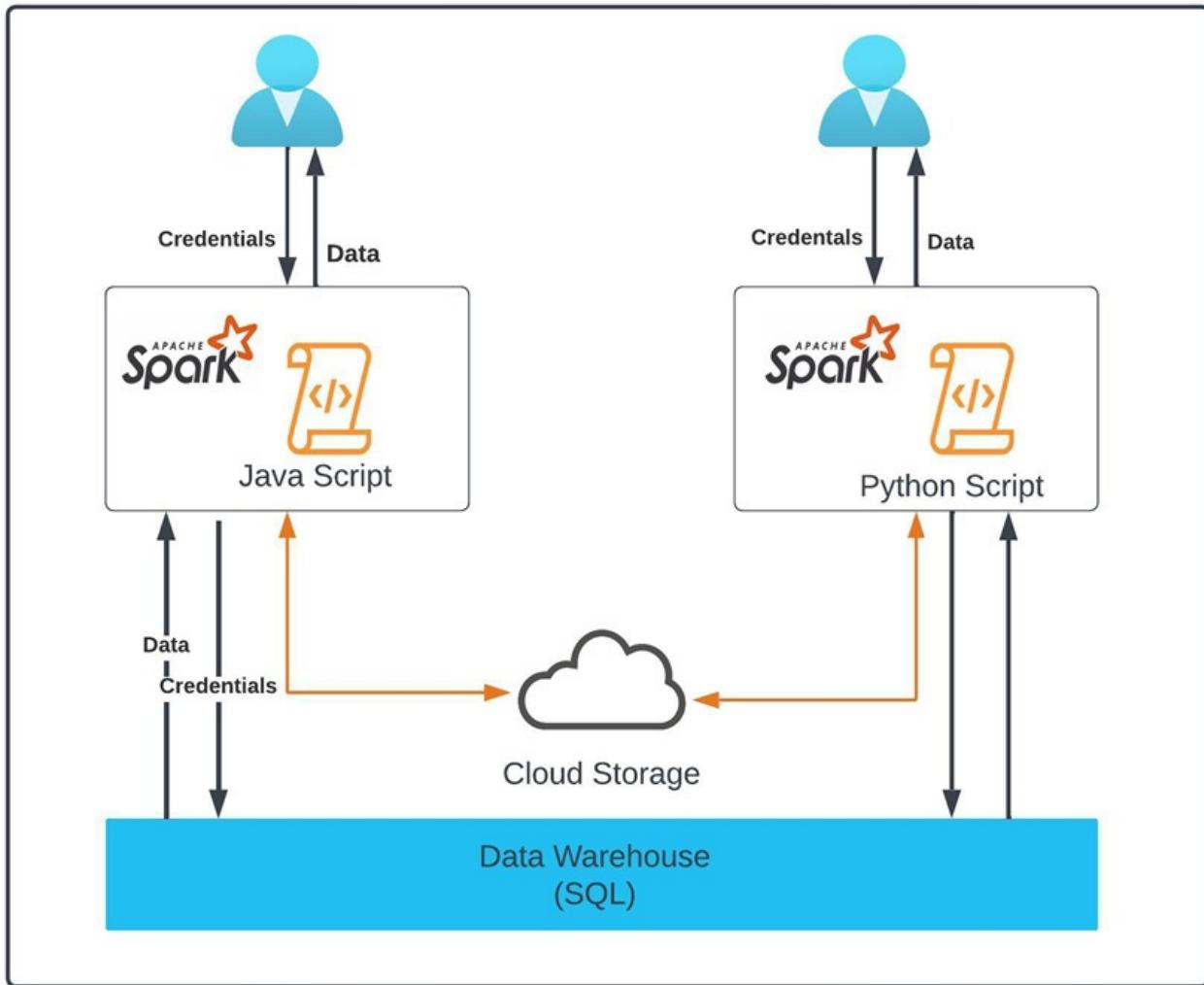


Figure 6.1: Typical Spark Environment

Snowpark is designed to address most of these traditional challenges. Snowpark offers below features to support the application design needs:

- **Streamline architecture:** This offers support to the different programming languages of choice.
- **Scalable and optimized pipelines:** Snowpark leverages Snowflake features to offer great performance, scaling, and optimized workloads.
- **Platform consistency:** This also offers enterprise-grade controls, security,

and governance across all applications. This helps to enforce strong consistency.

You will learn more about Snowpark's streamlined architecture, use cases, and support for the programming languages in the next section.

Understanding Snowpark

Snowpark is the only platform that has native support for different languages like Python, Java, SQL, and more. This platform allows users to address all of the typical challenges from earlier or old implementations. Snowpark has the following features:

- **One platform with support for different languages:** This eliminates the need to maintain different environments to support the programming choice of users.
- Simplest capacity management and resource sizing
- Streamline architecture with simpler collaboration of data within the same Snowflake data platform.
- Consistent data governance and security implementation across applications.

You can observe all these features and relate them with the architecture represented in [*Figure 6.2*](#):

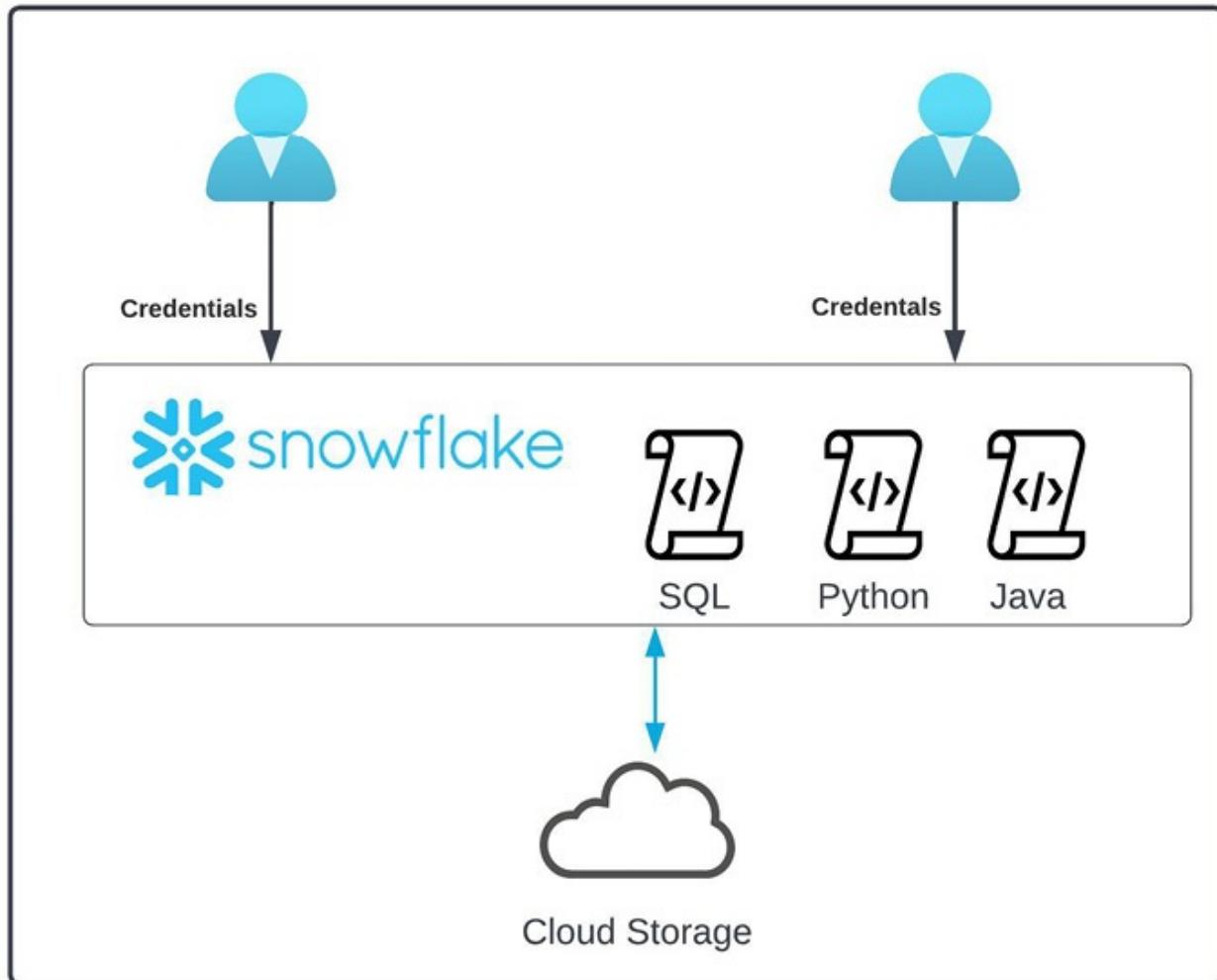


Figure 6.2: Snowpark streamline architecture

You can continue coding the same way as old applications and choice of your programming language. Snowpark keeps the development experience the old way. However, the execution experience is more efficient in comparison with earlier tests or architecture.

Snowpark offers an intuitive library to support querying and processing data at scale in Snowflake. You can use this library for any of three languages – Java, Python, and Scala. You can develop and deploy applications that process data in Snowflake without data movement. This platform allows you to run application code and process it at a scale as part of the Snowflake engine. This supports native features like elastic and serverless execution of the code.

Snowflake also supports Snowpark offering corresponding warehouse options. You have learned about the warehouses in the initial chapters and about using them for given workloads. Snowflake offers Snowpark warehouses to support

Snowpark workloads.

Snowpark-optimized warehouse

Snowpark workloads can be run on standard warehouses as well. Snowflake offers Snowpark-optimized warehouses and recommends using them for any workloads that need huge memory for processing. Snowpark-optimized warehouses can also be used for machine learning model training use cases using a stored procedure on a single virtual warehouse node. Snowpark-optimized warehouses might take some time to set up or for initial creation and resumption in comparison with standard warehouses.

These warehouses can be created using a **CREATE DDL** and providing the type of warehouse you want to create:

```
CREATE OR REPLACE WAREHOUSE snowpark_demo_wh WITH  
WAREHOUSE_SIZE = 'LARGE'  
WAREHOUSE_TYPE = 'SNOWPARK-OPTIMIZED';
```

Also, when you are running any stored procedures or UDFs or UDFTs, then Snowflake recommends setting up the concurrency level to 1 to fully utilize warehouse capacity to support your workload. You can use the following command to set the concurrency level:

```
alter warehouse snowpark_demo_wh set  
max_concurrency_level = 1;
```

You can use this feature while running on any ML workloads or SQL intense workloads using Snowpark, where high memory operations are being run on the data.

Snowpark-optimized warehouse billing

Snowpark warehouses are not available in **Extra Small (XS)** and **Small (S)** sizes. These warehouses are available between M to 6XL, and the credits consumed for these warehouses are separate from standard ones. The billing is done every minute, the same as in standard warehouses. The following table illustrates the details:

Size	M	L	XL	2XL	3XL	4XL	5XL	6XL
Credits	6	12	24	48	96	192	384	768

Table 6.1: Snowpark Warehouse Sizes

Snowpark-ML

Snowpark also supports ML libraries, and you can implement, develop, and deploy ML models using Snowpark ML. You will learn more about this feature in the next section.

Snowpark-benefits over Spark Connector

Snowflake also offers support using libraries and connectors. You can also write a program using Spark and connect to Snowflake using Spark connector. This uses Spark and Snowflake connectors to perform the workload developed. This is separate from Snowpark. You can use Snowpark over Spark connector and gain the following benefits:

- Support the libraries to interact with Snowflake data without impacting the performance and functionality.
- Support to tools like Jupyter, Visual Studio code or IntelliJ to develop the code
- Support to push down to perform all operations. This includes all types of workloads and UDFs. This pushes down all the operations to Snowflake and leverages the power of Snowflake features.
- This does not need any separate cluster outside Snowflake to execute or maintain application workloads or code.
- All the computations are done within Snowflake.
- No need to move data while defining data frames, you can define frames using select statement, and it will read data from the underlying Snowflake table.
- Allows you to create UDFs in line with the Snowpark app.

Implementing Snowpark

Snowpark library offers an API to be used for querying as well as processing data. This library can be used to build applications to process data in Snowflake, and you do not need to move data from Snowflake. Your application code and data run in the same environment. These data transformations can be automated

using stored procedures, and you can leverage Snowflake native tasks to schedule these using tasks.

You can write and use Snowpark in Java, Python, and Scala. This section covers details of the Python implementation and shares references to the Java and Scala implementations.

Environment setup

You can set the following components to get started:

- **Python code:** You can use the local development environment. You can also use the Python worksheet in Snowsight to develop code.
 - Snowsight can be used to develop Python code and is preferred when you need to develop stored procedures and set them up using tasks. You can setup and use Python worksheets easily. You can refer to <https://docs.snowflake.com/developer-guide/snowpark/python/python-worksheets> to get more insights on setup and usage.
- In the case of client application development, set the following:
 - Setup development environment (Preferred Development Environment: PDE) to develop Snowpark applications. You can also refer to <https://docs.snowflake.com/developer-guide/snowpark/python/setup> to setup the development environment.
 - Configure and establish a connection to the Snowflake instance. You can refer to <https://docs.snowflake.com/developer-guide/snowpark/python/creating-session> to set up a connection.

Using Python and developing code

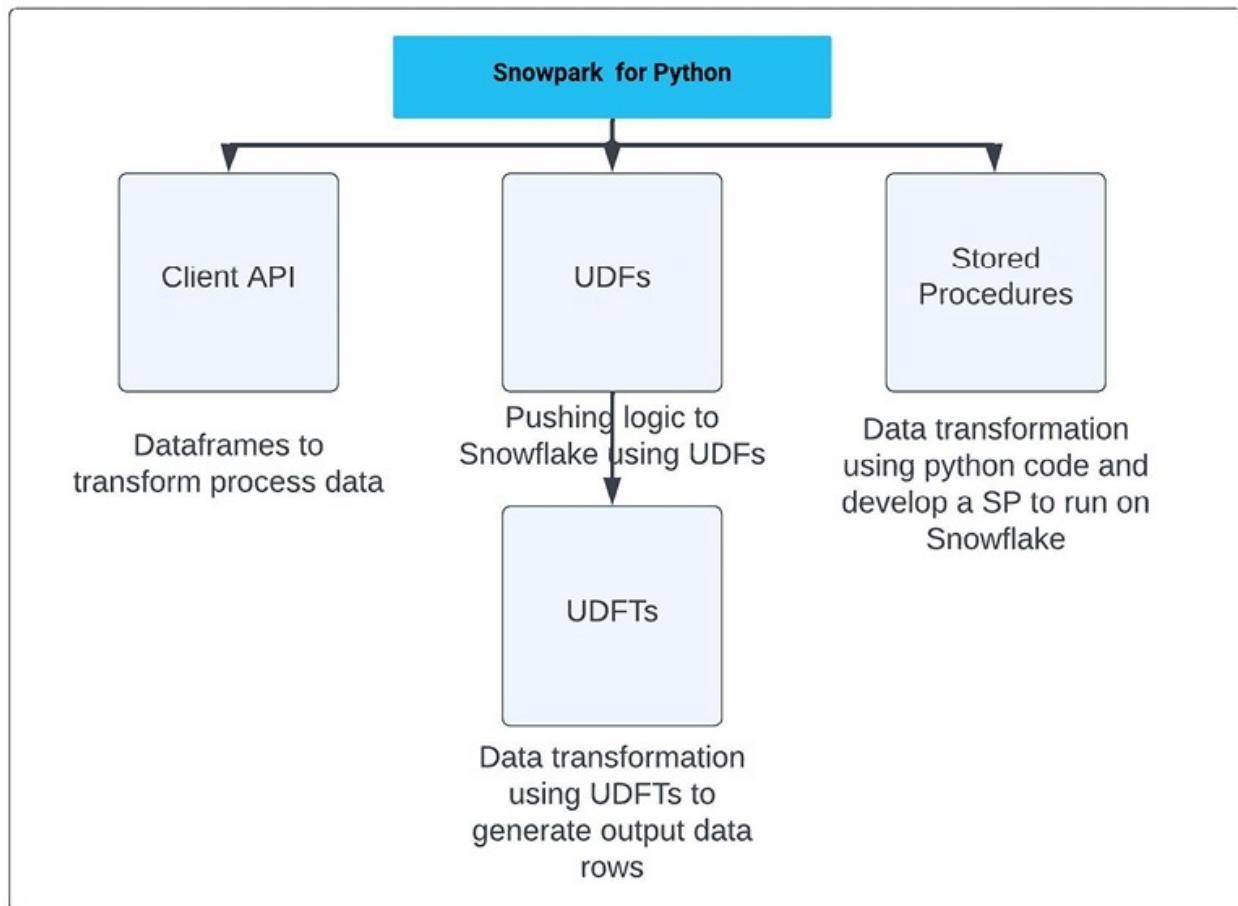
Snowpark Python offers a variety of ways to connect and read data from Snowflake. These ways are discussed as follows:

- Process data and run query using Dataframe object.
- Create **User Defined Functions (UDFs)** to represent custom lambda functions. These can be used to process the data.
- Snowflake supports **User Defined Table Functions (UDFT)**. You can create these UDFTs to process data and return the required result in the

form of multiple rows and columns.

- You can also write a **Stored Procedure (SP)** to process the data. These SPs can be scheduled using Snowflake tasks to develop as data pipelines.

The following figure depicts the various ways to connect and read data:



How does Python dataframe API work?

You can use data frame API to develop the data processing logic using data frames. These data frames can be used to read, process, transform, and write data to the Snowflake tables. These dataframes run queries against Snowflake tables, and this internally gets interpreted and converted as SQL logic to run against the SQL engine.

You can refer to [Figure 6.4](#), which demonstrates the execution of data frame API query. This query gets translated over query translator, and using Python

connector for Snowflake, the query gets pushed to the processing engine and gets executed over SQL engine:

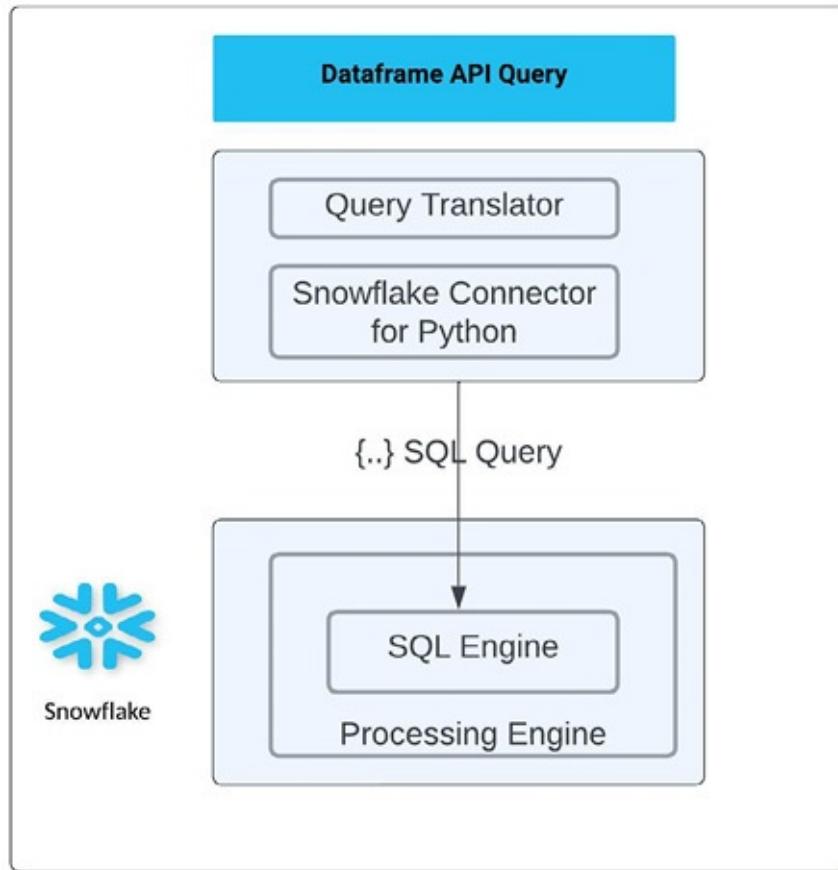


Figure 6.4: Dataframe API query execution

This is 100% push down implementation to Snowflake. The queries getting executed leverages Snowflake native features to achieve performance optimization.

You can also bring in the custom Python code to Snowflake as UDF, UDFTs and Stored Procedures. Code gets serialized and pushed down to run in a secure sandbox environment. This allows to access third party packages and Anaconda integration seamlessly.

Using Snowpark for ML

Snowpark can be used to develop ML pipelines using stored procedures. You can train machine learning models by writing stored procedures. Also, you can train, score, and tune ML models using Snowpark Python's stored procedure and deploy using user-defined functions.

Snowpark ML offers a set of tools with SDKs to leverage underlying infrastructure to build and deploy machine learning models. This allows you to benefit from Snowflake like performance, scalability, stability, and governance at all stages of ML workflows.

Use cases to implement Snowpark

As you all know by now, Snowpark supports both data engineering as well as machine learning type of workloads. The most common use cases to implement Snowpark are the same – Engineering as well as ML. This is illustrated in the following figure:

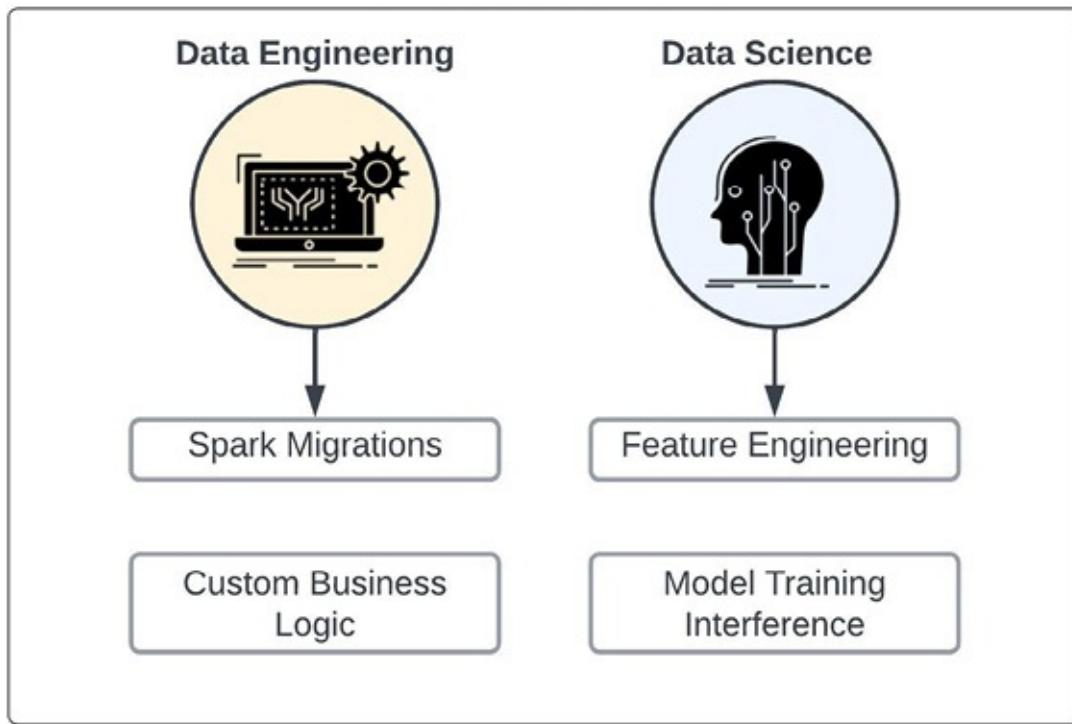


Figure 6.5: Snowpark Use cases

Data engineering use cases

Data engineering use cases involve two major categories – migrating Spark workloads and developing new workloads using Snowpark.

Migrating Spark workloads

This is the most common use case of data lake modernization where data lake from Hadoop or Spark ecosystem is being migrated to Snowflake. As you know,

Snowflake supports data platform implementation for Data Warehouse, Data Lake, Lakehouse, and so on.

In any other migration use cases, these spark workloads need to be converted or hosted on services to set up these spark jobs. In the case of Snowflake, users can leverage Snowpark API and migrate or set up their existing Python + Spark or Spark + Scala workloads. Users can use Spark-optimized warehouses and leverage dataframe APIs to push down queries to Snowflake. You will get performance efficient workloads upon implementation.

Data science (ML) use cases

This is the type of use case where you can migrate your existing ML workloads or set up new ML workloads. Snowpark ML offers support to various phases of the ML lifecycle, like model development and model ops. You can refer to the following figure that represents the ML lifecycle in Snowflake:

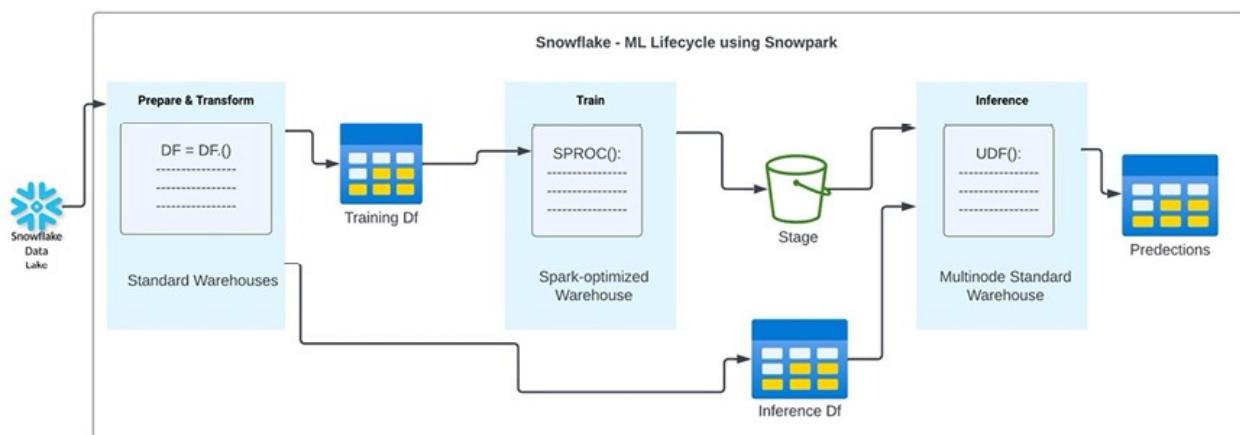


Figure 6.6: Snowpark ML lifecycle

This lifecycle represents effortless, scalable, and secure processing without moving data across multiple environments.

Snowpark ML capabilities allow users to develop their own models in the Snowflake platform itself, unlike other implementations where you need to bring data out of the platform to support or use ML services or use cases. This is the foundation of possible ML capabilities of the Snowflake platform. In the future, this will also allow users to run heavy duty ML models, processing, running **Large Language Models (LLMs)**, and implementing Generative AI on top of Snowflake data.

In the next section, you are going to perform hands-on labs leveraging the

Snowflake Quick Start program. This allows users to explore, learn, and implement various use cases with an easy, quick tour guide.

Practical: Setting up Snowpark and use cases

As you know, Snowpark can be used for data engineering as well as ML use case implementations. Snowpark can be used with your preferred choice of programming. Below are some of the reference links to get started with Snowpark and hands-on labs.

Snowpark engineering use cases

This section shares details of the dataframe API use case in Python implementation. You can refer to the reference links mentioned to choose between Java or Scala implementations.

Dataframe use case

1. Using DataFrames in Python:
 - a. Snowpark processing queries in the form of dataframes.
2. Implementation reference:
 - a. Refer: <https://docs.snowflake.com/en/developer-guide/snowpark/python/working-with-dataframes> for Python dataframe implementations.
3. Other references:
 - a. Java reference for dataframes: <https://docs.snowflake.com/en/developer-guide/snowpark/java/working-with-dataframes>.
 - b. Scala reference for dataframes: <https://docs.snowflake.com/en/developer-guide/snowpark/scala/working-with-dataframes>.

Data engineering pipeline

1. Snowflake offers quickstarts to help users to learn and explore various use cases.

Refer:

https://quickstarts.snowflake.com/guide/data_engineering_pipelines_wi_index=..%2F.index#0 to implement data engineering pipeline with Snowpark.

2. **Pre-requisites:** Users with following knowledge can perform the lab mentioned and details shared in above step:
 - a. Understanding of Python
 - b. Familiarity with the DataFrame API
 - c. Understanding Snowflake
 - d. Familiarity with Git repositories and GitHub

Conclusion

This chapter provided a summary of Snowflake's data engineering as well as data science capabilities and implementation using Snowpark. This chapter helped you to understand various programming languages supported while building Snowpark workloads. This also helped you to learn Snowpark use cases and ML lifecycle. You will learn more about other Snowpark architecture and underlying implementation leveraging Snowflake native offerings. You will learn more about Snowflake features and offerings as you go along with the next chapters in this book.

In the next chapter, you will learn about Snowflake's access control, users, and roles. You will also learn how **Role Based Access Control (RBAC)** is implemented in Snowflake. You will learn more about user creations, Snowflake's default or native roles as well as custom roles users can create.

Points to remember

- Snowpark is a single platform that supports different programming languages – Java, Python, and Scala.
- Snowpark can be used for data engineering as well as ML workloads.
- You can use engineering modernization – data lake modernization or spark workload migration to Snowpark.
- Snowpark ML leverages Snowflake native features and platform

capabilities to ensure seamless integration, and scalable and performance efficient implementation.

Questions

1. How does Snowpark data frame query API run queries on Snowflake?
2. How does dataframe query API convert queries to Snowflake SQL and works as push down optimization?
3. How Snowpark is different from Spark and Spark connector?
4. What are the Snowpark optimized warehouses? When can you use these warehouses vs standard warehouses?
5. Can you run Snowpark jobs or workloads on standard warehouses? What is the scenario when you need Snowpark optimized warehouses?

Multiple choice questions

1. **What are the programming languages supported in Snowpark? (Select One)**
 - a. Spark
 - b. Python
 - c. SQL
 - d. JavaScript
2. **What is the implementation method used by Snowpark? (Select One)**
 - a. Push Up
 - b. Push Down
 - c. Equal distribution
 - d. Leveraging Snowflake native features
3. **What is the type of workloads supported by Snowpark? (Select Two)**
 - a. Engineering

- b. Machine Learning
 - c. DataOps
 - d. Both a and b
4. **What is the type of phases supported for ML workloads supported by Snowpark? (Select Two)**
- a. Engineering
 - b. Machine Learning
 - c. DataOps
 - d. Both a and b

Answers

1	b
2	b
3	d
4	d

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 7

Access Control and Managing Users Roles

Introduction

Data governance is one of the most critical pillars of data architecture and platform design. Data governance has pillars that guide users to design and implement appropriate rules, and policies while designing the data platform. Access control is one of the critical elements of data governance implementation. User onboarding, assigning required or granular roles, role creation, and management are a part of the access control setup and design. Snowflake supports **Role Based Access Control (RBAC)** implementation. This means you can create a set of custom roles using Snowflake native roles. Users can be assigned to the defined custom roles or default Snowflake roles.

This chapter covers scenarios to help you understand the RBAC implementation of Snowflake. This also covers Snowflake hierarchy, default roles, custom roles, creation and use of custom roles, assigning and managing users, access to the Snowflake objects, and so on. Snowflake follows the role hierarchy while defining the roles and accesses to the users.

You will learn about user management, role management, access controls, and access management at the role level within Snowflake. Sample examples and lab questions in this chapter will help you learn with labs. There are a set of questions to check your knowledge of understanding concepts explained in this chapter.

Structure

This chapter consists of the following topics:

- Snowflake access control overview
- Understanding RBAC
- Understanding default roles in Snowflake
- Implementing RBAC with Snowflake
- Understanding access hierarchy
- Managing users and roles

Objectives

By the end of this chapter, you will be able to understand Snowflake access control setup, design, implementation and share best practices. This chapter helps you to create the roles required to support various business requirements, managing users and applications for platform access and usage. You will be able to implement them using the trial account setup in *Chapter 1, Getting Started with Snowflake*.

Snowflake access control overview

Access controls are essential while implementing and designing any of the applications as well as platforms. Access control is nothing but a framework that allows users to access the platform, set up access rules, and permissions to the users. This is used to define the policies for the users as well as for the components or objects to allow or grant permissions to read, write, list as well and access components.

Snowflake access control is the framework that defines control policies and supports the following implementation models:

- **Role Based Access Control (RBAC):** This model supports access granted to the roles. Each user is assigned a role, and each role has all required permissions and privileges granted.
- **Discretionary Access Control (DAC):** This model supports access at the object level. Each object has an owner who creates it, and that owner can

grant access to the users.

It is always recommended to use the RBAC model to support and manage access grants at a role level. You need to understand the Snowflake concepts to work on implementing RBAC. Following are some of the key concepts:

- **Securable objects:** This is the entity or object to which access is granted.
- **Role:** This is the entity that needs to be granted access. Roles are assigned to the users. Role can also be assigned to another role creating a role hierarchy.
- **Privilege:** This is the level of access role that needs to be granted. There are multiple privileges like read, write, operate, manage, and so on, that can be granted to the roles.
- **User:** This is the entity used to identify the user at Snowflake. Each user onboarded has a user ID and is assigned to the default role while onboarding.

Snowflake role and users

You can create roles and assign required privileges to access Snowflake objects. Once roles are created, you can start onboarding users to Snowflake and assign one or more roles to the user, depending on the access requirements.

Let us consider that you create a **Role 1** that has access to create objects in a given database and schema. Once you have these objects created, you have development and reporting users to grant access to the objects **Role 1** creates. You can grant access to the roles with a grant so that they can grant any additional privileges to other roles that are dependent or downstream application consumers. The following figure explains this process:

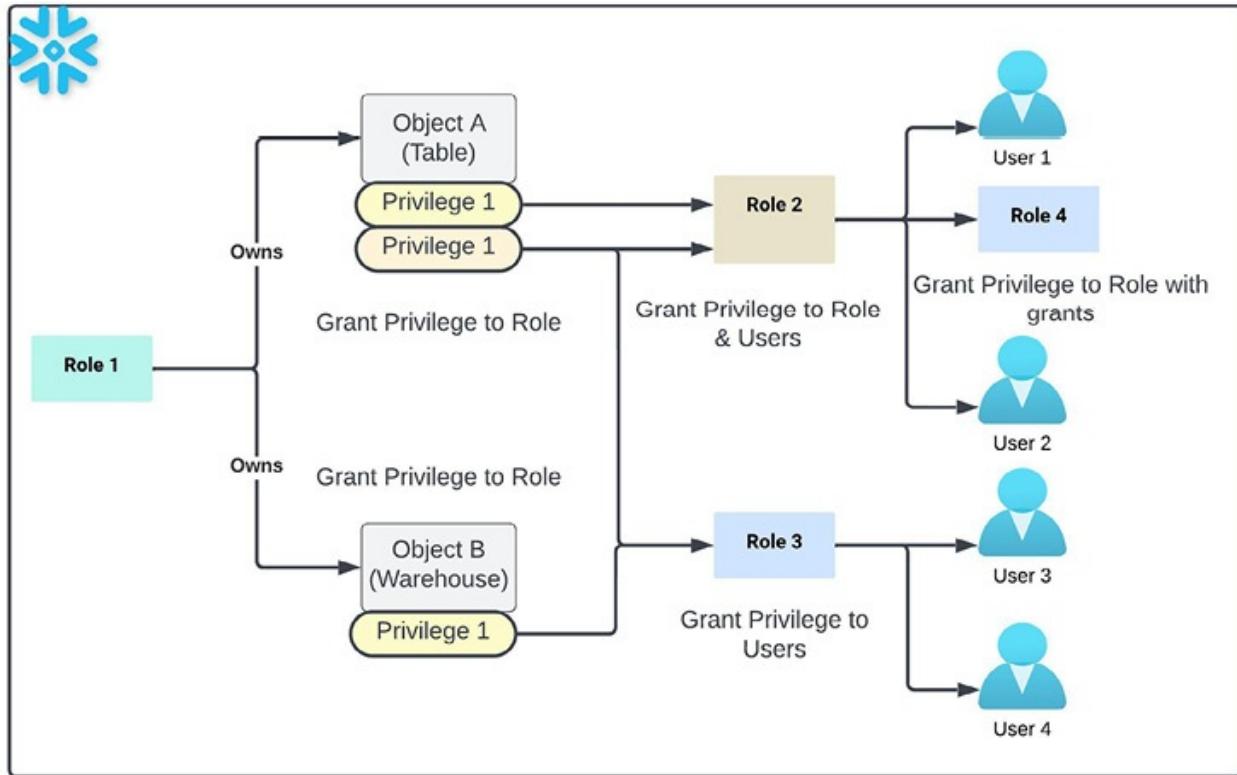


Figure 7.1: Granting roles to users

Snowflake has system defined roles. These are the default roles available in Snowflake. You can create custom roles using these default roles and DCL commands to grant permissions to the required objects. You can create custom roles and drop them whenever not needed. However, system roles cannot be dropped. The accesses granted to custom roles can be granted as well as revoked depending on the requirements, but the access granted to the system roles cannot be revoked. Snowflake supports a variety of roles. You will learn more about these roles in the next section.

Types of roles

You have learnt about system defined roles as well as custom roles. There are a few additional roles, and this section covers details of all types of roles available in Snowflake:

- **Account roles:** Permissions at the account level can be applicable to all objects in that account hierarchy. Grant access or privileges to the role with access to all objects in an account.
- **Database roles:** Some permissions can be granted at a database level. Users or roles can be granted access only at a database level to create,

manage, and use objects created within that database.

- **Active roles:** These are roles or users active in a session. You can use commands to set different users for a session or use different roles for a given session to perform necessary operations. You can use SQL commands like `USE ROLE xxx`.
- **System-defined roles:** These are the roles that are available by default in any Snowflake account for security, user management, and resource management. Following are the set of default roles available:
 - **ORGADMIN:** This role manages resources at an organizational level. This can be used to create accounts in an organization. This can also be used to list all accounts in a given organization using `SHOW ORGANIZATION ACCOUNTS`.
 - **ACCOUNTADMIN:** This role manages resources at an account level. This is the highest role at an account level and encapsulates other roles like **SYSADMIN** and **SECURITYADMIN**.
 - **SECURITYADMIN:** This role is used to manage roles and users at the account level. This is used to manage resource or object privileges for users as well as roles. This user management is part of the **USERADMIN** role that is also encapsulated as part of **SECURITYADMIN**.
 - **SYSADMIN:** This role is used to manage resources at an account level. This manages, uses, and creates resources and objects at the database and account level.
 - **USERADMIN:** This is the role used to manage users at account level. This role can be used to create or drop users. This can also manage the resources that are owned by this role.
 - **PUBLIC:** This is a pseudo role that is assigned to every user getting onboarded to a Snowflake account.
- **Custom roles:** These are the user-defined roles using system-defined roles as well as adding some of the additional privileges required at the Snowflake objects level. These custom roles can be created by **ACCOUNTADMIN**, **SECURITYADMIN**, and **USERADMIN**. You can create a role using the `CREATE ROLE` statement and grant privileges using `GRANT` statements. You will learn more about creating custom roles, assigning privileges, and

using them in the next section of this chapter.

Snowflake securable objects

Snowflake follows a hierarchy for the objects as well. All objects created in Snowflake are part of the logical container in the hierarchy. These logical containers are nothing but account, warehouses, databases, schemas, roles, users, and so on. These logical containers store securable objects. These securable objects are nothing but the tables, views, functions, stages, and objects that can be created within a schema. The hierarchy can be represented in the following figure:

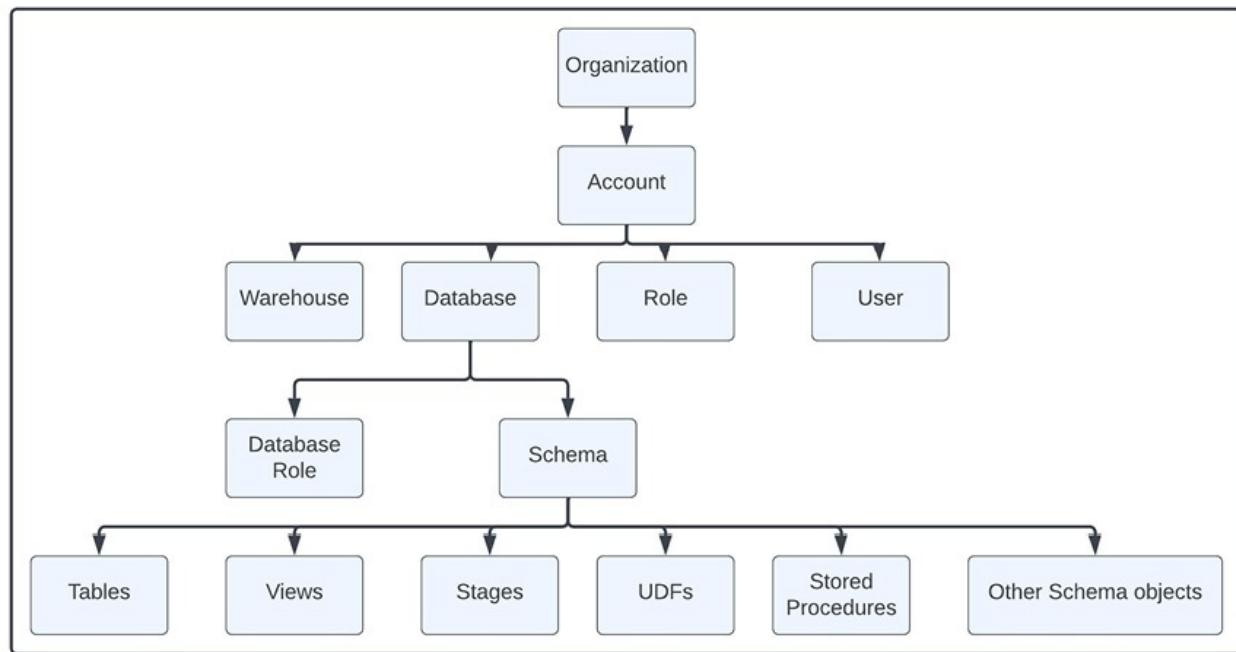


Figure 7.2: Snowflake objects hierarchy

In this section, you have learnt the various roles supported in Snowflake and the securable objects hierarchy. In the next section, you will learn more about using RBAC and DAC, along with creating custom roles and using them.

Understanding RBAC

As you know, Snowflake recommends using **Role Based Access Control (RBAC)**. As part of RBAC implementation and setup, you can create custom roles and onboard users with assigned privileges using these custom roles. You can refer to [Figure 7.1](#), which represents a role that can be assigned to a user as

well as to a role. You can create one or more custom roles to support your business and application needs.

You can also use system-defined or the default roles of Snowflake and assign them to the users directly. This role assignment depends on the type of resources and accesses required to perform the set of operations. In this section, you will primarily learn about the custom role use cases and defining the roles as per business and application requirements.

Consider a scenario where you are working on designing a unified data platform, and you have a set of applications and users to be onboarded to the platform. Following are some of the application teams and corresponding users:

Teams	Name	Details of access required
Development team	APP_DEV	Development team needs to read, write, and manage access to the Snowflake objects at database level in development region or environment.
QA Team	QA_DEV	QA team needs read access to review and validate test cases. Read only access is required at the database level in QA environment.
BI Team	BI_DEV	BI team or Reporting team that is working on building reports, needs to read, write, and manage access to Snowflake objects at the database level.
Data Science Team	DS_DEV	This is the team that works on performing ML POCs and ML development in DEV environment.

Table 7.1: RBAC Use case and Users

As you have learnt in the earlier section on Snowflake, secure objects, access privileges, and hierarchy need to be set up at a defined level – account or database or schema, etc. You have to consider the mentioned teams, users, and their requirements to grant them necessary privileges. You can follow the below steps to derive the privileges:

- Identify the user team: DEV, BI, QA and data science.
- Identify the privileges needed: read, write, manage, and so on.
- Identify the boundaries of accesses: database, account, schema, and so on.
- Identify the environments: DEV, QA.

Now, you can consider the users, teams, environments, and boundaries to design the custom roles. In the given scenario, you can design below custom roles and use Snowflake objects as shown:

Type	Objects	Action needed
Snowflake Objects	Databases Schema DB objects like – tables, views, stages, and so on.	Dev team needs full access to the snowflake objects. QA team needs read only. DS team needs read, write and manage objects.
Resources needed	Warehouses Roles	Create a warehouse and grant usage to the teams. Create custom roles to be assigned to the users in these teams.
Environments	DEV QA	Manage environments at the database level. For example, DEV database and QA database. DEV consists of all development efforts and DEV accesses. QA consists of all QA loads and read only to the teams.
Users	DEV team QA team DS team	

Table 7.2: Custom users and objects

You can create the custom roles with privileges as shown in the following figure:

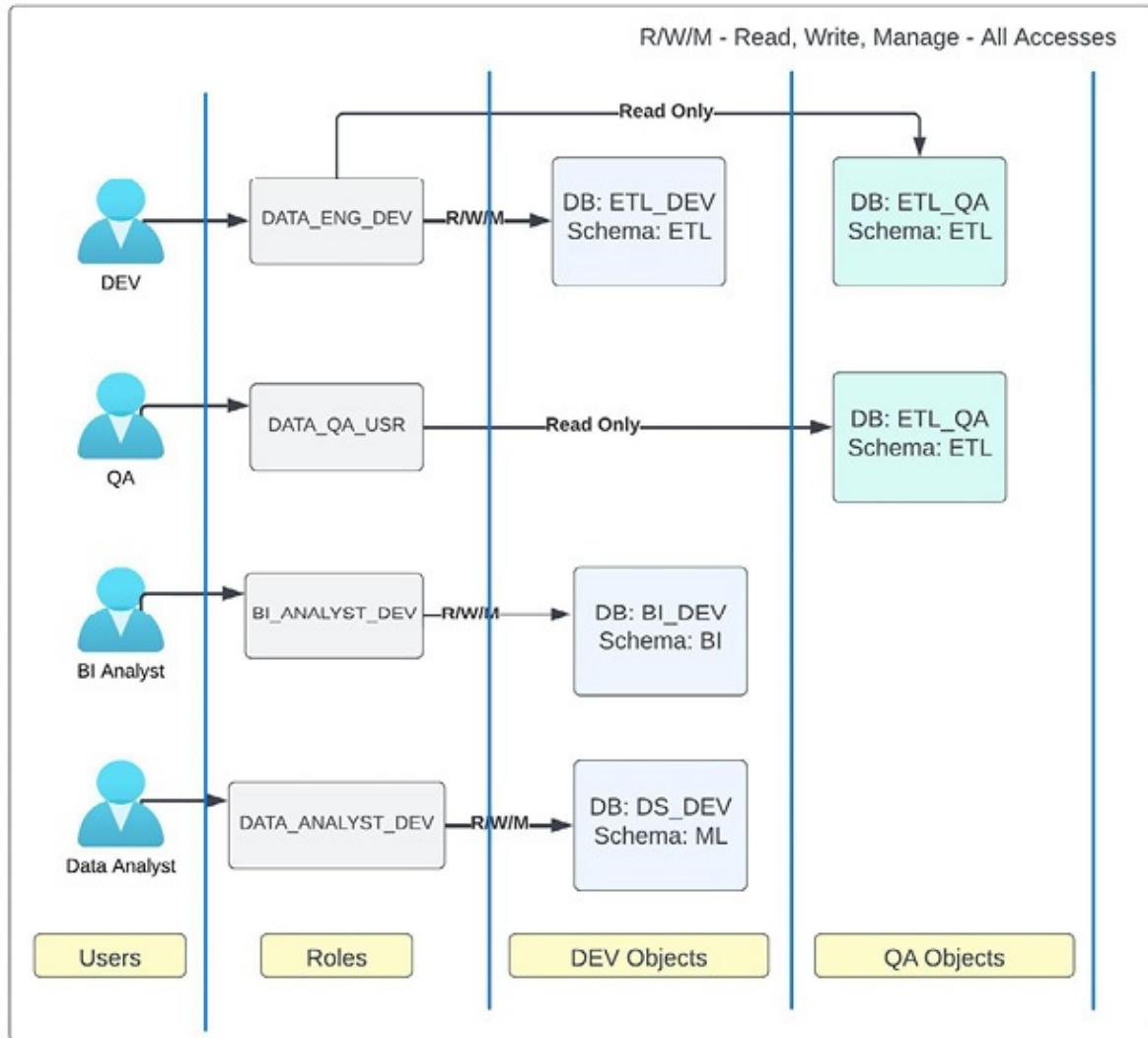


Figure 7.3: Custom role creation use case

Now, your roles are defined, and you know the set of privileges required for each user. You can start onboarding users and assign roles using SQL commands. Refer to the following list of commands to create roles, users and assign permissions to them:

- **CREATE USER** command to create users.
- **CREATE ROLE** command to create roles.
- **GRANT ROLE** to user to grant privileges.
- **GRANT USAGE, GRANT OWNERSHIP** to roles.

You will learn more about implementing these commands in the upcoming section. The same use case will be explained along with set of SQL commands

to set these users.

Understanding default roles in Snowflake

You have learnt Snowflake default roles in the first section. There are set of system defined roles that are also referred as default roles. You can use these default roles to create custom roles and grant additional privileges as required. Following are the system roles available:

ACCOUNTADMIN

SYSADMIN

SECURITYADMIN

USERADMIN

PUBLIC

ACCOUNTADMIN is the highest role that can be used to create and manage all resources and objects within the Snowflake account. **ORGADMIN** can be used to create new Snowflake accounts as well as manage existing Snowflake accounts. These default roles also follow a hierarchy of privileges and one role granted can overwrite or overpower privileges granted with additional role to the user. This is depicted in the following figure:

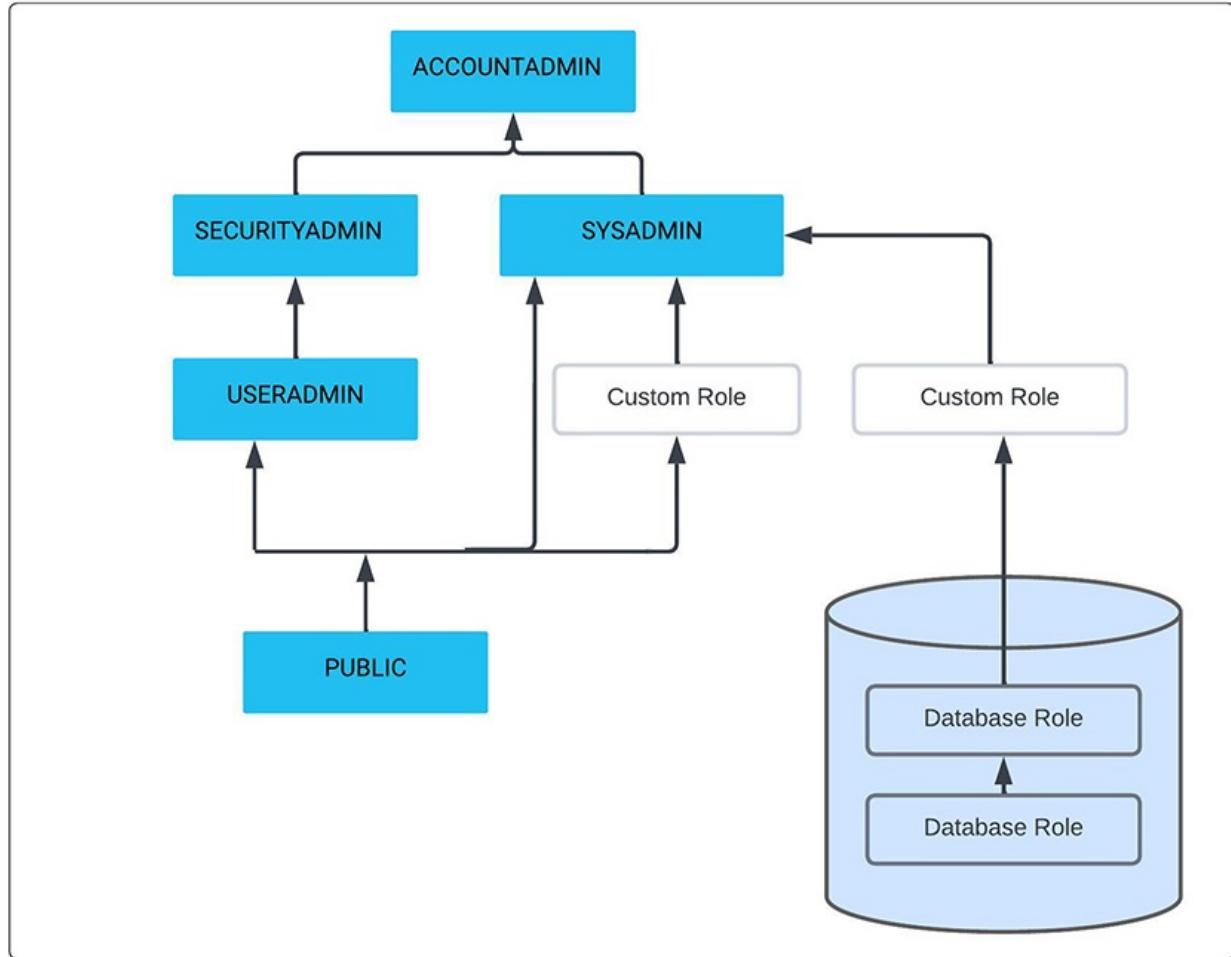


Figure 7.4: Default Role Hierarchy

As you can observe in *Figure 7.4*, **USERADMIN** privileges are encapsulated with **SECURITYADMIN** as the **SECURITYADMIN** role is associated with implementing security measures on the platform. You can create custom roles using **USERADMIN** as well as **SECURITYADMIN**. User onboarding is usually taken care of while designing the data platform or onboarding applications or users. This role may not be granted to the custom role as this is associated with user onboarding, setting up password policies, enabling MFAs for users etc. As an architect, you will never have this admin role granted to a custom role until it is a specific **ADMIN** role created for the data administrator.

SYSADMIN is the system role that allows users to create warehouses, tables, objects, Snowflake resources etc. This is the role granted to the users who need privileges to create and manage objects. This is typically granted while onboarding users in **DEV** environments. This is also one of the most common systems roles granted while creating a custom role that allows you to add

privileges to manage resources at Snowflake. You can consider this to create **DEV** user custom role, as mentioned in [Table 7.1](#). This is recommended to create a custom role first and grant **SYSADMIN** role to it while creating any users with these privileges.

ACCOUNTADMIN is the highest role in the account. This role encapsulates the privileges of all other system roles. You will need to be careful assigning **ACCOUNTADMIN** privileges to any users or roles in the system as these grant **FULL** permission on the account. In a real time scenario, there will be very few – 2 or 3 members in the Snowflake account that are granted **ACCOUNTADMIN** privileges. Only **ADMIN**s need to have this privilege and are recommended to enable **Multi Factor Authentication (MFA)**for these **ADMIN** users.

You can use the following commands to grant system roles to users:

- Create user:

```
use role useradmin;    --Set context to use
useradmin system role

create user "joy.hardwick"  -create user for joy
with default properties

default_warehouse=dev_wh

password='Demo@2023' must_change_password = true;
```

- Grant a system role to the new user created:

```
use role useradmin;

grant role sysadmin to user "joy.hardwick"
```

Now, you have learned about system users and using them to assign to users. You will learn more about custom roles and using them in the next section.

[Implementing RBAC with Snowflake](#)

You have learned about RBAC and how roles are used to onboard users in Snowflake platform. This section helps you learn more about the commands to create, manage, and assign roles as part of the access control setup. You will learn with real time use case shared in the above section. Refer to [Table 7.1](#) and [Table 7.2](#) to get more details on the workloads, users and privileges required on

Snowflake objects, warehouses, and so on.

You can split the implementation setup in the following steps:

1. Creating Snowflake resources.
2. Create custom roles.
3. Granting access to the custom roles.
4. Create users (user onboarding).
5. Assign custom roles to the users.
6. Using the custom roles.
7. Validating the access privileges.

These steps are explained in detail in upcoming parts of this section.

Step 1: Creating Snowflake resources

Refer to the following table to list down the objects and resources that need to be created:

Type	Objects	Action needed
Snowflake Objects	Databases Schema DB objects like – tables, views, stages etc.	Dev team needs full access to the snowflake objects. QA team needs Read only. DS team needs read, write and manage objects.
Resources needed	Warehouses Roles	Create warehouse and grant usage to the teams. Create custom roles to be assigned to the users in these teams.
Environments	DEV QA	Manage environments at the database level. For example, DEV database and QA database. DEV consists of all development efforts and DEV accesses. QA consists of all QA loads and read only to the teams. Databases that represent each environment and underlying schemas
Users	DEV team QA team DS team	Create users

Table 7.3: Custom role use case details

Create environments

Consider the above scenario use case and the following assumptions:

- There is a single Snowflake account.
- Different environments are maintained at a database level.
- Users to be onboarded as per the given use case and privileges.

Create DEV databases and schemas

To create DEV databases and schemas:

```
Use role SYSADMIN;  
Create database DEMO_APP_DEV;  
Use database DEMO_APP_DEV;  
Create schema ETL_DEV;  
Create schema TXM_DEV;  
Create schema TGT_DEV;
```

Create QA databases and schemas

To create QA databases and schemas:

```
Use role SYSADMIN;  
Create database DEMO_APP_QA;  
Use database DEMO_APP_QA;  
Create schema ETL_QA;  
Create schema TXM_QA;  
Create schema TGT_QA;
```

Create BI databases and schemas

To create BI databases and schemas:

```
Use role SYSADMIN;  
Create database DEMO_BI_DEV;  
Use database DEMO_BI_DEV;
```

```
Create schema BI_DEV;
```

Create ML databases and schemas

To create ML databases and schemas:

```
Use role SYSADMIN;
```

```
Create database DEMO_ML_DEV;
```

```
Use database DEMO_ML_DEV;
```

```
Create schema ML_DEV;
```

Create warehouses for users and use cases

To create warehouses for users and use cases:

```
Use role SYSADMIN;
```

```
CREATE OR REPLACE WAREHOUSE app_dev_wh  
WAREHOUSE_SIZE=SMALL INITIALLY_SUSPENDED=TRUE;
```

```
CREATE OR REPLACE WAREHOUSE bi_dev_wh  
WAREHOUSE_SIZE=SMALL INITIALLY_SUSPENDED=TRUE;
```

```
CREATE OR REPLACE WAREHOUSE ml_dev_wh  
WAREHOUSE_SIZE=MEDIUM INITIALLY_SUSPENDED=TRUE;
```

```
CREATE OR REPLACE WAREHOUSE app_qa_wh  
WAREHOUSE_SIZE=MEDIUM INITIALLY_SUSPENDED=TRUE;
```

Step 2: Create custom roles

You can create custom roles using following commands.

Create DEV custom role

Use the following command to create DEV custom role:

```
Use role USERADMIN; --you can also use SECURITYADMIN  
or ACCOUNTADMIN
```

```
Create role DATA_ENG_DEV;
```

```
Create role DATA_QA_USR;
```

```
Create role BI_ANALYST_DEV;  
Create role ML_ANALYST_DEV;
```

Step 3: Granting access to the custom roles

You can grant required privileges to the custom roles using the following commands.

Grant access of DATABASES to custom role

Use the following command to grant access of DATABASE to custom role:

```
Use role SYSADMIN;  
grant all privileges on database DEMO_APP_DEV to role  
DATA_ENG_DEV;  
grant all privileges on database DEMO_BI_DEV to role  
BI_ANALYST_DEV;  
grant all privileges on database DEMO_ML_DEV to role  
ML_ANALYST_DEV;
```

Grant access to WAREHOUSES to the custom role

Use the following command to grant access to WAREHOUSE to the custom role:

```
Use role SYSADMIN;  
grant usage on warehouse app_dev_wh to role  
DATA_ENG_DEV;  
grant usage on warehouse bi_dev_wh to role  
BI_ANALYST_DEV;  
grant usage on warehouse ml_dev_wh to role  
ML_ANALYST_DEV;  
grant usage on warehouse app_qa_wh to role  
DATA_QA_USR;
```

Grant access to QA custom role

Use the following command to grant access to QA custom role:

```
Use role SYSADMIN;

grant select on all tables in schema ETL_QA to role
DATA_QA_USR;

grant select on all tables in schema TXM_QA to role
DATA_QA_USR;

grant select on all tables in schema TGT_QA to role
DATA_QA_USR;
```

Grant access to DEV custom role in QA environment

Use the following command to grant access to DEV custom role in QA environment:

```
Use role SYSADMIN;

grant select on all tables in schema ETL_QA to role
DATA_ENG_DEV;

grant select on all tables in schema ETL_QA to role
BI_ANALYST_DEV;

grant select on all tables in schema ETL_QA to role
ML_ANALYST_DEV;
```

Step 4: Create users (user onboarding)

You have roles, warehouses, databases and schemas set, as well as assigned privileges to the custom role created. Now, you are all set to onboard users and assign them the required roles. You can use the given set of statements below to create users and assign roles.

Create users

Use the following command to create users:

```
Use role useradmin;

create user "joy.hardwick" /* dev user */
    default_warehouse=app_dev_wh password='Demo@2023'
    must_change_password = true;
```

```
create user "claire.moorie"
    default_warehouse=app_dev_wh password='Demo@2023'
must_change_password = true;
create user "Natalie.p" /* BI user */
    default_warehouse=bi_dev_wh password='Demo@2023'
must_change_password = true;
create user "Robert.patrick" /* data scientist */
    default_warehouse=ml_dev_wh password='Demo@2023'
must_change_password = true;
create user "joe.cione" /* data scientist */
    default_warehouse=ml_dev_wh password='Demo@2023'
must_change_password = true;
```

Step 5: Assign custom roles to the users

You can use grant statements to assign custom roles created to the users onboarded in step 4. Refer to the following commands to be used.

Assign custom roles to the users

Use the following command to assign custom roles to the users:

```
use role useradmin;
grant role DATA_ENG_DEV to user "joy.hardwick" ;
grant role DATA_ENG_DEV to user "claire.moorie";
grant role BI_ANALYST_DEV to user "Natalie.p";
grant role ML_ANALYST_DEV to user "Robert.patrick";
grant role ML_ANALYST_DEV to user "joe.cione";
```

Step 6: Using the custom roles

Now, you have users onboarded and assigned custom roles as per their business requirements and job roles. You can pretend to be one of the users and check the

access granted using the following SQL commands.

Using custom roles created

Use the following command to use the created custom roles:

```
use role DATA_ENG_DEV;  
use database DEMO_APP_DEV;  
use schema ETL_DEV;
```

Testing and validating access of the custom roles created

To test and validate the access of the custom roles created:

```
use role DATA_ENG_DEV;  
use database DEMO_APP_DEV;  
use schema ETL_DEV;  
use app_dev_wh;
```

```
--create sample table to test the CREATE privileges  
create test_table (col1 int, col2 string);  
insert into test_table values (1, 'one');
```

You get a table created and record inserted message in case of a successful operation. In case of access issues, you will get insufficient privileges.

Step 7: Validating the access privileges

To test the privilege issues, you can use the QA database and try to create an object in QA as follows.

Using custom roles created

Use the following command:

```
use role DATA_ENG_DEV;  
use database DEMO_APP_QA;
```

```
use schema ETL_QA;

--create sample table to test the CREATE privileges
create test_table (col1 int, col2 string);
insert into test_table values (1, 'one');
```

The same set of statements is used in QA to test the privileges **DEV** custom role assigned. The moment you run these SQL statements, you will get an insufficient privileges message on the QA database as the DEV user is granted READ-only access to the QA database.

In this section, you have learned to create custom roles, assign privileges, create warehouses, Snowflake resources, and assign permissions to the custom roles. In the next section, you will learn about the access hierarchy and how this impacts the multiple roles assigned to a single user.

Understanding access hierarchy

Snowflake recommends RBAC implementation. A single user can be granted multiple roles. One role can be granted with multiple permissions. A user with multiple roles can roll up the permissions and have them all collated together. If one role overwrites the permission of another role, then the user gets the access permissions as per privilege or access hierarchy.

By now you know, **ACCOUNTADMIN** is the superuser or the role with the highest level of privileges at the account level. If you assign **USERADMIN** and **SYSADMIN** to the same user, then the user has privileges of both – user management and resource management. If you assign a user **SYSADMIN** as well as **ACCOUNTADMIN** then the privileges are rolled up and **ACCOUNTADMIN** privileges take precedence over **SYSADMIN**.

Refer to [**Figure 7.5**](#). This explains the use case where a role with specific privileges is granted to the same user: **DEV_USER**. In this case, the **DEV_USER** has all accesses as part of *Role A*, *Role B* and *Role C*:

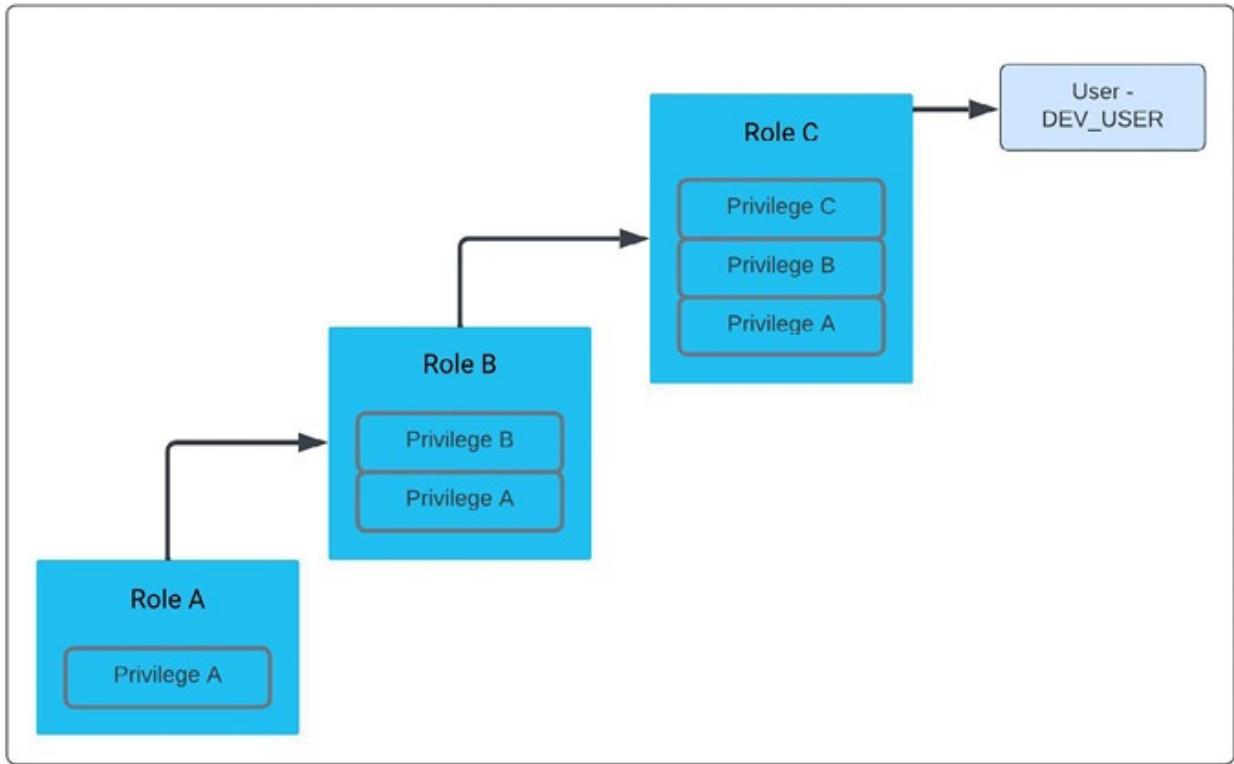


Figure 7.5: Role and Privileges sharing

You can observe the role and privileges consolidation if you grant multiple privileges to the same user. This is also true about system defined functions.

You can also refer to [Figure 7.4](#). This represents the hierarchy of system defined functions. You can create a database role and assign it to the custom role. This custom role gets a **SYSADMIN** system defined role assigned – this allows the role to perform the DB operations as well as extended permissions as part of the **SYSADMIN** role.

Remember, you can grant multiple roles to a single user. You can also grant custom roles, system defined roles, and object specific permissions to the same user or role. It is always recommended to manage the users at role level instead of assigning object specific permissions to the individual users. Create a custom role – assign all roles and required privileges and use this to onboard required set of users.

You can use DDL commands to create, alter, and delete users and roles from the Snowflake account. The next section helps you to learn more about these DDL commands and their usage.

Managing users and roles

You have learned a set of commands to create Snowflake objects, warehouses, roles, and users in the earlier section – *Implementing RBAC*. You can create, alter, and remove the roles. Users are created as part of custom implementations and user onboarding.

User commands

These are a set of DDL commands that are used to **CREATE**, **ALTER**, **DROP**, **DESCRIBE** users and roles created.

CREATE command

This is used to create or replace existing users. Only **USERADMIN** or **ACCOUNTADMIN** can create users and roles. Users with create permissions can also create users. However, this may not be the real time scenario of implementation:

```
CREATE role dev_role;  
CREATE USER demo_user PASSWORD='demo123' DEFAULT_ROLE  
= dev_role MUST_CHANGE_PASSWORD = TRUE;
```

DROP command

This is used to remove the specified user from the system:

```
DROP USER demo_user;  
DROP role dev_role;
```

ALTER command

This is used to change or modify user information. **ADMIN** can use this command to alter any of the user's properties. For example, you create a user who is facing a login issue due to their password. You can reset the password using the **ALTER** command:

```
ALTER USER demo_user SET PASSWORD = 'Demo2023'  
MUST_CHANGE_PASSWORD = TRUE;
```

DESCRIBE command

This is used to describe the properties of the user:

```
DESC demo_user;
```

SHOW command

This is used to list down all users in the account:

```
SHOW USERS;
```

```
SHOW USERS like '%DEV%';
```

You can use these SQL commands to manage users in your account. You can follow the practice exercises shared and use trial account setup at the beginning to complete the labs.

Practical

You can follow the given instructions to create Snowflake roles, users and manage users:

- Consider the following use case and setup the required environments, users, custom roles and grant required permissions to the role. You can refer to the section – *Implementing RBAC* to complete the lab and refer to the syntax shared in this section.

Category	Type	Details
Environment	DEV and PROD	Setup environment that supports DEV and PROD workloads.
Warehouses	DEV and PROD	Create warehouses as per DEV and PROD workloads.
Workloads	DEV – Application development – ETL or data engineering Analytics loads PROD – ETL Pipelines Reporting & BI loads	Create warehouses to support the workloads.
Roles	DEV Role for data engineering and analytical loads PROD read only for ETL and BI loads	Create custom roles for these workloads in DEV and PROD environment.
Snowflake Resources	Create required Snowflake objects – databases, schema to support the environment access permissions.	Create databases for DEV and PROD workloads.

Create Users	Create sample users for development, BI development, PROD read only user.	Create users for DEV and PROD.
Role assignment	Grant privileges for DEV and PROD	Assign DEV and PROD custom roles to the users created

Table 7.5: Lab exercise use case details

Conclusion

This chapter provided a summary of Snowflake's access control policies, support to RBAC, and recommendations to onboard applications and users. This chapter helps you to understand custom role creation, allocating and applying them to the business applications. This also helps you to learn Snowflake account usage and tracking for roles. You will learn more about user management, user onboarding, and resource allocations – warehouses, Snowflake objects etc. You will learn more about Snowflake's features and offerings as you go along with the next chapters in this book.

In the next chapter, you will learn about Snowflake's data protection and recovery. You will learn more about time-travel, failsafe, and data encryption.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake supports Role Based Access Control.
- Create custom roles to be aligned and assigned for dedicated workloads.
- You can manage users, workloads, and snowflake resources in a customized role.
- Snowflake roles can be created once and assigned to the users during onboarding.
- Snowflake roles follow a hierarchy and can overwrite access privileges depending on the roles granted to the users.

Questions

1. How is user management done in Snowflake?
2. How custom roles can be created? Can you create a new role or access privileges than default roles?

3. How is Snowflake's default role used to perform system security activities?
4. How can you onboard applications, workloads, and users aligned with custom roles defined?
5. Can you track account usage at workload, application and user levels?

Multiple choice questions

- 1. What are the default roles in Snowflake? (Select all applicable)**
 - ACCOUNTADMIN
 - SECADMIN
 - SYSADMIN
 - USRADMIN
- 2. What is the topmost role in Snowflake?**
 - SYSADMIN
 - SECURITYADMIN
 - ACCOUNTADMIN
 - USERADMIN
- 3. What is the role used to create users in Snowflake?**
 - SYSADMIN
 - SECURITYADMIN
 - ACCOUNTADMIN
 - USERADMIN
- 4. What is the role used to create resources in Snowflake?**
 - SYSADMIN
 - SECURITYADMIN

- c. ACCOUNTADMIN
 - d. USERADMIN
5. **What is the role used to create security policies in Snowflake?**
- a. SYSADMIN
 - b. SECURITYADMIN
 - c. ACCOUNTADMIN
 - d. USERADMIN

Answers

1	a, c
2	c
3	d, c
4	a, c
5	b, c

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 8

Data Protection and Recovery

Introduction

Data governance has a set of pillars that guide users to design and implement appropriate rules and policies while designing the data platform. Access control is one of the critical elements of data governance implementation that you have learned in [*Chapter 7, Access Control and Managing Users, Roles*](#). This chapter covers and focuses on the next important part of data governance and data security implementation. Data protection and recovery focuses more on implementing data security, that is, data encryption at rest and in transit. This also includes fallback and recovery. Snowflake offers support and features that can be used for data recovery and fallback.

This chapter covers scenarios that help you understand data protection – dynamic data masking, data security, data encryption, as well as data recovery methods like time travel and failsafe. Snowflake offers features not only to recover the data dropped but also objects dropped accidentally. You can **UNDROP** the objects that are dropped accidentally. This is one of the unique features Snowflake offers for data recovery.

The chapter includes sample examples and lab questions to help you learn these features with labs. There are questions to evaluate your understanding of the concepts explained in this chapter.

Structure

This chapter consists of the following topics:

- What is data protection?
- Implementing dynamic masking
- Understanding data recovery options
- Implementing time travel
- Implementing data replication

Objectives

By the end of this chapter, you will be able to understand Snowflake features that extend support to data governance – protection, security, and recovery options. This chapter helps you to understand and create dynamic masking as part of data security. It also helps to understand and use Snowflake data recovery features and differentiate between time-travel and failsafe. You will be able to implement them using the trial account setup in [*Chapter 1, Getting Started with Snowflake.*](#)

What is data protection?

Data protection is a process to secure data from unauthorized access, exposure and loss. You can implement security strategies and processes as part of data protection. Unauthorized access to data is referred to as data breach. Any type of information loss due to assets, data transmission, or any loss over a network is referred to as data loss. Data protection policies and strategies are used to prevent data loss and breach.

There are two principles of data protection – data availability and data management. Data availability ensures that data is available as part of the business continuity plan. This also consists of data recovery and retention to ensure data is available as and when needed.

Data management consists of data lifecycle and information lifecycle management. Data lifecycle management takes care of all phases of the data – data creation, storage, processing, analysis, archival, deletion, retention, and so on. Information lifecycle management takes care of information that is related to the data cataloging as well as storing information related to the datasets.

Data protection is an important pillar of the data platform design and one of the critical aspects of data governance implementation. This takes care of data loss, preventing any access to the data. This also includes monitoring and protecting

data. There are various methods that support data protection implementation – data recovery, data replication, data masking, data encryption, and data access control. Following are the Snowflake features that extend this support:

- **Data access control:** It is an authentication and authorization setup to ensure you grant only authorized users and grants to the users based on the custom roles. You can refer to [*Chapter 7: Access Control and Managing Users Roles*](#), for more details on access control using RBAC policies.
- **Data encryption:** Data stored in the Snowflake storage is encrypted automatically using AES-256. All data stored in the internal stage as part of loading and unloading is encrypted.
- **Data masking:** Data security feature where you mask the entire data or tokenize the data to be exposed to all users. Only selected users can see the original form of the data, and for the rest of the users, data is stored in masked format. Snowflake supports dynamic masking.
- **Data recovery:** Data recovery feature offers support to recover the data in the form of historical data that is being modified or deleted. Snowflake extends support using Time-travel and Failsafe. You will learn more about these features and their implementation in the upcoming section of this chapter.
- **Data replication:** Data is replicated to create an additional copy to restore data in case of data unavailability due to any disaster or system failures. You will learn about the data replication feature of Snowflake in the subsequent section of this chapter.

Data protection is an essential implementation of data platform. You will learn various data protection components and Snowflake implementation in upcoming sections of this chapter.

Implementing dynamic masking

Dynamic masking is an enterprise edition feature. This is used to implement column-level security in the form of masking policies. These policies are used to mask the data selectively, and plain text data gets masked. These policies are schema-level objects and can be created once. Dynamic masking can be implemented only on tables and views. These are applied to the run time whenever the mapped columns appear.

Implementation commands

Snowflake supports DDL commands to create, drop, alter, show, and describe masking policy. This section covers DDL commands.

Create masking policy

This is used to create a new policy in specified schema or used to replace policy in the schema.

Syntax:

```
CREATE [ OR REPLACE ] MASKING POLICY [ IF NOT EXISTS ]
<name> AS
( <arg_name_to_mask> <arg_type_to_mask> [ , <arg_1>
<arg_type_1> ... ] )
RETURNS <arg_type_to_mask> -> <expression_on_arg_name>
[ COMMENT = '<string_literal>' ]
```

Here, name is the name of masking policy

- **arg_name_to_mask:** Specifies input columns to be masked.
- **arg_type_to_mask:** Specifies input column data type to be masked.
- **arg_type_to_mask:** Specifies expression used to mask column data/value.
- **Comment:** Specifies details of the policy.

Usage:

Masking policies can be setup as normal policy, as well as policy to be implemented on a condition.

Normal masking policy

Consider that you want to set up a masking policy for PII data stored. You have the retail database, orders, and sales schema to store data. Now, you want to protect your data from being used by various users of Snowflake account. You have an analyst role created, and users onboarded to your Snowflake platform. You do not want to show the original values and protect data from being exposed to the analysts. Only **ACCOUNTADMIN** can see the original value.

PII data to be protected includes **email**, **account_number**, **SSN**, and so on:

```
Create database retail;
create schema orders;
use database retail;
use schema orders;
--mask the email ids to *****
CREATE OR REPLACE MASKING POLICY email_policy AS (val
string) returns string ->
CASE
    WHEN current_role() IN ('ACCOUNTADMIN') THEN VAL
    ELSE '*****'
END;

--mask the SSN and account_number to NULL for
unauthorized users
CREATE OR REPLACE MASKING POLICY PII_mask AS (val
string) returns string ->
CASE
    WHEN current_role() IN ('ACCOUNTADMIN') THEN VAL
    ELSE NULL
END;

--mask to the hash values instead of static values
CREATE OR REPLACE MASKING POLICY name_mask AS (val
string) returns string ->
CASE
    WHEN current_role() IN ('ACCOUNTADMIN') THEN VAL
```

```
ELSE sha2(VAL)
END;
```

You can also create a UDF to mask the values and use it in place of the hashing function to mask the column value.

Conditional masking policy

The earlier masking policies mentioned in normal masking are the static ones that check for the user and role before masking value. In conditional masking, you can check on the current role or visibility column value for the masking value.

Examples:

- Set the masking policy based on the role of email value and value in the visibility column in public:

```
create masking policy mail_visibility as
(customer_email varchar, visibility string) returns
varchar ->
case
    when current_role() = 'ACCOUNTADMIN' then
email
    when visibility = 'Public' then email
    else '***MASKED***'
end;
```

- Viewing email address or viewing only email domain or showing fixed masking value:

```
create masking policy mail_policy as
(Val string) returns string ->
case
    when current_role() = 'ACCOUNTADMIN' then Val
```

```

    when current_role() = 'ANALYST' then
regexp_replace(val,'.+@\w+','*****@')
    else '*****'
end;

```

Alter masking policy

This command is used to alter the masking policy defined to change the masking rule. Following is the syntax of the command:

```
ALTER MASKING POLICY [ IF EXISTS ] <name> RENAME TO
<new_name>
```

```
ALTER MASKING POLICY [ IF EXISTS ] <name> SET BODY ->
<expression_on_arg_name>
```

```
ALTER MASKING POLICY [ IF EXISTS ] <name> SET TAG
<tag_name> = '<tag_value>' [ , <tag_name> =
'<tag_value>' ... ]
```

```
ALTER MASKING POLICY [ IF EXISTS ] <name> UNSET TAG
<tag_name> [ , <tag_name> ... ]
```

```
ALTER MASKING POLICY [ IF EXISTS ] <name> SET COMMENT
= '<string_literal>'
```

```
ALTER MASKING POLICY [ IF EXISTS ] <name> UNSET
COMMENT
```

Here:

- **BODY:** Policy rules
- **COMMENT:** Set comment to the policy or remove the comment
- **TAG:** Used to set the **TAG** value or unset the value

Examples:

```
ALTER MASKING POLICY email_visibility SET BODY ->
CASE
WHEN current_role() IN ('ANALYST') THEN VAL
```

```
ELSE sha2(val, 512)  
END;
```

Drop masking policy

This is used to drop the policy.

Syntax:

```
DROP MASKING POLICY <name>;
```

Example:

```
DROP MASKING POLICY ssn_mask;
```

Show masking policy

This is used to list the existing policies in the database and schema as well as account.

Syntax:

```
SHOW MASKING POLICIES IN SCHEMA retail.sales;
```

Describe masking policy

This command is used to describe details of the masking policies.

Syntax:

```
DESC MASKING POLICY ssn_mask;
```

These are the DDL commands used to define masking policies. The following section covers using dynamic masking and applying policies to the databases, tables, views.

Using dynamic masking

This section covers details of defining masking policies, applying them to the database objects and using the rules to protect data being exposed. Once policies are defined, these can be applied and used by following the given steps:

1. Create a custom role to be assigned to the security officer.
2. Grant masking policy privileges to the custom role or security officer.
3. Grant custom roles to the users.

4. Use custom roles to define masking policies.
5. Apply masking policies to the database and objects.
6. Execute queries on the tables to validate the policy applied:
 - a. Snowflake rewrites the query that applies masking policies to the table columns.
 - b. This occurs at every place wherever the column is being used.
 - c. Users see masked data whenever they execute queries against tables and columns defined on the tables.

You can use a set of SQL commands to perform each step defined above as part of policy implementation. You will learn implementation using SQL commands and reference use case in this section. Below are the steps and commands to be used.

Step 1: Create custom role

You are an **ACCOUNTADMIN** and designing your data platform. As an architect, you need to define the custom roles to support your platform design, application development etc. You need to create a set of custom roles to manage Snowflake usage, such as warehouse, storage etc. In this use case, consider creating a role to be assigned and an onboard security admin. Security admin is the role that defines security, masking policies, and applies to the objects:

```
Use role ACCOUNTADMIN;
/*--create security role--*/
Create role security_admin;
/*--Grant permissions to the custom role --*/
GRANT CREATE MASKING POLICY on SCHEMA retail.orders to
ROLE security_admin;

GRANT APPLY MASKING POLICY on ACCOUNT to ROLE
security_admin;
```

Step 2: Grant custom role to a user

You can use the **GRANT** command to grant privileges to the user.

```
GRANT ROLE security_admin TO USER john;
```

Step 3: Create masking policy

You can use the custom role to create masking policies:

```
/*--use new custom role--*/
use role security_admin;
/*--use role and create policy--*/
CREATE OR REPLACE MASKING POLICY ssn_mask AS (val
string) returns string ->
CASE
    WHEN current_role() IN ('ACCOUNTADMIN') THEN VAL
    ELSE NULL
END;
/*--grant policy to table owner role--*/
GRANT APPLY ON MASKING POLICY ssn_mask to ROLE
table_owner;
```

Step 4: Apply masking policy to the database objects

You can apply the policies while creating objects. You can also apply the policies using the **ALTER** command. Policy can be applied to a table as well as view:

```
-- apply masking policy to a table column
ALTER TABLE IF EXISTS customer_details MODIFY COLUMN
cust_email SET MASKING POLICY email_policy;
```

```
-- apply the masking policy to a view column
ALTER VIEW v_customer_details MODIFY COLUMN cust_email
```

```
SET MASKING POLICY email_policy;
```

Step 5: Query data and validate masking policies

You can validate the data masked by using two different roles. You can use **ACCOUNTADMIN** to run the same query on the table and use another role to run the same query. Compare the result between two queries:

```
-- using the ACCOUNTADMIN role
```

```
USE ROLE ACCOUNTADMIN;
```

```
SELECT customer_email FROM customer_details; -- should  
see plain text value
```

```
-- using the ANALYST role
```

```
USE ROLE ANALYST;
```

```
SELECT customer_email FROM customer_details -- should  
see full data mask
```

You can create masking policies, grant them to the custom role, and apply them to the tables. You can follow the preceding steps and set up masking rules as per your business requirements. The next section covers details of the data recovery and options.

Understanding data recovery options

Snowflake data protection includes data recovery, data security, and data recovery as part of **Disaster Recovery (DR)** and business continuity. Snowflake offers two features for data recovery – Time travel and Failsafe. You will learn more about data replication in data recovery as part of DR in the upcoming section. This section focuses more on understanding data recovery using Time-Travel and Failsafe. The subsequent section covers details of the implementation of these two features.

Understanding time travel

The time travel feature allows users to access historical data at any point in time. The historical data consists of any data that has been changed, modified, added

or deleted. You can access the deleted data as well using this feature. Time travel offers the following features:

- Restore any deleted objects (accidentally or intentionally dropped) – databases, schemas as well as tables.
- Data backup as well as data duplication based on the key points or events in the past.
- Analyze data usage or manipulation over a specific period of time.

Time travel offers data history of up to 90 days, depending on the type of objects and edition of Snowflake. Different time travel durations or periods are supported for different types of tables like transient, temporary, or permanent tables. Not all operations are allowed with this feature. You can restore data using **SELECT** or **CLONE** commands at a given point in time up to time travel retention. You can undo the objects deleted or dropped accidentally using **UNDROP** commands. You will learn more about these commands in the next section. Once you have data restored into a database object, you can use it as a standard database object and run all types of standard operations. You cannot combine standard operations with the time travel command. The following figure refers to this command:

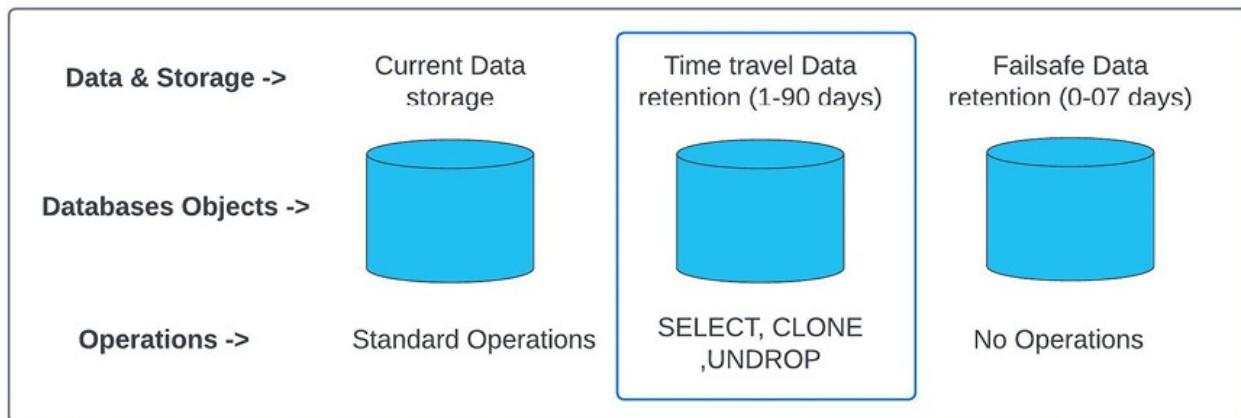


Figure 8.1: Time travel (Continuous Data Protection)

Following are the operations supported in Time travel:

- Select or query historical data up to the retention period.
 - Create clones of databases or schemas entirely at a specific point before or after an event using event timestamp.

- Restore objects that have been dropped or deleted.

Time travel SQL extensions

Time travel SQL extensions can be used to identify the time or period to restore or access history data:

- **AT | BEFORE:** This is a clause that identifies the exact time or period to restore the data using **CLONE** or query the data using **SELECT**. This accepts the following parameters to identify the event or point in time:
 - Timestamp
 - **Offset:** Time difference in seconds from current timestamp. This can be used to get history data from current time.
 - **Statement:** You can give statement or Query ID to identify the operation that has changed the data or deleted the objects.
- **UNDROP:** This is used to restore deleted or dropped objects:
 - **UNDROP DATABASE**
 - **UNDROP SCHEMA**
 - **UNDROP TABLE**

Time travel data retention

Data retention specifies the number of days data can be stored and maintained as part of historical data. This is enabled for 1 day (24 hours) by default to all Snowflake accounts. Snowflake data retention period varies based on the editions:

- **Snowflake Standard Edition:** Data retention can be set to 0 in case no retention is required. The default is 1 day.
- **Enterprise Edition and higher:** Data retention is based on the type of objects as well:
 - Transient databases, schemas, and tables can be set to 0 or use the default 1 day data retention
 - Temporary tables can be set to 0 days retention or use default 1 day retention.

- Permanent tables, schemas, and databases can have upto 90 days data retention. You can set any period, depending on your requirements, to maintain historical data between 0 to 90 days.

Note: For any objects with 0 data retention, time travel is disabled as there is no history to be maintained.

Date retention period can be specified and used while updating time travel. You can use the following commands:

- **DATA_RETENTION_TIME_IN_DAYS:** This is used to set the default retention period of the Snowflake account. Users with **ACCOUNTADMIN** role can set this up.
- **DATA_RETENTION_TIME_IN_DAYS:** This can also be used while creating database, schema and tables to overwrite the retention set at account level.
- Databases, schemas and tables retention period can be changed at any time.
- **MIN_DATA_RETENTION_TIME_IN_DAYS:** This is used to set the minimum retention period at the account level. This parameter is separate from **DATA_RETENTION_TIME_IN_DAYS** and cannot overwrite the overall retention time at the account level.

Time travel setup

Time travel is set automatically. You do not need to set it up separately. Time travel duration can be set at the account level as well as the object level. You cannot remove the time travel at the account level, but you can remove it for an object using 0-time travel days to maintain history.

Time travel storage cost

Data retention as part of Time travel contributes to the overall storage cost. Time travel storage cost is calculated for every 24 hours from the time it has changed. The storage cost is dependent on the time travel setup, depending on the type of table.

Snowflake maintains the storage based on the amount of data changed or modified. This helps to minimize the overall storage cost. The overall storage usage is calculated based on the % of the table modified. A full copy or data is maintained only if the object is dropped or deleted.

Time travel storage for temporary and transient tables

Temporary and transient tables have different storage costs. The storage cost is dependent on the following factors:

- Transient tables can have 0 to 1 day retention period
- Temporary tables can have a retention of 0 to 1 day. This retention ends as soon as the table is dropped or the session ends.
- Temporary and transient tables do not have failsafe.

The maximum storage cost incurred is only for one day. The following table represents the type of table and data retention period for time-travel and failsafe:

Type of table	Time travel	Failsafe	Min days history maintained	Max days history maintained
Standard (Permanent) table	0 to 1 (Standard edition) 0 to 90 (Enterprise edition)	7 days	7	8 (Standard Edition) 97 (Enterprise Edition)
Transient table	0 or 1 days	0 days	0	1
Temporary table	0 or 1 days	0 days	0	1

Table 8.1: Data retention based on type of table

Time travel is a data retention feature, and failsafe is an additional data recovery feature. You will learn more about failsafe in the following section.

Understanding Failsafe

The failsafe feature is separate from time travel. Time travel maintains the history of data for up to 90 days. Once data is maintained in time travel for 90 days, it is moved to fail safe to be maintained as part of disaster recovery in case of system failure or any other event. The following figure illustrates the working of failsafe:



Figure 8.2: Failsafe

Failsafe can store data for up to 7 days and this can be recovered only by Snowflake. Unlike time travel, you cannot restore the data on your own using SQL statements.

Note: Fail safe is a data recovery service offered to restore the data and this is intended to be used only when all other data recovery options are exhausted. This is not an alternate or extension to the time travel, this is meant to offer feature to store data to be restored by Snowflake.

Failsafe storage cost

Snowflake failsafe stores data for up to 7 days and this requires storage. Failsafe storage is also considered as one of the storage components that contributes to the overall data storage of an account. You can view this storage as part of usage using UI - Snowsight as well as query on metadata view. Login to Snowsight | Change role ACCOUNTADMIN | Admin | Usage | select **Storage** from the filter next to all tags:

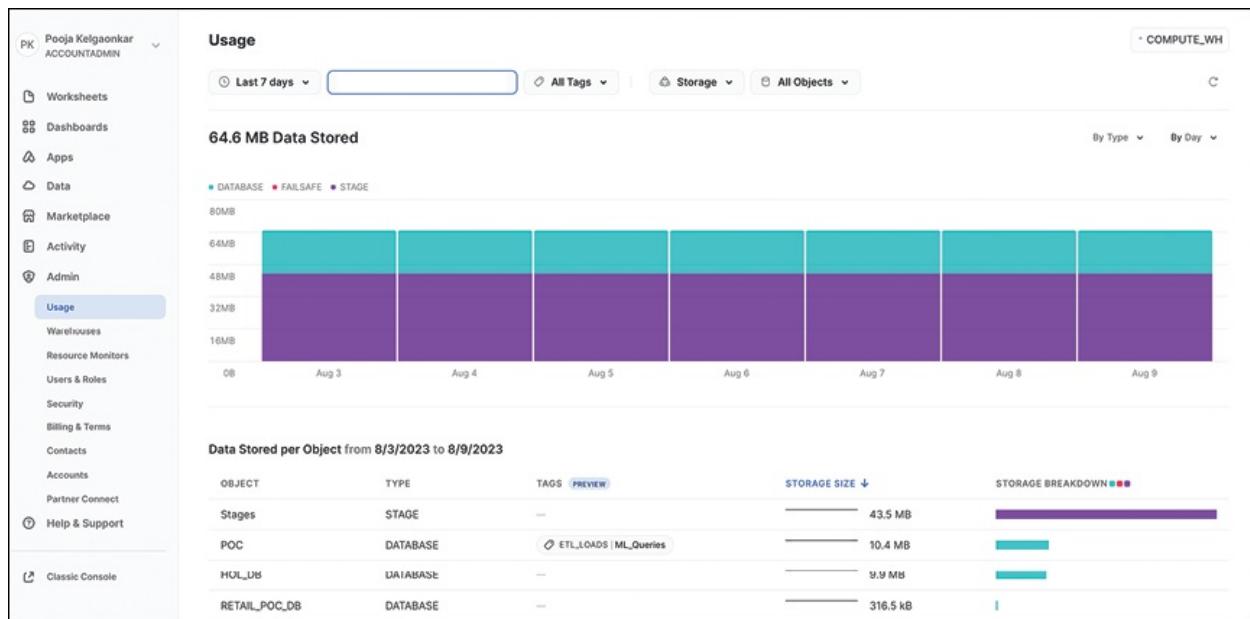


Figure 8.3: Failsafe storage view in account usage

Failsafe cannot be enabled or disabled at the account as well as object level. There are no SQL commands to access historical data, and this can only be recovered by Snowflake. You will learn more about SQL commands to be used for time travel in the following section.

Implementing time travel

Time travel can be enabled as well as disabled using SQL commands. This section covers various scenarios and using SQL commands.

Setup time travel at create table

Time travel can be set at the account level as well as at the time of object creation. The object creation parameter overrides object retention. Create a table with data retention of 90 days:

```
CREATE TABLE store_details(store_name
String, Store_number NUMBER, Store_location String,
Store_start_dt DATE, Store_end_dt DATE)
DATA_RETENTION_TIME_IN_DAYS=90;
```

Alter table to change the data retention to 30 days:

```
ALTER TABLE store_details SET
DATA_RETENTION_TIME_IN_DAYS=30;
```

Access historical data

Time travel maintains data history for a set retention period. The data can be accessed using the SELECT query and access clause. As mentioned in the above section, data can be accessed based on the timestamp, offset, and statement (query IDs).

Query to select historical data as of the date and time represented by the specified timestamp:

```
SELECT * FROM Store_details AT(TIMESTAMP => 'Mon, 07 Aug 2023 15:30:00 -0700'::timestamp_tz);
```

Query to select historical data as of 5 minutes ago:

```
SELECT * FROM Store_details AT(OFFSET => -60*5); -- OFFSET is always in the form of seconds
```

Query to select historical data before executing the query statement and any changes made by the specified statement:

```
SELECT * FROM Store_details BEFORE(STATEMENT => '2e7d0ca5-006e-33e6-b354-a8f4b29c2618');
```

Note: Data retention and historical data can be accessed only up to a specific retention period. If the **TIMESTAMP**, **OFFSET**, or **STATEMENT** mentioned in the clause is outside the retention period, then the query fails and returns an error.

Cloning objects

You can also use **AT** | **BEFORE** clause while cloning the objects using **CLONE**. You can use **CLONE** command to create or duplicate objects logically at a specified point in time.

- **CREATE TABLE** command to create a clone of a table as of the specified timestamp:

```
CREATE TABLE store_restored CLONE Store_details  
AT(TIMESTAMP => 'Mon, 07 Aug 2023 00:01:00 +0300'::timestamp_tz);
```

- **CREATE SCHEMA** command to create a clone of a schema with all its objects before 1 hour of the current time:

```
CREATE SCHEMA sales_schema_restored CLONE  
sales_schema AT(OFFSET => -3600);
```

- **CREATE DATABASE** command to create a clone of a database with all its objects as of the specified statement(query ID):

```
CREATE DATABASE sales_restored CLONE sales  
BEFORE(STATEMENT => '2e7d0ca5-006e-33e6-b354-  
a8f4b29c2618');
```

Commands to access dropped and restored objects

Whenever a table, schema, or database is dropped, it is not removed immediately from the system. It is retained for a specified retention period. If you drop an object and create a new object with the same name, it does not restore the deleted object but creates a new object with the same name. The original deleted objects can be restored using data retention commands. You can use commands to list the dropped objects and restore dropped objects using SQL commands.

List dropped objects

You can use the **SHOW** command to list down the objects that are dropped using the **HISTORY** clause. **SHOW** commands support:

- **SHOW TABLES**
- **SHOW SCHEMAS**
- **SHOW DATABASES**

List tables that are dropped:

```
SHOW TABLES HISTORY LIKE 'store%' IN  
sales.sales_schema;
```

List databases and schemas dropped:

```
SHOW SCHEMAS HISTORY IN sales;  
SHOW DATABASES HISTORY;
```

Note: These objects are available to be listed in the **SHOW** command until they exist in data retention period. These are not available to be listed after the data retention period is over.

Restore dropped objects

Dropped objects can be restored using the **UNDROP** command. This command is used to restore objects that are dropped and restored in the same state of object before dropping:

- **UNDROP TABLE**
- **UNDROP SCHEMA**
- **UNDROP DATABASE**

Restore database and schema:

```
undrop database sales;  
undrop schema sales_schema;
```

Restore table from a database:

```
undrop table store_details;
```

If you undrop the object to restore with the same name that already exists in the database schema, then the **undrop** command fails.

Reference use case

You can consider the following use case where a developer is testing a transformation pipeline and deletes tables accidentally. Now, the developer wants to check if it exists and restore the latest version or state of the table before dropping:

1. Create orders table in the retail database:

```
create database retail;  
create schema orders;  
use database retail;  
use schema orders;  
  
CREATE TABLE order_details (order_num number,  
amount number, currency string, store string,  
store_loc string)  
DATA_RETENTION_TIME_IN_DAYS=30;;
```

2. Load data to the table:

```
use database retail;  
use schema orders;  
INSERT INTO order_details VALUES (123456, 150,  
'USD', 'Macys', 'NY');  
INSERT INTO order_details VALUES (563459, 350,  
'CAD', 'Macys', 'Toronto');
```

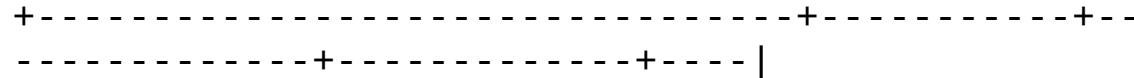
3. Drop objects:

```
use database retail;  
use schema orders;  
DROP TABLE order_details;
```

4. List the dropped object:

```
use database retail;  
use schema orders;  
SHOW TABLES HISTORY;
```

created_on	name	database_name	schema_name	kind	comment	cluster_by	rows	bytes	owner	retention_time	dropped_on
Mon, 07 Aug 2023 17:41:55 -0700	order_details		Retail	Orders					SYSADMIN	1	Fri, 11 Aug 2023 19:04:46 -0700



5. Restore objects:

```
use database retail;  
use schema orders;  
UNDROP TABLE order_details;
```

Time travel and failsafe are used to restore historical data. Note that the time travel data can be restored using SQL. However failsafe data can only be restored by Snowflake team. Apart from historical data, Snowflake also offers support for data replication. You will learn more about replication in the next section.

Implementing data replication

Data replication is the process to replicate or copy the data from one server to another server for better data availability and recovery in case the data is not available on the primary server. In typical data replication and legacy era, you might have noticed having two separate data centers which are mirror images of each other. One of the **Data Center (DC)** was set up as primary and another one as secondary or fail over DC. Usually, the primary data center carries all workloads, pipeline executions, and consumer integrations. Once data is available in primary, copy it to the secondary data center to ensure data is consistent across the two regions. In case of any failure or DR activities, the workloads are redirected to the secondary data center where data is consistent, and you can continue executing your workloads and consumer loads until the primary is restored. Once the primary is restored, it becomes the new secondary or the fail-over server in DC.

Data replication is a critical process to ensure your primary and secondary data centers are in sync and ready to switch at any time in case of a disaster. This is one of the critical parts of business continuity and is implemented for legacy cloud or cross-cloud applications. In the case of Snowflake, data replication is available for specific editions. Not all replications are available for all editions. Database and sharing replication are available to all accounts. Replication or failover and client redirect are available only for the critical and higher business accounts. You will learn more about Snowflake replication in subsequent

sections of this chapter.

Database replication

This feature supports database replication between Snowflake accounts that are in the same organization. This also supports cross region and cross cloud platform replication. This feature enables users to keep database objects as well as data in sync.

You can enable replication for existing transient as well as permanent database objects. Once you enable replication, the databases are treated as primary databases. You can have multiple databases defined as primary databases. These primary databases can be replicated to multiple accounts. You can create a secondary database or databases in the same or cross cloud or accounts within the same organizations. Note that all DML and DDL operations that run on these primary databases are replicated or refreshed periodically.

Database replication supports only database objects. The database replication does not replicate the privileges or permissions granted to existing or new objects in the databases. You can replicate other objects as part of account replication. You will learn more about account replication in the upcoming section. There are a few limitations with database replication. They are as follows:

- The refresh feature will not work if the primary database contains an external table.
- The databases created from shares cannot be replicated.
- If the primary database consists of event tables, then the replication operation fails.
- You can create a replica using `SQL DDL - CREATE DATABASE secondary_database as REPLICA`. However, this command does not support the `TAG` clause. This is not supported as the secondary database is a read only copy of the data.

Note: Snowflake recommends account replication to replicate database data.

Share replication

You might have read about data shares in Snowflake. It is a wonderful data sharing feature that allows users to share the data without moving data out from

the platform. You will learn more about this in [Chapter 12, Data Sharing](#). Share replication offers replication support to the data shares. You have learnt that database and account replication can replicate some objects to secondary databases or accounts. These are read only instances to store replica of the primary data source. Data sharing replication needs a few additional steps to define the replication group, setting up schedule to replicate, and applying privileges to them.

Replication group

This is a group of objects that are defined to be replicated to one or more accounts. This group offers read-only access to the replicated objects. This group offers point-in-time consistency for objects within this group. Similar to sharing, providers and consumers, the provider can define a primary replication group and enable replication to one or more target accounts for consumers. Unlike database replication, share replication carries the objects shared as well as the privileges along with shared objects. Objects in the secondary share can be refreshed manually and set up a schedule to refresh them.

Replication schedule

Snowflake recommends setting up automatic refresh using the **REPLICATION SCHEMA** parameter. The first refresh is done automatically as and when the secondary is created. For any next refresh, you need to setup a schedule with a given interval. Snowflake executes only one refresh at a time - for example, if you have set refresh every 15 minutes and your first refresh cycle is running more than 15 minutes, then the next cycle gets delayed and waits to finish the first cycle.

You can use SQL DDL commands to define custom roles for replication and use DDL commands to define the replication group and schedule. You can use the following sample code:

1. Create a custom role and grant replication group access in source account:

```
USE ROLE ACCOUNTADMIN;  
CREATE ROLE src_replicate_user;  
GRANT CREATEreplication GROUP ON ACCOUNT  
TO ROLE src_replicate_user;
```

2. Create a replication group for source account:

```
USE ROLE src_replicate_user;  
CREATE REPLICATION GROUP replicate_src_poc  
    OBJECT_TYPES = DATABASES, SHARES  
    ALLOWED_DATABASES = RETAIL_DEV_POC, DEV_POC  
    ALLOWED_SHARES = s1  
    ALLOWED_ACCOUNTS = org.account2, org.account3  
    REPPLICATION_SCHEDULE = '15 MINUTE'; /* this  
is used to automate the refresh */
```

3. Create a role in target account:

```
USE ROLE ACCOUNTADMIN;  
CREATE ROLE tgt_replicate_user;  
GRANT CREATE REPLICATION GROUP ON ACCOUNT  
    TO ROLE tgt_replicate_user;
```

4. Create a replica in the target account using source replica:

```
USE ROLE tgt_replicate_user;  
CREATE REPLICATION GROUP replicate_tgt_poc  
AS REPLICA OF org.account1.replicate_src_poc;
```

5. Setup a user to execute refresh manually at source and target

```
/* grant for source account group */  
GRANT REPLICATE ON REPLICATION GROUP  
replicate_src_poc TO ROLE src_replicate_user;  
/* grant for target account group */  
GRANT REPLICATE ON REPLICATION GROUP  
replicate_tgt_poc TO ROLE tgt_replicate_user;
```

6. Manually refresh from target account:

```
USE ROLE tgt_replicate_user;  
ALTER REPLICATION GROUP replicate_tgt_poc  
REFRESH;
```

You can also use the same steps to replicate databases.

Business continuity

As you know, business continuity is a process defined and used to recover data in case of disaster. You have learned two types of failovers using database and share replications. Snowflake offers two additional features as part of business-critical editions. The following are the features:

- Replication and failover
- Client redirect

These features are designed to support various business continuity scenarios:

- **Planned failover:** To test readiness of DR.
- **Unplanned failover:** In case of any outage in any region.
- **Migration:** Used to move an account to a different region or cloud platform without impacting current business.
- **Multiple readable secondaries:** Objects replication across clouds, multiple accounts.

This section helps you learn more about replications, types of replications supported and features that you can use to implement replication across accounts within the organization. The next section provides details of a retail use case that can be used to implement data masking.

Practical: Create Snowflake policies, tagging objects

Consider the following use case to complete the hands-on labs:

Use case: Data classification and dynamic masking.

Domain: Retail

Details: You are working on a retail use case and need to implement data classification, data masking and protecting data from misuse. You are getting data feeds like product, sales, customer details, transactions, and inventory from

your source systems. You need to load these to the Snowflake landing layer. Once data is available in landing, you need to process, transform, and run data enrichment processes to prepare the target consumer layer. You have data analysts, a sales team, and an ML team consuming your target datasets. Now, you need to define the classification and masking policies, and apply them to the target tables. You can also look back to [Chapter 7: Access Control and Managing Users Roles](#) for access control and roles setup reference. Following are the considerations and objects to be setup in Snowflake to implement the use case:

- **Snowflake objects setup:** This is for the development region.
 - **Create database:** RETAIL_DEV_POC
 - **Create schemas:** Landing, Transform and Target
 - **Create tables:** Sales, Product, transactions, order, and customer
- **Roles:** Create custom roles for data analyst, sales, ML and Development team.
- **Privileges:** Grant the following privileges to the roles:
 - Sales, data analyst, and ML team need read-only access to the target layer that is represented by a database or schema.
 - The development team has write access to the DEV database and schemas within this database.
- **Rules or policies:**
 - Highly confidential data is completely masked.
 - Only ACCOUNTADMIN can view the original values.
 - Private or internal access data is tokenized data.
 - Data format can be maintained with tokens.
 - All custom roles can see tokenized data except ACCOUNTADMIN.
- **Data classification:**
 - Highly confidential / restricted
 - Data that represents PII
 - Customer details, transaction details

- Fully masked data
- Columns like `customer_name`, `customer_phone`, `account_number`, `card_details`, `tin` or `ssn` or `sin`, and so on.
- Private / internal access
 - Tokenized data
 - Data that can be accessible only Snowflake users
 - Sales, product, order details, and so on.
 - Columns like `customer_email`, `address`, `zip_code`

Exercises

- Create snowflake objects
- Create custom roles
- Implement dynamic data masking:
 - Create dynamic policies
 - Apply policies to the Snowflake tables
 - Validate applied policies to the tables using various roles. You can refer to below sample query:

```
Use role ACCOUNTADMIN;
select * from
RETAIL_POCTEST.target.customer_details;
use role data_analyst;
select * from
RETAIL_POCTEST.target.customer_details;
```
- Implement object tagging:
 - Create object tags for the objects created for retail use case
 - Apply those tags to the objects using SQL DDL commands

This use case is designed to help you understand end to end role of a security

admin or someone with masking privileges. You need to design the policies and setup, followed by implementing masking policies and applying them to the objects. These policies can be defined once and used for existing as well as any new Snowflake objects being deployed.

Conclusion

This chapter provided a summary of Snowflake's access control policies, support to RBAC, and recommendations to onboard applications and users. This chapter helped you to understand custom role creation, allocating and applying them to the business applications. This also helps you to learn Snowflake account usage and tracking for roles. You will learn more about user management, user onboarding, resource allocations – warehouses, Snowflake objects etc. You will learn more about Snowflake features and offerings as you go along with the next chapters in this book.

In the next chapter, you will learn about Snowflake's data protection and recovery. You will learn more about time-travel, failsafe, and data encryption.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake offers time-travel and failsafe.
- You can recover not only data but also database objects that are dropped accidentally.
- Time-travel feature varies based on the database object and Snowflake edition.
- Snowflake enterprise edition offers 90 days' time-travel.
- Snowflake offers 7 days fail-safe for enterprise and above editions. This can also be used as part of disaster recovery. Snowflake support is required to retrieve data as part of failsafe.
- Snowflake encrypts data stored as part of database objects.
- Snowflake also offers dynamic data masking that allows users to create policies to mask or tokenize data depending on the business requirements as well as classification rules.

Questions

1. How is data classification implemented in Snowflake?
2. How can time-travel be used? Can you create a table from time-travel data?
3. How does Snowflake's data protection- encryption work? What is the type of encryption supported?
4. How can data masking be implemented?
5. Can you calculate usage consumed by time-travel and failsafe?
6. How can you control the usage and monitor storage consumed by time-travel and failsafe?
7. How to retrieve data from time-travel as well as failsafe? why snowflake support is required for failsafe?

Multiple choice questions

1. **What are the default data protection available in Snowflake? (Select all applicable)**
 - a. DR
 - b. Time-travel
 - c. Failsafe
 - d. UNDROP
2. **How many maximum days is data available in Snowflake as part of time-travel? (Select all applicable)**
 - a. 07
 - b. 21
 - c. 90
 - d. 60
3. **How many maximum days is data available in Snowflake as part of**

failsafe? (Select all applicable)

- a. 07
- b. 21
- c. 90
- d. 60

4. How usage can be calculated for time-travel and failsafe? (Select all applicable)

- a. Storage required to store data
- b. Compute required to retrieve data
- c. Storage +compute
- d. All of the above

5. Which role can be used to create masking policies? (Select all applicable)

- a. SECURITYADMIN
- b. ACCOUNTADMIN
- c. USER with SECURITY ROLE
- d. All of the above

6. Which role can be used to apply masking policies? (Select all applicable)

- a. SECURITYADMIN
- b. ACCOUNTADMIN
- c. SYSADMIN
- d. All of the above

7. What does the maximum time travel available for temporary and transient tables?

- a. 0
 - b. 01
 - c. 07
 - d. 60
8. **What are the objects supported in dynamic masking? (Select all that apply)**
- a. UDFs
 - b. Tables
 - c. Views
 - d. Schemas
9. **Who can restore data as part of Failsafe? (Select all that apply)**
- a. Users
 - b. Snowflake
 - c. Customers
 - d. End Users
10. **How to restore data as part of Time travel? (Select all that apply)**
- a. SELECT
 - b. CLONE
 - c. AT | BEFORE
 - d. All of these
11. **What are the parameters supported to restore data as part of Time travel? (Select all that apply)**
- a. OFFSET
 - b. STATEMENT

c. AT | BEFORE

d. All of these

Answers

1	b, c
2	c
3	a
4	a
5	a
6	d
7	b
8	b, c
9	b
10	d
11	d

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 9

Snowflake Performance Optimization

Introduction

Platform performance is one of the key features to evaluate a platform. This is a measure used to capture the platform response to the workloads. Performance optimization is the process by which you can improve the performance of the platform by improving warehouse performance, query performance, and better workload management. Typical performance optimization implementation consists of query optimization, data model optimization, resource optimizations, workload optimizations, and so on.

Snowflake performance is measured, and details are captured in metadata views. These native views store details of query performance, storage, and warehouse usage. You can leverage the native metadata views and analyze the performance of your Snowflake account. Snowflake recommends warehouse usage strategies to optimize the performance of warehouses. Snowflake also offers a set of features like caching, query acceleration service, and so on.

Structure

This chapter consists of the following topics:

- Understanding Snowflake performance
- Understanding Snowflake metadata objects
- Introduction to **ACCOUNT_USAGE**
- Introduction to **INFORMATION_SCHEMA**

- Calculating and understanding performance measures
- Understanding `QUERY_HISTORY`
- Introduction to Query Acceleration Service
- Implementing performance optimization

Objectives

By the end of this chapter, you will be able to develop an understanding of performance measure, performance optimization techniques, and the need to implement optimization. This also covers understanding of Snowflake metadata objects that help to measure the performance of the platform. Snowflake offers a set of features to improve performance as well as recommend workload performance. You can implement optimization analysis and usage queries using the trial account set up in the first chapter.

Understanding Snowflake performance

Snowflake architecture is a three-layered architecture: Cloud Service layer, Warehouse layer, and Storage layer. Snowflake also maintains three types of cache: metadata cache, warehouse cache, and storage cache. Snowflake performance depends on the platform design, warehouse setup, workload setup, and query performance. Snowflake performance consists of the following components:

- Query performance
- Warehouse performance
- Cache usage
- Warehouse setup (workload management)

Snowflake performance components are explained in the next section.

Query performance

Query performance is the overall time taken by a query to be executed. The overall execution time consists of the time a query was in flight or queue, query compilation time, query execution time, and so on. Query performance can be measured and analyzed using a query profile. Query profile shares execution

plan, stepwise execution, time taken for each stage, disk spilling, and so on. This represents the overall time the query takes to execute. One important aspect of query execution is the way the query has been designed and developed. The query complexity, joins, aggregates, and transformations used, as well as tables used in the query, play a vital role in generating the execution plan. Snowflake uses a pruning mechanism to scan the table data stored and access data through queries.

Snowflake associates each query with a unique **query_id**, and you can use the same query id to analyze performance from the **QUERY_HISTORY** view. The query profile is available to be accessed based on the **query_id**. You can view query profile from Snowsight as well as classic UI.

To view query profile from classic UI, go to **History or Worksheets | Query ID**, as shown in the following figure:

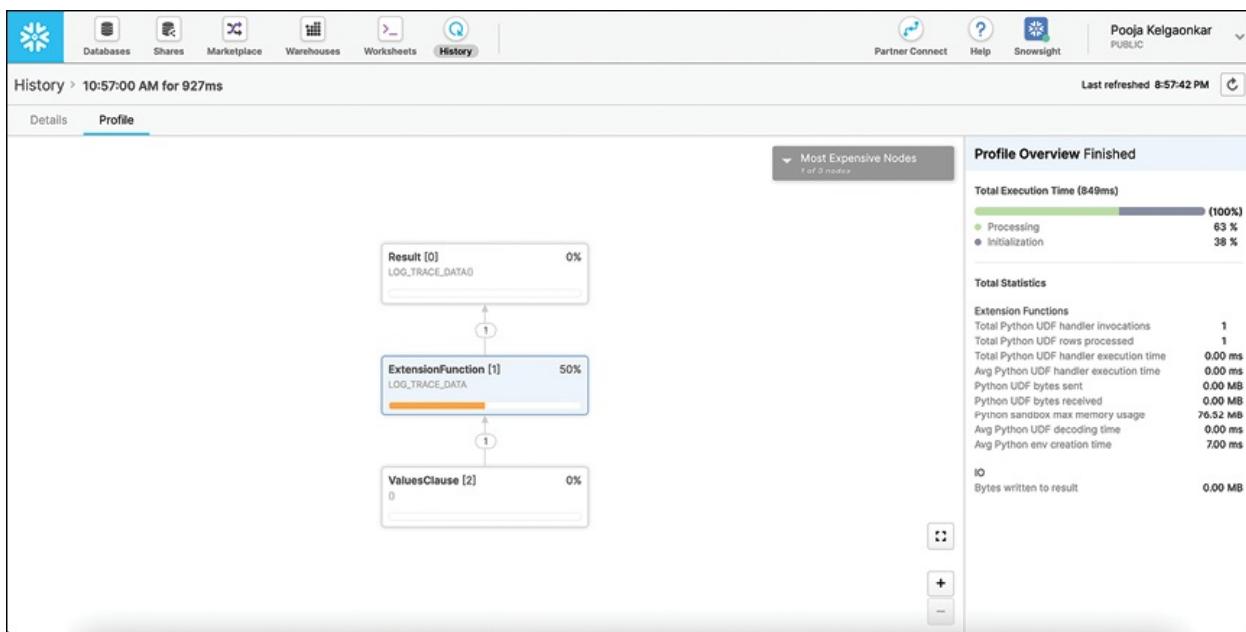


Figure 9.1: Query profile from Classic UI

Query profile from Snowsight is viewed from the query execution. Once you execute the query, you can see the **query_id** on right hand side menu. You can click on the **query_id** to view the query profile, as shown:

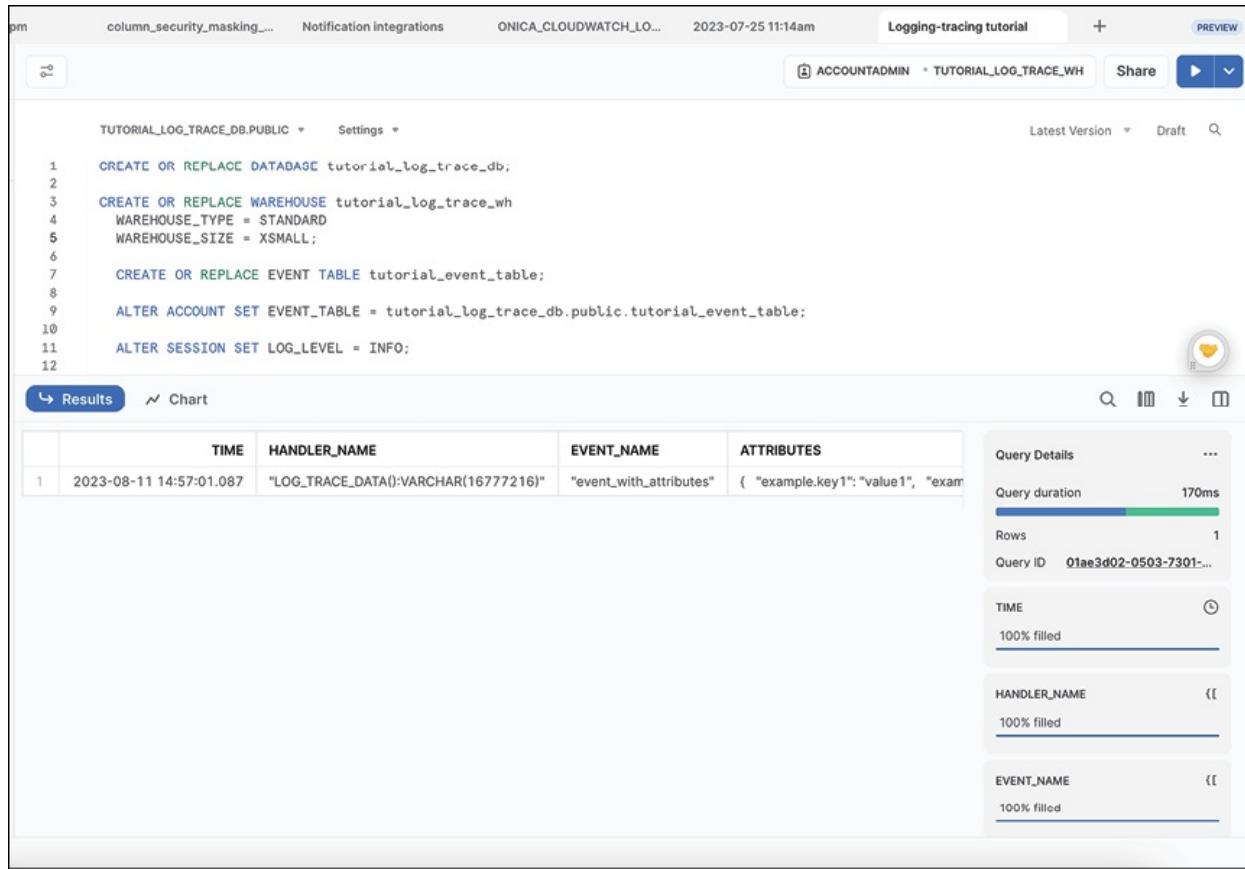


Figure 9.2: Query profile in Snowsight

Snowflake query performance can be measured, analyzed, and viewed using **QUERY_HISTORY** metadata view. You will learn more about this view in the upcoming section of this chapter. Snowflake also offers Query Acceleration Service that enables users to identify the workload or query that is eligible for acceleration. You will also learn more about this in the next section of this chapter.

Understanding Snowflake metadata objects

Snowflake shares metadata information with **INFORMATION_SCHEMA** and **ACCOUNT_USAGE** schema. Information schema contains a set of system defined views and functions to provide metadata information about Snowflake objects. This is the default schema Snowflake adds as and when a database is created.

INFORMATION_SCHEMA

This is one of the metadata object schemas present in the Snowflake database. This can be used to reduce the DDL or metadata queries cost, like **SHOW**, **LIST**,

and so on. The following are the features of the schema:

- This is a read-only schema.
- Queries on the views may not provide consistency with the current DDL of the object.
- The view or function output is based on the privileges granted to the current user.
- These are Snowflake specific views that may change. It is recommended to use specific column names from the view while using them to analyze usage.

Using INFORMATION_SCHEMA

This schema consists of a set of views that can be used to query to gather information and details on Snowflake standard and non-standard objects. Below are the details of one of the scenarios where you can use **INFORMATION_SCHEMA**:

- This schema offers an SQL interface to the metadata information provided by the **SHOW** command.
- These views can be used to replace the **SHOW** commands.
- **SHOW** commands are run on metadata, cache stored in cloud services layer.
- Querying on **information_schema** views may cost some additional cost as this needs the warehouse in a running state to run any queries on views.

ACCOUNT_USAGE

This is another schema in the Snowflake shared database. You can use this schema object to gather information of Snowflake components to generate usage metrics. Following are some of the key features of this schema:

- This is a read-only schema.
- There are two types of **ACCOUNT_USAGE** views: **ACCOUNT_USAGE** and **READER_ACCOUNT_USAGE**.
- This schema has the same views and functions as **INFORMATION_SCHEMA**.
- There are few key differences between **INFORMATION_SCHEMA** and **ACCOUNT_USAGE** as follows:

- **ACCOUNT_USAGE** includes objects that are dropped, whereas **INFORMATION_SCHEMA** does not maintain any dropped objects.
- **ACCOUNT_USAGE** has 45 minutes to 3 hours latency to reflect details in metadata views and functions. However, there is no latency for **INFORMATION_SCHEMA**. Metadata information is immediately reflected in these views and objects.
- **ACCOUNT_USAGE** has data retention of 1 year and **INFORMATION_SCHEMA** has retention of 7 days to 6 months, depending on the type of object: views or functions.

DATA_SHARING_USAGE

This is one of the schemas in Snowflake share that consists of metadata information of data shares of the account. Following are some of the key features of this schema:

- This schema consists of metadata information of the data listed in the marketplace or data exchange.
- This schema has telemetry as well as consumption data.
- Only **ACCOUNTADMIN** can access this schema. However, permissions can be granted to other roles to access this schema.

ORGANIZATION_USAGE

This is another schema in the Snowflake database with a set of views to provide metadata information on the historical usage of accounts present in an organization. Below are some of the key features of schema:

- This schema consists of metadata information at the **ORGANIZATION** level.
- You can have multiple accounts in one organization, and this schema consists of historical data of usage.
- This is not accessible to all accounts.
- This can be accessed only from an account that has **ORGADMIN** role enabled. **ORGADMIN** can enable this role for an account using SQL command.
- Users with the appropriate role and privilege can access this **USAGE** view. Not all users have access to this schema.

You will learn more about **USAGE** and **INFORMATION_SCHEMA** in subsequent sections of this chapter.

Introduction to ACCOUNT_USAGE

ACCOUNT_USAGE is a Snowflake database schema that enables users to query the object metadata and historical usage data. There are two schemas that you can use to get the **USAGE** details of your **ACCOUNT** as well as the **READERS** account created for Snowflake data shares. You can create **READ ONLY** users to share the Snowflake data with non-snowflake consumers. You will learn more about this in the upcoming *Chapter 12: Data Sharing*. You have learned the **ACCOUNT_USAGE** features and how it is different from **INFORMATION_SCHEMA** in the above section. In this section of the chapter, you will learn more about usage details captured and how you can use this metadata information to generate overall account usage details.

Dropped object information

Account usage views store metadata information of all the objects that are dropped. Most of the views in the schema have an additional column called **DELETED**, that represents the timestamp of object deletion. In the case of the objects that are dropped and re-created with the same name, this view creates an ID to differentiate between the dropped and re-created table. This is maintained and used as an internal ID assigned to each object.

Latency of the details

Account usage extracts details from Snowflake's internal metadata store and has some latency to reflect the operations to the usage views. Some views are reflected within 2 hours, and some of them are reflected within 3 hours.

Data retention

Account usage views also provide historical metrics. The data retention of account usage views is 1 year. You can query account usage views for 1 year and generate the usage over a period of a year to derive the usage metrics, trends, spends, and so on.

Account usage views

These are a set of views and functions in the account usage schema. There are a set of views in this schema that store historical details of various events, alerts, transactions, operations, accesses, logons, loads, queries, and other operations in the account. This metadata information is crucial to derive the usage metrics, track usage, and monitor performance of the account and components like warehouses, queries, and so on. The following table represents some of the critical views along with short summary of each view:

View name	Description	Latency
COPY_HISTORY	Maintains historical details of data loaded using COPY as well as Snowpipe.	1 Year
LOAD_HISTORY	Maintains details of data loaded using COPY INTO .	1 Year
LOGIN_HISTORY	Contains information of the account or user logons for a year.	1 Year
METERING_HISTORY	Stores hourly credit usage for an account in a year.	1 Year
PIPE_USAGE_HISTORY	Maintains history of the data loaded using snowpipe in one year. Shows history of data loaded as well as credits billed for a year.	1 Year
QUERY_ACCELERATION_HISTORY	Details of the queries and usage where query acceleration service is being used.	1 Year
QUERY_HISTORY	Contains details of the historical queries being run in an account for a year.	1 Year
SEARCH_OPTIMIZATION_HISTORY	Stores details of search optimization service, usage, credits consumed to use the service.	1 Year
SERVERLESS_TASK_HISTORY	Maintains usage and history details of the serverless tasks being	1 Year

	used. Stores usage information of each of the task.	
SNOWPIPE_STREAMING_FILE_MIGRATION_HISTORY	Maintains details of the history data migrated using Snowpipe.	1 Year
STORAGE_USAGE	Maintains the average daily usage of data stored in bytes for a year.	1 Year
TASK_HISTORY	Maintains history of the tasks used for an account over the period of a year.	
WAREHOUSE_EVENTS_HISTORY	Contains details of the various events triggered for single as well as multi-cluster warehouses like suspend, start, and so on.	1 Year
WAREHOUSE_LOAD_HISTORY	Stores details of the workloads being run on a warehouse.	1 Year
WAREHOUSE_METERING_HISTORY	Contains details of the hourly usage information for a single warehouse in an account for a period of a year.	1 Year

Table 9.1: Account usage views information

Account usage functions

There is only one function: **TAG_REFERENCES_WITH_LINEAGE** in this schema. This can be used to derive the association between tag and object. This can be used to derive the data lineage with tags associated with objects.

Reader account usage views

Reader accounts are created for non-Snowflake users to share the data as a part of the data sharing feature of Snowflake. In the case of reader accounts, the credits of Snowflake usage are charged to the account that creates the reader account. You will learn more about this in [Chapter 12, Data Sharing](#). Following are the views available in this schema to track usage details of the reader accounts:

View name	Details	Data

		retention
LOGIN_HISTORY	Maintains details of account logon.	1 year
QUERY_HISTORY	Contains information of queries run using the Reader account.	1 year
RESOURCE_MONITORS	Maintains details of the resource monitors used to monitor the credit consumption for reader accounts.	1 year
STORAGE_USAGE	Contains information on storage used as daily average in bytes.	1 year
WAREHOUSE_METERING_HISTORY	Maintains information of warehouse credits consumed by each reader account for a period of a year.	1 year

Table 9.2: READER_ACCOUNT_USAGE views

Roles and permissions

This schema is accessible to the **ACCOUNTADMIN** by default. You have to run the permission query to enable other users or roles access to the views. You can use the **GRANT SQL** command to allow users or roles as follows:

```
USE ROLE ACCOUNTADMIN;
```

```
GRANT IMPORTED PRIVILEGES ON DATABASE snowflake TO
ROLE SYSADMIN;
```

```
GRANT IMPORTED PRIVILEGES ON DATABASE snowflake TO
ROLE DATA_ENG;
```

Once you grant the permission, you can use the role and test the permission by using **SELECT** on the snowflake database schema, as shown:

```
USE ROLE DATA_ENG;
```

```
SELECT database_name, database_owner FROM
snowflake.account_usage.databases;
```

You have learned about account usage and reader account usage schema in this section. You can use these various views in the schema to track the account usage, and performance of your queries, use them to optimize the performance

of the account as well as workloads. You will learn more about utilizing these views in the upcoming section.

Introduction to INFORMATION_SCHEMA

Information schema is also referred to as a data dictionary. This consists of system defined views and a table function that provides details or metadata information of objects created in an account. Most of the information schema views and functions have data retention of 7 days to 6 months based on the type of views. Some specific views maintain details only for 14 days.

Information schema views

Similar to **ACCOUNT_USAGE**, information schema consists of various metadata views. These views and queries can be used to replace DDL commands to list metadata information, object definitions, and so on. **LOAD_HISTORY** maintains load metadata for 14 days. Following are some of the commonly used views in this schema:

View name	Description	Specific retention time
COLUMNS	Maintains the details of columns.	
DATABASES	Contains details of the databases in an account.	
FILE FORMATS	Maintains metadata information of file formats created in an account.	
FUNCTIONS	Maintains metadata information of functions created in an account.	
LOAD_HISTORY	Maintains metadata information of loads in an account.	Data retained for 14 days.
PIPES	Maintains metadata information of pipes created in an account.	
PROCEDURES	Maintains metadata information of procedures created in an account.	
SEQUENCES	Maintains metadata information of sequences created in an account.	
STAGES	Maintains metadata information of stages created in an account.	
TABLES	Maintains metadata information of tables created in an account.	

USAGE_PRIVILEGES	Maintains metadata information of privileges, roles granted in an account.	
VIEWS	Maintains metadata information of views created in an account.	

Table 9.3: Information Schema Views

Information schema functions

These are a set of functions that can be used to get account level usage of accounts for storage, logins, queries, and warehouses. Following are the functions:

Table function name	Data retention period
AUTOMATIC_CLUSTERING_HISTORY	14 days
AUTO_REFRESH_REGISTRATION_HISTORY	14 days
COMPLETE_TASK_GRAPHS	60 minutes
COPY_HISTORY	14 days
CURRENT_TASK_GRAPHS	N/A
DATA_TRANSFER_HISTORY	14 days
DATABASE_REFRESH_HISTORY	14 days
DATABASE_REFRESH_PROGRESS , DATABASE_REFRESH_PROGRESS_BY_JOB	14 days
DATABASE_REPLICATION_USAGE_HISTORY	14 days
DATABASE_STORAGE_USAGE_HISTORY	6 months
EXTERNAL_FUNCTIONS_HISTORY	14 days
EXTERNAL_TABLE_FILES	N/A
EXTERNAL_TABLE_FILE_REGISTRATION_HISTORY	30 days
LOGIN_HISTORY , LOGIN_HISTORY_BY_USER	7 days
MATERIALIZED_VIEW_REFRESH_HISTORY	14 days
NOTIFICATION_HISTORY	14 days
PIPE_USAGE_HISTORY	14 days
POLICY_REFERENCES	N/A
QUERY_ACCELERATION_HISTORY	14 days
QUERY_HISTORY , QUERY_HISTORY_BY_*	7 days
REPLICATION_GROUP_REFRESH_HISTORY	14 days

<code>REPLICATION_GROUP_REFRESH_PROGRESS,</code>	14 days
<code>REPLICATION_GROUP_REFRESH_PROGRESS_BY_JOB</code>	
<code>REPLICATION_GROUP_USAGE_HISTORY</code>	14 days
<code>REPLICATION_USAGE_HISTORY</code>	14 days
<code>REST_EVENT_HISTORY</code>	7 days
<code>SEARCH_OPTIMIZATION_HISTORY</code>	14 days
<code>SERVERLESS_TASK_HISTORY</code>	14 days
<code>STAGE_DIRECTORY_FILE_REGISTRATION_HISTORY</code>	14 days
<code>STAGE_STORAGE_USAGE_HISTORY</code>	6 months
<code>TAG_REFERENCES</code>	N/A
<code>TAG_REFERENCES_ALL_COLUMNS</code>	N/A
<code>TASK_DEPENDENTS</code>	N/A
<code>TASK_HISTORY</code>	7 days
<code>VALIDATE_PIPE_LOAD</code>	14 days
<code>WAREHOUSE_LOAD_HISTORY</code>	14 days
<code>WAREHOUSE_METERING_HISTORY</code>	6 months

Table 9.4: Information Schema Functions

You can query information schema views and functions using a fully qualified name. Below are some of the sample commands:

- You can use a fully qualified name:

```
SELECT table_name FROM
retail_poc.INFORMATION_SCHEMA.TABLES WHERE
TABLE_SCHEMA = 'POC';
```

- You can set the context and database name before running a query on information schema:

```
USE database retail_poc;
```

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'POC';
```

- You can also use information schema as session schema using the `USE` command:

```
USE database retail_poc;
```

```
Use schema INFORMATION_SCHEMA;  
  
SELECT table_name FROM TABLES WHERE TABLE_SCHEMA =  
'POC';
```

You can use information schema to retrieve metadata information. This schema and usage are different from account usage schema as account usage maintains historical data that can be used to calculate account credit consumption, storage usage, and so on. You will learn more about various metrics that can be derived and about using them in the next section.

Calculating and understanding performance measures

Performance can be measured using various metrics. You can define the metrics to compute account usage based on Snowflake's three-layered architecture: Cloud Service Layer, Compute Layer, and Storage Layer. These metrics can be defined and used to derive overall account usage in terms of storage usage, compute usage, credit consumption, cloud service layer usage etc. This section covers various metrics and their usage to understand overall account performance, billing, and usage.

You can define metrics based on the Snowflake layers and their corresponding usage. You can use the **ACCOUNT_USAGE** schema and its views to derive the overall usage for the metrics defined. You can refer to some of the metrics as defined below:

- **Storage layer usage:** This is the overall usage of storage used for database, table data stored as well as data protection usage – failsafe and time-travel. This is calculated as the daily average in bytes. You can define some of the metrics for this usage as below:
 - Total storage usage for an account
 - Storage usage by databases
- **Compute or warehouse layer usage:** This is the overall usage of the credits consumed by warehouses. This is the total usage of active warehouses being run for a duration. If you are not using the warehouse, then you will not be billed for the warehouse. The default or minimum bill of a warehouse is the minute as soon as the warehouse is set in **START** state. You can consider the metrics as the following:

- Analysis of warehouse performance
- Warehouse consumption or credit usage (Total)
- Warehouse usage by warehouse
- Queries per warehouses
- Query Execution trend (based on time)
- **Cloud service layer usage:** This is the usage of the Cloud service layer. As you know, this is the first layer of the architecture and maintains metadata information that is required for query parsing, query execution plan, maintaining cache etc. You can define metrics to check the service layer usage to identify the overall consumption of cloud services against the total usage of an account. Service layer consumption is not billed until it is below 10% of credits consumed overall. If the usage exceeds the 10% threshold, then this is billed for the account. You can define and use the following metrics:
 - Cloud service layer consumption (Total)
 - Top logon failures

You can consider some of the above metrics and define your set of metrics per your needs. You can now start using the **USAGE** schema views and functions to generate metrics to calculate performance. Below are some of the sample queries that you can use to generate these metrics:

- **Total storage usage for an account per month:** Run a query on **STORAGE_USAGE** view to count the total storage in TB:

```
select date_trunc(month, usage_date) as
usage_month
, avg(storage_bytes + stage_bytes +
failsafe_bytes) / power(1024, 4) as billable_tb
from storage_usage
group by 1
order by 1;
```

- **Total queries by warehouses:** Run a query on **QUERY_HISTORY** view to

count the number of queries being run on a warehouse to date:

```
select WAREHOUSE_NAME, count(*) as  
number_of_queries  
  
from query_history  
  
where start_time >= date_trunc(month,  
current_date)  
  
group by warehouse_name;
```

- **Total number of jobs running in an account:** Run a query on **QUERY_HISTORY** view to consider the queries as jobs being run actively in an account to date:

```
select count(*) as number_of_jobs  
  
from query_history  
  
where start_time >= date_trunc(month,  
current_date);
```

- **Total credit consumption by warehouse:** Run a query on **QUERY_HISTORY** to calculate the overall credits consumed to date:

```
select warehouse_name,  
  
       sum(credits_used) as total_credits_used  
  
from warehouse_metering_history  
  
where start_time >= date_trunc(month, current_date)  
  
group by 1  
  
order by 2 desc;
```

- **Overall loads by warehouse:** Run a query on **WAREHOUSE_LOAD_HISTORY** to calculate the overall load:

```
select  
  
warehouse_name,
```

```
sum(avg_running) as avg_load_time  
from  
SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_LOAD_HISTORY  
group by warehouse_name;
```

- **Overall credits consumed by warehouse:** Run a query on **WAREHOUSE_METERING_HISTORY** to calculate overall credit consumption:

```
select  
warehouse_name,  
sum(credits_used) as total_credits_used  
from  
SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY  
group by warehouse_name;
```

You can derive the metrics and categorize them into operational, monitoring, engineering, and administrative. You can derive the metrics leveraging **USAGE** views for your account. You can also build dashboards using Snowsight. You will learn more about the **QUERY_HISTORY** view and metrics that can be derived from this view.

Understanding **QUERY_HISTORY**

QUERY_HISTORY is an account usage view that can be used to get the query history in an account. You can run various queries to derive metrics using dimensions like time, session, user, warehouse etc. You can use the view in **ACCOUNT_USAGE** as well as **READER_ACCOUNT_USAGE**. You can generate the metrics leveraging the dimensions. You can refer to some of the metrics as following detailed:

- Total active queries in an account
- Total failed queries by users
- Top failed queries
- Long-running queries based on the execution time
- Total jobs running per warehouse

- Total queries by database
- Total queued queries

You can use the **QUERY_HISTORY** view to derive the above-mentioned metrics. You can use view columns along with dimensions. This view has 68 columns that cater to the metadata information of each query being run in an account.

Snowflake associates each query with a unique identified/ID called **query_id**. You can use this ID column to track the progress of a query, view query profile, or view the consumption details of queries. Refer to some of the critical view columns and their description in the following table:

Column name	Description	Data type
QUERY_ID	System-generated unique identifier for the SQL statement.	TEXT
QUERY_TEXT	SQL statement text. This stores query text up to 100K. If the query is longer, then it gets truncated.	TEXT
DATABASE_NAME	Database used in the query.	TEXT
SCHEMA_NAME	Schema used in the query.	TEXT
QUERY_TYPE	Type of query. For example, SELECT, SHOW, USE etc.	TEXT
USER_NAME	User who ran the query.	TEXT
WAREHOUSE_NAME	Warehouse used to execute the query, if any.	TEXT
WAREHOUSE_SIZE	Size of the warehouse used to execute the query.	TEXT
WAREHOUSE_TYPE	Type of the warehouse used for query statement being executed.	TEXT
CLUSTER_NUMBER	The cluster number is executed on multi-cluster warehouse.	NUMBER
QUERY_TAG	Query tag set for query execution using QUERY_TAG session parameter.	TEXT
EXECUTION_STATUS	Status of the query execution values: success, fail, incident.	TEXT

ERROR_CODE	Error code, if the query had an error	NUMBER
ERROR_MESSAGE	Error message, if the query had an error	TEXT
START_TIME	start time of query(in the UTC time zone)	TIMESTAMP_LTZ
END_TIME	end time of query (in the UTC time zone).	TIMESTAMP_LTZ
TOTAL_ELAPSED_TIME	Elapsed time to execute query (in milliseconds).	NUMBER
BYTES_SCANNED	Number of bytes scanned by this query.	NUMBER
PERCENTAGE_SCANNED_FROM_CACHE	The percentage of data scanned from the local disk cache.	FLOAT
BYTES_WRITTEN	Number of bytes written in case of load	NUMBER
BYTES_WRITTEN_TO_RESULT	Number of bytes written to a result object.	NUMBER
BYTES_READ_FROM_RESULT	Bytes read from a result object.	NUMBER
PARTITIONS_SCANNED	Number of micro-partitions scanned.	NUMBER
PARTITIONS_TOTAL	Total micro-partitions of all tables included in this query.	NUMBER
BYTES_SPILLED_TO_LOCAL_STORAGE	Data spilled to local disk.	NUMBER
BYTES_SPILLED_TO_REMOTE_STORAGE	Data spilled to remote disk.	NUMBER
COMPILATION_TIME	Query Compilation time (in milliseconds)	NUMBER
EXECUTION_TIME	Query Execution time (in milliseconds)	NUMBER
QUEUED_PROVISIONING_TIME	Time (in milliseconds) spent for warehouse provisioning or resizing	NUMBER

Table 9.5: Query history view columns

You can use these columns to derive performance metrics. This view is one of the critical views to derive the warehouse loads, warehouse performance, warehouse credit consumption etc. You can refer to some of the following sample use cases:

- **Warehouse sizing:** You can generate metrics that can be used to help derive the warehouse configuration: resize or cluster reconfiguration. You can analyze the workloads running in warehouse using their execution, compilation, queue time etc. You can compare the performance to derive the warehouse configuration. It is recommended to use resize to change the size of the warehouse in case of file loads. In case of concurrent queries, and workloads being run on a clustered warehouse – you can plan on adding or removing clusters using re-cluster.
- **Query performance analysis:** You can use this query view to compute the performance of queries. You can use elapsed time, compilation time, provisioning time, and queue time for query execution to identify queries that are running for a long duration. You can also use query profile to validate the query plan, bytes scanned, processing steps to fine-tune the query, or use a query acceleration service. You will learn more about this in the next section.

Along with analysis, you can also use **QUERY_HISTORY** to monitor workloads, queries/jobs being executed as part of ops, and monitoring. You can consider some of the below metrics and use SQL to monitor query performance:

- **Long running queries:** You can use the **QUERY_HISTORY** view to capture the queries running for a long time. You can set the threshold to capture the long-running queries. For example, you can capture the queries that are running for more than an hour, as shown:

```
select
query_id,
total_elapsed_time,
warehouse_name
from QUERY_HISTORY
where total_elapsed_time >= 3600000;
```

- **Long queued queries:** You can run a query on **QUERY_HISTORY** to capture details of queued workloads:

```
select
```

```

(QUEUED_PROVISIONING_TIME+QUEUED_REPAIR_TIME+
QUEUED_OVERLOAD_TIME) as QUEUED_TIME,
QUERY_ID
from QUERY_HISTORY
where QUEUED_PROVISIONING_TIME > 0;

```

You can derive custom metrics from the query history view. You can use these to identify queries that need to be optimized. Snowflake also offers a query acceleration service that can help you to identify queries that need acceleration and improve the performance of queries. You will learn more about this service in the next section.

Introduction to Query Acceleration Service

This is an enterprise edition feature that offers acceleration by accelerating parts of the workload in a warehouse. You can enable this service for a warehouse. This helps in improving the performance of a warehouse by reducing the impact of queries that need more resources than typical queries. The service improves performance by offloading work to shared compute resources provided by the service. Following are some of the examples that can be benefited from this service:

- Ad-hoc analysis
- Workload with unpredicted data volume
- Queries with large data scans

These workloads can be handled by the acceleration service in a better way as this adds resources to perform workload in parallel and reducing scanning, filtering time. This service is dependent on the server availability. You can use this service to improve the performance of the warehouse. You can identify the eligible workloads and warehouses using the system function

SYSTEM\$ESTIMATE_QUERY_ACCELERATION or use the **QUERY_ACCELERATION_ELIGIBLE** view to identify warehouses that might benefit from this service.

You cannot accelerate all types of workloads using this service. Following are some of the loads that are ineligible queries:

- Queries with no filters and aggregates

- Query with filters that are not selective enough
- Queries where there are not enough partitions
- Queries with `LIMIT` clause

SYSTEM\$ESTIMATE_QUERY_ACCELERATION function

You can use this system function to identify if queries are eligible for acceleration. You can use the function with `query_id`, and this returns an estimated query execution time:

```
SELECT
PARSE_JSON(SYSTEM$ESTIMATE_QUERY_ACCELERATION('1be89bf0-2315-6b2c-ac0c-8a4295ebd70f'));
```

Result:

```
{
  "estimatedQueryTimes": {
    "1": 270,
    "10": 105,
    "2": 112,
    "4": 125,
    "8": 129
  },
  "originalQueryTime": 310.201,
  "queryUUID": "1be89bf0-2315-6b2c-ac0c-8a4295ebd70f",
  "status": "eligible",
  "upperLimitScaleFactor": 10
}
```

If this query is eligible, then this results in `estimatedQueryTimes`, which represents the estimated time. If the query is not eligible, then it shows blank `estimatedQueryTimes`.

QUERY_ACCELERATION_ELIGIBLE view

This is the view that shows queries and warehouses eligible for acceleration. You can run queries against this view to identify workloads eligible for a warehouse:

```
SELECT query_id, eligible_query_acceleration_time  
      FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_ACCELERATION_ELIGIBLE  
     WHERE warehouse_name='DEV_POC_WH'  
ORDER BY eligible_query_acceleration_time DESC;
```

You can also use this view to find which warehouses are eligible for acceleration:

```
SELECT warehouse_name,  
       SUM(eligible_query_acceleration_time) AS total_eligible_time  
    FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_ACCELERATION_ELIGIBLE  
   GROUP BY warehouse_name  
ORDER BY total_eligible_time DESC;
```

Enable query acceleration service

You can use and enable this service by using a parameter while creating a warehouse or using the **ALTER** command. You can use the following command:

```
Create warehouse DEV_POC_WH with  
ENABLE_QUERY_ACCELERATION = TRUE;
```

Similar to the type of workloads, this service supports only a set of the following SQL commands:

- **SELECT**
- **INSERT** with **SELECT** clause

- **Create table as (CTAS)**

Query acceleration usage

Once you enable the service, you can also monitor the usage of the query acceleration service. This can be done using Web UI as well as **QUERY_HISTORY** view. You need to monitor the service cost for its usage as it contributes to the overall cost of the Snowflake account. Query profile represents acceleration service usage.

You can use the query history view to identify queries or workloads that benefit from this service. The query history view contains the following columns that are used to identify service usage:

`QUERY_ACCELERATION_BYTES_SCANNED`

`QUERY_ACCELERATION_PARTITIONS_SCANNED`

`QUERY_ACCELERATION_UPPER_LIMIT_SCALE_FACTOR`

You can use the below query to identify the workload that is consuming most of the bytes:

```
SELECT query_id,  
       query_acceleration_bytes_scanned,  
       query_acceleration_upper_limit_scale_factor,  
       query_acceleration_partitions_scanned  
  FROM SNOWFLAKE.ACOUNT_USAGE.QUERY_HISTORY  
 WHERE query_acceleration_partitions_scanned > 0  
 ORDER BY query_acceleration_bytes_scanned DESC;
```

You also need to understand the billing of Snowflake service. You can view the billing cost from Web UI as well as run queries on the **QUERY_ACCELERATION_HISTORY** view.

Query acceleration billing

You can refer to the billing of the acceleration service. This billing is similar to any other serverless service of Snowflake. The billing is computed based on the

usage per second. You can view the usage from the web UI billing serverless section or query usage view:

```
SELECT warehouse_name,
       SUM(credits_used) AS total_credits_used
    FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_ACCELERATION_HISTORY
   GROUP BY 1
  ORDER BY 2 DESC;
```

Query acceleration service can be used for performance optimization. This service can also be used along with search optimization to optimize the performance of queries and workloads. You will learn more about performance optimization in the next section.

Implementing performance optimization

Performance of a data platform can be measured in various ways. There are a few key considerations to the performance computation of a data platform. You can consider query performances, workload performances, resources consumed, credit consumptions, storage consumption, warehouse utilization, queued loads, request and response time of queries over a period of time, and so on. Once you know the performance of your data platform, you can define the thresholds based on the workloads, business requirements, **Service Level Agreements (SLAs)** and **Service Level Objectives (SLOs)**. You can evaluate the usage and performance against these thresholds and analyze whether your platform is performant or needs optimization to improve the performance.

Optimization is the method or process through which you improve the performance or execution of workloads based on the analysis of the platform. You can analyze the platform by computing the query execution, query profile, performance evaluation of other workloads, storage, concurrent workloads, warehouse configurations, and so on. You can take these as input checkpoints to define an optimization plan to improve the performance of the platform.

You have learned about **USAGE** schemas: **ACCOUNT_USAGE**, and **INFORMATION_SCHEMA**, which can be used to analyze, review usage, and use it to monitor overall account consumption. You can also use it as input to optimize

the performance of queries, warehouses, and accounts. You need to use metadata views in these schemas to generate the metrics, evaluate them against the thresholds defined, and identify the components eligible for optimization.

Based on the layered architecture of the Snowflake platform, the performance optimization can be divided into 3 categories:

- Query performance optimization
- Warehouse performance optimization
- Storage performance optimization

You will learn more about each of these categories in the upcoming section of this chapter. Each of these categories includes the steps to analyze performance and techniques to improve the performance. This also includes recommendations to use Snowflake services to achieve performance optimization.

Query performance optimization

This section covers query analysis for historical queries as well as tasks. You can consider the query execution time and identify the queries that are potential candidates for optimization. You can view the historical details or performance of queries using Snowsight, or you can also write queries on the views in **ACCOUNT_USAGE**.

View historical performance in Snowsight

Snowsight can be used to get historical performance data as well as visual insights into the query performances, task performances or warehouse performances. You can follow the given steps to open a visual presentation of query performances using Snowsight:

1. Logon to Snowsight (account URL)
2. Go to **Activity | Query History**.
3. You can see an activity window as shown below:

SQL TEXT	QUERY ID	STATUS	USER	WAREHOUSE	DURATION
SHOW GRANTS TO USER identifier('POOJAKE...')	01ae7270-0603-76f1-0046-be830059136e	Success	POOJAKELGAONKAR	—	26ms
with active_contracts as (select disti...	01ae7270-0603-76f2-0046-be83005913ba	Success	POOJAKELGAONKAR	COMPUTE_WH	453ms
with active_contracts as (select disti...	01ae726f-0603-76f1-0046-be8300592342	Success	POOJAKELGAONKAR	COMPUTE_WH	1.3s
with active_contracts as (select disti...	01ae726f-0603-76f2-0046-be83005913a2	Success	POOJAKELGAONKAR	COMPUTE_WH	1.7s
with active_contracts as (select disti...	01ae726e-0603-76f2-0046-be83005913...	Success	POOJAKELGAONKAR	COMPUTE_WH	4.2s
select * from IDENTIFIER('SNOWFLAKE...',	01ae7244-0603-76f1-0046-be83005922fe	Success	POOJAKELGAONKAR	COMPUTE_WH	3.3s
select * from IDENTIFIER('SNOWFLAKE...',	01ae723c-0603-76f1-0046-be83005922fa	Success	POOJAKELGAONKAR	COMPUTE_WH	3.4s
select * from IDENTIFIER('SNOWFLAKE...',	01ae7235-0603-76f1-0046-be83005922...	Success	POOJAKELGAONKAR	COMPUTE_WH	2.5s
select * from IDENTIFIER('SNOWFLAKE...',	01ae722e-0603-76f2-0046-be83005912d2	Success	POOJAKELGAONKAR	COMPUTE_WH	879ms
select * from IDENTIFIER('SNOWFLAKE...',	01ae722d-0603-76f1-0046-be83005922...	Success	POOJAKELGAONKAR	COMPUTE_WH	3.3s
select * from IDENTIFIER('SNOWFLAKE...',	01ae720f-0603-76f2-0046-be83005912b2	Success	POOJAKELGAONKAR	COMPUTE_WH	3.4s
select * from IDENTIFIER('SNOWFLAKE...',	01ae720d-0603-76f2-0046-be83005912a2	Success	POOJAKELGAONKAR	COMPUTE_WH	3.7s
select * from IDENTIFIER('SNOWFLAKE...',	01ae7fde-0603-76f2-0046-be830059129e	Success	POOJAKELGAONKAR	COMPUTE_WH	3.9s
select * from IDENTIFIER('SNOWFLAKE...',	01ae7fde-0603-76f1-0046-be8300592222	Success	POOJAKELGAONKAR	COMPUTE_WH	1.1s
select * from IDENTIFIER('SNOWFLAKE...',	01ae7fdf-0603-76f1-0046-be830059220e	Success	POOJAKELGAONKAR	COMPUTE_WH	1.2s
select * from IDENTIFIER('SNOWFLAKE...',	01ae7fd7-0603-76f1-0046-be830059220a	Success	POOJAKELGAONKAR	COMPUTE_WH	1.1s

Figure 9.3: Activity Query history view

4. You can notice the **DURATION** column with a colorful representation of the time it has taken to execute a query. This color code represents the compilation and execution time with the total time to complete the query.
5. You can further use the filters on top to select a specific user or status or duration.
6. You can also view the task history the same way.
7. Click on **Activity | Task History**.
8. You can see a visual representation as follows:

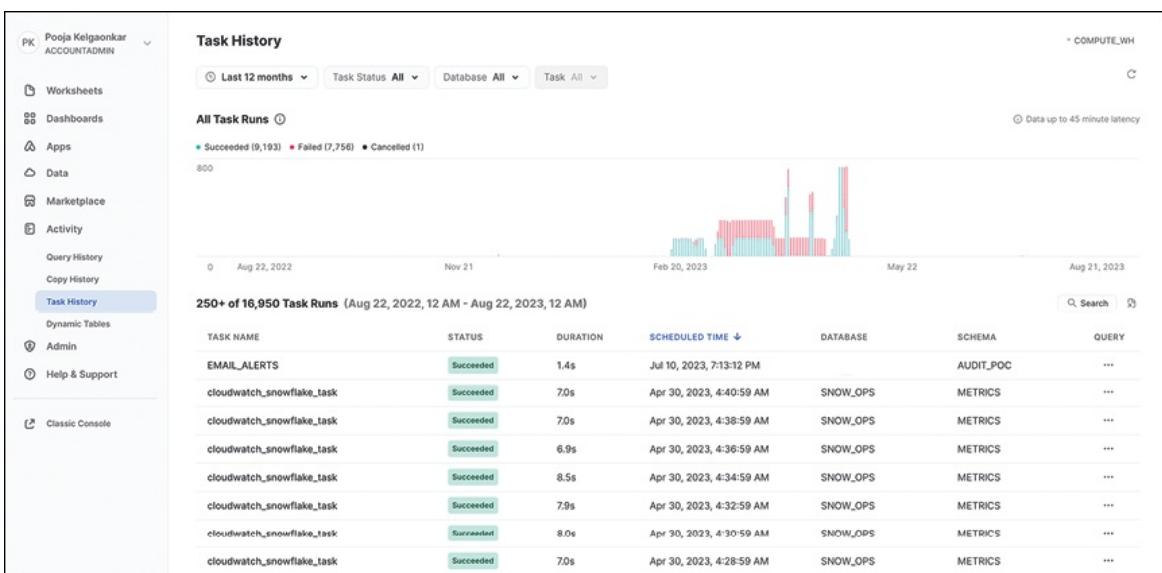


Figure 9.4: Activity- Task History View

9. You can use filters from the top, the same as the query history view.

Now, you can view the performance from Snowsight or run the following queries on **ACCOUNT_USAGE** views to get similar details:

- Get query details of today's run from **QUERY_HISTORY** view:

```
select
query_id, total_elapsed_time, warehouse_name
from SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
where start_time >= date_trunc(month,
current_date);
```

- You can get a historical view of the queries and performance trends using the **start_time** column of view to filter specific duration or weeks or months to analyze queries.

- Get tasks details of tasks run for last one hour from **TASK_HISTORY** view:

```
SELECT query_text, completed_time
FROM snowflake.account_usage.task_history
WHERE COMPLETED_TIME > DATEADD(hours, -1,
CURRENT_TIMESTAMP());
```

- You can also run the task history for a duration using the **COMPLETED_TIME** column from the view.

Once you have execution details for evaluation, you can review the top executed queries using query profile.

[View query profile Snowsight](#)

Query profile provides details of the query execution plan along with the granular details of each step of query execution. This is a powerful tool to use query plan and execution. You can use this to analyze the query further and understand the technical issues in the queries. You can work on query performance and tuning required to re-write the SQL query once you have technical issues using query profile.

Query profile provides a brief overview of the plan with the following sections:

- Query execution plan
- Operator node
- Query profile navigation
- Information panes

You can access query plan by following the given steps:

1. Logon to Snowsight (account URL)
2. Go to **Activity | Query_History**
3. Select or click on any **query_id** from this view.
4. Go to the **Query Profile** tab.
5. You can view the query profile as shown below:

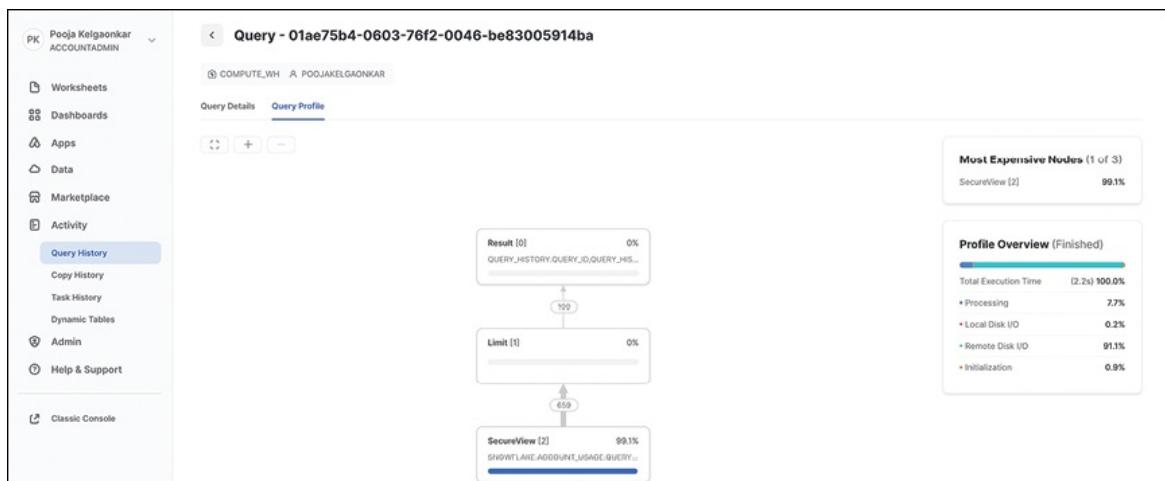


Figure 9.5: Query Profile View

You can use query profile view to analyze the queries and technical issues to rewrite SQL. You need to identify the queries before you start analyzing the queries using query profile. You can identify and bucket the queries for performance review using the **QUERY_HISTORY** view. You can run a few queries to generate the metrics as well as queries that qualify the threshold values for analysis. You will learn more about query performance in the next section.

Listing queries for performance optimization

You can run queries on the **ACCOUNT_USAGE** schema and **QUERY_HISTORY** view to list the queries and bucket them based on their execution time and compilation time. Below are some of the sample queries to list the queries that take longer

time to execute. You can use the view and columns to derive your own set of metrics and list down the queries for optimization.

- **Top 20 longest running queries to date:**

To get these queries, there are a few pre-requisites and assumptions:

- Queries that have taken more than 0 milliseconds to run (this is required to identify the queries that required compute to execute).
- Queries are successful. Failed queries are not considered for optimization.
- Queries resulting from cache do not fall into the optimization category.
- List only the top 20 queries to start the analysis.

You can use `QUERY_HISTORY` view and columns to satisfy the above conditions to list down the queries that need to undergo optimization:

```
SELECT
    query_id,
    ROW_NUMBER() OVER(ORDER BY partitions_scanned
DESC) AS query_rank,
    total_elapsed_time/1000 AS
    query_execution_time_seconds,
    partitions_scanned,
    partitions_total,
FROM snowflake.account_usage.query_history
WHERE warehouse_name = 'DEV_POC_WH'
AND error_code IS NULL
AND total_elapsed_time > 0
AND partitions_scanned IS NOT NULL
AND TO_DATE(Q.start_time) >
DATEADD(day, -1, TO_DATE(CURRENT_TIMESTAMP()))
```

```
ORDER BY total_elapsed_time desc  
LIMIT 20;
```

You can also use the rank derived in this query to list the top queries. Here, the limit clause is used to restrict the output to 20 rows. You can use `query_rank` to get the top 5, top 10, or top 50 queries for analysis.

- **Queries running for more than one hour:** Get a history trend. You need to consider the above conditions as well to analyze these queries:

```
SELECT  
query_id,  
total_elapsed_time  
FROM snowflake.account_usage.query_history  
WHERE  
total_elapsed_time >= 3600000  
AND total_elapsed_time > 0  
GROUP BY 1;
```

You can use `start_time` view column to bucket the duration: month, week, or day for queries. You can also bucket the queries using execution duration like queries less than 30 minutes, queries greater than 30 minutes, and queries less than 60 minutes etc. You can customize it based on your query trend and requirements of optimization. You can also use query acceleration service, query acceleration eligible view, or system function in case the query is technically correct and cannot be rewritten to optimize it further. You can optimize such workloads using Snowflake optimization services. You have learned about query optimization techniques in this section. You will learn more about warehouse optimization in the next section.

Warehouse performance optimization

As you know, the warehouse layer or compute layer is the processing layer of Snowflake. All your workloads that run on Snowflake need a warehouse associated to execute the processes. You can leverage the cache: result cache, warehouse cache, or metadata cache, to execute some of the queries that do not

need warehouse. However, such queries are a very small percentage in comparison with the overall workload being run on Snowflake. You also need to consider warehouse optimization to optimize the workload and warehouse configurations, along with query optimization.

Warehouse performance can be measured using the **WAREHOUSE** view in the **ACCOUNT_USAGE** schema. You can review the performance, warehouse configurations, workload concurrency, and warehouse compute credits using these views. You can adjust the warehouse by resizing or reconfiguring, and execute the workloads to compare the performances. You can use the **execution_time** column in the **query_history** view to compare the execution times before and after. Snowflake recommends using the following strategies to optimize the warehouse and improve the performance of queries:

- Reducing queue time
- Resolving or avoiding memory spillage
- Changing warehouse size (resize)
- Using query acceleration service
- Optimizing warehouse cache
- Limiting concurrent workloads or queries

You will learn more about each of these strategies in the upcoming section of this chapter.

Reducing queue time

This is one of the warehouse optimization strategies where you can reduce the queue time of the queries or workloads. You can get the queue time from the **query_history** view. When you run multiple concurrent queries on the warehouse, the resources get exhausted. If you are still submitting more queries to be run on the warehouse, then these queries end up in the execution queue until resources are made available. You can identify the queue from Snowsight or running queries.

1. Finding queue using Snowsight:
 - a. Logon to Snowsight.
 - b. Go to **Admin | Warehouses**.

- c. Select a **Warehouse**.
 - d. You can see the activity chart and associated color to identify queued loads.
2. Running query to get details:

```

SELECT
    warehouse_name,
    TO_DATE(start_time) AS date,
    SUM(avg_queued_load) AS avg_queued,
    SUM(avg_running) AS avg_run_time
FROM
    snowflake.account_usage.warehouse_load_history
    where avg_queued_load > 0
GROUP BY 1,2;

```

Once you identify the queries that are most queued and the warehouses where these are getting queued. You can use some of the following strategies to reduce the queue:

- **For standard warehouses:** You can consider adding one more warehouse based on the workloads and split the loads between warehouses. For example, you can have one warehouse for all load activities and one for transformation queries or other queries. You can also consider converting a standard warehouse to a clustered warehouse. Standard warehouses cannot be scaled if the workload increases or there is a spike in executions. You can use the clustered warehouse to replace the standard, as this can scale only if needed, depending on the workload.
- **For clustered warehouses:** You can consider changing the cluster size by adding more clusters to the warehouse. You can change it from 6 to 8 or 8 to 10 nodes in a cluster.

Resolving memory spillage

Performance gets impacted if the warehouse runs out of memory while

executing the query because it has to spill the memory to the local disk. If your query needs even further memory, then it spills to the cloud storage by the cloud provider. It then results in the worst performance. You can use **QUERY_HISTORY** view to identify the queries that get spilled by running the following query:

```
SELECT
    query_id,
    bytes_spilled_to_local_storage as spilled_bytes,
    bytes_spilled_to_remote_storage as storage_bytes,
    user_name,
    warehouse_name
FROM snowflake.account_usage.query_history
WHERE (bytes_spilled_to_local_storage > 0
       OR bytes_spilled_to_remote_storage > 0 )
ORDER BY bytes_spilled_to_remote_storage,
bytes_spilled_to_local_storage DESC
LIMIT 20;
```

You can avoid memory spillage by converting your warehouse to a Snowpark-optimized warehouse. If you do not want to change the warehouse type, then you can use query profile to identify which portion of the query is being spilled and optimize the query.

Changing warehouse size

As you know, a warehouse is nothing but the compute available to execute the query or workload. The bigger the warehouse size, the more resources are available to run the queries. You can consider changing the size of the warehouse if there are queries getting queued and waiting for resources. You can consider the warehouse size and uptime used to calculate the cost of the warehouse. It is the same for small warehouses running for long or large warehouses running for an equivalent smaller duration. The cost will be the same, considering the execution time and resources required to complete the queries or workloads.

You can use the **WAREHOUSE_LOAD_HISTORY** view to identify warehouses that

qualify for resizing:

```
SELECT  
    warehouse_name,  
    SUM(avg_running) AS avg_run_time,  
    SUM(avg_queued_load) AS avg_queued_time  
FROM snowflake.account_usage.warehouse_load_history  
where avg_queued_load > 0  
GROUP BY 1,2;
```

You can also use `start_time` to get the trend or usage for a week, month, or for a specific duration to evaluate warehouse queued times.

Optimizing cache

As you know, there are three types of cache that are maintained and used for query optimization. Warehouse maintains cache of table data, and if you are running subsequent queries, they can consume table data cache instead from the table. Warehouse cache is active and maintained only till the time is up and running. If the warehouse is suspended, then the corresponding cache is also deleted.

You need to consider the cache availability, queries executed, and the warehouse auto-suspend set up. It costs the same if you keep the warehouse in a run state or read data from the storage layer. You have to calculate the cache usage and warehouse auto schedule to maintain the cache and lower cost as well as the performance of queries. You can identify such queries using the following query:

```
SELECT  
    warehouse_name  
    ,SUM(bytes_scanned) AS sum_bytes_scanned  
    ,SUM(bytes_scanned*percentage_scanned_from_cache) /  
    SUM(bytes_scanned) AS percent_scanned_from_cache  
    ,SUM(bytes_scanned*percentage_scanned_from_cache) AS  
    sum_bytes_from_cache
```

```
, COUNT(*) AS count  
FROM snowflake.account_usage.query_history  
WHERE  
bytes_scanned > 0  
GROUP BY 1  
ORDER BY 3 DESC;
```

Limiting concurrent queries

Warehouse provides the resources required to execute queries. The parallel queries executed on warehouse consume more or all resources, hence putting the rest of the queries in a queue. The queries in queue wait for the resources to be available to execute. You can limit the number of concurrent runs on a warehouse using the **MAX_CONCURRENCY_LEVEL** parameter. You can use the below command to set the level:

```
ALTER WAREHOUSE DEV_POC_WH SET MAX_CONCURRENCY_LEVEL = 4;
```

If you lower the limit of concurrent runs, then you can see an improvement in the performance of queries. This can be implemented especially for larger complexes and multi-statement queries. This also leads to putting up more queries in the queue; hence it is recommended to test it thoroughly to ensure you get the desired result.

Using query acceleration service

You can use the system function or the eligible view to get the list of queries that can be accelerated. You can refer to the above section to revisit the acceleration service and its benefits.

With warehouse or compute optimization strategies, you can optimize the performance of the warehouse and reduce the query wait time and query performance by reducing the execution time. The next important optimization is storage optimization. You will learn more about this in the next section.

Storage performance optimization

This section focuses on optimizing the performance of queries using storage

optimization techniques. Snowflake offers different strategies to optimize the storage performance. These strategies can be used to optimize the performance of queries. However, the same cannot be applicable to queries that are already running less than a second or faster. There are three different strategies:

- Automatic clustering
- Materialized views
- Search optimization

You will learn more about these strategies, applying them, and using them to optimize performance in this section.

Automatic clustering

Snowflake stores table data in the form of micro partitions. Snowflake organizes the data based on the dimensions of the data. If any query uses these columns to join, aggregate, or filter records, then Snowflake uses micro partitions and reads only required partitions eliminating other partitions from the scan. This helps to improve the performance and result query faster.

Snowflake also clusters the data based on micro partitions and dimensions, and this is referred to as cluster key. Snowflake does this automatically; however, you can also define the clustering key. Cluster key is used to organize the data within micro partitions. The clustering key can improve the performance of a query that contains filters, joins, or aggregates by the column defined in cluster key. Once automatic clustering is enabled, it updates micro partitions as soon as new data is loaded to the table.

Materialized views

Materialized views are pre-calculated data stored from a **SELECT** statement. Queries running on materialized views have better performance as they run on pre-calculated aggregated data. These are used to simplify the processing of queries. You can implement these views to improve the performance of complex queries with aggregations wherever you need to generate data from multiple tables joined together to result in smaller datasets. These views can be implemented on a single table. You cannot implement them on more than one table. This is an enterprise edition feature and above.

Search optimization service

This is the optimization service that improves the performance of queries that use point lookup and returns a small number of rows with selective filters to filter table data. This is recommended to be used when it is critical to have a low latency lookup. This is optimized by building a persistent data structure that can be used for a particular type of search.

You can enable this service for a table or specific columns. The only required pre-requisite is a column with selective values, substring searches, geo searches, and equality searches against those columns. This can be used for structured as well as semi-structured data. This is an enterprise edition and the above feature.

Choosing the right strategy

Query optimization and the benefits of query performance improvement are dependent on the strategies defined above. Based on the strategies, different types of queries or workloads are benefited. You can pick up the strategy using the following techniques:

- Automatic clustering is used to get benefits from a range of queries that access the same set of columns from a table. Typically, the admin analyzes and picks up the queries depending on the frequency and latency requirements to choose the clustering key. Clustering key can be used to maximize the performance of queries where filters, joins, or aggregates on the same set of columns.

You can choose the clustering key based on the frequent queries and columns of the table being used frequently. This strategy is used for buckets of queries that are of similar types of workloads.

- Search optimization, as well as materialized views, have limited scope. This can be used for tables and queries where the subset of table data is accessed. You can use building persistent data or materialize them using materialized views. If you want to apply multiple strategies to improve the query performance, then you can get started with automatic clustering or search optimization, as this could probably improve the performance of similar queries on the table.

You can use the strategies to improve the performance of the queries and apply techniques to benefit other queries running on a table accessing the same set of columns.

Performance considerations

You can consider these strategies and techniques to compare performance optimization. You can follow the above techniques to get performance improvement and compare the benefits of these techniques:

Technique	Considerations
Automatic clustering	Huge performance benefit with WHERE clause to filter on the column of cluster key.
	This can also improve the performance of other SQL clauses and functions in the same column.
	Recommended to be used for queries that have inequality filters. This can also be used for an equality filter; however, search optimization gives better performance over clustering in an equality filter.
	Available in Snowflake's Standard Edition
	Only one cluster key can be defined on the table. If different queries use different columns than the cluster key, then you can use a search optimization service.
Search optimization service	Improves performance of lookup queries that return a small number of rows.
	Can be used to support lookup queries that uses LIKE and RLIKE . Search specific fields in VARIANT, ARRAY, or OBJECT Use Geo functions for GEOGRAPHY values.
Materialized views	Used to improve expensive and frequent calculations, for example, aggregates, and analysis of semi-structured data.
	Focused on a subquery or specific query calculation.
	Used to improve the performance of queries against external tables.

Table 9.6: Optimization technique and benefits

You can use these strategies to improve the performance of queries as well as workloads. You can follow the analysis steps defined in the section and identify queries from **USAGE** views. You can bucket the queries depending on the patterns of data access and apply the storage optimization techniques using the considerations mentioned in [Table 9.5](#).

Conclusion

This chapter provided a summary of Snowflake's performance optimization. It covered information on various **USAGE** schemas, metadata views, and functions. This chapter also covered various performance optimizations that can be implemented in Snowflake, such as query performance, warehouse performance, and storage performance optimization. It also helped you to understand various techniques as part of query performance, warehouse performance, and storage performance optimization. You will learn more about query analysis, warehouse

workload analysis, and considerations to implement optimization strategies. You will learn more about Snowflake **USAGE** schema and queries that can be used to identify workloads, and queries for optimization in the upcoming chapters.

In the next chapter, you will learn about Snowflake's costing and billing utilization. You will learn more about computing usage, credit consumption, and understanding the billing components of Snowflake.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake shares metadata information using the Snowflake database.
- Snowflake database contains **INFORMATION_SCHEMA**, **ACCOUNT_USAGE** and **READING_ACCOUNT_USAGE** schemas.
- Snowflake performance optimization includes three types of optimizations: Query performance, Warehouse performance, and Storage performance.
- Snowflake's query performance analysis can be done using the **QUERY_HISTORY** usage view.
- Snowflake's warehouse performance analysis can be done using **WAREHOUSE_LOAD_HISTORY**, and you can use optimization techniques to resize or reconfigure the clustered warehouses.
- Snowflake's storage analysis can be done using the **QUERY_HISTORY** view by calculating bytes of data spilled to the disk or cloud storage.
- Snowflake also offers query acceleration service, search optimization service, materialized views, and automated clustering to optimize query performance.

Questions

1. How can Snowflake usage schemas be used?
2. How can you use **WAREHOUSE** usage views to calculate the performance of warehouse?
3. How does Snowflake's **query_history** cater to both storage and query performance analysis?

4. How does query profile help to identify the query to be optimized?
5. Can you calculate storage spillage for a query?
6. How can you optimize concurrent workloads?
7. How to optimize warehouses for similar workloads as well as different types of workloads?

Multiple choice questions

1. **What is the schemas present in the Snowflake database? (Select all applicable)**
 - a. INFORMATION_SCHEMA
 - b. ACCOUNT_USAGE
 - c. READER_ACCOUNT_USAGE
 - d. All of the above
2. **What are the query profile sections? (Select all applicable)**
 - a. Steps
 - b. Operator tree
 - c. Node list
 - d. All of the above
3. **What are the query profile sections? (Select all applicable)**
 - a. Steps
 - b. Operator tree
 - c. Node list
 - d. All of the above
4. **What are the storage optimization techniques? (Select all applicable)**
 - a. Search Optimization Service

- b. Materialized View
- c. Automated Clustering
- d. All of the above

Answers

1	d
2	d
3	d
4	d

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 10

Understanding Snowflake Costing and Utilizations

Introduction

Snowflake platform is a three-layered architecture. Platform cost consists of three components cost as per architecture design. Snowflake cost is calculated as total with warehouse cost, storage cost, and Cloud services cost. Snowflake cost also includes serverless services cost and performance optimization services usage. Cost is the most critical component of platform design, implementation, usage, and maintenance. This chapter guides readers on developing an understanding of cost measures, cost optimization techniques, and the need to implement cost optimization.

Snowflake platform cost is measured in terms of credits consumed by each layer. These details are available and captured in the metadata views. These metadata views store information on query usage, storage usage, warehouse usage, and serverless usage. You can leverage the metadata views to analyze the performance of your Snowflake account and compute Snowflake credits consumed every month.

Structure

This chapter consists of the following topics:

- Understanding Snowflake costing
- Component costing in Snowflake

- Using Snowflake metadata objects
- Monitoring Snowflake costs

Objectives

By the end of this chapter, you will be able to develop an understanding of the Snowflake platform cost measure, cost optimization techniques, and the need to implement cost optimization. This also covers understanding of Snowflake metadata views that help to measure credits as cost consumption of the platform. Snowflake offers a set of features to improve cost performance as well as recommend cost optimization techniques. You can implement cost optimization and usage queries using the trial account setup in [*Chapter 1: Getting Started with Snowflake*](#).

Understanding Snowflake costing

Snowflake's three-layered architecture offers services that are essential to support various workloads. Snowflake costing includes the usage of Cloud Services Layer, Compute Layer, Storage Layer, and Serverless services usage. Snowflake costing can be managed efficiently by following a costing framework. The cost framework consists of three parts – control, visibility, and optimization, as represented below:

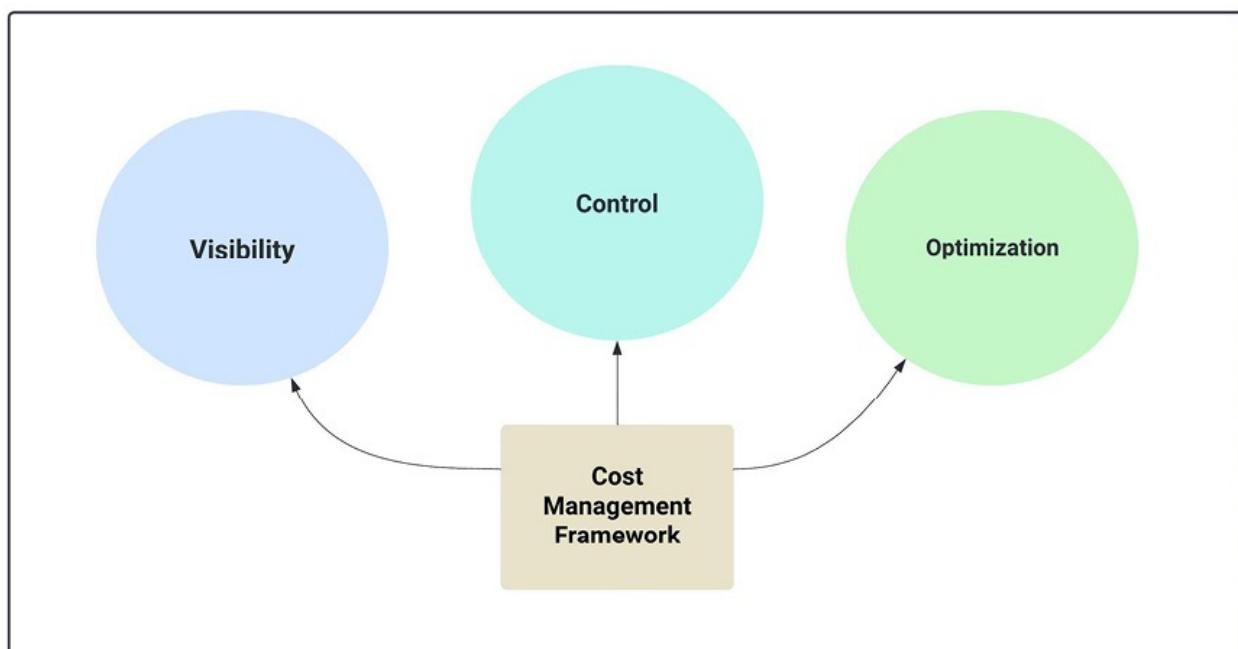


Figure 10.1: Costing framework

Managing cost using the costing framework offers features that help to minimize the total cost. Parts of the costing framework are explained in the following section.

Visibility

This part of the framework provides an understanding of cost and the ability to analyze the cost in detail. As the name says, this includes the cost to the right objects and services in the organization and monitoring the cost to avoid any additional or undefined costs. Below are the benefits and features of this part of the framework:

- **Understanding the cost:** To gain more visibility of the Snowflake cost, one starts with understanding the basics of the cost. Snowflake cost consists of different types of usage that consume credits as well as factors that attribute to the cost of overall Snowflake resources based on their consumption.
- **Exploring the cost:** You can start exploring the cost in detail once you develop an understanding of the cost components and cost accumulated in the Snowflake account or organization. You can use Snowsight – web UI to explore the usage and use pre-built dashboards that help to visualize the credit consumption and usage of the Snowflake account. You can also use **USAGE** schema views and write custom queries to analyze the overall credits consumption as well as the dedicated consumption of various workloads.
- **Attribute of the cost:** This is the next part of visibility, which helps you to identify various stakeholders or teams that contribute to the Snowflake usage and use the platform. Once you have visibility for cost, understand the components of the usage, and explore the usage with workloads, you can start adding chargebacks to the teams based on their consumption. For example, if your Snowflake platform is used by Development, BI, and QA teams, then you can create chargebacks to these teams based on their consumption of the platform.
- **Monitoring the cost:** Now, you know the usage and attribute the usage and overall spending of your account. You can create monitoring, visualizations and setup alerting to notify teams based on their spending.

You can also define thresholds based on their average consumption to generate alerts for the attributing teams. You can also use resource monitors to take action to suspend the warehouses depending on the consumption to avoid overspending of credits.

Control

As the name represents, this is to control and put safeguards to avoid any unexpected cost. Snowflake allows users to set up guardrails using its native features and objects to prevent any unexpected spending. This implementation can include setting up query execution time, that is, how long a query can run before it gets terminated, how long warehouses can be up, or how often warehouses can be spun down automatically, and so on. Implementing such measures and controls can help you to reduce as well as avoid any additional costs.

Optimization

Optimization is nothing but improving the areas where you can see a scope for improvement. This can be in terms of query performance, warehouse performance, and storage performance to reduce the overall execution time and cost. Snowflake offers native metadata **USAGE** views that provide all the required information to identify and optimize such resources. You can use the following resources to optimize the performance of your Snowflake account:

Category	Description	Metadata views
Query Performance Optimization	You can enhance the performance of queries by analyzing the detailed information of execution from metadata view.	Query_history - View in USAGE schema Query_profile - Query profile to view the execution plan and steps
Warehouse Performance Optimization	You can analyze the performance of existing warehouses, the size of warehouses, and workloads on warehouses using detailed information from metadata views.	Warehouse_Load – from Query_history Warehouse_Metering – to get credits consumed
	Analyze the events of the warehouse – suspend and resume to identify the cost of cache vs cost of the warehouse for optimization.	Warehouse_events_history – USAGE view

Table 10.1: Optimization features

The cost framework of Snowflake – Visibility, Control, and Optimization is

helpful for analyzing and enhancing the performance of the platform as well as controlling the cost of the account. You will learn more about Snowflake costing features, credits consumption, and using USAGE views to monitor cost spend in upcoming sections. The next section will help you to understand Snowflake's cost components and use them to derive the overall cost of the account.

Component costing in Snowflake

As you know, Snowflake is a three-layered architecture and offers many serverless services like Snowpipe, Query acceleration, and search optimization services. Your account usage is the total usage of Snowflake components being used. Snowflake is data on cloud implementation, and billing of the account is done as per the usage. You can host your workloads and store your data in Snowflake to serve your various workloads. Your overall account bill is computed in the form of Snowflake credits, and it is calculated by adding your warehouse cost, storage cost, cloud services cost, and serverless cost. Snowflake follows a hierarchy of resources. If you have multiple accounts in one organization, then you can view the overall usage at the organization level as well. Below are the components that contribute to the overall cost:

- Cloud services cost
- Compute or warehouse cost
- Storage cost
- Serverless cost
- Data transfer cost

You will learn more about each of these components in this section.

Cloud services cost

Cloud services layer is the first layer of architecture that performs various operations as part of services that tie different components of Snowflake processing. Cloud services takes care of services like logon, user access – authentication and authorization, query plan, query cache, metadata management, and so on. The compute resources required to perform cloud services are managed by Snowflake. Cloud services are also billed the same as compute or warehouse resources. Snowflake credits are also used to pay the cloud services usage.

Cloud services are free for up to 10% of the compute credit consumption. This is charged only when the compute exceeds 10% of the daily warehouse usage. The consumption is calculated daily to ensure the charges are calculated and applied accurately every day. These are some of the important factors to remember:

- Serverless services are not part of the cloud services 10% bucket.
- The cloud services 10% bucket is calculated on a daily basis. This is calculated by multiplying the warehouse usage by 10%.
- The overall monthly adjustment is similar to the sum of daily adjustments.

Snowflake uses credits to calculate the overall cost and consumption. Snowflake credit is nothing but a unit to measure Snowflake consumption. You can use **USAGE** views to view the consumption in credits. Snowflake credits are associated with \$ value depending on the Snowflake edition and pricing to convert credits to equivalent \$ value.

Compute cost

Compute in Snowflake is also referred to as virtual warehouses. The credit consumption is calculated based on the warehouse used to perform queries, data loads, and DML operations as part of different workloads. The warehouse cost is calculated based on its uptime in seconds. The moment the warehouse is spun up by the default value of 1, minimum billing is calculated and added to the warehouse credit consumption. Warehouse billing is calculated based on its usage – the time the warehouse is up and running, warehouse size etc.

As you know, warehouse comes in different sizes – standard and clustered warehouses. The credits vary based on the size of the warehouse. The credits consumed doubles based on the warehouse size as it grows. Snowflake offers two types of warehouses – Standard and Snowpark optimized warehouses. The following table represents the warehouse size and the credits consumed per hour:

Type of warehouse	Extra Small (XS)	Small	Medium	Large
Standard Warehouse	1	2	4	8
Snowpark- Optimized Warehouse	N/A	N/A	6	12

Type of warehouse	Extra Large (XL)	2XL	3XL	4XL	5XL	6XL
Standard Warehouse	16	32	64	128	256	512

Snowpark Warehouse	24	48	96	192	384	768
--------------------	----	----	----	-----	-----	-----

Table 10.2: Warehouse credit consumption chart

Compute credits are consumed only when the warehouse is in running. Users are not billed when the warehouses are in suspended state. The table values represent credit consumption for an hour, and the compute billing is calculated based on the usage per second.

Storage cost

Storage cost is calculated based on the overall storage of the data – in tables, stages as Snowflake objects, data storage for fail safe and time-travel, data replicated, data cloned etc. The storage cost is calculated on a monthly basis at a flat rate of per TB of data. This is calculated as the daily average of the data and the cost is charged based on the type of Snowflake account and region. Below are the components of the storage cost:

- Database data stored in tables
- Data recovery storage – Time travel and fail safe
- Data cloned
- Data stored in stages for data load and unload

Data is stored in the form of tables. Storage cost stores data stored in the table and historical data as part of time travel. The data stored is in compressed format, and compressed size is used to calculate the database cost. Data stored in permanent tables and historical data stored as part of time travel incurs cost. You can reduce the database cost by using temporary and transient tables. You can use these tables to store intermediate data as part of data processing. Transient and temporary tables are used to store data for a session or temporary basis. These tables do not have time travel and fail safe hence not adding any cost to the overall storage cost.

Serverless cost

Serverless compute is the compute consumed by serverless services. This usage is of Snowflake managed resources. These are managed and automatically resized, scaled up and down based on the demand of each workload. The usage is similar to serverless compute usage of cloud, where you are billed only for the compute usage for a given duration. This is an efficient model in comparison

with warehouse billing as this consumes usage while executing the workloads, whereas user-managed warehouses cost if they are in a running state even though they execute any workloads or not.

The overall cost of serverless is calculated based on the compute hours. Compute hours are calculated based on the usage per second. The serverless charges are computed and mentioned as a separate line item of the billing. The serverless costing is shown on the web UI usage page. Snowflake offers a variety of serverless features, as follows:

Features	Consumption
Automatic clustering	Resources used to maintain clustered tables in the background.
External tables	Automatic data is refreshed with metadata and the latest set of files in the external stage.
Materialized views	Resources used to sync data automatically in the background.
Query acceleration service	Resource usage for the execution of the part of the queries.
Replication	Automated data copy between accounts.
Search optimization service	Resource usage to maintain automated background search access paths.
Snowpipe	Automated processing with pipe.
Snowpipe streaming	Snowpipe streaming cost.
Tasks	Snowflake tasks that execute SQL code.

Table 10.3: Serverless services and usage

Data transfer cost

Data transfer is nothing but the process of loading or unloading data in the account. The charges are calculated based on the data ingress and egress. Snowflake charges per byte fee for data egress into different regions on the same cloud platform or a different cloud platform. Note that data transfers between the same region are free. Following are the Snowflake features that can incur the transfer cost:

- **Data unloading:** Data unloading is done using `COPY INTO` to cloud storage services like Amazon, Google Cloud Storage, or Microsoft Azure. The data unloading to an external stage also adds up to the overall cost as storage costs along with egress charges.

- **Data replication:** Data snapshots to another database as a secondary database.
- **External functions:** External functions can be used to transfer data from Snowflake to the cloud providers- Amazon, Microsoft Azure, or Google Cloud.
- **Cross cloud auto fulfillment:** This is incurred when you are using listings in other cloud regions.

These are the Snowflake components that incur the cost. You can view the overall usage by leveraging the Snowflake USAGE metadata views. The next section focuses more on using metadata objects and leveraging them to derive the overall consumption.

Using Snowflake metadata objects

As you know and learned in [Chapter 9: Snowflake Performance Optimization](#), USAGE views can be used to derive the consumption of Snowflake services. You can use these metadata views to generate the overall cost and set up the alerting to avoid any unexpected costs.

View compute usage

You can use warehouse views: `warehouse_metering_history`, `warehouse_load_history` as well as `query_history` to identify the workloads on a warehouse and calculate cumulative usage of the services and workloads. Following are some of the reference use cases:

- **Hourly usage of warehouse for the previous day:**

```

SELECT
    warehouse_name,
    DATE_PART('HOUR', start_time) AS hour_ofthe_day,
    credits_used_compute
FROM
    snowflake.account_usage.warehouse_metering_history
WHERE start_time >= DATEADD(day, -1,
```

```
CURRENT_TIMESTAMP())
    AND warehouse_id > 0
ORDER BY 1 DESC, 2;
```

Here, the `warehouse_id` condition helps to filter out queries that need a warehouse to execute queries.

- **Usage of warehouse trend for a month:**

```
SELECT
    warehouse_name,
    DATE_PART('DAY', start_time) AS DAY_ofthe_month,
    credits_used_compute
FROM
    snowflake.account_usage.warehouse_metering_history
WHERE start_time >= DATEADD(day, -30,
CURRENT_TIMESTAMP())
    AND warehouse_id > 0
ORDER BY 1 DESC, 2;
```

- **Usage of warehouse till date:**

```
SELECT
    warehouse_name as warehouse,
    sum(credits_used_compute) as credits_consumed
FROM
    snowflake.account_usage.warehouse_metering_history
GROUP BY 1
ORDER BY 2 DESC;
```

You can use the `start_time` column of the view to add date conditions to filter data for a month, week, year, and so on.

View Cloud services usage

Cloud services usage can be viewed in `query_history`. Following are some of the reference use cases:

- **Usage of services credits per query type for the previous day:** The `credits_used_cloud_services` are the view column in `query_history` that represents usage of cloud services layer:

```
SELECT
    query_type,
    SUM(credits_used_cloud_services) AS
    cloud_services_credits,
    COUNT(1) num_queries
FROM snowflake.account_usage.query_history
WHERE true
    AND start_time >= TIMESTAMPADD(day, -1,
CURRENT_TIMESTAMP)
GROUP BY 1
ORDER BY 2 DESC;
```

- **Listing warehouses with high cloud service usage:**

```
SELECT
    warehouse_name,
    SUM(credits_used) AS credits_consumed,
    SUM(credits_used_cloud_services) AS
    credits_for_cloud_services,
    SUM(credits_used_cloud_services)/SUM(credits_used)
    AS percent_of_cloud_services
FROM
```

```

snowflake.account_usage.warehouse_metering_history
WHERE TO_DATE(start_time) >=
DATEADD(month, -1, CURRENT_TIMESTAMP())
    AND credits_used_cloud_services > 0
GROUP BY 1
ORDER BY 4 DESC;

```

View serverless usage

To view the usage of serverless services of Snowflake, you can query corresponding views from the **USAGE** schema. Refer to the below sample queries to list down the usage of services:

- **Viewing usage of automatic clustering history:** You can use **automatic_clustering_history USAGE** view to get details of automatic clustering service usage:

```

SELECT
    database_name,
    schema_name,
    table_name,
    TO_DATE(start_time) AS date,
    SUM(credits_used) AS credits_used
FROM
    snowflake.account_usage.automatic_clustering_histo
WHERE start_time >=
DATEADD(month, -1, CURRENT_TIMESTAMP())
GROUP BY 1,2,3,4
ORDER BY 5 DESC;

```

- **Viewing usage of search optimization history:** You can use **search_optimization_history USAGE** view to get details of search

optimization history service usage:

```
SELECT
    database_name,
    schema_name,
    table_name,
    TO_DATE(start_time) AS date,
    SUM(credits_used) AS credits_used
FROM
    snowflake.account_usage.search_optimization_histories
WHERE start_time >=
    DATEADD(month, -1, CURRENT_TIMESTAMP())
GROUP BY 1,2,3,4
ORDER BY 5 DESC;
```

- **Viewing usage of query acceleration history:** You can use `query_acceleration_history USAGE` view to get details of query acceleration service usage:

```
SELECT
    warehouse_name,
    TO_DATE(start_time) AS date,
    SUM(credits_used) AS total_credits_used
FROM
    SNOWFLAKE.ACCOUNT_USAGE.QUERY_ACCELERATION_HISTORY
WHERE start_time >= DATE_TRUNC(month,
CURRENT_DATE)
GROUP BY 1,2
ORDER BY 2 DESC;
```

- **Viewing usage of tasks:** Snowflake tasks are serverless components for running an SQL to perform a particular task at a given schedule:

```
SELECT  
    task_name,  
    start_time,  
    end_time,  
    credits_used,  
    schema_name,  
    database_name  
FROM  
snowflake.account_usage.serverless_task_history  
ORDER BY start_time;
```

These are some of the references for viewing and using **USAGE** views to list down the usage of various components of Snowflake cost. The next section covers monitoring the cost and taking automated actions based on the limit of monitors.

Monitoring Snowflake costs

You can monitor the usage and spending using Snowsight Web UI as well as by using resource monitors. You can set up monitoring dashboards with widgets to represent compute, storage, cloud services, and workload usage.

Resource monitor is used to monitor the credit usage by warehouse and cloud services. You can set up the action on the threshold limit to suspend the warehouse. Resource monitors are also treated as Snowflake objects with the following set of properties:

- **Credit quota:** This is used to specify the Snowflake credits allocated to the monitor. This is used to track both virtual warehouse credits as well as cloud services credits. The overall quota consists of compute usage, services usage, and serverless usage to be tracked.
- **Monitoring level:** The level can be defined to be used for an account or

set of warehouses in an account. This is a required property. If it is not set, then the monitor remains dormant.

- **Monitoring schedule:** You can assign a schedule to the monitors. The schedule is set up and run for a month, and it gets reset every first of the month after that. It supports frequencies like daily, weekly, monthly, yearly, or never. You can also set the start time to start immediately or at a given time. The start time represents the start of the resource monitor, and the end time represents the end of the monitor.
- **Monitoring actions:** Resource monitors support a set of actions as listed below:
 - **Notify and suspend:** Notifies and suspends the warehouse once all loads on the warehouses are finished. It waits for the workload to be finished.
 - **Notify and suspend immediately:** Notifies and suspends warehouse immediately.
 - **Notify:** Just send a notification on the usage with no action to be taken.
- **Using monitoring:** You can set up resource monitors at an account, at a user, or for warehouses to monitor credit consumption. You can set it up for a warehouse as well as for a set of warehouses with a resource monitor.

You can refer to *Figure 10.2* and have multiple resource monitors set up for an account:

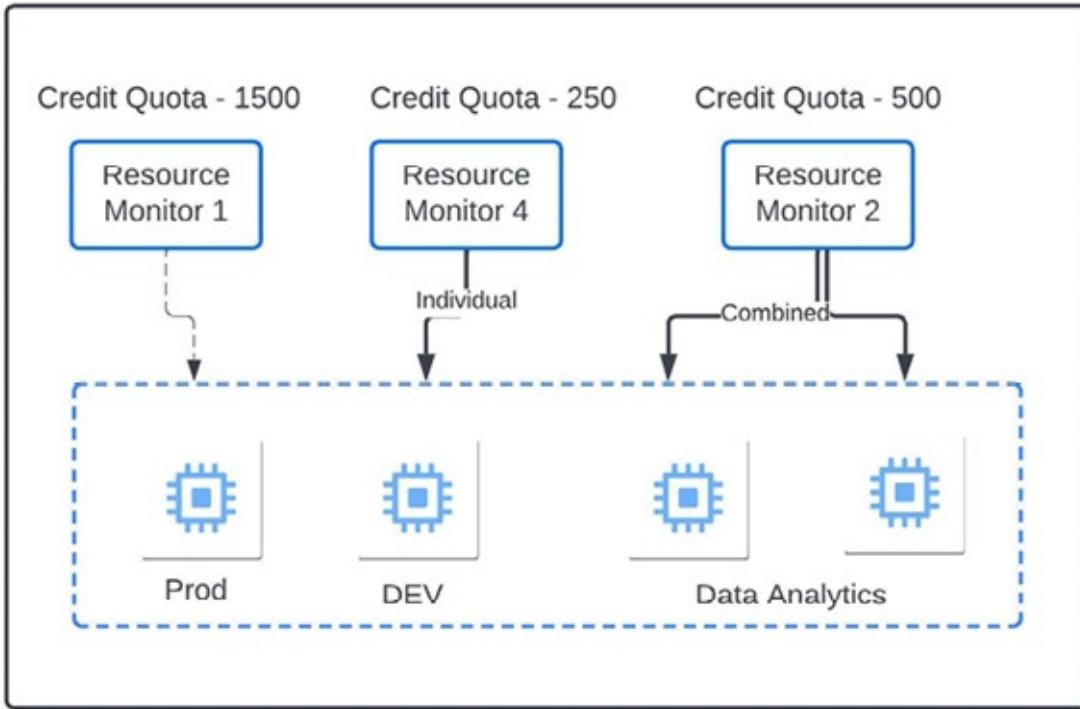


Figure 10.2: Resource Monitor Setup

The resource monitors setup based on the above diagram is explained as follows:

- The overall credit quota for an account is 1500 and managed by resource monitor1. Once this is reached, the action defined in the resource monitor is applied to all warehouses in the account.
- The individual monitor 4 is defined only on one warehouse for 250 credits.
- The data analytics team has 2 warehouses, and they can consume a maximum of 500 credits.

Resource monitor can take action to suspend the warehouse, and if the warehouse is suspended then it cannot be brought back using an auto resume. If you get to use the warehouse in case of exhausted credits, then the resource monitor will not allow you to use the warehouse. You have to change the quota or remove it from warehouses or create a new one without a monitor to be used by teams.

Conclusion

This chapter provided a summary of Snowflake's cost components, costing

optimization, and monitoring costs. This chapter covers various components of Snowflake cost, and how users can use **USAGE** schemas, and metadata views to view their consumption. This also covered cost optimization strategies to reduce the compute cost. You can also monitor the usage using resource monitors that can be implemented in Snowflake.

In the next chapter, you will learn about Snowflake's cost optimizations. You will learn more about overall usage, credit consumption, and optimizing usage of Snowflake.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake cost is associated with compute cost, storage cost, cloud services layer, and serverless services.
- Snowflake database contains **USAGE** schemas and a bunch of views that can be used to view detailed information on Snowflake consumption.
- Snowflake's serverless cost is associated with all serverless services whose compute is managed by Snowflake.
- Snowflake's cost analysis can also be done using **QUERY_HISTORY** usage view to view various credit components.
- Snowflake's storage analysis can be done using **QUERY_HISTORY** view by calculating bytes of data spilled to the disk or cloud storage.
- Snowflake serverless credit consumption includes Query acceleration service, search optimization service, materialized views, and automated clustering to optimize query consumption using corresponding **USAGE** views.

Questions

1. How can Snowflake usage schemas be used?
2. How can you use **WAREHOUSE_METERTING_HISTORY** usage views to calculate the performance of the warehouse?
3. How does Snowflake's **query_history** cater to compute performance analysis?

4. How can **query_history** offer to compute vs. cloud services consumption?
5. Can you calculate the usage of storage for external functions?
6. How can you monitor Snowflake costs?
7. How can you view and set up the monitoring dashboards for the Snowflake consumption using **USAGE** views?

Multiple choice questions

1. **What are the USAGE views for serverless services in the Snowflake database? (Select all applicable)**
 - a. QUERY_ACCELERATION_SERVICE
 - b. SEARCH_OPTIMIZATION_SERVICE
 - c. READER_ACCOUNT_USAGE
 - d. All of the above
2. **How can you monitor the cost of Snowflake and which object can be used? (Select all applicable)**
 - a. RESOURCE_MONITOR
 - b. DATABASE
 - c. WAREHOUSE
 - d. All of the above
3. **What are the actions defined in the resource monitors setup for the Snowflake account? (Select all applicable)**
 - a. NOTIFY
 - b. NOTIFY & SUSPEND
 - c. NOTIFY & SUSPEND IMMEDIATELY
 - d. All of the above

Answers

1	a, b
2	a
3	d

CHAPTER 11

Implementing Cost Optimizations

Introduction

Snowflake platform cost is measured as credits consumed by each layer. Snowflake credit consumption is calculated as the overall and total cost of the compute layer, cloud services layer, storage layer, and serverless services. Platform usage details are available and captured in the **USAGE** metadata views. You can leverage the metadata views to compute and analyze the performance of the Snowflake account and compute credits consumed every day or month.

Snowflake cost is measured by leveraging details captured in the metadata views. You can leverage the native metadata views and enhance the cost of your Snowflake account. Snowflake shares best practices and strategies to reduce the overall costing.

Structure

This chapter consists of the following topics:

- Components costing in Snowflake
- Implementing cost optimizations
- Implementing cost dashboards

Objectives

By the end of this chapter, you will be able to develop an understanding of

Snowflake costing and implementing cost optimization. This also covers implementing cost and usage dashboards using Snowsight metadata views. Snowflake shares best practices and recommends using warehouses and services to save costs. You can implement optimization analysis and usage queries using the trial account setup in [Chapter 1, Getting Started with Snowflake](#).

Components costing in Snowflake

Snowflake architecture has 3 layers and a set of features that contribute to the overall Snowflake cost. Refer to [Figure 11.1](#) which lists down the Snowflake layers and serverless services as part of Snowflake costing components. [Chapter 10: Understanding Snowflake Costing](#), focuses more on costing components in detail. This section focuses on a quick review and recap of the costing components of Snowflake:

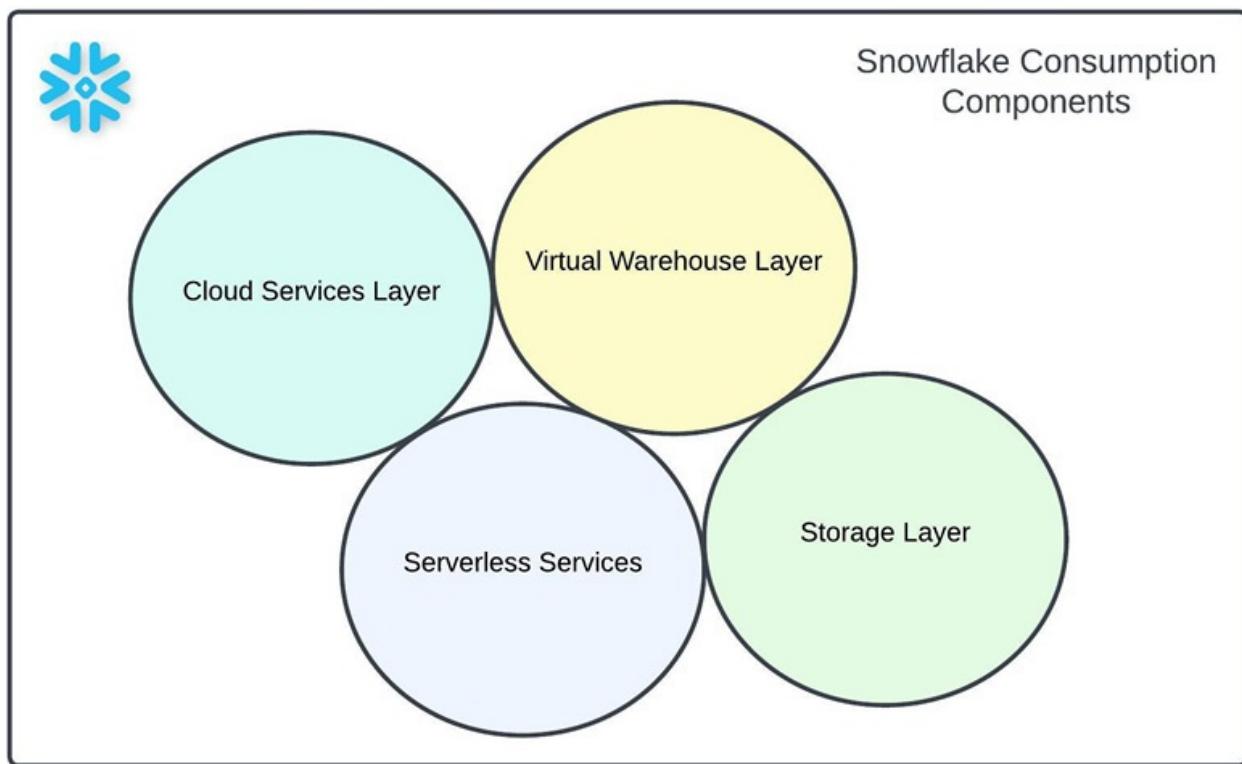


Figure 11.1: Snowflake Cost Components

Compute or virtual warehouse layer

Credit consumption is calculated based on the warehouse size, credits per hour rate, and the warehouse up time. Minimum credits consumed is for a minute

when the warehouse is set to active state. Post that, the usage is calculated based on every second of consumption.

Cloud services layer

This is the cost of Snowflake managed compute resources calculated every day. This is calculated in accordance with the compute usage on a daily basis. The 10% of compute usage is considered in the cloud service bucket as FREE usage. For example, if your computer usage is 200 credits per day, then 20 credits are considered FREE for cloud services usage. If your total cloud services consumption is 25 credits per day, then you will be billed for $25 - 20 = 5$ credits per day for cloud services. If your consumption is less than or equal to 10% of compute, then you will not be billed for any additional credits for the cloud services layer.

Storage layer

Storage is calculated as the daily average of data stored in compressed format. Data is stored in tables, stages, table recovery – time-travel, and failsafe. Data transfer cost is also considered as the overall cost of the account.

Serverless services

Serverless services like Snowpipe, Snowpipe streaming, and optimization services are run by Snowflake and use Snowflake managed compute resources. Their usage is calculated based on their uptime and use.

Sample use case

You can consider the below retail use case and Snowflake account setup, pipelines, and usage to calculate the overall usage of Snowflake account. Below are the current stats of the applications setup on the platform:

Use case details:

Category	Details	Specifications
Snowflake Edition	Enterprise	
Data Stored	70TB compressed	
Data Loading	Real time data loads – 24x7	Small continuous data loads
Users	Sales consumers	7 users, 5 days in week, 8am to 5pm

	Internal consumers	12 users, 16 hours in day, 5 days in a week
Warehouses	1 Large VW	Sales team
	1 Medium VW	Consumers
	1 2XL VW	Reporting
	1 S VW	Loading
Consumers	BI Reporting	Weekly report, takes 2 hours to run

Table 11.1: Snowflake component details

You can calculate the overall usage of your account as follows:

Component	Computation	Credits
Compute usage: Data loading (24 hours in a day, 31 days in a month)	$1 * 24 * 31 = 744$ hours	2 credits/hour for standard VW Credits = $744 * 2 = 1488$ credits per month
Storage computation (flat rate per TB storage)	70 TB with compression	\$23 per TB, Usage = $70 * 23 = \$1610$
Compute usage: Sales (9 hours in a day, 5 days in week, 20 days in a month)	$1 * 9 * 20 = 180$ hours	8 credits/hour for Large warehouse Credits = $180 * 8 = 1440$ credits per month
Compute usage: Internal consumers (16 hours, 5 days in week, 20 days in a month)	$1 * 16 * 20 = 320$ hours	4 credits/hour for Medium warehouse Credits = $320 * 4 = 1280$ credits per month
Compute usage: BI queries (Weekly report – 4 times, 2 hours each time)	$1 * 2 * 4 = 8$ hours	32 credits per hour for 2XL warehouse Credits = $8 * 32 = 256$ credits per month

Table 11.2: Use case consumption

Total compute credits per month:

Compute cost (add all VW cost computed above)	$= 1488 + 1440 + 1280 + 256 = 4464$ Credits
Credits to \$ conversion - consider \$2 per credit	$= 4464 * 2$ $= \$8928$ per month

Table 11.3: Credits table

Total cost of the account per month:

The total usage of an account is the sum of compute and storage costs. You can calculate the total cost for the given use case as follows:

Total Cost = Compute cost + Storage cost

Total Cost = \$8928 + \$1610

Total Cost = \$10538

Hence, the total cost of the given use case is \$10538 per month. You can add data transfer cost in case the data is moved in or out from a stage. Now, you know how you can calculate the overall cost of the Snowflake account for a given use case. As the next step, you can optimize the overall use of these services and Snowflake resources to reduce the monthly usage. You can use the strategies outlined in the below section.

Implementing cost optimization

Snowflake costing has various components, and to optimize the overall cost of Snowflake, you can optimize the use of components like Compute, Storage, Cloud Services, and so on. Compute cost and optimizing compute cost is one of the critical components of optimization.

Compute optimization

Compute is used to perform SQL, DML, and data load operations. Warehouses can be created, used, auto suspended, and auto resumed, based on the usage. You can follow the below strategies to optimize the cost.

Access setup to warehouse

You can restrict the usage of warehouses as well as access to the warehouses. You can set up the warehouses to be used by teams with different types of workloads. You can grant access to the users to use the warehouse and restrict granting edit access.

As a best practice, Snowflake recommends a centralized approach to create and manage warehouses in an account. Grant access to only limited users with custom roles to manage warehouses.

Setup the right size of the warehouse

Snowflake warehouse credits are based on the warehouse size and their uptime. Snowflake recommends using the warehouse with different sizes and different workloads before sizing a warehouse. It is recommended to use a smaller size and increase the size of the warehouse to test for the workload if you are not sure

of the warehouse sizes.

Setup query time limit

As you know, query executions need compute credits to complete. Queries can be executed for a longer duration, and this can cause unexpected costs. You can set the time limit to the query executions and terminate the queries to avoid additional costs. You can use the **STATEMENT_TIMEOUT_IN_SECONDS** parameter to set the limit.

Setup query queue time limit

If any query is in queue and waiting for resources, then it will not be charged. It does not cost any credits until it is in the queue. However, when it comes to the execution state, it may not be relevant and consumes credits for execution. You can avoid such consumption of irrelevant queries that stay in a queue for a long time using the **STATEMENT_QUEUED_TIMEOUT_IN_SECONDS** parameter. This is a parameter that can be set at account, at user, at session, or at a specific warehouse.

Setup auto suspend and auto resume

By default, every warehouse has auto suspend enabled. You can use this to suspend the warehouse. If the warehouse is in a suspended state, then it does not cost any credits. You can set auto suspend and resume to warehouses to avoid the cost in case the warehouse is not being used. It is recommended to calculate the cache maintained against the auto suspend before setting up auto suspend, as maintaining a cache will be less costly than warehouse costs.

Enforce spend limits

You can enforce the limits on the warehouse spend using resource monitors. You can set the threshold values and compare them against the usage to generate the alerts, notifications or take any specific actions. You will learn more about this feature in the next section.

Storage optimization

Storage cost is calculated as the daily average of data stored in compressed data format. The storage cost is computed on the flat rate of storage. This is not computed in terms of credits. The storage cost is on the higher side in comparison with credits used for warehouses. You can optimize storage costs by

using the following strategies.

Store only required data

Use permanent tables to store the final or target state of the data. You can use transient or temporary tables for intermediate processing. You can also use internal or external stages to reduce the storage cost of the data that is meant to be archived, restored, or used as a backup copy. You can use the stage in the same region where data copy is FREE. As a best practice, you can always store the data necessary or the data used frequently, usually termed HOT data, in the storage layer.

Cloud services optimization

If your cloud services cost is within 10% of the compute usage, then you still have room to use the SQL commands that leverage the cloud services bucket. Suppose your cloud services usage is more than 10% of the compute. In that case, you can leverage the usage views to analyze the usage with detailed information of cloud services to analyze queries that are consuming more of services layer cost. You can check on some of the following recommendations to reduce the services cost:

- Reduce queries that set the context using USE queries to set database, schema, role, warehouses etc. You can configure the connections or use profile configurations to set these parameters. The more you use them as session variables with every pipeline or every step of your data pipeline, the more you consume the service layer.
- Queries that run **count(*)** or **count(1)** on tables or using table information from schema views
- Evaluate the queries to check if they can be run from the cache stored at various layers.

You can use these optimization strategies to evaluate and reduce the usage cost of the account. You can also implement another important aspect of cost optimization is cost monitoring. You can use Resource monitors to monitor the cost, as you can plan on using Snowsight to implement dashboards leveraging metadata views. You will learn more about Snowsight dashboarding in the next section.

Implementing cost dashboards

This section focuses more on implementing dashboards that can be used to monitor the performance of the Snowflake account. You can plan on implementing Query metrics, Warehouse metrics, Storage metrics, as well as overall compute, cost, and user analytics using Snowflake metadata views. You can use Snowsight web UI to build a dashboard using the following steps:

1. Login to Snowsight web UI: <https://app.snowflake.com/>
2. Select the appropriate role: **ACCOUNTADMIN/SYSADMIN** or a custom role with the required privileges to read data from the **USAGE** schema.
3. Navigate to Dashboards from the left side navigation bar:

TITLE	VIEWED	UPDATED	ROLE
Account Usage Dashboard -	2 months ago	5 months ago	ACCOUNTADMIN
Engineering - Demo Dashboard	7 months ago	5 months ago	ACCOUNTADMIN
Operations - Demo Dashboard	5 months ago	5 months ago	ACCOUNTADMIN
Usage - Demo Dashboard		5 months ago	ACCOUNTADMIN

Figure 11.2: Snowsight Web UI – Dashboard View

Refer to [Figure 11.1](#), where you can see the dashboards page. If you have any existing dashboards, you will see the list of dashboards as shown. If there are no dashboards then you will see a blank page. You can click on **+Dashboard** button on the right top corner to create a new dashboard and provide name of the dashboard and click on **Create Dashboard**.

4. Click on **new Tile** button on the screen to create a tile /widget, as shown:

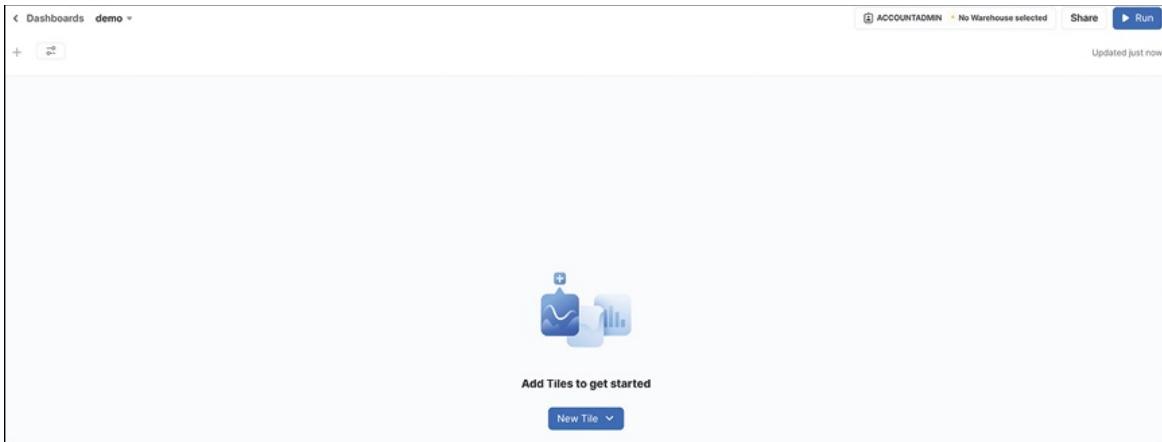


Figure 11.3: Adding New Tile to the dashboard

5. New tile has two options: SQL Worksheet and Python Worksheet:

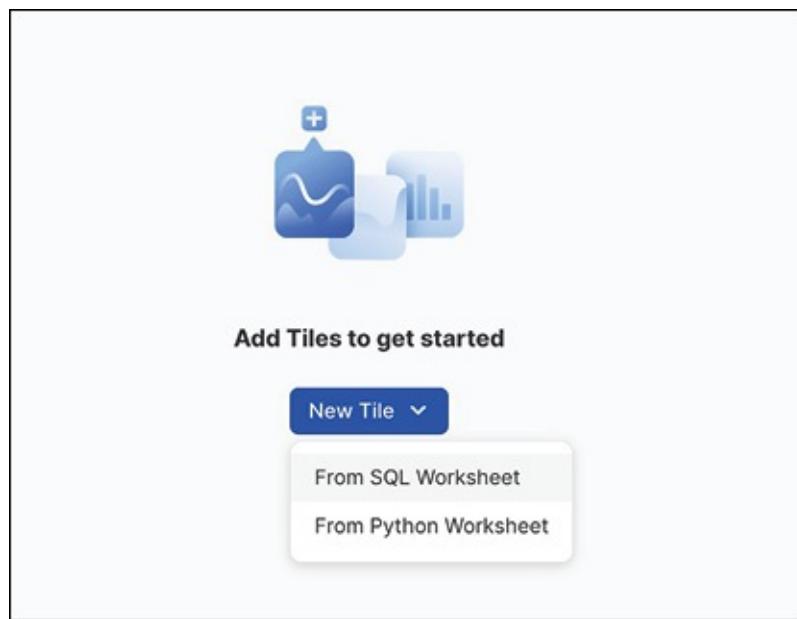


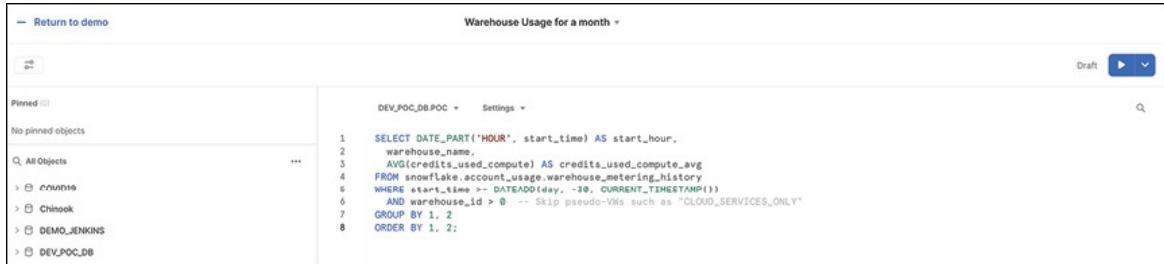
Figure 11.4: New Tile Options

6. Select **SQL worksheet** if you want to generate dashboard using **USAGE SQL** queries, as shown in the following screenshot:

A screenshot of a SQL worksheet interface. At the top, there's a header with 'Return to demo' on the left, the date '2023-08-27 1:57pm' in the center, and a play/pause button on the right. On the left side, there's a sidebar titled 'Pinned' with a note 'No pinned objects'. Below this, there's a list of objects: 'All Objects', 'COVID19', 'Chinook', 'DEMO_JENKINS', and 'DEV_POC_DB'. The main area shows a database connection named 'TUTORIAL_100_TRACE_DB.PUBLIC' with a 'Settings' button. A single-line query is displayed: '1 select col from table where created = :daterange'. There's also a search bar at the bottom right.

Figure 11.5: SQL Screen

7. Write SQL to generate **USAGE** data from **USAGE** views, as shown:



```

-- Return to demo
Warehouse Usage for a month *

Pinned ⓘ
No pinned objects
Q All Objects
> ⚡ covid19
> ⚡ Chinook
> ⚡ DEMO_JENKINS
> ⚡ DEV_POC_DB

DEV_POC_DB.POC + Settings ⓘ
Draft ⏪

SELECT DATE_PART('HOUR', start_time) AS start_hour,
       warehouse_name,
       AVG(credits_used_compute) AS credits_used_compute_avg
  FROM snowflake.account_usage.warehouse_metering_history
 WHERE start_time >= DATEADD(day, -30, CURRENT_TIMESTAMP())
   AND warehouse_id > 0 -- Skip pseudo-WHs such as "CLOUD_SERVICES_ONLY"
 GROUP BY 1, 2
 ORDER BY 1, 2;

```

Figure 11.6: SQL to get Warehouse usage details from USAGE view

8. Run the SQL to generate result data:

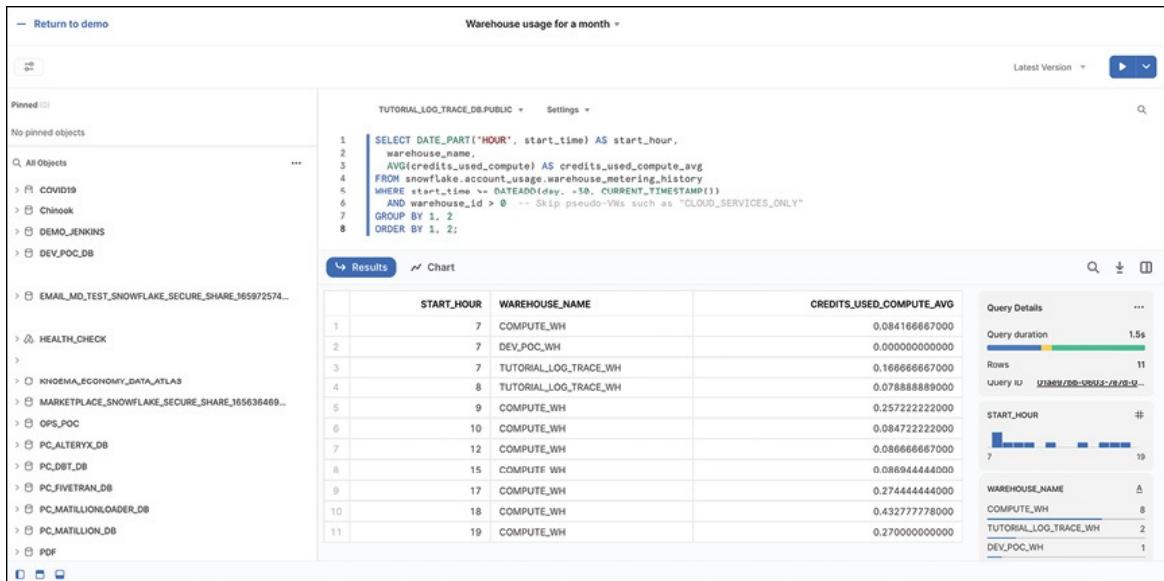


Figure 11.7: Execute SQL and Generate result

9. Once you have the result set generated, you can see the **Chart** option. You can select Bar, Line, Heatgrid, Scatter, Scorecard, and so on, as shown:

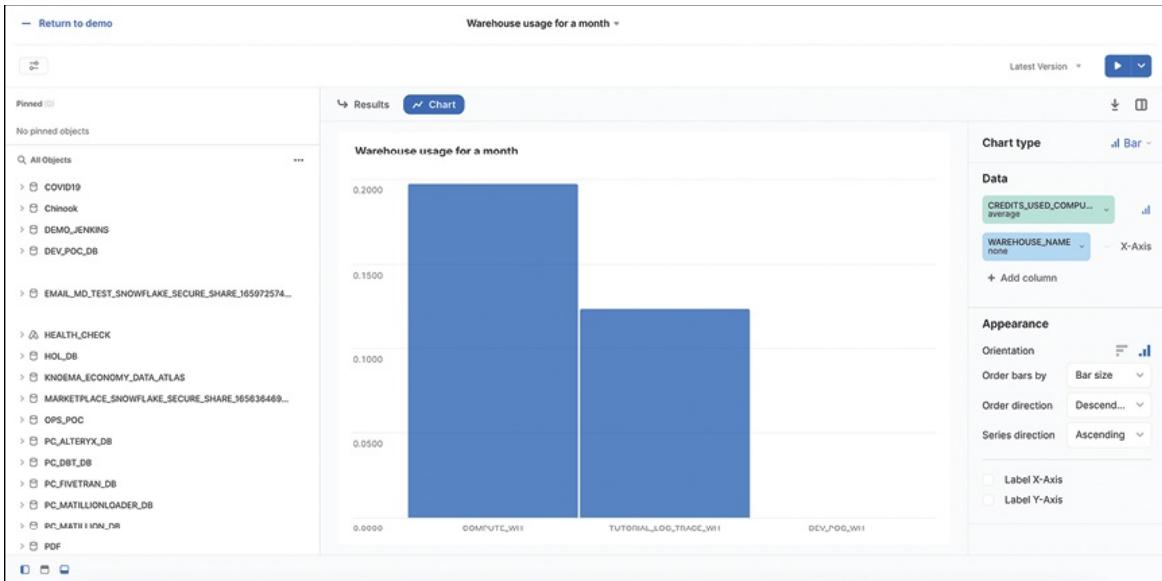


Figure 11.8: Deriving appropriate chart with result set

10. You can add more widgets to the dashboards.
11. You can refer to below reference metrics for engineering as well as operational dashboards:
 - a. Long running queries
 - b. Query that is most queued
 - c. Warehouse performances
 - d. Cloud Services consumption
 - e. User logon issues
 - f. Top 10 failed queries
 - g. Query compilation time vs execution time

You can generate multiple metrics depending on your business requirements. You can create these dashboards and share them with your team using the share option on the dashboards. These are developed once for each account and region to be used by operations and engineering teams.

Conclusion

This chapter provided a summary of Snowflake's costing, dashboards, and cost

optimization. We covered information about various cost optimization strategies and costing factors. This also covers various cost optimization strategies and recommendations that can be implemented in Snowflake to reduce usage. The chapter also helped you to understand costing factors with a sample use case that illustrates various parameters and components to be considered as Snowflake cost per month. You can use a sample use case as a reference to understand the usage of the Snowflake account, and you will be able to derive Snowflake account usage for your account based on the workloads, warehouses, cloud services, serverless, and storage components.

In the next chapter, you will learn about Snowflake's data sharing feature. You will learn more about secure data sharing with Snowflake as well as non-Snowflake consumers.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake offers **INFORMATION_SCHEMA**, **ACCOUNT_USAGE**, and **READING_ACCOUNT_USAGE** schemas and metadata views to generate account usage.
- Snowflake cost optimization includes strategies to define warehouse sizes, using optimal warehouses, dedicated workloads and warehouses, and automated suspend and resume to avoid additional, unexpected costs.
- Snowflake recommends using cost strategies and setting up cost dashboards for monitoring.
- Snowflake's cost components include compute, storage, serverless, and cloud services layer. Most of the Snowflake components' cost is calculated based on the credits, whereas the storage layer's cost is calculated as a flat rate.
- Snowflake serverless services cost is based on the resource usage and uptime usage of the services.

Questions

1. How is Snowflake cost calculated?
2. How can you use usage views to derive usage?

3. How can cost be optimized?
4. Can you calculate the monthly cost of your account?
5. Can you set up a monitoring dashboard using Snowsight?

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 12

Data Sharing

Introduction

Data collaborations, data integrations, and integrating with consumer applications is essential for business growth and requirements. Data needs to be shared with various consumer applications and users to consume data. Typically, the data is shared in the form of file feeds and database extracts, and you need to develop pipelines to generate the extracts. Maintaining pipelines becomes challenging. This may also lead to issues like stale data, managing data consistency, and so on. Snowflake offers a data sharing feature where you can create data shares that can be shared with consumers, and you no longer need to maintain pipelines to generate extracts and maintain data consistency.

This chapter helps you to understand data sharing, implementing it, and secure data sharing with Snowflake as well as non-Snowflake consumer groups. This also helps to understand the various sharing options available in Snowflake. This chapter covers data sharing, data exchange, data marketplace, secure sharing, and data sharing with read-only accounts.

Structure

This chapter consists of the following topics:

- Data sharing needs
- Implementing data sharing
- Implementing sharing with Snowflake users

- Implementing sharing with non-Snowflake users
- Understanding data sharing options

Objectives

By the end of this chapter, you will be able to develop an understanding of Snowflake data sharing, data marketplace, and sharing data with Snowflake as well as non-Snowflake users. This also covers implementing data sharing – understanding sharable objects, secure sharing, and managing non-snowflake data consumers. You can implement secure data sharing, set up read-only consumer accounts, and access data using read-only accounts using the trial account setup in [*Chapter 1, Getting Started with Snowflake*](#).

Data sharing needs

In a typical enterprise data platform implementation, you would have observed the need to bring in data from heterogeneous sources, loading them to the platform in the form of raw data or landing or staging layer, processing the data using native or third-party tools, and transforming the data and store it in data marts or target layer. The data stored in the target layer is in the form of consumable data where consumer applications and users can connect and consume data for their data use cases. The consumer use cases can be BI/Reporting, Data Analytics, AI/ML, Downstream applications consuming data for further processing, or business users accessing data for their analysis. These consumers consume data in the form of tables by connecting directly to the platform or in the form of file feeds in case no connectivity needs to be set up. These file feeds are extracted and FTPed to the common or shared location or **Network Attached Storage (NAS)** to be consumed by one or more applications. In this scenario, you need to design and develop an additional set of pipelines to generate the data feeds. This also needs additional maintenance and costs associated with extracts. This may lead to data consistency if it is not maintained or updated as a data feed or file feed.

Data also needs to be prepared and shared with the analytics team, where they can build AI/ML models on top of the data, train them with new datasets as and when the data is updated, use them to make predictions, and save the prediction result set. In earlier implementations, AI/ML needed a separate environment, and data needed to be moved between systems to train your models. This is another scenario where data needs to be prepared, extracted, and exchanged between

systems that cause additional cost and maintenance. This may also lead to data inconsistencies if not maintained.

In the case of BI/Reporting scenarios, you can use BI tools to connect to the data platform, and read data from the databases or schema for reporting needs. This can be beneficial and without moving any data until your queries are not complex enough. In case of complex, repeated subset of data – you might prefer creating an additional layer or database to create materialized views or views or additional aggregated data in tables for reporting. In typical platforms, you must develop pipelines to generate and store aggregated data to the materialized views. This may again need maintenance and can cause data inconsistencies.

These are some of the most common use cases where data needs to be moved or extracted or shared between consumer applications. Snowflake offers a data sharing feature that enables users to create data shares and share data between consumer applications in a secure fashion. You can also share data between Snowflake as well as non-Snowflake users. You no longer need to develop and maintain any additional pipelines to maintain the state of the data. The data shares are built on top of your databases or tables that need to be shared and refreshes automatically as and when data is updated. This avoids data extractions, additional pipelines, data state maintenance, as well as additional setup to share across applications like SFTPs, stage extracts, and so on.

You can refer to [*Figure 12.1*](#), where data is sourced from heterogeneous sources to external stages, and these stages are integrated to read data in raw format with Snowflake external tables. You can have the processing layer using Snowflake native – SQLs, database objects like stored procedures, user-defined functions, and Snowpark to transform the data and store them as data marts in another database schema. You can create data shares from the data marts and share them with consumer applications to consume the data.

The data shared is secure, read-only, and the access is managed by RBAC. You can share them with Snowflake users, and they can use their own warehouses to query the data from shares. Consumers can also create a database from the data share if granted appropriate privileges to copy the data. The moment the consumer creates their own copy from share, the copy is not updated automatically for any new data changes as it is the consumer's own version of the data. Data share will continue to reflect the new data from the source database and objects from the source account. Refer to the following figure:

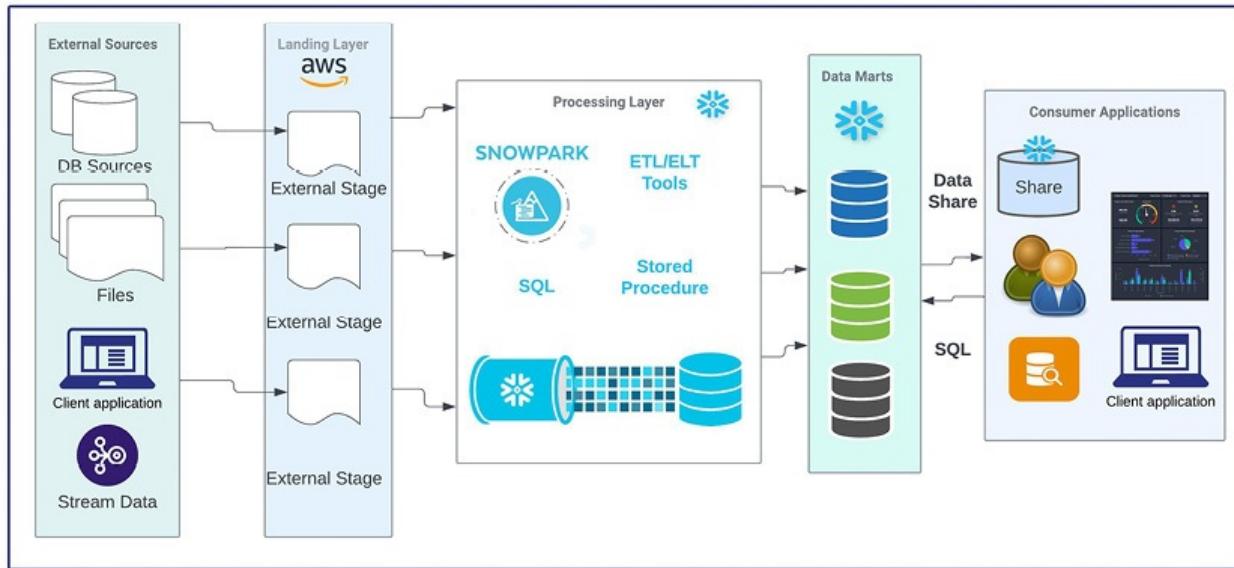


Figure 12.1: Snowflake Data Platform

You can also share the data with non-Snowflake accounts where you need to create a reader account, assign a warehouse, manage warehouse utilization, and cost management. Reader accounts get their login URL, and they can log in to the Snowflake account and read data from the shares. They get only read copies of share, run queries, and use them for analysis. They cannot copy the data from share and create their own versions until they are assigned the required privileges, as well as the compute capacity. The cost of the consumer is owned by the source account and needs to be managed to avoid unexpected costs.

You will learn more about creating shares and assigning them to Snowflake, as well as non-Snowflake consumer accounts, in the subsequent sections of this chapter.

Implementing data sharing

Snowflake data sharing enables users to share the data without moving data out of the platform. Snowflake data shares have providers and consumers. Providers are the users that create data shares to be shared with consumers. Consumers are the users who can read data from the shares. A single account or user can be a provider as well as a consumer of data shares. You can refer to the following figure that represents various providers and consumers sharing data with Snowflake and non-Snowflake users:

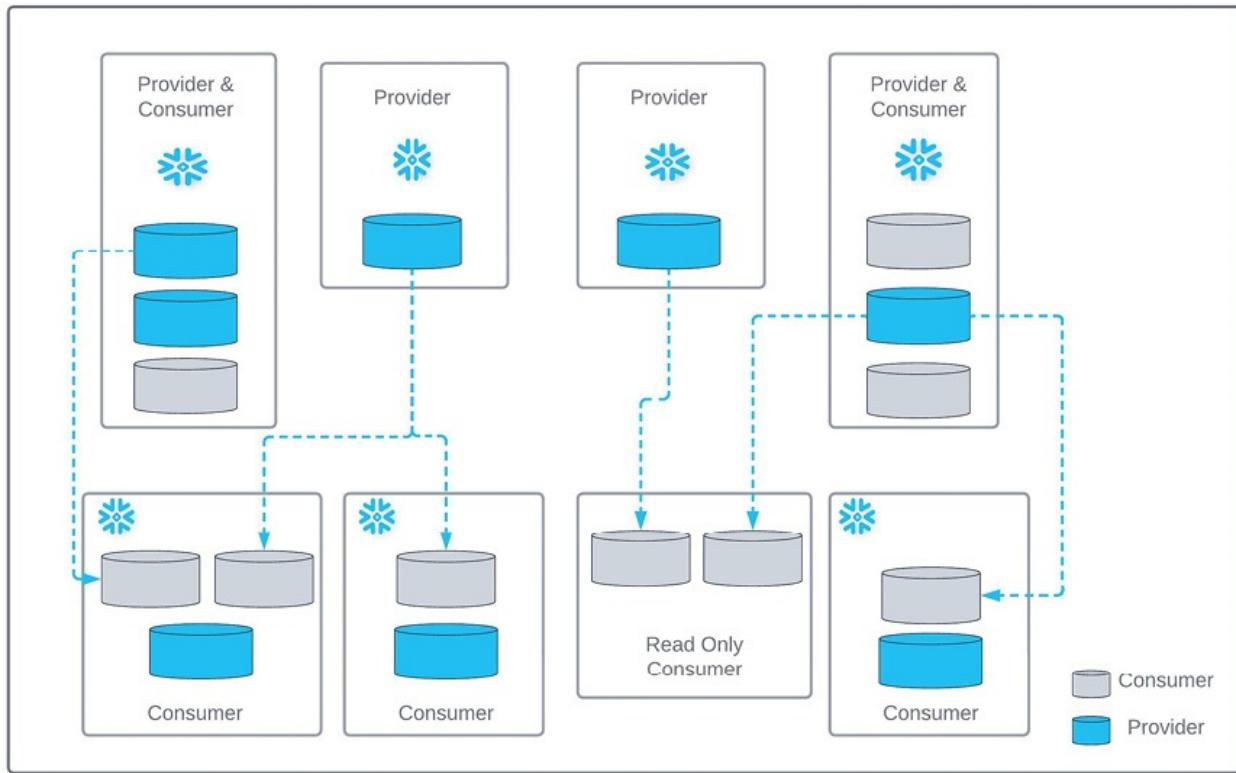


Figure 12.2: Snowflake data sharing

Snowflake data sharing allows users to share the objects in an account. You can share the following Snowflake objects as part of data sharing and sharable objects:

- Tables
- External tables
- Secure views
- Secure materialized views
- Secure **User Defined Functions (UDFs)**

All shared objects and data are in read-only mode. Consumers cannot write, update, or modify anything in shared databases or objects. Snowflake offers secure data sharing that allows users to share the data between accounts without transferring or copying data across.

How does data sharing work?

Data sharing is sharing data in read-only mode without transferring data. Have you ever wondered how sharing can be done without moving data? Snowflake

data share does it with the help of the cloud services layer and metadata store. Data shared is not copied anywhere else. It is instead stored from the existing storage. Consumers can read data from data share, and query data from shares, and they need warehouses or compute to query the data. Consumer cost includes only the compute cost as data does not get stored on the consumer's side. There is no storage cost associated with data shares to the consumer accounts.

Data sharing is implemented quickly, as no data is to be moved or copied over. The data shares provide data in a near-instantaneous fashion. The providers are the users or accounts that share the data, and consumers are the users or accounts that read data from shares. Providers can share objects from one or more databases in the same account as part of data sharing. Consumers can read data but are not allowed to modify any objects or data in the shares.

Whenever a user creates a share, it consists of the objects to be shared and inherits the privileges granted to the shared objects. If the consumer creates the database from the share, it inherits the privileges and properties of the objects from the share. Shares are secured objects managed by providers. If the provider adds any new objects to the share, then it becomes available to the consumers immediately. Providers can control access to the shares, and they can grant or revoke access at any time. Based on the data and consumers, Snowflake offers data to be shared in 3 ways or options:

- Listing
- Direct share
- Data exchange

You will learn more about these sharing options in the upcoming section of this chapter.

Implementing sharing with Snowflake consumers

You can create data shares as providers and share with consumer account or multiple accounts. As you know, Snowflake access control is maintained and managed with roles. You need to grant the required privileges to the role and grant the role to the user to create the secure data share as a provider. You can also manage the privileges of the consumer's side by creating a share by using a database role or by granting the privileges to the share. In this section, you will learn about creating a share, sharing a share with Snowflake accounts, and accessing share as a Snowflake account consumer.

Data sharing considerations

You can share data with consumer accounts. However, below are some of the considerations to share data with accounts or users:

- Data can be shared across cloud platforms and regions.
- Data share can consist of objects from multiple databases.
- Data share can consist of only secure views due to privacy reasons. You cannot add standard views to the share. This gives an error while adding a standard, non-secure view to the share.
- Consumer accounts added to the share are reflected immediately. Consumer accounts can access data as soon as share is shared with them.
- Data modified in shared objects is available to the consumers immediately. However, a new object added to the database shared is not reflected automatically. You can run the **GRANT** privileges command to make the newly added object available to the consumers.

Data sharing commands

You can use DDL commands to create, manage, modify, drop, list, and describe data shares available in an account. Snowflake also supports additional DDL commands to manage the database role as well as offers a set of **GRANT** privilege commands to manage accesses of the objects in shares. The following are the supported commands:

- **DDL commands for share:** These are the DDL commands to create share. You can use the below commands to create and manage share in an account:
 - **CREATE SHARE**
 - **DROP SHARE**
 - **ALTER SHARE**
 - **SHOW SHARES**
 - **DESCRIBE SHARE**
- **Database role commands:** Database role is created to manage access to the objects. You can use the following commands to define and manage a

role:

- **CREATE DATABASE ROLE**
- **ALTER DATABASE ROLE**
- **DROP DATABASE ROLE**
- **SHOW DATABASE ROLES**
- **GRANT** commands to manage shared objects: You can also manage the access of the shared objects by using **GRANT** and **REVOKE** commands. You can use the below commands to allow or revoke access to the shares:
 - **GRANT DATABASE ROLE __ TO SHARE**
 - **REVOKE DATABASE ROLE __ FROM SHARE**
 - **GRANT <privileges> __ TO SHARE**
 - **REVOKE <privileges> __ FROM SHARE**
 - **SHOW GRANTS TO SHARE __**

Data sharing pre-requisites

You can create a data share to share the data with consumers. However, you need to identify the objects and define the access policies before you start creating the data shares. You can follow the below steps to meet your pre-requisites before the creation of data shares:

1. Identify the objects that need to be shared. As you know, all objects cannot be shared in a data share. You can identify shareable objects like databases, tables, and secure objects - user-defined functions, views, and materialized views.
2. You can share the databases and tables without any preparation. Also, if you choose to share the entire table in a database, then you will not need to prepare anything additional to set up the table for data share.
3. You can share the filtered or sample data from a table with share. You need to create a secure view on top of your table to filter the data. Only secure views are allowed in data share. Hence, create a secure view to hold the required data to be shared with consumers.
4. You can identify and share secure materialized views, secure user-defined

functions, and secure views with shares.

5. Consumers can get read-only data from providers. However, consumers can also get a copy of data, create a database from share, or add a stream on top of the shared object to track the changes. To allow consumer accounts to track changes, you need to grant additional access to the shared objects and consumers.

Creating a data share

You can create a share if you are using **ACCOUNTADMIN** role or using a role that has privilege **CREATE** share granted. As you know, you have two options to create the shares. You can use Snowflake UI to create the shares, or you can use SQL commands to create the data shares.

You can create the direct data share in Snowsight by following the given steps:

1. Log on to Snowsight (your Snowflake account URL).
2. Go to **Data | Private sharing**.
3. Select **Share | Create a direct share** (This opens a new dialog).
4. In the new dialog, select the data that you want to share:
 - a. You can select a database as a source.
 - b. Select objects from the source database.
 - c. You can then select the accounts to share the data.
 - d. Click on create share, and share is created.

You can use the following steps and SQL commands to create a share and grant access by using a database role:

1. Create a share.
2. Create a database role.
3. Grant database, schema, and table access to the database role.
4. Grant database access to the share.
5. Grant database role to the share.

Follow the following block of SQL commands to create share, grant

permissions, and grant role to the share:

```
/* Create Share */
CREATE SHARE marketing_share;
/* Create Database Role */
CREATE DATABASE ROLE marketing_db.consumer_role;
/*Grant required usage permissions */
GRANT USAGE ON DATABASE marketing_db TO DATABASE ROLE
marketing_db. consumer_role;
GRANT USAGE ON SCHEMA marketing_db.Target_Layer TO
DATABASE ROLE marketing_db. consumer_role;
GRANT SELECT ON TABLE
marketing_db.Target_Layer.promotions TO DATABASE ROLE
marketing_db. consumer_role;
GRANT USAGE ON DATABASE marketing_db TO SHARE
marketing_share;
/* Grant role to the share */
GRANT DATABASE ROLE marketing_db. consumer_role TO
SHARE marketing_share;
```

You can use the following steps and SQL commands to create a share and grant access using **GRANT** privileges to the share:

1. Create a share.
2. Grant database access to the share.
3. Grant select to the share, as shown:

```
CREATE SHARE marketing_share;
GRANT USAGE ON DATABASE marketing_db TO SHARE
marketing_share;
GRANT USAGE ON SCHEMA marketing_db.Target_Layer
TO SHARE marketing_share;
GRANT SELECT ON TABLE
```

```
marketing_db.Target_Layer.promotions TO SHARE  
marketing_share;
```

You can add consumer accounts while creating a share, or you can also extend the share to the accounts by using an **ALTER** command:

```
ALTER SHARE marketing_share ADD ACCOUNTS=ab98765,  
mn43097;
```

You can also list down the shares and list the various grants of the share using DDL commands, as shown:

```
SHOW SHARES; --list all available shares in an account
```

```
SHOW GRANTS OF SHARE marketing_share; --list down the grants of the share
```

Creating a data share with multiple databases

You can create a data share that consists of objects from multiple databases. As you know, a secure view can be shared in a data share. You can build a secure view to share the data from multiple databases or tables.

Consider a scenario where you are working on a **RETAIL** use case, and you need to create a data share that consists of data of the customers and sales together, to be shared with consumers to get a view of the overall orders placed and delivered. You can refer to the below sample code:

```
-- Define database customer  
  
CREATE DATABASE customer;  
  
CREATE SCHEMA customer.poc;  
  
CREATE TABLE customer.poc.cust_info(cust_id  
int,cust_name string, addr string, order_id int);  
  
CREATE VIEW customer.poc.cust_view AS SELECT * FROM  
customer.poc.cust_info;
```

```
-- Define database sales
```

```
CREATE DATABASE sales;
```

```
CREATE SCHEMA sales.poc;

CREATE TABLE sales.poc.order_details(cust_id
int,order_id int, order_date date, order_status
string, cust_id int);

CREATE VIEW sales.poc.sales_view AS SELECT * FROM
sales.poc.order_dtls;

--Define share database and secure object

CREATE DATABASE cust_sales;

CREATE SCHEMA poc;

CREATE view cust_sales.poc.v_cust_sales as
(
    select
        cust_id,
        order_id,
        order_date,
        order_status
    from
        customer.poc.cust_view a
    left outer join
        sales.poc.sales_view b
    on a.cust_id = b.cust_id
    where order_status is not NULL
);
```

```
--Define share and grant accesses  
CREATE SHARE orders;  
GRANT USAGE ON DATABASE cust_sales TO SHARE orders;  
GRANT USAGE ON SCHEMA cust_sales.poc TO SHARE orders;
```

--Grant access to reference databases

```
GRANT REFERENCE_USAGE ON DATABASE sales TO SHARE  
orders;  
GRANT REFERENCE_USAGE ON DATABASE customer TO SHARE  
orders;
```

--Grant access to the view to the share

```
GRANT SELECT ON VIEW cust_sales.poc.v_cust_sales TO  
SHARE orders;
```

Using a data share

Once a share is created, you can share it with consumer accounts. In this scenario, you will learn how Snowflake consumer accounts can access data. For non-Snowflake reader accounts, you will learn more about the process in the next section.

As a consumer, shared data can be accessed in read-only mode. However, shares also have a few limitations:

- In read-only shares, consumers cannot write or update anything to the data shares.
- Some of the below features and actions are not supported:
 - Time travel for shared database and objects.
 - Creating a clone of data share.
 - Modifying share comments.
- Shared objects cannot be replicated.

- Share cannot be re-shared with other consumer accounts.

Consumers can view the shares using Snowsight as well as use SQL commands to list the shares. You can follow the below steps to view the shares using Snowsight:

1. Log on to Snowsight (Snowflake account URL).
2. Go to **Data | Private sharing**.
3. You can select the tab – Shared with you. This will list down the shares shared with the account.

You can also list the shares by running: **SQL – SHOW SHARES;**

Consumers can create a database from data share. You can use data share as a consumer user and create a database using Snowsight as well as SQL. Follow the following steps to create a database using Snowsight:

1. Logon to Snowsight (Snowflake account URL).
2. Go to **Data | Private Sharing** to list down the data shares.
3. Go to **Shared with you** to list down the data shared.
4. Click on the **Ready to get** section and select Share to create a database.
5. Select the database role that is allowed access to the database.
6. Click on **Get Data**.

You can also run the following SQL command to create a database:

```
CREATE DATABASE <database_name> FROM SHARE
<share_provider_account>.<share_name>;
```

For example, **CREATE DATABASE cust_orders FROM SHARE orders;**

As a consumer, you can access data shared in read-only mode. You can also create a database from share and grant privileges to access the data. The compute required to query the data from share is owned by the consumer account. Note that there is no storage cost associated with this consumer account.

You can also share the data with non-Snowflake accounts. You will learn more about creating reader accounts, granting access, and sharing data with reader accounts in the next section.

Implementing sharing with non-Snowflake consumers

Snowflake data share can be shared with consumers who are non-Snowflake users. To share data with non-Snowflake accounts or users, you need to create a reader account. Reader account allows non-Snowflake consumers to access and query data shared by the provider account. Reader account does not need to manage any compute or need any licensing to access Snowflake. These accounts, as well as computes, are created, managed, and owned by provider accounts.

Providers create and manage warehouses required and assigned to reader accounts. Reader accounts can only consume data in the form of queries. These users cannot create a database from the shares as they are only reader accounts. The reader account uses the assigned warehouse to query the data. The warehouse usage can be unlimited. Hence, as a provider, you need to assign the threshold resource monitor to monitor the usage and add appropriate action to suspend the warehouse depending on the usage.

Reader account has a few limitations in comparison with data share in Snowflake accounts. Below are some of the limitations of reader accounts:

- Reader accounts can only read data from share.
- Reader accounts or consumers cannot perform the following operations on share:
 - Upload new data
 - Modify existing data
 - Unload data to external stages using the **COPY** command.
 - Reader accounts cannot run any DML commands to operate the data or any DDL commands to create/update/delete/drop objects.

In the following figure, reader accounts represent the reader accounts created to share data for an account. You can have multiple reader accounts created and managed for data share. You can also have data shared across reader accounts:

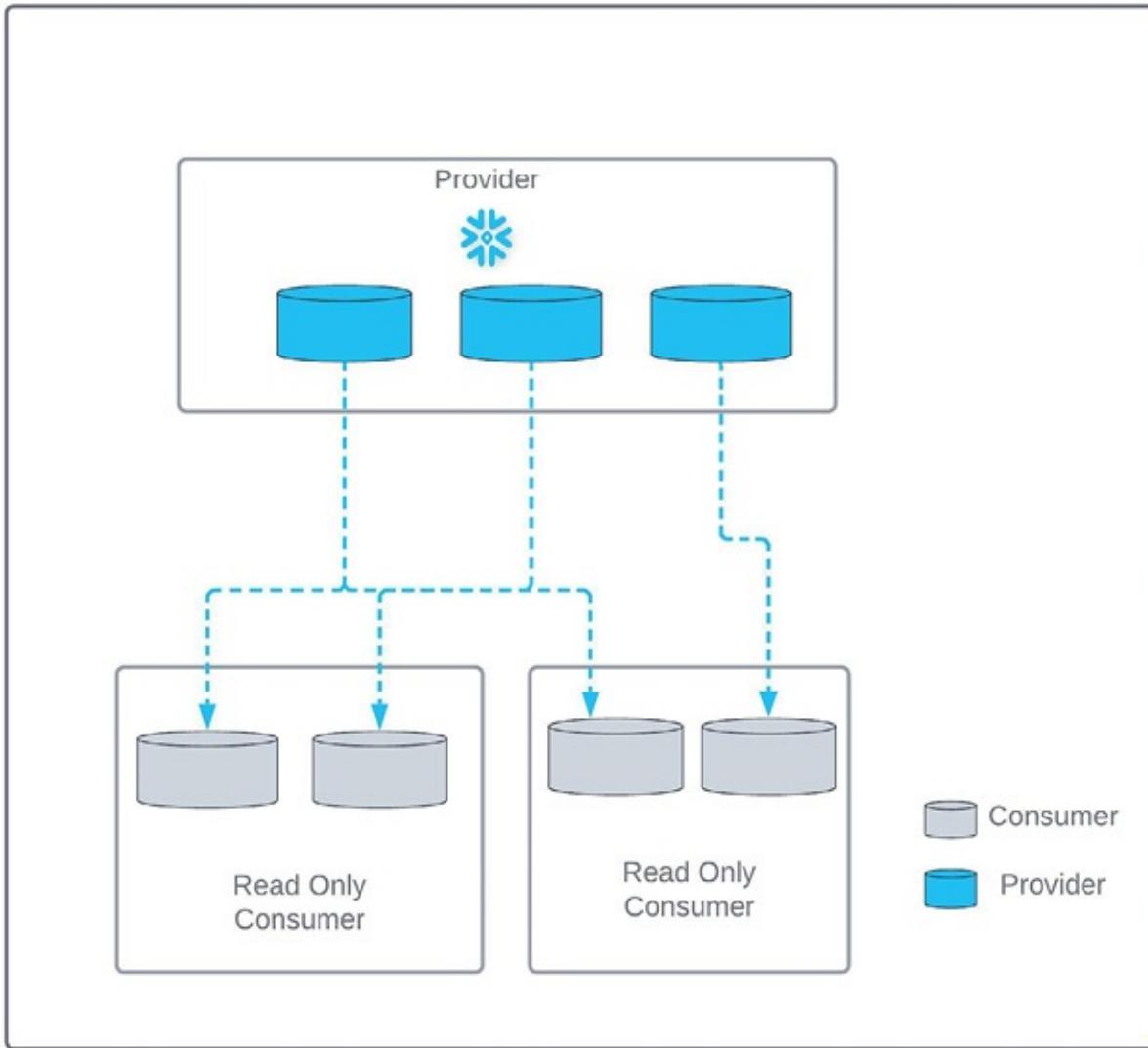


Figure 12.3: Reader Accounts

Creating reader account

You can create a reader account if you have **ACCOUNTADMIN** or **CREATE ACCOUNT** privilege assigned to you or role. Similar to any other operations in Snowflake, you can create an account using Snowsight, as well as use SQL command to set up an account.

You can follow the below steps to create a reader account using Snowsight:

1. Logon to Snowsight (Account URL).
2. Go to **Data | Private Sharing**.
3. Go to **Reader Accounts**.

4. You can manage existing accounts or create a new account here:
- Click on **+ New** button to add a reader account.
 - You can create a reader account by providing account details and mail ID to share the data, as shown:

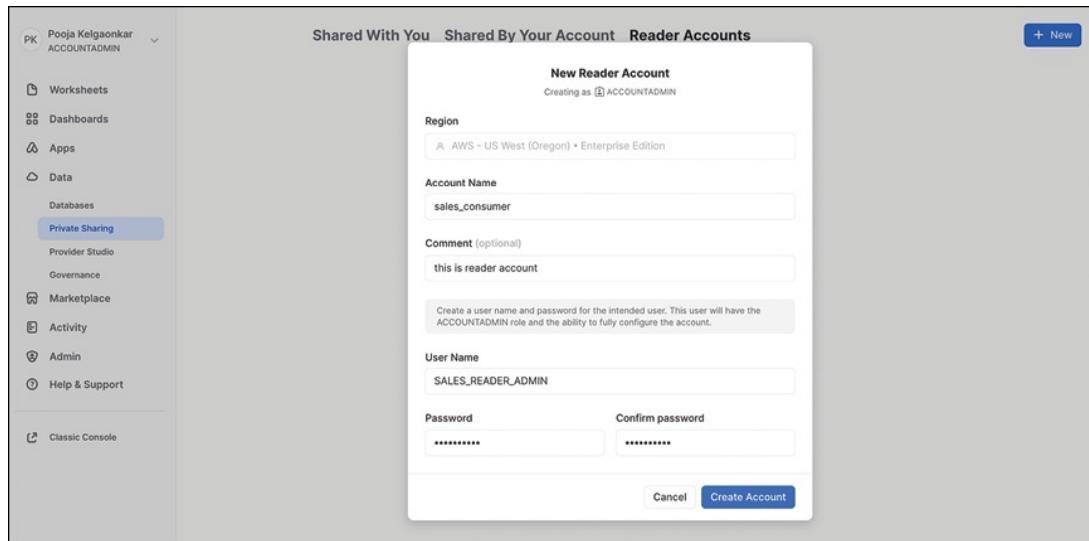


Figure 12.4: Create Reader Account

You can list existing Reader accounts using Snowsight:

- Go to **Data | Private Sharing | Reader Accounts** tab, as shown in the following figure:

Shared With You Shared By Your Account Reader Accounts						
<input type="text"/> Search						
1 Reader Account ⓘ						
NAME ↑	CLOUD	REGION	LOCATOR	LOCATOR URL	CREATED	...
SALES_CONSUMER	AWS	US West (O...	ORB27542	🔗	1 minute ago	⋮

Figure 12.5: List Reader Accounts – Snowsight

- You can use an SQL command to create a reader account using the **CREATE MANAGED** account. Following are the sample commands to create accounts.

Syntax:

```
USE ROLE ACCOUNTADMIN;  
  
CREATE MANAGED ACCOUNT <account_name>  
    ADMIN_NAME = <username> , ADMIN_PASSWORD =  
    '<password>' ,  
    TYPE = READER;
```

Example:

```
USE ROLE ACCOUNTADMIN;  
  
CREATE MANAGED ACCOUNT sales_consumer  
    ADMIN_NAME = sales_user , ADMIN_PASSWORD =  
    'Test123' ,  
    TYPE = READER;
```

- You can also use SQL commands to list the users and drop the reader account:

```
SHOW MANAGED ACCOUNTS; --list all reader accounts existing in the account
```

```
DROP MANAGED ACCOUNT sales_consumer; --drop or remove any existing reader account
```

- Once you create a reader account, you can get the logon URL for consumers. You can share the logon URL, username, and password with non-Snowflake accounts. You can create a warehouse and assign it to the reader account. Provider can create a resource monitor to track the warehouse utilization.

Snowflake offers three options to share the data – Listing, Direct Share, Data Exchange. You have learned direct sharing with Snowflake and non-Snowflake accounts. You will learn more about the additional approaches – Data Exchange and Listing in the next section.

Understanding data sharing options

Snowflake offers listing and data exchange options to share the data with one or more accounts. You will learn more about these two options in this section. Data sharing options follow the provider and consumer model. You have providers that share data and consumers that consume data shared by the provider.

Listing

Listing is an enhanced method of secure data sharing. The provider can share data with other Snowflake accounts privately or by using listing in the Snowflake marketplace. Consumers can use listing to access data shared by the provider as listing or marketplace. Listing is also secure data sharing along with some additional capabilities, such as:

- Public data share in the Snowflake marketplace
- Can charge consumers to access the data
- Monitor interest and usage of the data in the listing
- Provide metadata of provider

As a provider, you can create a listing and choose to share the data on the marketplace, secure data share, or public data share. Snowflake offers various types of listing depending on the consumers, usage, and charges to the consumers. They are discussed in the following sections.

Marketplace listing

Providers can share data on Snowflake data marketplace across clouds. Share on Snowflake marketplace allows multiple consumers to access data simultaneously. In this case, the provider does not need to maintain any individual users as consumers. This public listing can be free, paid, or personalized, depending on the consumer's needs.

Private listing

With private listing, providers can share data directly with the consumer accounts. This listing can be free or paid. They are described as follows:

- **Free listing:** This type of listing is free and is available privately or over the marketplace. This offers instant access to the data. This can be entirely free to access, or providers can negotiate fees with consumers to consume

the data. Typically, this is used to share generic, aggregated, or non-consumer specific data.

- **Paid listing:** This type of listing is paid and is shared privately or over Snowflake marketplace. Providers can create paid listings and charge consumers to access the data. These are available to consumers in a specific region, or in specific region providers.

Personalized listing

This listing is for a specific consumer request. This is treated as a personalized request to access the specific data for specific consumers. This listing can contain premium data or specific data for which the provider can charge.

Data exchange

Data exchange is nothing but a data hub that allows users to share and access data securely within a group of members based on the invitations. As a provider, this allows you to publish data that can be accessed and discovered by consumers in exchange.

You can easily share data with data exchange. You can share data with specific groups of internal consumers, users, business consumers, vendors etc. via data exchange. You can define listings, manage access to consumers, audit usage data, and membership of consumers. The following figure represents the data sharing across Snowflake users, accounts within organizations as well as across organizations:

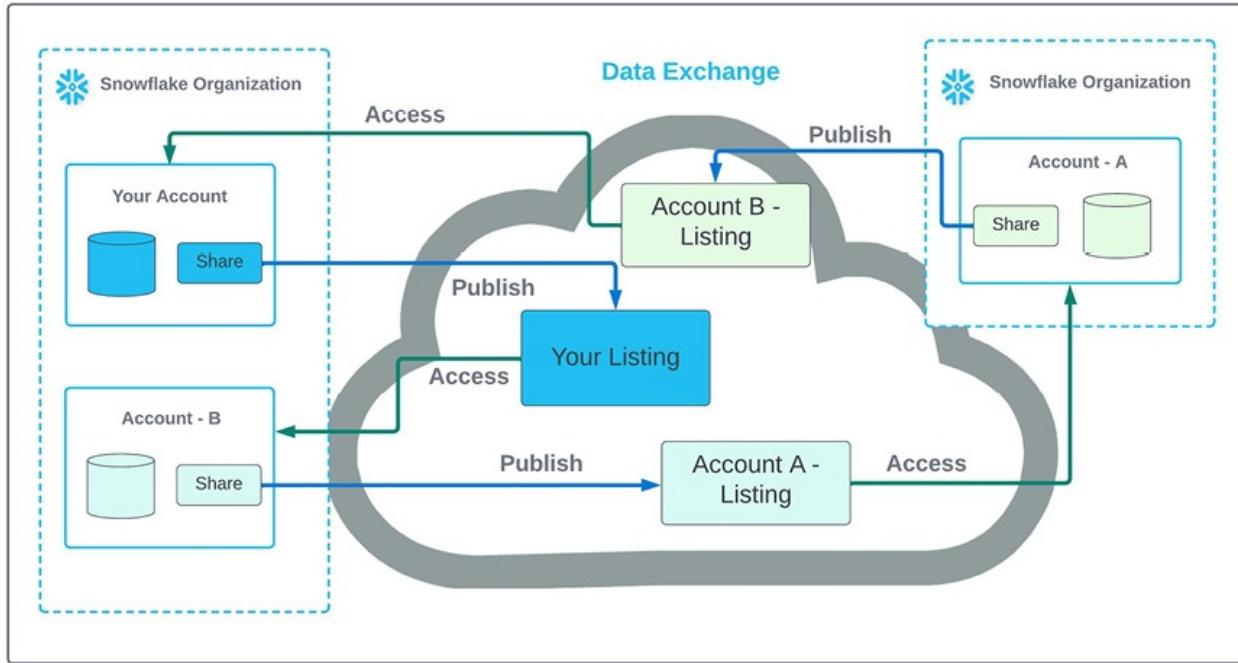


Figure 12.6: Snowflake Data Exchange

Note that data exchange is not enabled for all accounts. You can inquire about it with Snowflake support.

You can set up the data exchange with a Snowflake account, and the account that hosts it is referred to as the data exchange admin. This admin configures the data exchange as well as manages members – data providers and data consumers. You can set up an **ACCOUNTADMIN** to a role to assign it as a data exchange admin. This exchange admin can add Snowflake accounts as providers, consumers, or both. Once you are added to the exchange, you can perform the tasks discussed in the following sections as a provider or consumer.

Providers tasks

Accounts added as providers can perform the following tasks:

- Create a listing
- Define access to the listing – free or paid
- Publish listing
- Manage and grant access to the personalized listings

Consumer tasks

Accounts added as consumers can perform the following tasks:

- Discover datasets by browsing listings available in exchange.
- Can switch between data exchange and Snowflake marketplace.
- Can consume datasets- on request or instantly.

You can view the Snowflake data marketplace from UI – Snowsight. Search for the available listings and get the data that is required – free or paid, depending on your business requirements to access data from the marketplace. You can follow the given steps to explore data in the marketplace:

1. Go to Snowsight.
2. Logon to Snowsight – Account URL.
3. Go to **Marketplace**.

You will see the datasets available and some of the tags associated with the datasets. You can also filter based on the domain, and your business requirements. You can click on get data to get it added as a database to your account. You can use the data sharing features to share the data with consumers. You can also avoid any additional pipelines to manage data and data consistency issues. You will learn more about the data sharing implementation with labs and scenarios in the next section.

Practical: Create Snowflake shares

In this section, you will implement the given scenario and a customer use case. You need to follow the below sections to understand the overall scenario, the customer use case, technical requirements, account setup, and data sharing needs before you proceed with the exercise.

Scenario

Consider a retail use case where you have a data platform setup with Snowflake. You will be processing sales, orders, customer, and purchase data with various data pipelines. You are required to share data with the internal sales team for the analysis of sales. You also need to share the inventory data with your stores and capture the order details to maintain the inventory of products. You also have a requirement to share the sales data with the marketing team to process and generate promotional offers for the customers.

Technical requirement

As an admin, you can set up a consumer role, create data shares for internal teams, such as sales and marketing. You can create a data share for the external team – inventory data who are non-Snowflake user.

Account setup

You need to set up a database, schema, and tables for your retail requirements. The following are the pre-requisites to be set up before data sharing:

- **Create a database:** RETAIL_POC
- **Create a schema:** POC
- **Create tables:**
 - Customer_details
 - Sales
 - Order_details
 - Inventory_details

Data sharing

Now, you have an account set up and sample data loaded. Next, you need to set up role for data sharing and create shares.

- **Create roles:**
 - Data sharing admin – Share_admin
 - Assign privilege to share the data to the retail objects
- **Create shares:**
 - Create sales share
 - Create marketing share
 - Create inventory share
- **Reader account setup:**
 - Create reader account for inventory data sharing
 - Create a warehouse for reader account

- Assign warehouse for reader account
- Create a resource monitor to review reader usage
- **Validate data shared as reader account:**
 - Logon as reader account
 - List the data shared
 - Access inventory data

Conclusion

This chapter provided a summary of Snowflake's data sharing feature and options to share the data with consumers. It covered information on various data sharing options – private sharing, listing, data exchange, and Snowflake marketplace. We also covered details of data sharing with Snowflake as well as non-Snowflake accounts. This also helps you to understand public data sharing and data listing. You can use a sample retail use case as a reference to understand data sharing, and you will be able to set up direct data sharing with Snowflake and non-Snowflake consumers.

In the next chapter, you will learn about Snowflake's data cloning feature. You will learn more about data cloning and its benefits.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake offers instant data sharing with consumers.
- Data sharing is one of the distinguishing features of Snowflake that allows users to share data with consumers without data movement.
- Data is shared at the cloud services layer and metadata level. No actual data copy or movement is done.
- Consumer accounts do not need to maintain any copy of the data until they create a new database from share.
- Consumer accounts do not need to pay for storage. They only need to pay for compute required to query data.
- You can share data with consumers in private, public, free, or paid

offerings.

- You can also act as a data provider – list your data in the marketplace and share it as a free or paid listing.
- Snowflake data exchange is an account that allows various providers and consumers to publish or access data across.
- Snowflake's reader accounts are the accounts created, managed and assigned to the users that need access to the data share but do not have a Snowflake account.
- Reader accounts do not need to maintain Snowflake or warehouse resources to query or access data shared.
- Provider account manages the resources required for reader account – warehouse or compute required to query data.

Questions

1. How does Snowflake data sharing work?
2. How can you share data with Snowflake accounts?
3. How can you manage data shared with non-Snowflake accounts?
4. Can you manage the warehouse setup for reader accounts?
5. Can you setup monitoring of warehouse allocated to reader accounts?
6. How can you share data with multiple consumer accounts?
7. How is data storage calculated for provider and consumer accounts?
8. Can consumers add, remove, or edit data in data shares?
9. As a consumer, you need to update the data shared in data share. What access privileges are required to set this up?

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 13

Data Cloning

Introduction

The data pipeline, as well as the data lifecycle, follows certain stages before it gets deployed to production and starts running live to serve the business use cases. These pipelines are to be developed, tested, integrated in different environments, and validated before deploying to the production environment. In a typical scenario, developers or data engineers develop pipelines in a DEV environment and perform unit testing to validate the logic developed. Once it is tested in DEV, this is moved to the QA environment, where it gets integrated with other pipelines to build an application or enhance existing application flow. The QA team validates and runs QA test cases to pass the pipelines being developed. Once this passes through QA, it is moved to UAT for user acceptance testing, where a set of users validate the data being processed. Once this is done, it is set to be deployed in PROD. The pipeline lifecycle is typically followed in most of the use cases to ensure it follows the standards and meets the business requirements before going live. In a typical scenario, the environments are maintained separately with minimal or sample data for development, getting specific datasets for QA and UAT to meet and test the requirements.

With continuous development and deployment, all phases might not be needed, but in most cases, DEV, QA, and PROD are maintained to ensure the lifecycle is followed, and automated deployments carry the required dependencies across environments.

Managing environments like DEV, QA, or UAT and PROD, becomes challenging when the user wants to develop and test their features with greater

data volumes or production datasets. This chapter guides how to set up environments using the data cloning feature and helps to understand how easy it is to clone and maintain copies of data across environments. This chapter also helps you understand data cloning, how to implement it, and how to use data clones.

Structure

This chapter consists of the following topics:

- Data cloning needs
- Zero copy cloning
- Implementing data cloning
- Understanding the usage of cloned objects

Objectives

By the end of this chapter, you will be able to develop an understanding of Snowflake data cloning and zero copy cloning. This also covers implementing data cloning, understanding cloned objects, and using cloned data and its usage. You can implement data cloning, set up multiple environments like DEV and PROD, and access PROD data using data clones from the **DEV** environment using the trial account setup in [Chapter 1: Getting Started With Snowflake](#).

Data cloning needs

In a typical data implementation, you need to set up separate environments for the data and pipeline lifecycle. You also need to maintain the internal teams and stakeholders who are onboarded to the data platform to consume the data from the platform. In an enterprise data scenario, you will have multiple teams accessing your platform to consume data, build their applications, data analytics or BI, and so on. You need to ensure that you are maintaining separate resources: warehouses, storage- databases, and objects for these teams. While you align the Snowflake resources for internal stakeholders, you also need to ensure you are providing the required datasets for these consumer applications. Typically, you end up creating an additional pipeline to move or load the required data to another database for consumer applications. You develop pipelines to load transformed and aggregated data to the tables or materialized views.

In the case of Snowflake, you can create materialized views on top of tables for consumers, and you no longer need to maintain the load of these views.

Snowflake takes care of these operations and charges you accordingly. You can also implement data sharing to share the data. However, this is preferably used for external stakeholders or consumers. In the case of internal stakeholders and teams, you can use data cloning to create datasets for consumer applications.

Similar to data sharing, data clones copy the data at the metadata and cloud services layer. Data does not get duplicated or copied to incur any additional storage costs. The data clones are a little different than data sharing in terms of maintaining changes in the data and data state. You will learn more about clones in the upcoming sections.

Zero copy cloning

Data clones are snapshot of data. These are the snapshots of the data at the time of creation. For example, if you create a clone at 12 noon, then the clone created consists of the data state at noon. If any other records are added or removed post-noon, they will not be reflected to the clones automatically. These cloned objects are treated as separate objects in a database. If any changes are made to the clones, then it costs additional storage charges as per the changes.

With Snowflake, you can create multiple parallel environments leveraging the organizational hierarchy. You can create separate databases representing **DEV**, **QA**, and **PROD** environments. Onboard users manage their roles and accesses at the database level. You can have a data model implemented in all three databases; however, the data can be productionized and live in the **PROD** database, and other databases can have cloned data from **PROD** depending on the need to have required datasets or duration data for development and **QA**.

As the name says, the clones are often referred to as zero copy clones, as there is no actual data copy that takes place when you define a clone. The data clone is created, and it points to the same storage or memory location or micro-partitions as the source table. The data is read from the primary storage as and when the queries are run on data clones. If you modify anything in the clone, then it starts creating additional micro-partitions to maintain new data. In this case, the old data is still pointed to the primary memory, and the new data starts pointing to the new micro partitions created. In this case, the single primary memory can be associated with multiple cloned objects. This is managed at the metadata and cloud services layer.

You can refer to [*Figure 13.1*](#), which represents the zero-copy cloning scenario. The figure consists of two layers: Cloud Services, which are the Metadata Layer and Storage Layer. There are two primary tables: **T1** and **T2**. These primary tables are currently pointing to the primary storage partitions in the Storage layer as presented with actual data storage in partitions. There are two clones created from these two tables: **T1_C1** is the clone of table **T1**, and **T2_C2** is the clone of table **T2**. These two clones are pointing to the actual data storage partitions for old data that point in time of clone creations. Later, the clones are used by developers, and new data is added to them. The new data is stored in new micro partitions at the storage layer as represented. Any new data in the clone gets stored in the new micro partitions. For any old data reference, it will continue to read from the actual partitions. There is an additional storage cost associated with new data partitions created at the storage layer. You will pay an additional cost only for new data. In the case of old data, it will not be duplicated or charged again.



Data Cloning in Snowflake

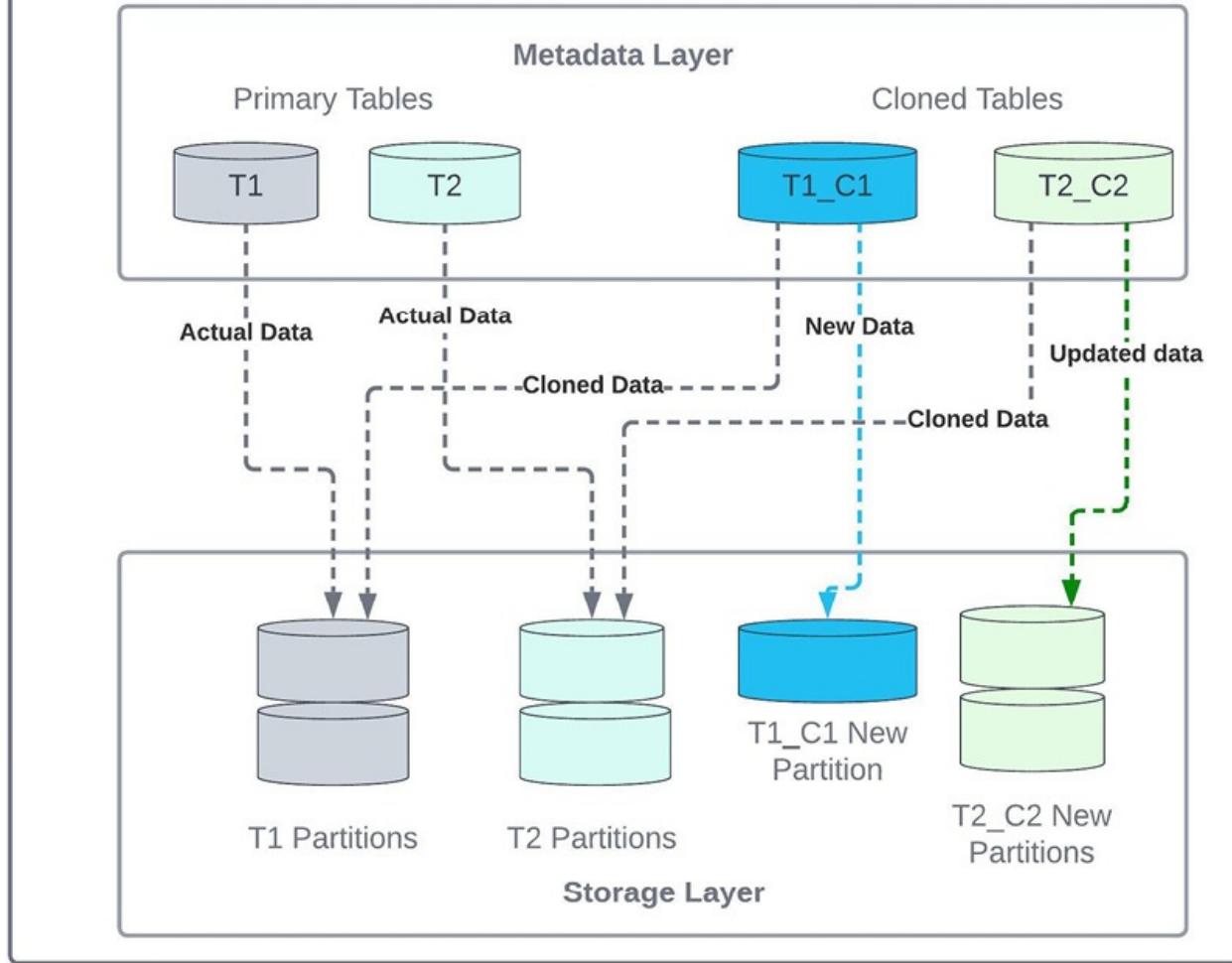


Figure 13.1: Data Cloning in Snowflake

Like any other Snowflake objects, you can create clones using DDL commands. You can create clones of certain objects in databases and schemas. You will learn more about creating and managing clones in the next section.

Implementing data cloning

Data cloning can be created using **CREATE DDL**. However, there are a few considerations while creating a clone of Snowflake object. The following section will help you understand these considerations to clone Snowflake objects.

Cloning considerations

You need to consider the Snowflake objects, schema, permanent or standard tables, DDLs and DMLs of Snowflake objects that can affect the cloned objects. Data retentions like time travel and retention duration can also affect cloning.

As you know, Snowflake users are granted access based on their roles. Users need access to create cloned objects. You can clone the database or schema, and this allows the clone to inherit all grants and privileges of cloned child objects in the cloned database or schema. The **CREATE CLONE** command does not copy grants on the source objects to the clone. Below are some of the considerations for Snowflake objects before cloning them:

- **Cloning sequences:** If you clone a database that consists of clones or if you are cloning a table that references a sequence column in the database, then the cloned table can refer to the sequence from the source or cloned database.
 - If you clone a database that consists of the table referencing the sequence column and the sequence also gets cloned, then the cloned object refers to the cloned sequence.
 - If you clone only a table without sequence, then the cloned table refers to the source sequence.
 - You can use the **ALTER** command to point the sequence to a new value or sequence:

```
ALTER TABLE <tablename> ALTER COLUMN  
<columnname> SET DEFAULT <newsequence>.nextval;
```

- **Cloning constraints:** A table can have a foreign key constraint and point to another table that has a primary key. You can clone a table with constraint.
 - If you clone a table with constraint, then it points to the primary key of a table from the clone if the table is also cloned. The primary key and foreign key constraint tables are part of the same cloned objects.
 - If the cloned objects do not consist of the table with the primary key table, then it points to the table from the source.
- **Cloning clustering keys:** A table can have a subset of columns as clustering keys that help to co-locate rows in micro-partitions. If you clone a table with clustering keys, then it clones the table with clustering keys,

but the clustering is not enabled automatically. You need to run SQL to resume the auto clustering:

```
ALTER TABLE <table_name> RESUME RECLUSTER
```

- **Cloning stages:** Snowflake supports various stages: internal, external, table, and user stages. You cannot clone the internal stage. You can copy external stages. The cloned object, which is the external stage, does not have any impact on the referenced cloud storage. If you clone the table, then the table stage also gets cloned. However, the data files from the source stage are not cloned.
- **Cloning pipes:** Snowflake pipe loads data from stages to the tables. Snowflake pipes loading data from internal stages are not cloned. Any pipes referencing to the external stages can be cloned.
 - Pipe internally uses the **COPY** command to load the data. If the **COPY** statement in pipe is fully qualified, then this loads duplicate data into source table in place of cloned object.
 - If **COPY** does not have a fully qualified name, it loads data to the source and cloned objects.
 - This is a critical use case where you have to be careful while cloning objects, otherwise, you might end up duplicating data as the cloned object does not clone the metadata information to avoid duplicate file loads.
- **Cloning streams:** Snowflake streams are used to capture the changes as part of CDC. You can clone the streams. If you clone the streams, then any changes from the source stream that are not cloned are inaccessible.
- **Cloning tasks:** Snowflake tasks can be cloned. If you clone the tasks, then the cloned tasks are in suspended state by default. You need to run the **ALTER** command to **RESUME** tasks.
- **Cloning governed objects:** Snowflake supports masking, row access policies, tags, and tag-based masking policies.
 - You cannot copy individual object policy.
 - Tags associated with objects get cloned with source database or schemas.

You need to consider the above-mentioned considerations while working on data cloning. You can create clones using DDL commands and use DML commands on cloned objects.

Cloning implementation

You can create clones using the **CREATE** command. This creates a copy of the object and is usually used to create zero copy clones of databases, schemas and tables. This statement is also used to create clones of other objects like sequences, formats, and external stages.

Syntax of CREATE

Refer to the following syntax to create a cloned object:

```
CREATE
<database|schema|table|stage|fileformat|sequence|stream>
  objectname CLONE <source
  database|schema|table|stage|fileformat|sequence|stream>
```

You need to consider some of additional consideration on top of considerations mentioned in preceding section. You can use syntax shared above to create cloned objects. Some features of clones are:

- Clones are writable objects that are independent of source.
- Clones carry the parameters, grants, privileges, parameters, permissions from the source to the cloned objects.
- You need to have appropriate privileges to clone the objects:
 - You need **SELECT** privilege to tables.
 - You need **USAGE** privileges for other objects.
 - You need **OWNERSHIP** privileges for pipes, streams, and tasks.
 - You will also need privileges to the target or cloned objects.
- You can clone most of the objects within the database or schema. However you cannot clone a few objects like external tables, internal stages, pipes leading to internal stages, etc.
- Cloned objects like tables or pipes do not carry the load history or metadata information to clone. You may end up loading duplicate files to

the cloned objects, as there is no metadata to validate the duplicate loads.

- You can also provide **COPY GRANTS** along with the **CREATE CLONE** statement:
 - You can use **COPY GRANTS** to inherit any explicit privileges to the new object that are granted on the original table. However, this does not inherit any future grants.
 - If you are not using **COPY GRANTS**, then it does not inherit any privileges to the new object from the source. However, this inherits any new future grants from the source to the cloned object.
- You can also create a clone with time-travel. You can use the **AT** clause while creating a clone to create objects from the past.

Reference use cases and examples

Let us take a look at some of the use cases and examples:

- **Create clone of database and its objects:** Use the following command to create the clone. This will clone all cloneable objects from the source database:

```
CREATE DATABASE RETAIL_CLONE CLONE RETAIL_DB;
```

- **Create a clone of schema and objects:** Use the following command to create a clone of schema, and this clones all the allowed objects from the schema:

```
CREATE SCHEMA POC_CLONE CLONE POC;
```

- **Create clone of a table at given timestamp:**

```
CREATE TABLE SALES_ORDERS_RESTORE CLONE  
SALES_ORDER AT (TIMESTAMP =>  
TO_TIMESTAMP_TZ('09/27/2023 09:41:30', 'mm/dd/yyyy  
hh24:mi:ss'));
```

You can also create clone of schema at given timestamp. You can create clones of objects based on your requirements and privileges granted on objects. You have learnt that cloned objects do not incur any additional costs and storage is not duplicated until you add any new data or modify data of cloned objects. In the next section, you will learn more about the usage of Snowflake resources while creating zero copy cloning.

Understanding the usage of cloned objects

Clones are also referred to as snapshot of tables. This is also used to take instant backup of the tables or objects as this does not incur any additional storage cost. Clones do not cost any additional credits until the data is modified. Clones make storage computation complex as they need to maintain any new changes separately and have a separate lifecycle. These changes need to be tracked independently for source and cloned objects.

You can consider a scenario to understand the utilization better: Consider that you have a table, and you create a clone of that table. The cloned object points to the same data stored in micro partitions initially. You can run DML operations on clones. Consider that you run a few updates, inserts, and deletes to the cloned object, and then you need to maintain the storage and track operations separately on cloned objects. These are tracked independently from the original table. Any new changes to the clones are managed separately and new micro-partitioned created and protected through CDP.

This becomes more complex as you are allowed to clone the cloned object. You can create multiple clones from cloned objects. With this n-level hierarchy, it becomes complex to maintain a separate data lifecycle with own and cloned data. This also protects data through CDP.

Whenever you create a clone of a new table **ID**, **CLONE_GROUP_ID** gets assigned to the original table. All micro partitions are shared as soon as the table is cloned. The original partitions and data are owned by the original oldest table in the cloned group, and the cloned table references micro partitions. Once a clone is created, the original and cloned objects are tracked separately and have a separate life cycle. You can get storage details of these micro partitions from the **RETAINED_FOR_CLONE_BYTES** column in the **TABLE_STORAGE_METRICS** view.

You can calculate the usage and monitor overall usage for original as well as cloned objects using metadata view: **TABLE_STORAGE_METRICS**.

Practical: Create Snowflake clones

1. Create a clone of database created for the following retail use case:
 - a. Create clone of POC schema and objects.
 - b. Create clone of a table from POC schema.

- c. Create a data pipeline that loads from files to the cloned object.
- d. Calculate the data storage from cloned objects.

Conclusion

This chapter provided a summary of Snowflake's data cloning feature and options to clone or take snapshot of the data. This chapter covered information of various data cloning considerations, data objects allowed to be cloned, privileges required to clone objects from databases or schemas. This also helps you to understand the storage, storage usage and maintaining data micro partitions for original, cloned as well as objects in a cloned group. You can use sample tables from the retail use case as reference to understand data cloning and you will be able to setup data clones in an account.

In the next chapter, you will learn about Snowflake's web UI classic console as well as Snowsight. You will learn more about Snowsight and its features.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake offers data cloning, instant data backup, and data snapshot.
- Data cloning is one of the distinguishing feature of Snowflake that allows users to create data backup or data copy without data movement.
- Data is cloned at cloud services layer and metadata level. No actual data copy or movement is done.
- Data clones can be created at database or schema with all existing cloneable objects.
- You cannot clone external tables, internal stage, or pipe loading from the internal stage.
- A table ID and cloned group ID is created for the original table and tables in clone.
- A separate data lifecycle is maintained, and data is protected through CDP.
- You may end up duplicating data to source tables if the cloned pipe refers to the fully qualified table name.

Questions

1. How does Snowflake data cloning work?
2. How can you clone data within Snowflake accounts?
3. How can you manage data cloned?
4. How can you measure the storage of data cloned?
5. Can you setup monitoring of storage utilization of cloned objects?
6. How can you clone data from clones?
7. How data storage is calculated between original and cloned objects?
8. Can users add, remove or edit data in data clones?
9. How is data manipulation done on data clones?
10. Can you setup multiple clones from a single table?

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 14

Understanding Snowsight

Introduction

Snowflake's web interface is called Snowsight. This chapter helps you to understand the features of Snowsight. You will learn more about using the SQL interface, creating and managing multiple worksheets, data governance, and the admin and data features of Snowsight.

Structure

This chapter consists of the following topics:

- Understanding Snowsight
- Implementing dashboards with Snowsight

Objectives

By the end of this chapter, you will be able to develop an understanding of Snowsight and its features. This also covers implementing various use cases, dashboards, and new UI features. You must be using Snowsight as Web UI with the trial account setup in *Chapter 1, Getting Started with Snowflake*.

Understanding Snowsight

Snowsight is Snowflake's web UI. Once you sign up for a trial account, you get the app link: app.snowflake.com, and you can log on to the account using the

user ID setup. This is the default app and UI link for all new account setups. This section will help you as a guide and tour to Snowsight.

Once you log in to Snowsight, you can create and manage all types of Snowflake objects. You can manage warehouses, databases, database objects, and load small datasets to the Snowflake tables. You can create SQL worksheets, execute SQL queries, and view past queries in the query history. You can also create and manage account users using Snowsight. Snowsight has many features embedded for developers, data engineers, architects, and admins. Following are some of the key features:

- **Improved productivity:** Snowsight adds an auto-complete feature. This allows users to select objects and SQL functions. This is one of the favorite features of developers.
- **Visualization:** This feature allows users to build insights. You can create widgets with supported graph formats and develop dashboards.
- **Collaboration:** This is one of the most used features where you can create worksheets and dashboards and share them with the team. You can create worksheets as part of development and share them with developers for collaboration. Users can also add, modify, review, and provide inputs on the content being developed.
- **Worksheet management:** As you know, you can create worksheets and share them with teams. You can also maintain multiple worksheets and create folders to organize them as part of worksheet management.
- **Data usage:** You can view the actual usage for users, warehouses, queries etc. You can view it in tabular as well as graphical format.
- **History:** Snowsight offers a more intuitive way to view the query history, copy history, and data loading history. You can also view the workloads in history.
- **Manage roles and users:** You can view the users, roles, and their grants in a new Tree view.

Snowsight interface

Refer to [*Figure 14.1*](#) for the three sections of the Snowsight interface. Following are the sections shown in the figure and overview of each section:

- **User menu:** This is the menu section where you can change role, update your account profile, documentation, or log out.
- **Navigation menu:** This menu consists of various items such as worksheets, data, dashboards, marketplace, activity, admin, and help support.
- **Content pane:** The content pane displays corresponding content as per the navigation menu item selected. Some of the menu items may also have additional child windows.

TITLE	TYPE	VIEWED	UPDATED	ROLE
2023-10-03 12:07pm	SQL	just now	7 hours ago	ACCOUNTADMIN
2023-07-13 1:42pm	Python	7 hours ago	7 hours ago	ACCOUNTADMIN
Snowflake Usage metrics	SQL	7 hours ago	7 hours ago	ACCOUNTADMIN
2023-09-10 12:01am	SQL	7 hours ago	7 hours ago	SYSADMIN
2023-09-18 4:38pm	SQL	7 hours ago	7 hours ago	ACCOUNTADMIN
2023-07-17 5:21pm	SQL	3 weeks ago	7 hours ago	ACCOUNTADMIN
column_security_masking_samples	SQL	3 weeks ago	3 weeks ago	ACCOUNTADMIN
Notification integrations	SQL	3 weeks ago	3 weeks ago	ACCOUNTADMIN
2023-07-25 11:14am	SQL	3 weeks ago	7 hours ago	ACCOUNTADMIN

Figure 14.1: Snowsight Page

Some of the commonly used navigation menu items in development are **Worksheets**, **Dashboards**, **Data**, and so on. The following section lists down the various menu and content pages.

Worksheet pages

You can create SQL as well as Python worksheets. SQL worksheet is used to write DDLs, DMLs, and queries to develop Snowflake objects as well as pipelines. You can also view the result set and share the worksheets with other developers for collaboration. You can perform the following tasks as the user:

- You can develop and test SQL queries.
- You can share worksheets, review SQL queries, and develop SQL queries.
- You can also export results, view results, as well as use graphical representation to view the result data set.
- Each worksheet is an individual session for Snowflake.

- You can set the context using SQL commands as well as context using web UI. Every worksheet can have a separate context and session.

You can refer to *Figure 14.2*, the content pane is where you can write the SQL queries:

The screenshot shows the Snowflake Worksheet Page interface. On the left, there's a sidebar titled 'Databases' and 'Worksheets'. Under 'Databases', several databases are listed, including COVID19, CYBERSYN_FINANCIAL_ECONOMIC_ESSENTIALS, Chinook, DEMO_JENKINS, DEV_POC_DB, DFS_POC_DB, EMAIL_MD_TEST_SNOWFLAKE_SECURE_SHARE_165972574..., FROSTY_SAMPLE, GUZZLE, HEALTH_CHECK, HOL_DB, KNOEMA_ECONOMY_DATA_ATLAS, MARKETPLACE_SNOWFLAKE_SECURE_SHARE_165636469..., OPS_POC, PC_ALTERYX_DB, PC_DBT_DB, PC_FIVETRAN_DB, and PO_MATILIOHLOADCR_DB. Under 'Worksheets', it says 'No pinned objects'. The main content area has a header 'DEV_POC_DB.POC' with a 'Settings' dropdown. Below the header is a text input field containing '1 Write your queries here |'. At the top right, there are buttons for 'ACCOUNTADMIN - COMPUTE_WH', 'Share', and a play button. The status bar at the bottom indicates 'Draft'.

Figure 14.2: Worksheet Page

Refer to *Figure 14.3*, where you can write SQL queries, execute them, and view results as a result set. You can also download using the Result set options. Figure content represents the below menu items:

1. **Worksheet query:** You can develop SQL queries here.
2. **Result set:** You can view the results in this pane.
3. **Role & Warehouse:** You can set the context here.
4. You can select the database schema from a drop-down menu.

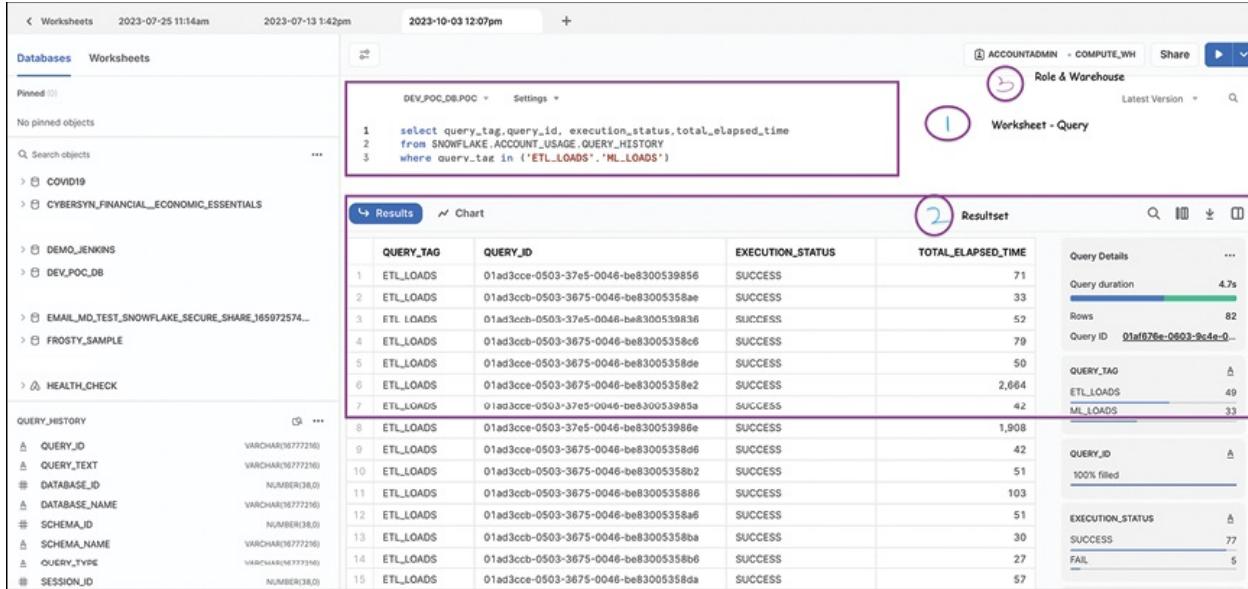


Figure 14.3: Worksheet and content pane

Data pages

You can view the data page from the navigation menu item. You can get a list of all existing Snowflake data objects. You get a consolidated view of databases, schemas, and objects within this hierarchy. You also get the privileges granted on the database representing ownership, usage, and other privileges granted to the Snowflake users.

You can refer to [Figure 14.4](#), where you will get a list of all databases, schema, and Snowflake objects. Once you click on the database, you get a list of all objects and schemas within the Snowflake database. You can also list the privileges by selecting the object. You can refer to items 1, 2, and 3 from the following figure:

1. This represents data from navigation menu items.
2. Represents a list of databases in the database menu pane.
3. Represents the privileges and schema details of the databases selected in the database pane.

The screenshot shows the Data and Content pane. On the left, the navigation menu is open, with 'Data' selected under 'Apps'. A circled '1' is next to the 'Data' icon. The main content area shows a list of databases. A circled '2' is next to the 'INFORMATION_SCHEMA' entry under the 'COVID19' database. On the right, a detailed view of the 'DEV_POC_DB' database is shown, including its creation details and a 'Privileges' section. A circled '3' is next to the 'ACCOUNTADMIN (Current Role)' entry in the privileges list.

Figure 14.4: Data and Content Pane

Dashboard page

You can go to the dashboard from the navigation menu. You will have the dashboard content pane opened once you select dashboards. This dashboard pane lists all existing dashboards in the account. If you do not have any dashboard created, you can choose the **Create Dashboard** option in the content pane. You can refer to [Figure 14.5](#) to get a list of the menu items and content pane:

1. Dashboards menu item from the navigation menu.
2. List of dashboards from the dashboard content pane:

The screenshot shows the Dashboards page. The navigation menu on the left has 'Dashboards' selected under 'Worksheets', with a circled '1'. The main content area displays a table of existing dashboards. A circled '2' is next to the 'demo' dashboard in the list. The columns in the table are TITLE, VIEWED, UPDATED, and ROLE.

TITLE	VIEWED	UPDATED	ROLE
demo	1 month ago	1 month ago	ACCOUNTADMIN
Account Usage Dashboard -	3 months ago	6 months ago	ACCOUNTADMIN
Engineering - Demo Dashboard	3 weeks ago	6 months ago	ACCOUNTADMIN
Operations -Demo Dashboard	6 months ago	6 months ago	ACCOUNTADMIN
Usage - Demo Dashboard	6 months ago	6 months ago	ACCOUNTADMIN
Organization Usage Dashboard -	3 months ago	8 months ago	ACCOUNTADMIN

Figure 14.5: Dashboards page

You can create dashboards using **+Dashboard** menu item from the top right corner. You can also share the dashboards created using the share option. You will learn more about this in the next section.

You will find new features like Streamlit and Apps that have been recently added to Snowsight.

Data page

This page lists down the various data objects deployed in an account. This includes the databases, schemas, and Snowflake objects, such as tables, views, functions, file formats, etc. You can also get the DDLs of Snowflake objects from this page.

Marketplace page

This page lists down the various datasets available to share and be shared. You can get the list of various datasets available to the consumer and as a consumer. You get free, paid, and available data to be consumed in the marketplace.

Activity page

This page lists the four types of activities: Query history, Copy history, Task history, and Dynamic tables. The query history page lists the queries and workloads. You get filters on top of the page to filter the workloads for the last day, 3 days, 7 days, 14 days, and custom dates. You get a variety of filters to filter the workloads using SQL ID or query ID, warehouse, statement type, query tag, and many more. The copy history page displays and lists the copy and data loaded to databases. The task history page displays as well as lists down the various tasks running and setup for a database with date and task status filter. Dynamic tables list the tables available.

Admin page

This page lists child pages that cater to usage, warehouses, resource monitors, users and roles, security, billing and terms, accounts, contacts, and partner connect. You will need **ACCOUNTADMIN** access to view these pages and access details for an account.

In the next section, you will learn more about deploying dashboards using Snowsight.

Implementing dashboards with Snowsight

Snowsight dashboard feature is one of the most interesting and widely used features to build operational and monitoring dashboards. You can set up the dashboards and share them with teams as per your business requirements.

You can consider a use case to understand the dashboarding needs, features, implementation, and sharing with teams. Consider an enterprise use case where you have Snowflake as the unified data platform, and multiple teams are onboarded to your platform to read, write and process their data. The Snowflake onboarding, user management, and RBAC are well defined, where you have separate roles and privileges assigned to these internal and external stakeholders. You have separate warehouses assigned to each group to track their overall utilization, workloads, and manage resources. You can consider the following key architectural design choices implemented:

- Unified data platform
- Internal and external stakeholders accessing the data.
- Separate roles created for each team.
- **Internal stakeholders:** Data engineering, data analysts, sales, marketing, and business teams
- **External stakeholders:** Data shared with sales partners and vendors.
- **User management:** Individual roles created for internal and external stakeholders.
- **Resource management:** Separate warehouses allocated to each team.
- **Key asks:** Following are some of the key asks of the given use case:
 - Build an operational dashboard to monitor the overall utilization of resources allocated to the teams.
 - Monitor the warehouse credit consumption.
 - Monitor the failed workloads and generate alerts to the Ops team.
 - Query performance metrics to capture long-running queries, and most queued queries to analyze the query and warehouse performance.
 - Admin dashboard to monitor user login and access issues for the admin team.

- **Engineering workload monitoring:** Dashboard with engineering metrics that are essential for engineering teams to implement optimization.

Now, you need to define the metrics for each of the key asks mentioned above and start building the dashboard. You can also implement alerting using Snowflake native features. You will learn more about them in [Chapter 16, Workload Patterns with Snowflake](#).

Next, you need to define the metrics for each team to monitor their utilization. You can derive the metrics by using **ACCOUNT_USAGE** views. Following are some of the sample metrics for your reference. You can derive more based on your business as well as application requirements:

Category	Metrics	Usage view
Engineering	Long Running Queries	Query_history
	Queries over period of Time	Query_history
	Average query queue per Warehouse	Query_history
	Average Queries per Warehouse	Query_history
Ops Team	Top 10 Queries with failed status	Query_history
	Most failed workloads	Query_history
Admin Team	Most failed logins	Login_history
	Admins without MFA	Users

Table 14.1: Sample Usage Metrics

To create the dashboard, you can use the Snowsight dashboard feature. Follow the below steps to set the dashboard and add widgets to the board:

1. Create a Dashboard, as shown:

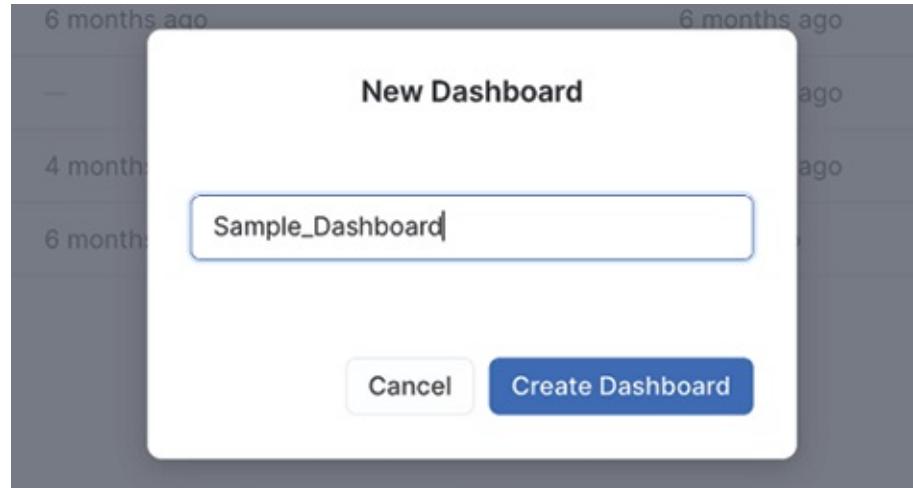


Figure 14.6: Create Dashboard

2. Add the first widget: Select **New Tile** drop down and select **From SQL Worksheet** as shown in [Figure 14.7](#) and [Figure 14.8](#):

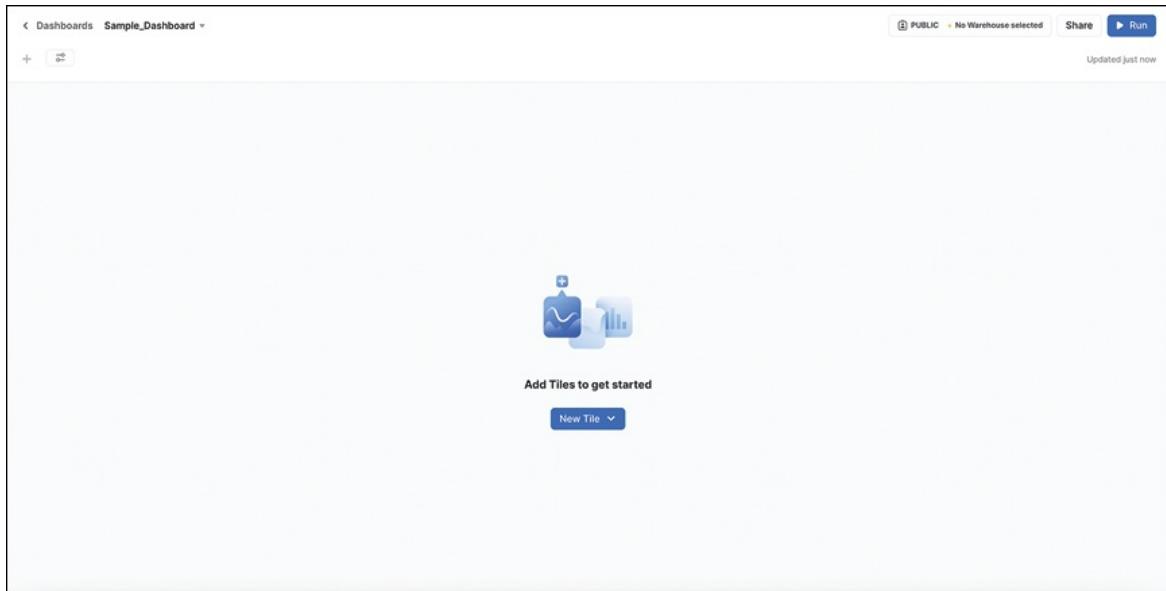


Figure 14.7: Adding Tile to Dashboard

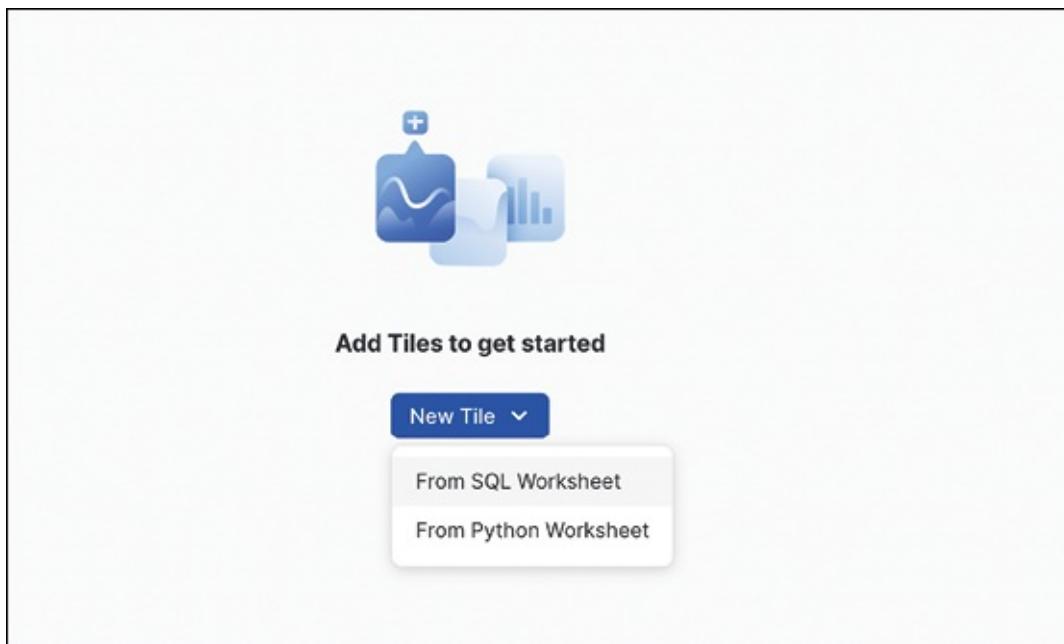


Figure 14.8: Select SQL Worksheet

3. Start writing SQL query on top of the **USAGE** view to generate the metrics required to be captured. This sample engineering dashboard demonstrates the creation of query metrics, as listed in *Table 14.1*. Refer to *Figure 14.9* and *Figure 14.10* to write SQL and execute to generate graph:

```

OPS_POC_METRICS + Settings +
1 select * from
2 (
3   select
4     query_id,
5     start_time,
6     total_elapsed_time,
7     warehouse_name,
8     ROW_NUMBER() OVER (partition by warehouse_name order by total_elapsed_time desc) as rank
9   FROM snowflake.account_usage.query_history
10  WHERE
11    QUERY_TYPE NOT IN ('DESCRIBE', 'SHOW')
12    \long_queries
13    where rank between 1 and 10;

```

Figure 14.9: Writing long running queries

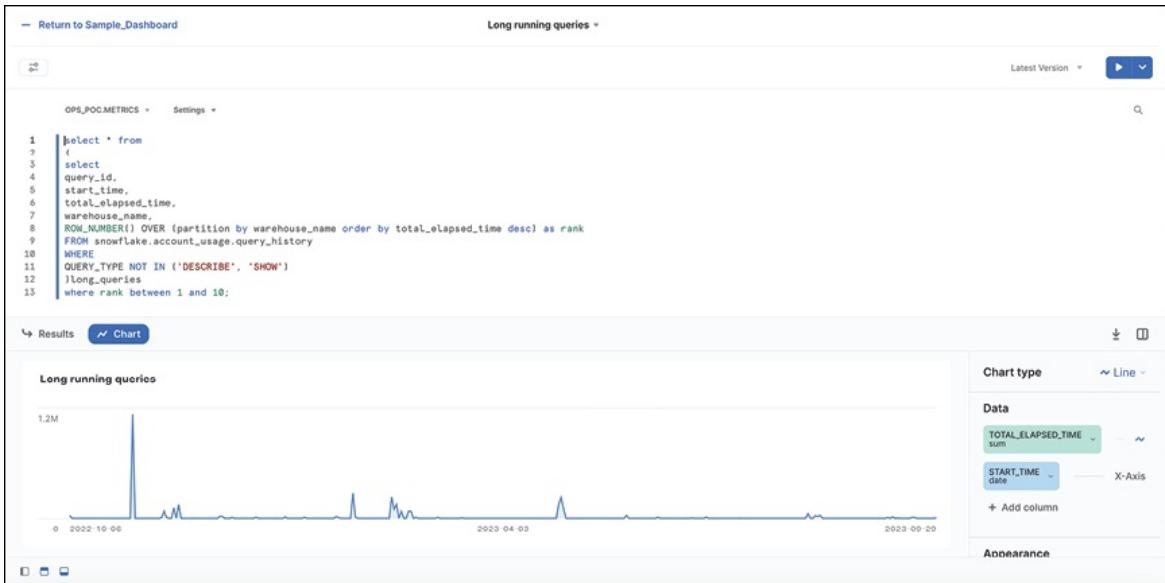


Figure 14.10: Executing Query and Building graph

4. This is the first widget added. Similarly, you can write the SQL for the rest of the three metrics and add widgets to the sample dashboard. Refer to [Figure 14.11](#):
5. Add **Queries Over Period of Time** Tile. Refer to [Figure 14.12](#):

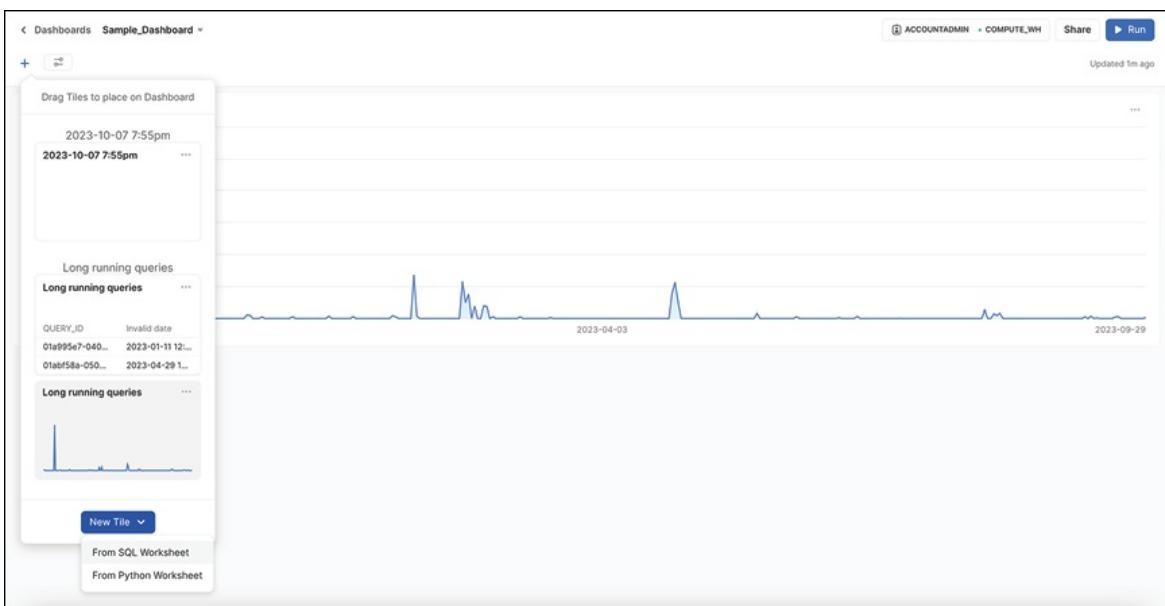


Figure 14.11: Adding New Tile to the dashboard



Figure 14.12: Writing Query metrics and generating graph

- Add the next metric, which is queue time per warehouse. You can query **query_history** view to get the average queue time to analyze the performance of warehouse loads. Refer to [Figure 14.13](#):

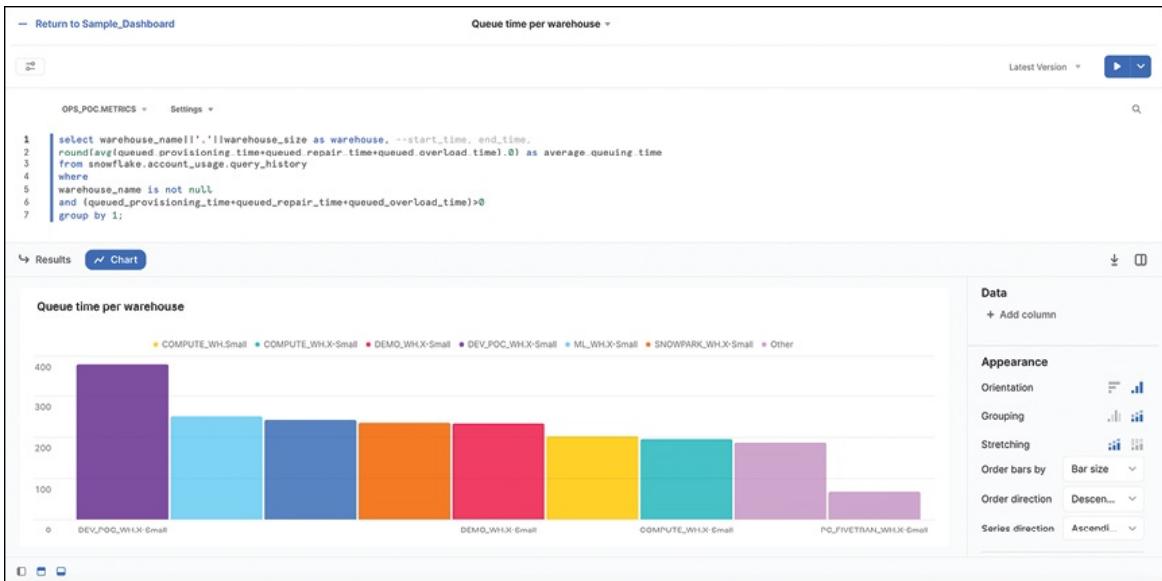


Figure 14.13: Execute Query and Add Graph

- Add the next tile, generate metrics: average workloads per warehouse. Run a similar query to capture total queries, also known as workloads or jobs running per warehouse. You can also generate it per database to capture workload details. Refer to [Figure 14.14](#):

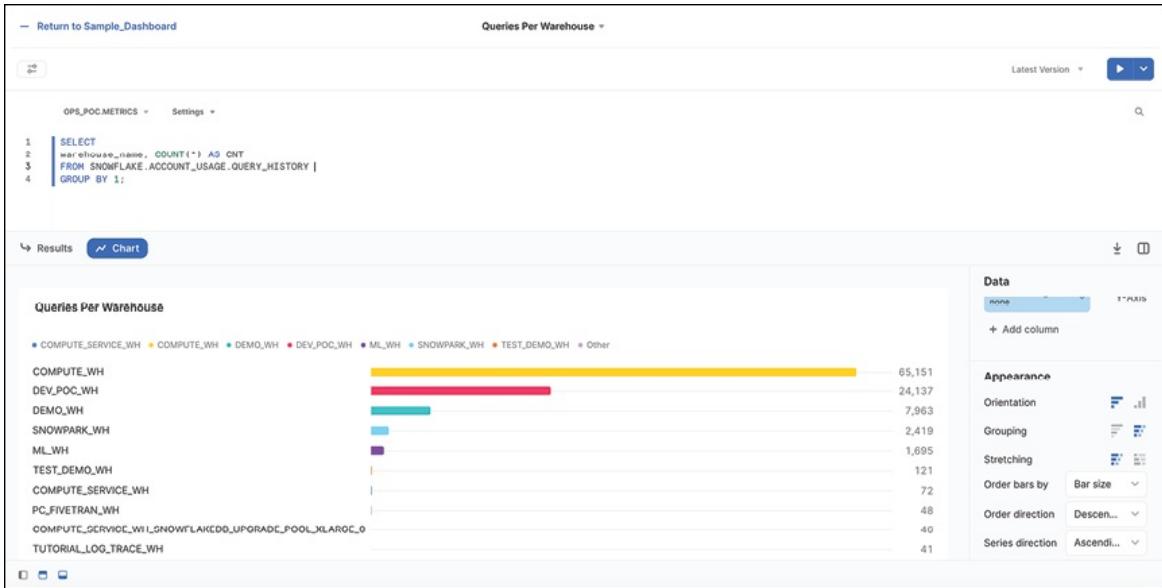


Figure 14.14: Write SQL and Generate Graph

- Add the tile to the dashboard and your engineering dashboard is ready to be shared with the engineering teams. This consists of the metrics defined in *Table 14.1*. Refer to *Figure 14.15*:

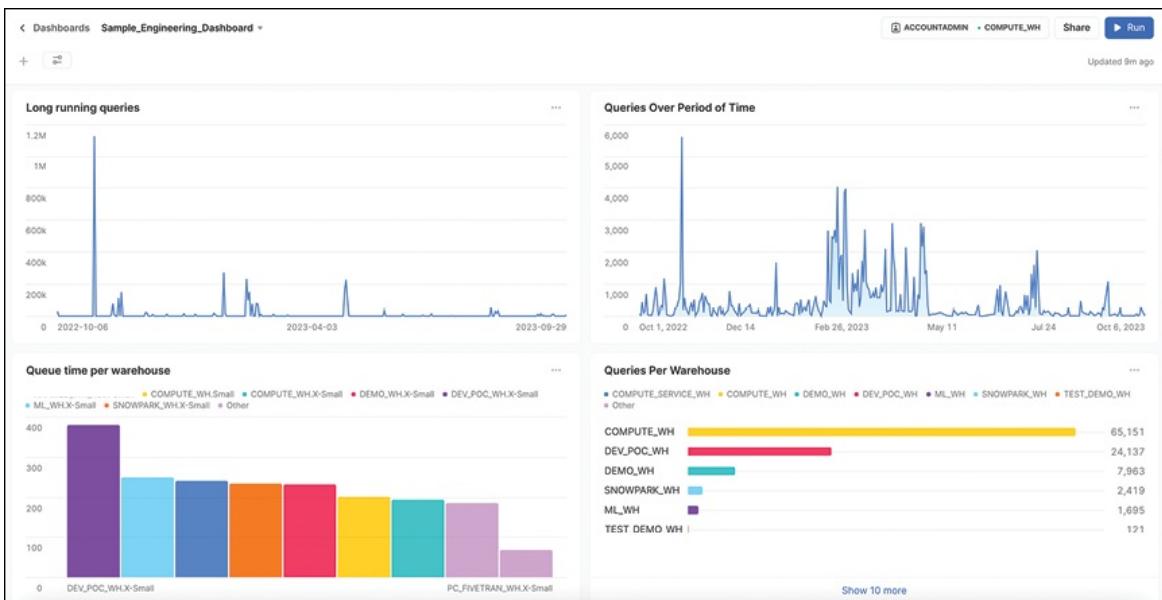


Figure 14.15: Sample dashboard

You can create Ops and Admin dashboards following the same steps as mentioned above. You can add the Ops or Admin metrics to those dashboards. You can also refer to the next section to get an additional use case as part of exercise of this chapter.

Practical: Create Snowflake dashboards

Refer to the following use case and requirements to set up the Snowflake dashboard for monitoring.

Use case

You are an admin of the Snowflake account, and you have only one Snowflake account hosted on AWS. Your account is used by 5 different teams: Engineering, Analysts, HR, Marketing, and Learning and Development. You have set up separate resources for each team. As an admin, you need to monitor the usage of each team to ensure they are consuming allocated resources and credits well. You are also keen to know if there are any resources that are overutilized or underutilized.

Initial set up

This will be the initial setup:

- You have 5 teams onboarded: 5 roles created for each team.
- You have 5 databases created for each team.
- You have a separate warehouse allocated for each team:
 - **HR:** XS (single warehouse)
 - **Learning and development:** XS (single warehouse)
 - **Marketing:** XS (single warehouse)
 - **Analysts:** XS (Cluster Warehouse of 3 nodes)
 - **Engineering:** S (Cluster warehouse of 3 nodes)
- You have different credits allocated for each team per month:
 - **HR:** 100
 - **Learning and development:** 100
 - **Marketing:** 100
 - **Analysts:** 200
 - **Engineering:** 350

Key asks

Following will be the key asks:

- Monitor workloads of each team.
- Monitor credit consumption for each team.
- Monitor warehouse performance.
- Monitor queued workloads, if any, for warehouse optimization.
- Set up a dashboard for Engineering and Ops team.
- Share the dashboard with the engineering team.

You can refer to the dashboard section and set up a new dashboard for the use case shared as part of the exercise.

Conclusion

This chapter provided a summary of Snowflake's Web UI: Snowsight. This chapter covered information on various features of Snowsight. It also helped you to understand new features like data governance and implementing dashboards using Snowsight.

In the next chapter, you will learn about Snowflake's drivers and connectors. You will learn more about various connectors used while designing data pipelines and integrations with Snowflake.

Points to remember

Following are the key takeaways from this chapter:

- Snowsight is a one-stop solution to design, develop, monitor, and implement operational dashboards.
- All types of DDLs, DMLs, and DCLs commands can be executed from Snowsight.
- Create all types of database objects, integrations, resource monitors, and tasks using the SQL worksheet of Snowsight.
- Snowsight can also be used to implement data governance features like object tagging, query tagging, dynamic masking etc.

- Snowsight can also be used to implement dashboards for operational and monitoring activities.
- Snowsight dashboards can be implemented to use dashboards, and operational dashboards.
- Snowsight dashboards can be shared with other users.
- Snowsight worksheets can be shared with other developers for better collaborations.
- With the new features of Snowsight, the admin can get a view of the data governance, which is a visual representation of governance implementation.

Questions

1. How can Snowflake's Snowsight be used?
2. How can you set context with Snowsight?
3. How can you manage data objects using Snowsight?
4. How can you capture metrics and build a dashboard?
5. Can you set up monitoring of storage utilization and warehouse utilization of Snowflake objects?
6. How can you view data lineage and data tagging using the data governance feature of Snowsight?
7. What are the types of widgets that can be built for dashboards?
8. How can you manage data shares, data clones, and reader accounts from Snowsight?
9. What are the admin activities visible and accessible from Snowsight?
10. Can you view data objects and data preview in the data list?

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 15

Programming Connectors and Drivers

Introduction

Snowflake offers various connectors and drivers to work with a set of programming languages. This chapter helps to understand the various connectors and drivers. This chapter also guides you in setting up a sample driver and a Snowflake native command line interface: Snowsql.

Structure

This chapter consists of the following topics:

- Understanding programming connectors
- Understanding drivers
- Understanding Snowsql: CLI

Objectives

By the end of this chapter, you will be able to develop an understanding of programming connectors and drivers. This will also cover the installation and setup of Snowsql, which is a Snowflake **Command Line Interface (CLI)**.

Understanding programming connectors

Programming languages like Python, Java, Go, and a few others are widely used to build applications. Snowflake offers a wide range of connectors and drivers to integrate, read, write, and process data. Applications that need Snowflake interface use supporting drivers and connectors. Programming connectors like Python, Kafka, and Spark are used to set up interface with Snowflake. You will learn more about these connectors in this section.

Snowflake connector for Python

Python applications use connectors to connect to Snowflake. This provides an interface to develop an application with Python and interact with Snowflake to perform all required operations. This is a native package that does not have any dependencies on JDBC or ODBC drivers. This can be installed using Linux command using `pip`. You can also install using commands on Windows and Mac platforms. You can refer to the annexure to get a reference to the installation and setup connection.

Snowflake connector for Kafka

Kafka connector for Snowflake can be used to read data from Kafka. This can also be referred to as a Kafka connector. This is used to read data from one or more Kafka topics and load data to the Snowflake tables. Kafka works on the publish and subscribe model. This is used to get real-time data in the form of streams. Kafka publishes messages to the Kafka topic and receives messages. These Kafka topics are mapped to the Snowflake table in the Kafka configuration. This connector maps and loads data from a Kafka topic to the corresponding Snowflake table. If the target table does not exist in Snowflake, it then creates a table for each topic.

Snowflake connector for Spark

Spark connector helps to bring Snowflake into the Spark ecosystem. This enables applications to read, write, and process data to and from Snowflake. This can also be used as any other Spark source like HDFC, PostgreSQL etc. There are various versions available as per Spark versions. You can refer to the annexure to get Spark installation and configuration.

Snowflake also offers a variety of drivers that can be installed and configured to access Snowflake. You will learn more about the supported drivers and their use in the next section.

Understanding drivers

You can use languages like Go, C#, and Python to develop applications on top of the Snowflake platform. You can use the drivers discussed in this section to access data and platforms via applications developed in supported languages. Following are the set of drivers supported:

- JDBC Driver
- ODBC Driver
- Go Driver
- .NET Driver
- Node.js Driver
- PHP PDO Driver

These drivers can be used to set up connectivity with Snowflake to perform all standard operations as a part of the application. You need to follow a set of steps to set up and configure drivers to be used with your application. You will learn more about these drivers in the upcoming section.

JDBC driver and ODBC driver

Snowflake offers JDBC type 4 driver to support JDBC functionality. JDBC driver needs to be installed in a 64-bit environment and requires Java long term support versions 1.8 and higher. This driver can be used with most of the applications, tools, and clients that support JDBC to connect to the database. You can follow the steps shared as per the annexure and use the JDBC driver. You can download and set up ODBC driver as per your application environment and requirements. Snowflake **GET** and **PUT** commands support the ODBC version, and they are different for the cloud supports.

GO driver

This is the driver used with Go programming language. The applications built with Go need connectivity with Snowflake to perform operations. The driver installation installs Go database/SQL package to set up connectivity with Snowflake. You can refer to the annexure for the installation guide and configuration steps.

Snowflake offers **Command Line Interface (CLI)**. Snowsql. CLI is used to

execute SQL queries and perform all operations. You will learn more about Snowsql in the next section of this chapter.

Understanding Snowsql: CLI

Snowsql is the command line interface of Snowflake. You can install, set up, and configure Snowsql to connect to Snowflake. This interface offers an interactive way to execute all types of SQL queries, DDL, and DML operations. You can also use this interface to load and unload data. SnowSQL is also used as an interactive shell or batch mode using corresponding options.

Installing Snowsql

Snowsql is also an example of an application that is built using a Python connector for Snowflake. You do not need to set up the Python connector to use Snowsql. The Snowsql package consists of all required installers together. You can refer to the annexure reference link to download and install Snowsql. You can also follow the commands to download and install Snowsql. You can run the command: `snowsql -v` to get the version of Snowsql installed.

Connecting to Snowsql

The Snowsql command is used to connect to Snowflake. There is a set of parameters that can be used to set up and configure connection details that are required to connect to Snowflake.

Command:

```
$ snowsql <connection_parameters>
```

Following are the connection parameters that can be used with the Snowsql command:

Parameter	Description	Value
-a --accountname	Used to provide account_name.	Unique identifier of Snowflake account.
-u --username	Used to provide the username.	Your username to connect to Snowflake.
-d --dbname	Used to provide database name.	Your database name to set the context.
-s	Used to provide Schema name.	Provide your schema name to set

--schemaname		the context.
-r --rolename	Used to provide role name to be used to perform operations.	Provide your role name to be used.
-w --warehouse	Used to provide warehouse name.	Provide the warehouse name to be used.

Table 15.1: Snowsql connection parameters

Using password to connect via Snowsql

You cannot pass the password parameter and provide a password to connect to Snowflake. You can instead follow the below steps to provide the password:

- **Interactively from Snowsql:** Once you provide the connection string, it prompts you to provide the password to the username to connect to the host.
- Setting up a password in an environment variable: **SNOWSQL_PWD**
- Using key pair authentication:
 - Create a set of public and private key pair.
 - Set the private key path using the below command:

```
private_key_path = <path>/rsa_key.p8
```

You can also use the variable to set the private key:

```
export SNOWSQL_PRIVATE_KEY_PASSPHRASE=
<passphrase>
```

- **Use the following Snowsql command with the private key path to connect:**

```
$ snowsql -a account_name -u user_name --private-key-path <path>/rsa_key.p8
```

- **Define the password in the Snowsql configuration file:**
 - Create a config file: Create a config file in `~/.snowsql/` path
 - Add connection details:


```
[connections.your_connection_config]
accountname = your_account
```

```
username = john  
password = xxxxxxxx  
dbname = retail_db  
schemaname = poc  
warehousename = demo_wh
```

- You can create a configuration file and use the command to connect to Snowflake:

```
snowsql -c your_connection_config
```

You can use one of the above configuration methods to set the password to connect to Snowflake. Snowsql can be used to develop and test applications interactively.

Conclusion

This chapter provided a summary of Snowflake's programming connectors and drivers. Snowsql is Snowflake's CLI and can be used to connect to Snowflake and execute SQL queries and workloads from CLI. You can also run most of the workloads from Snowflake's web UI: Snowsight. You can use various programming connectors to integrate Snowflake with your existing pipelines or applications. These are typically used in ETL, BI, or ML workloads to train the models. Snowflake also recommends using Snowpark for any ML workloads where you do not need to move your data to the models.

In the next chapter, you will learn about Snowflake's workload patterns with real time use cases. You will learn more about various workload design patterns that you have learned in [Chapter 2, Three Layered Architecture](#). You will learn the reference architecture, use cases, features, and services used while designing workloads on Snowflake.

Annexure

This section lists the steps, details, and reference links to set up the drivers. You can refer to the following section to get details on installing drivers:

- **Install and configure JDBC driver:**

<https://docs.snowflake.com/en/developer-guide/jdbc/jdbc-download>

- Download and install ODBC driver: <https://docs.snowflake.com/en/developer-guide/odbc/odbc-download>
- Configure ODBC driver: <https://docs.snowflake.com/en/developer-guide/odbc/odbc>
- Installation steps and developer notes: <https://github.com/snowflakedb/gosnowflake>
- .NET installation steps: <https://github.com/snowflakedb/snowflake-connector-net>
- Node.js installation guide: <https://docs.snowflake.com/en/developer-guide/node-js/nodejs-driver>
- Snowflake Python connector: <https://docs.snowflake.com/en/developer-guide/python-connector/python-connector>
- Kafka connector installation and configure: <https://docs.snowflake.com/en/user-guide/kafka-connector-install>
- Spark connector installation and configure: <https://docs.snowflake.com/en/user-guide/spark-connector-install>
- Snowsql download: <https://developers.snowflake.com/snowsql/>

Points to remember

Following are the key takeaways from this chapter:

- Snowflake programming connectors and drivers extend the capabilities to develop and deploy applications.
- Snowsql, Snowflake's CLI, allows users to interact with Snowflake interactively via command line.
- You can use connectors to connect and integrate various applications such as, engineering, programming, ETL and ELT tools, BI and Reporting Tools, data analytics, and so on.

Questions

1. How is Snowsql used?
2. How can you set context with Snowsql?
3. What are the types of queries and workloads you can design with Snowsql?
4. How can you use Snowflake's drivers to connect to Snowflake?
5. Can you set up multiple connectors while developing an application?
6. How can you use Snowflake's data to build any reporting using BI tools available in the market like Tableau, Power BI, and so on?
7. How will you setup users as well as resource monitoring for any BI applications using Snowflake's connectors?

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 16

Workload Patterns with Snowflake

Introduction

Snowflake is a data platform that supports various data implantations. This chapter helps to understand data implementations like data warehouse, data lake, and data mesh. This also covers the platform requirements, understanding design aspects, and implementing them with Snowflake. This also covers a variety of real-time use cases and data architectures for reference.

Structure

This chapter consists of the following topics:

- Understanding platform needs
- Implementing data solutions with Snowflake
- Reference use cases and architecture
- Snowflake recommendations

Objectives

By the end of this chapter, you will be able to develop an understanding of workload designs and patterns with Snowflake. You will learn with real-time use cases, reference architectures, and Snowflake native features learned in earlier chapters.

Understanding platform needs

You might have heard about ETL or ELT while working on data projects. This section focuses more on understanding these two implementation patterns and the need to select one to design the platform.

Extract, Transform, and Load

As the name says, **Extract, Transform and Load (ETL)**, is the type of workload pattern where you design processes and pipelines to extract the data from sources, transform the data on the fly while reading from the source before loading it to the target and then load data to the target system.

You must have heard of various ETL tools available in the market. There are many tools that offer ETL functionalities. Some of them are Informatica, DataStage, Ab-Initio, or Pentaho. These tools have various drivers and connectors to connect to most of the heterogeneous systems as source or target.

Consider a scenario where you need to read data in the form of mainframe files, Unix files, one of the database sources as Oracle, and have a target system like Teradata. You can use the same ETL tool to connect to these heterogeneous sources, read data from these files and tables, and run transformations using ETL features like joiners, lookups, routers, filters, and aggregators to generate the target dataset. Once you have data transformed, you can map it to the target table using the target connector.

Refer to [*Figure 16.1*](#) to visualize the ETL functionalities to connect to sources, transform and load to the target systems. The processing and transformation are implemented using ETL features:

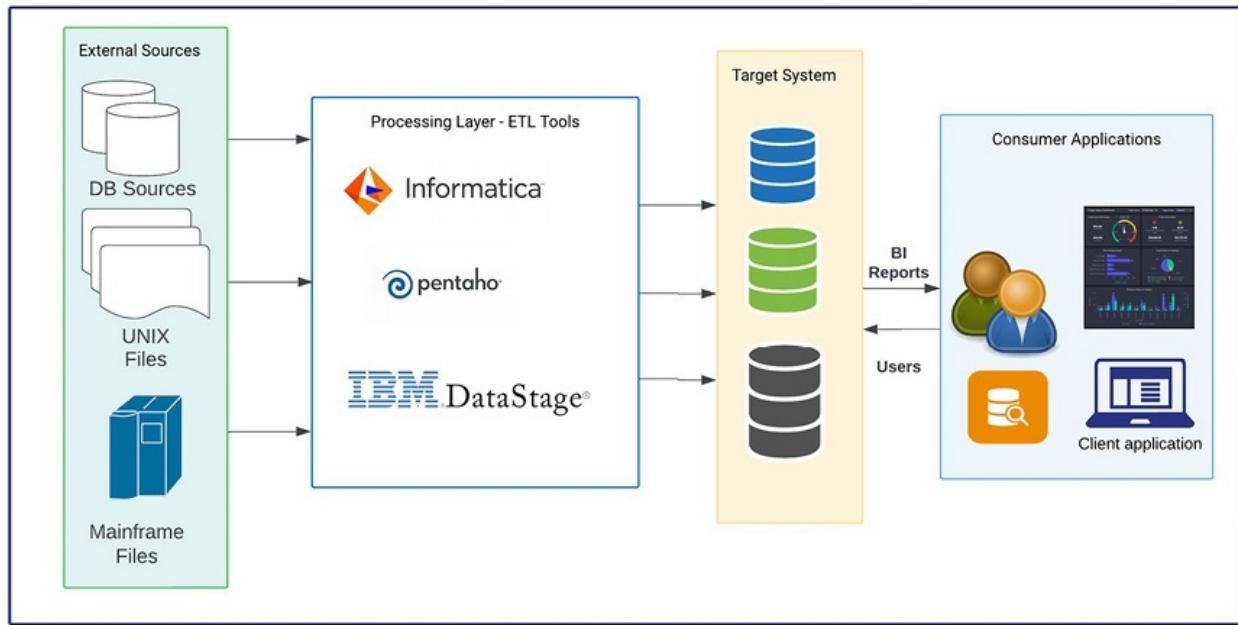


Figure 16.1: ETL reference architecture

You can continue to use ETL tools to process, transform, and load your data to Snowflake. Snowflake connectors and drivers are supported by almost all ETL tools. If you are already using ETL in your application, and your target system is migrating to Snowflake, then you can continue to use the ETL tool to point to Snowflake as the target system and load processed data. You will learn more about this in one of the real-time use cases in the upcoming section.

Extract, Load and Transform

Extract, Load and Transform (ELT), as the name specifies, is the design or implementation strategy to extract data and load it to the target system in the form of raw data or landing zone, followed by a running set of processes to transform the data, apply business rules to generate the required target datasets. In this type of implementation, the database or platform power is utilized to process the loads. This is also referred to as a push-down mechanism, where all complex logic and transformation logic is pushed to the database and leverages the power of the MPP design of the platform to achieve the performance of the processes.

You can refer to [Figure 16.2](#) to understand the overall design of the data processing flow from the source to the target system:

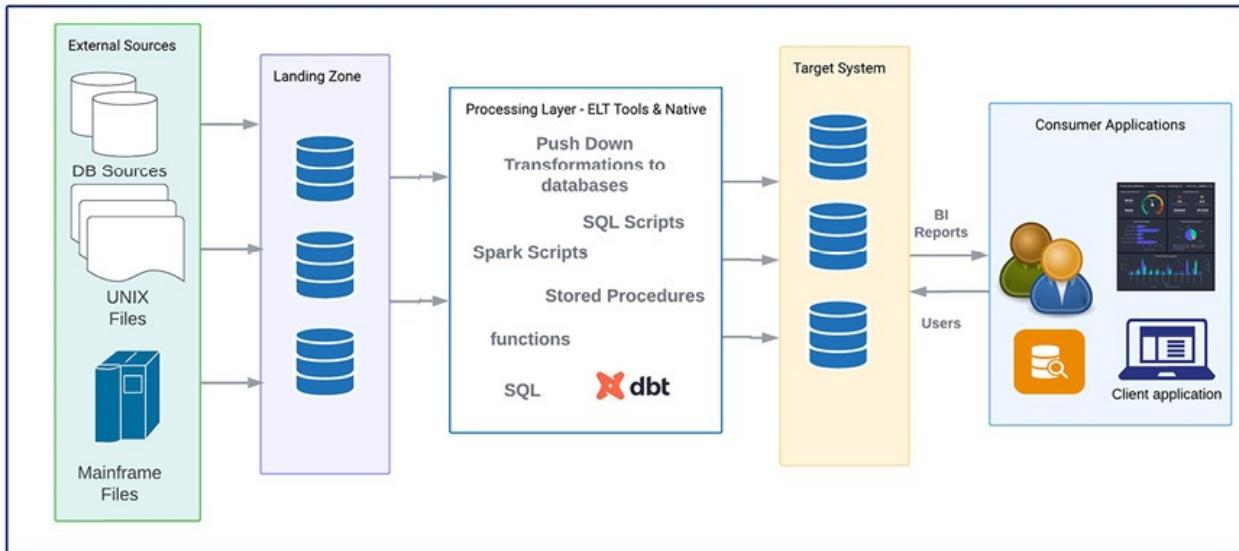


Figure 16.2: ELT Reference Architecture

You need to follow the given set of steps to implement the ELT pattern:

1. **Extract:** To extract or get data from source systems in raw or as-is format. In the case of the extract phase, you can use appropriate drivers, connectors, or tools or get files to shared locations or NAS.
2. **Load:** To load the data in raw or as-is format is a must. You get the data loaded to a set of tables in raw format without any changes or processing. This is often referred to as a RAW or Landing layer, where you get source data as-is.
3. **Transform:** In this phase, you set up your transformation jobs, SQL procedures, functions, SQL scripts or use a programming approach to leverage the power of the databases to execute the business logic.
4. **Database design:** Usually, you define 3-4 layers of data in this implementation. You get data in the RAW or Landing layer, process data and store it as intermediate to the transformed or processing layer, and store final datasets in the target layer, sometimes, you also need to set up a consumer layer for aggregated, consolidated data view depending on the needs.

There is also a set of tools available in the market that help to design, develop, and orchestrate data pipelines on your databases. They have interfaces similar to ETL tools. However, they run the processes on your databases. These are also referred to as ELT tools. DBT is one of the most commonly used ELT tools with Snowflake. You will learn more about designing and implementing ELT with

Snowflake in an upcoming section.

Implementing data solutions with Snowflake

Snowflake supports various types of workloads as you have learnt in [Chapter 1: Getting Started With Snowflake](#). This section focuses on understanding and implementing various workloads with Snowflake. You will learn these implementations with some of the real time reference use cases and architecture in the next section.

There are various types of solutions that you can implement using Snowflake. However, the choice of designing the platform is dependent on the business use case, data needs, and data accessibility within as well as outside the data platform. You can implement data warehouse, data lake, lakehouse, or data mesh using the native features available with Snowflake. However, the choice to implement one of these is based on your application and business requirements. You can design and implement these data solutions using a wide category of Snowflake's native features.

Snowflake's features

You can review the Snowflake features, and below are the most commonly used features:

- Loading and unloading capabilities
- Real-time streaming with Snowpipe streaming
- Near real-time loads with Snowpipe
- Data transformations using SQL and extended features
- Data transformations using Snowpark and data frames
- Semi-structure data processing using VARIANT
- Programming connectors and drivers for programmatic transformations
- Extended support to geospatial and other data types
- Alerting using native notification integrations
- Alerting using third party integrations: API, notification with other cloud providers

- Monitoring using USAGE views
- Dashboarding using Snowsight: Ops and engineering dashboards
- External integrations using external stages, external tables
- API integrations to integrate external sources (cloud buckets)
- Data sharing for consumer applications
- BI and reporting using BI tools like Tableau, Power BI etc.
- ETL tools integrations for ETL implementations: Informatica, DataStage etc.

These are some of the most commonly used features while designing and implementing data solutions using Snowflake.

Data solutions

You can implement all types of data solutions with Snowflake. The following is the list of some of the widely used workload patterns and data designs:

- Data Warehouse
- Data Lake
- Data Mesh

These workload patterns have a difference in their solution designs, as described:

- **Data Warehouse** is an enterprise platform where data from multiple applications are processed and stored at a centralized place. This stores the data processed historically and maintains the history of the processed data. The raw data or incoming data is always truncated and loaded as it gets processed daily and stored in the form of target tables.
- **Data Lake** is an enterprise platform and solution where data is usually stored in the raw and unprocessed format. Unlike a data warehouse, the raw data is also maintained and stored in the data lake. You can define the raw layer as the data warehouse with truncate and load before it gets processed, maintaining the copy of raw data. This also has various data marts defined when it comes to target layer design and consumer integrations. Data warehouses might not necessarily have data marts defined. This can be managed at the database or schema level.

- **Data Mesh** is a type of design where data is stored in domains, unlike the Data Lake or Data Warehouse, where the data is stored in a centralized platform. This is a comparatively new design pattern, a decentralized data store where data from various domains are processed separately and integrated with each other. You can consider this design pattern as a solution where you process your data domain-wise and store it at the domain level. However, the domain data sets are integrated with each other to refer to or consume data across domains. This is often used in implementing centralized data platforms for an organization where various domain data can be maintained in one place with appropriate standards, processes, practices, and boundaries. The data sharing and consuming data cross-domain becomes easy to implement.

There is also another pattern, a lake house, where you can combine the capabilities of data lake and warehouse. This essentially does not have a separate workload pattern as you can combine the features and implementations from the data warehouse and data lake into one. You will learn each of the design patterns mentioned above at length with reference use case and architecture in the next section.

Reference use cases and architecture

You have learnt about three design patterns that can be implemented using Snowflake and its native features. This section focuses on sharing real-time use cases, reference architecture, and understanding choices of Snowflake native features to design data platforms.

Data warehouse reference use case

You will learn with a real time use case to implement a data warehouse using Snowflake. The following sub-sections will help you to get details of the use case, proposed architecture, design key points, and end-to-end solution with Snowflake.

Use case

Consider you are working as an architect to design a data warehouse for a retail customer. Your customer is looking forward to setting up a data platform that gets batch and near real time data for sales, orders, and customers. You have a sales and marketing team reading data from your platform. They are the

consumers who read data and generate dashboards.

Technical requirements

Following will be your technical requirements:

- Total size of the platform: 20TB as of now
- Daily volume: 0.5 TB
- Type of loads: Batch and Near real time
- Type of data feeds: Pipe Delimited files
- Sources: Files sent to AWS S3 (External Stage)
- Consumers: BI dashboards and Ad-hoc User queries
- **Service Level Agreement (SLA): 8am EST**

Business requirements

The daily batch should run between 8 pm EST to 7 am EST. Data should be available before 8 am EST. Data processed is always day-1 data or previous-day data. Data processing should follow the transformation process from the raw layer to the consumer layer. The data quality and enhancements should run between the raw to the transform layer. The consumer layer should consist of the aggregated view for the consumers. The platform should have all standards implemented for data access control, data protection, and data security.

Proposed solution design

The data warehouse implementation will leverage the following Snowflake features:

- Data loading using COPY and Snowpipe
- Data processing using SQL and Stored Procedures
- Data layers using Databases and Schemas
- Data model will follow the Star schema
- BI integration using Snowflake connector
- User integration with role setup for Ad hoc queries
- Dynamic data masking for sensitive data

- Time travel for the transform layer
- Roles creation for engineering, BI, sales, and marketing teams
- Warehouse setup for engineering, BI, and Sales team for Ad-hoc
- Warehouse monitoring using resource monitors
- Email notification alerting as part of error handling
- Orchestration using AWS SQS to trigger pipeline
- Setup tasks to invoke transform jobs

Reference architecture

Refer to *Figure 16.3*, that illustrates the reference architecture to implement data warehouse using Snowflake native features and integrations:

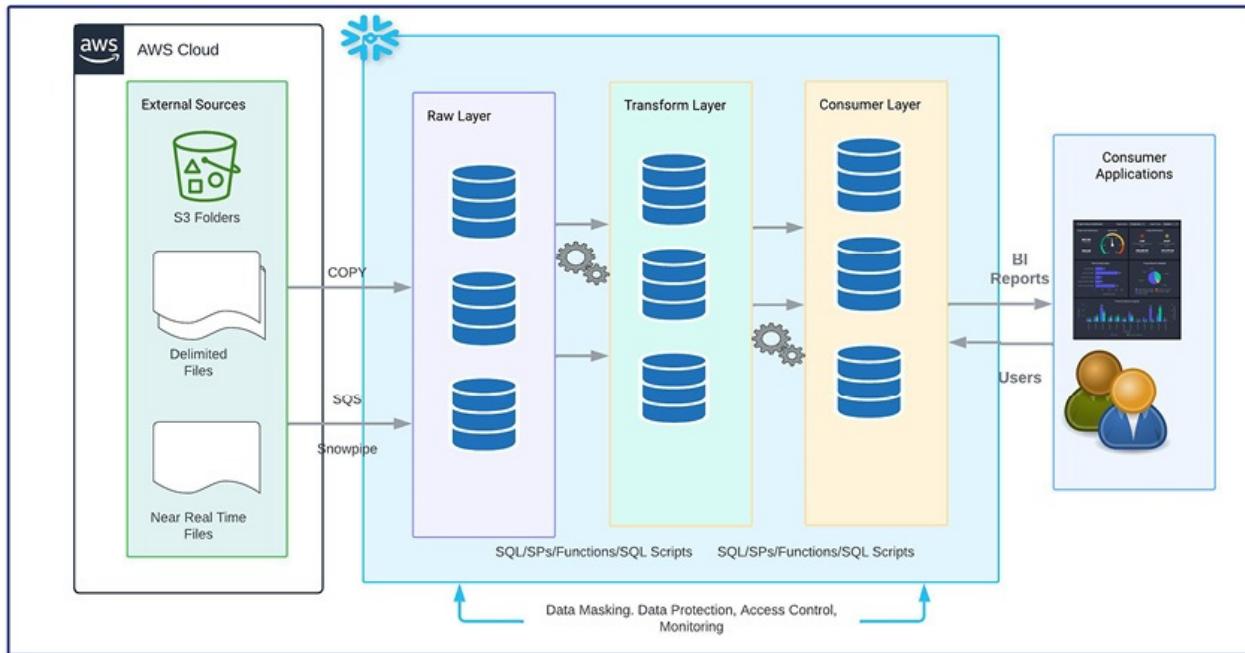


Figure 16.3: Datawarehouse Architecture

This architecture represents source integrations, consumer integrations, and Snowflake as processing layer where you can define the processing layers as databases: Raw, Transform and Consumer Layers. Data from one layer is transformed and enriched using SQL, SQL scripting and extended features.

Pipeline design

Data pipelines to load batch and near real time data are developed using **COPY** and Snowpipe. You can use **COPY** to load data from cloud storage to Snowflake table. You can integrate external cloud storage buckets as external stages. In this case, the source data is read from S3 buckets, these S3 storage buckets are integrated as stages. You can integrate and automate the Snowpipe load using AWS SQS. The file is loaded to the queue to be loaded as soon as files are arrived in storage buckets.

Platform design

Onboard all users with appropriate roles. Create roles, warehouses, and databases as per the teams. You can create small warehouses and assign them to the roles specific to users. Set up all loading pipelines as automated loads with storage triggers. Set up resource monitors at warehouses to monitor warehouse usage and take required action. Set up Snowsight dashboards to review performance metrics over widgets.

Data lake reference use case

Data Lake maintains the data in its original format before it gets processed. Data Lake designs using Snowflake follows the common strategy as Data Warehouse design however the RAW data or source data is maintained. Following sub-sections will help you to get details of use case, proposed architecture, design key points and end to end solution with Snowflake.

Use case

Consider you are designing a data lake for a telecom customer. Your customer is looking forward to setting up a data platform that gets batch data for customer, billing. You need to process the **Call Detail Records (CDRs)** to generate billing datasets. You are processing prepaid as well as postpaid records as two separate billing cycles. You also need to setup the monthly roll and archives to archive the data every month. The active data or data present in data marts for consumption is always 18 months of active data. Every month, the data gets rolled up, and a month's data gets archived to archive. There are sales and billing teams that read data from your platform. They are the consumers who read data and generate dashboards.

Technical requirements

Following are the technical requirements:

- Total size of the platform: 30TB as of now
- Daily volume: 1 TB
- Type of loads: Batch loads
- Type of data feeds: Delimited files
- Sources: Files sent to AWS S3 and Google Cloud Storage (External Stage)
- Consumers: Sales and Billing Team via BI dashboards, Downstream dataset and Ad-hoc User queries
- **Service Level Agreement (SLA):** 9am EST

Business requirements

The daily batch should run between 7pm EST to 8am EST. Data should be available before 9am EST. Data processed is always day-1 data or previous day data. Data processing should follow loading it to raw layer, processing and transforming CDRs to get call details, enriching the data with customer datasets and load the transformed layer to get billing data. The data is processed from the raw to the consumer layer. The data quality checks should run on transformed data. These data quality checks are for quality of data for billing, customer data records. The failed records are stored on another table with a quality indicator. The consumer layer is made up of data marts, also known as databases, that store aggregated billing datasets and customer datasets. Platforms should have all standards implemented for data access control, data protection and data security.

Proposed solution design

The data lake implementation will leverage the following Snowflake features:

- Data loading using **COPY** bulk utility
- Data processing using SQL scripting and Stored Procedures
- Monthly roll process using SQL scripting to check on the active 18 months and move oldest month data to archive database
- Data layers using databases: Raw, Transformed, Consumer
- Data marts at consumer layer maintained using databases and schema
- BI dashboards and reporting using Snowflake connector
- User integration with role setup for Ad hoc queries

- Dynamic data masking for sensitive data for customer and billing columns
- Data protection using Time travel for transform layer to recover records
- Roles creation for engineering, BI, sales and billing teams
- Virtual warehouse setup for engineering, BI and Sales team for Ad-hoc
- Warehouse monitoring using resource monitors, dashboards
- Email notification alerting as part error handling and auditing
- Setup audit table to maintain pipeline status in daily batches
- Orchestration using Snowflake tasks

Reference architecture

Refer to [Figure 16.4](#), which illustrates the reference architecture to implement data lake using Snowflake native features and integrations. The architecture design illustrates the various data layers: Raw, Transformed, and Consumer—source data integrations with AWS S3 and Google Cloud Storage as external stages. Data is read in raw format from these layers. The data can also be read as external tables on these stages. This also represents data generation for downstream applications and saving them to Google Cloud Storage – External stage:

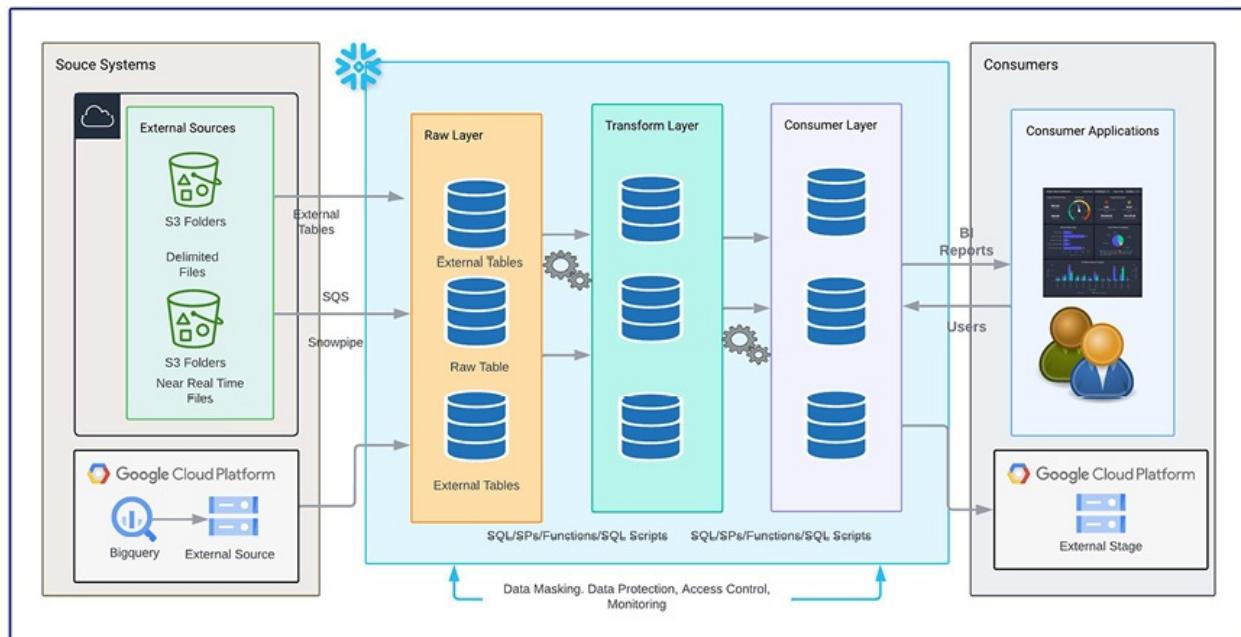


Figure 16.4: Data Lake Architecture

Pipeline design

Data pipelines to load batch data are developed using COPY bulk load. You can use COPY to load data from cloud storage to Snowflake table. You can integrate external cloud storage buckets as external stages. In this case, the source data is read from S3 buckets as well as Google Cloud Storage. These external storage buckets are integrated as external stages. You can also build external tables on top of external stage data. Data transformation, data enrichment, and data quality checks are implemented using SQL scripts, SQL native features – SQL jobs or Stored procedures. Design data quality table, audit table, and error table to maintain audit logs, and metadata of batch loads. Data pipelines are used to generate downstream data files and save them to Google Cloud Storage – an external stage for consumer applications. Notification alerting using notification integration to send error notifications.

Platform design

Onboard all users with appropriate roles. Create roles, warehouses, and databases as per the teams. Set up the consumer layer with databases and schemas for billing data. You can create clustered warehouses and assign them to the roles specific to users. Set up resource monitors at warehouses to monitor warehouse usage and take required action. Set up Snowsight dashboards to review performance metrics over widgets. External stages and notifications have to be set up as integrations.

Data mesh reference use case

Data mesh is a type of design pattern where you maintain your data at the domain level and the data is shared between these data domains. Each domain sources, processes, and maintains their relevant data within their domains. The data is shared between these domains either using data sharing or datasets. User management, access controls, and policies are maintained and implemented at each domain. The following sub-sections will help you to get details of the use case, proposed architecture, design key points, and end-to-end solution with Snowflake.

Use case

Consider you are designing a data mesh for a large and complex customer. Your customer is looking forward to setting up a data platform that gets data from heterogeneous sources. The customer is looking forward to setting up a data

platform that has cross-region or domain boundaries defined. You need to process the data as received and create data shares. The data shared across is in the form of accesses within an account. Define data boundaries, process and define policies to share data across.

Technical requirements

Following are the technical requirements:

- Total size of the platform: 20TB as of now
- Daily volume: 1 TB
- Type of loads: Batch loads, Real time loads, Data from other DB sources
- Type of data feeds: Delimited files
- Consumers: Internal teams

Business requirements

Data processing should follow multi-layer processing, the same as the data warehouse use case. Loading it to the raw layer, processing and transforming the data, and loading the transformed layer to the target layer. The platform should have all standards implemented for data access control, data protection, and data security. Policies and standards are to be identified as data access borders and set cross-regional access.

Proposed solution design

The data mesh implementation will leverage the following Snowflake features:

- Data loading using **COPY** bulk utility
- Programming interfaces, connectors to read data from databases as source.
- Data processing using SQL scripting and stored procedures
- Data layers using databases: Raw, Transformed, Consumer
- Consumer layer maintained using databases and schema
- User integration with role setup as per users per domain
- Dynamic data masking for sensitive data for customer and billing columns
- Data protection using time travel for transform layer to recover records

- Roles creation for development, consumers teams
- Virtual warehouse setup for teams
- Warehouse monitoring using resource monitors, dashboards
- Email notification alerting as part of error handling and auditing
- Setup audit table to maintain pipeline status in daily batches
- Data sharing with accesses to required consumer users, roles

Reference architecture

Refer to [Figure 16.5](#), which illustrates reference architecture to implement data mesh using Snowflake native features and integrations:

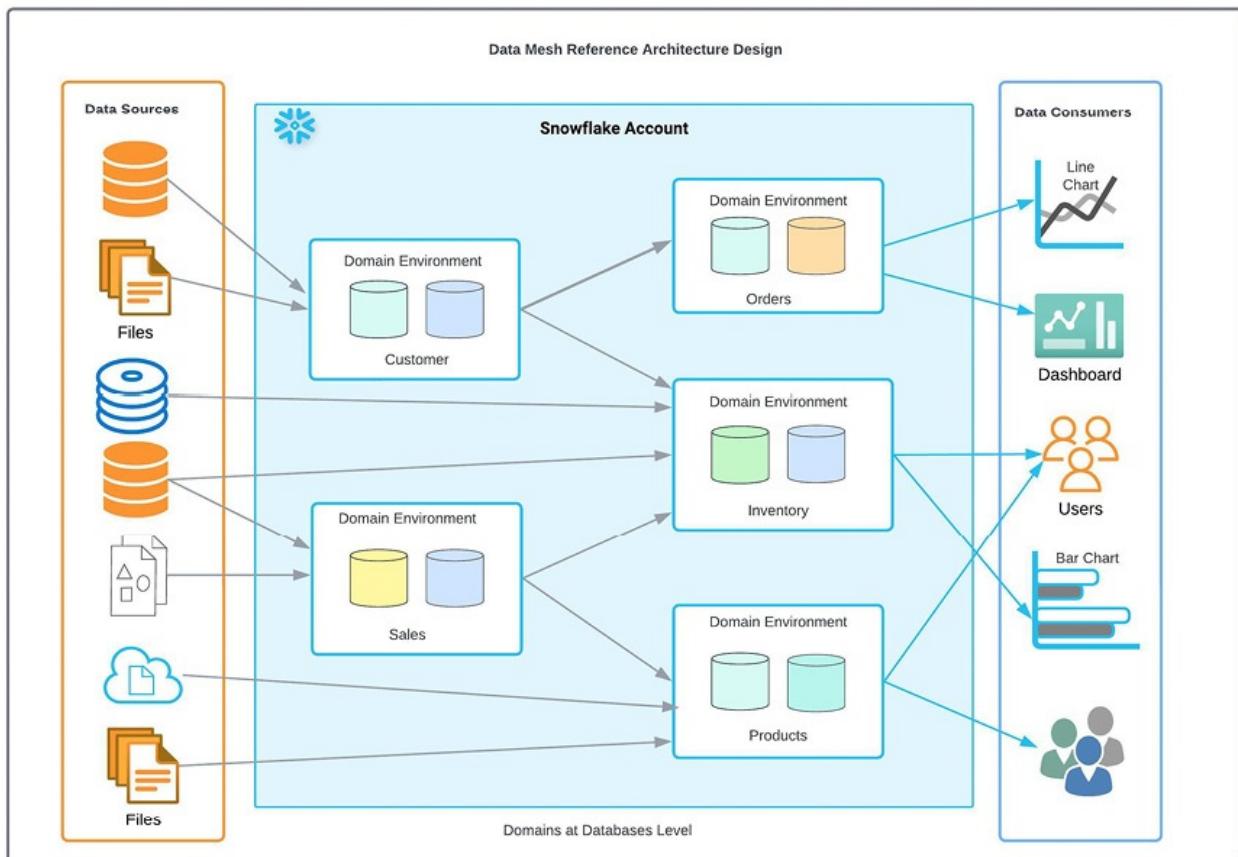


Figure 16.5: Data Mesh Reference Architecture

Platform design

Data mesh is a design pattern with domain or region set up to have data

maintained separately. You need to design the domains and these domains can be defined in two board ways, one at the account level, with multiple accounts for each region or domain. Secondly, you can have one account with multiple databases as domain boundaries. You can onboard all users with appropriate roles. Create roles, warehouses, and databases as per the teams. Set up consumer layer with databases and schemas for billing data. You can also create clustered warehouses and assign them to the roles specific to users. Set up resource monitors at warehouses to monitor warehouse usage and take required action. You can also design and set up Snowsight dashboards to review performance metrics over widgets. Design various integrations in the form of external stages and notifications to be set up as integrations.

You have learned use cases that define three major design patterns: data warehouse, data lake, and data mesh. You can use these references and architectures to implement data platforms.

Snowflake recommendations

Snowflake has a variety of features that architects use to design a data platform. Snowflake's **Massively Parallel Processing (MPP)** and three-layered architecture helps users to design performance efficient and scalable data platform. This section helps you understand some of the recommendations.

Recommendations

There are various Snowflake recommendations based on the Snowflake features. This section lists recommendations of some of the widely used features.

Warehouse recommendations

Warehouses are also referred to as compute power. Below are some of the recommendations for choosing warehouse size, managing warehouses, and usage of warehouses:

- Run different types of queries with different sizes of warehouses to derive the warehouse size for the workload.
- Use auto suspend and auto resume for clustered warehouses. This helps to manage the warehouse credit consumption.
- Warehouse cache is dropped as and when the warehouse is suspended. Compare query processing with cache, maintaining cache vs warehouse

suspension. Set up the processing.

- Set up resource monitors to monitor the usage of warehouses. Set up thresholds to monitor warehouses on frequency.

Storage recommendations

Snowflake storage cost is associated with various factors like data stored in physical tables, storage needed to maintain time-travel, failsafe, and temporary storage needed for transient or temporary tables. Following are some of the recommendations:

- Load only required data into physical tables.
- Maintain time-travel only for the tables required as per application and business requirements.
- Track the usage of storage from **TABLE_STORAGE_METRICS**.

Load recommendations

Loading is one of the integral processes of data platform. Snowflake offers native features to load data like **COPY**, Snowpipe and programming connectors, drives. Following are some recommendations for the same:

- Use **COPY** to load data from cloud storages to the table. This is used to load data in bulk.
- Use Snowpipe to load near real time and real time data to Snowflake tables.
- It is recommended not to use JDBC or ODBC to load large or massive data. You can use it only for smaller datasets, up to megabytes of data.

Transformation recommendations

Data transformation is one of the critical parts of data implementation. Snowflake offers various features and extended features to develop data transformations. Following are some of the recommendations:

- Avoid row-level processing.
- Develop simple, step-by-step processing to transform data in place of large complex overkilling queries.
- Use temporary tables to store intermediate results of large data processing.

- Use the query profile to view the performance of the query being executed.
- Use **QUERY_HISTORY** to review the performance of queries, query status, queued queries, long running queries.

Usage recommendations

Snowflake offers various **USAGE** views to track **USAGE** of queries, warehouses, storage metrics. Following are some of the common recommendations:

- Track warehouse usage to identify over utilized and underutilized warehouses.
- Use **USAGE** views to identify long running queries to optimize query performance as well as identifying queued queries.
- Set up **QUERY_TAGS** to track the performance of queries and capture metrics.
- Set up appropriate metrics and functions to track, and monitor using Snowflake tasks.
- Use resource monitors to track usage of the Snowflake account.

You can follow these recommendations while designing and implementing data solutions with the Snowflake platform, too. There are many other recommendations to use other features of Snowflake. However, these were some of the most commonly used features.

Conclusion

This chapter provided a summary of Snowflake's real time use cases, data solutions, and workload patterns. You can leverage the Snowflake native features to design data warehouse, data lake, data mesh data platforms. You have also learnt features of each workload pattern, reference architecture designs, solutions to implement them. This chapter also included Snowflake recommendations for platform design, management and monitoring.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake is data on cloud that allows users to design various workloads

on the same platform.

- Snowflake supports OLTP and OLAP workloads using native features.
- You can also implement ETL and ELT design patterns with Snowflake.
- Snowflake supports data loading and unloading with batch and real time use cases.
- Snowflake connectors and drivers allow users to extend integrations with various applications.
- Snowflake supports real time as well as batch loads using serverless and native features.
- Data warehouse, data lake, and data mesh are some of the widely used workload design patterns.

Questions

1. How can ELT be implemented using Snowflake native features?
2. How can you design a data lake with Snowflake?
3. What are the data integrations available and how can you connect to various sources and consumer applications?
4. How can you use Snowflake's integrations to get data from external sources or storage buckets?
5. Can you design an application with OLTP and OLAP needs at the same time?
6. How can you integrate consumer applications like BI, reporting, and dashboarding with Snowflake?
7. How can you use integrators like dbt and matillion?

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 17

Introduction to Snowflake's Advance Features

Introduction

Snowflake releases new features as well as extensions to the existing features with every release. This chapter helps to understand the new features released recently and extensions of the existing features of Snowflake. This also covers a variety of real-time use cases for reference.

Structure

This chapter consists of the following topics:

- Understanding new features and releases
- Understanding new types of tables
- Introduction to new Snowsight features
- Introduction to LLM capabilities

Objectives

By the end of this chapter, you will be able to develop an understanding of the new features of Snowflake. You will also learn the usage of these features with real use cases.

Understanding new features and releases

Snowflake continuously works on new features, improvements to existing features, enhancements, and fixes. Snowflake deploys these features in the background without any impact on the user experience and behavior changes. Snowflake releases features in **Private preview (Pr Pr)**, **Public preview (Pu Pr)**, and **General availability (GA)**. These features are released weekly and monthly. The behavior changes are released monthly, and feature releases are released weekly. You can find the release details here:

<https://docs.snowflake.com/en/release-notes/new-features>.

Snowflake releases

Snowflake releases two features every week. This feature release can be a full release or a patch. Patch release consists of fixes to the existing features. Following are the features included in a full release:

- New releases
- Feature enhancements
- Fixes
- Behavioral changes

New features

You will learn about Snowflake's new features in the subsequent sections. Below are some of the features released recently:

- **Native apps:** You can develop native apps and deploy them to be used with Snowflake. You can see this as an additional feature in the Snowsight menu. You will learn more about this in the upcoming section of this chapter.
- **Tables:** Snowflake offers additional tables along with the native tables – permanent, temporary, and transient tables. You can create dynamic, hybrid, event, and iceberg tables as an extension of the table feature of Snowflake. These tables were recently released, and you will learn more about them in the next section of this chapter.
- **Snowsight features:** New features are being added to Snowsight continuously. You will learn specific features in the next section of this

chapter.

SQL extensions

Snowflake's new fixes release support to the SQLs, functions, database objects, schemas, and extended features to implement SQL capabilities. You will learn some of the recent features released in this section. These features are categorized based on their usage in the data implementations. Refer to the categories and features in the following sections.

Schema features

Snowflake supports various schema features to derive schema and allows users to enable schema evolution to extend support to the incoming data. Below are the set of features used as part of schema features:

- The **INFER_SCHEMA** function is generally available. This can be used to automatically detect the schema of staged files and get column definitions. You can use this function to get the schema of files in Parquet, ORC, Avro, JSON, and CSV format files.
- **ENABLE_SCHEMA_EVOLUTION** is a parameter used in the table. You can set this while creating a table or use the **ALTER** command to set the parameter to track the changes to the table and types of columns. Schema evolution accommodates the data layout changes with incoming data. For example, in case of any new column or field being added to the incoming file, the table definition adds an additional column to preserve the incoming field in the file. You can use the **SET** command to enable this feature.

SQL features

You can refer to the latest release notes to get more information on the SQL updates. This section covers the updates as of the latest release in November 2023, which are as follows:

- **ARRAY_FLATTEN**: This semi-structure function is used to flatten the **ARRAY** into a single **ARRAY**.
- **CURRENT_ACCOUNT_NAME**: This is used to get the value of the current account. You can use **SELECT CURRENT_ACCOUNT_NAME** to get the account name as an account identifier.

- **TRY_DECRYPT** and **TRY_DECRYPT_RAW**: This is a version of the function that returns **NULL** whenever an error occurs in decryption
- **SEARCH_OPTIMIZATION**: It is generally available and can be used with substring and regular expressions for semi-structured data types – **VARIANT**, **ARRAY**, and **OBJECT**. Initially, the equality search on these semi-structured data types were supported for search optimization. Now, this can be used for substring queries using the following types of predicates:
 - **LIKE**: **LIKE**, **ILIKE**, **LIKE ALL**, **LIKE ANY**, **ILIKE ANY**
 - **Pattern matching**: **CONTAINS**, **ENDSWITH**, **STARTSWITH**, **SPLITPART**
 - **Regular Expressions (REGEX)**: **REGEXP**, **REXEP_LIKE**
- **Search optimization**: It can be added to a table using the **ALTER** command on specific fields, columns, and string functions:

```
ALTER TABLE customer_details ADD SEARCH
OPTIMIZATION ON
SUBSTRING(variant_col:data.cust_address);
```

- **UDF and stored procedure**: You can create a function and stored procedure using the **CREATE** statement and pass arguments as inputs to these SQL objects. With the new release, you can also make an argument as an optional argument using the **DEFAULT** keyword. If you define the argument as optional, then you may not need to pass it while invoking the function or procedure. You can do so by declaring the parameter or argument using the **DEFAULT** value as shown below:

```
CREATE FUNCTION convert_udf(
    args_1 VARCHAR,
    args_2 VARCHAR DEFAULT 'default'
    args_3 INTEGER DEFAULT 0)
```

You can also use the function to pass the value of arguments 2 and 3 marked as a default value. You can pass value as – **SELECT convert_udf(args_1 => 'value', args_2 => 'value')**;

- **ALLOWED_RECIPIENTS**: For this, notification integration is no longer required. Initially, you needed to have recipients validated as allowed

recipients. Now, with the new release, you can have this set to be used with the **SYSTEM\$SEND_EMAIL** procedure to send an email notification. You can create new notification integrations using the **TYPE=EMAIL** command to send emails to users with verified email addresses. For existing notifications, you can update this using the **ALTER** command: **ALTER NOTIFICATION INTEGRATION .. UNSET ALLOWED_RECIPIENTS**.

- **IF [NOT] EXISTS:** This is now supported in the **ALTER** command to **ADD** any column from a table. Also, **IF EXISTS** is supported in the **ALTER** command to drop any column from a table. You can use them as shown below:

```
ALTER TABLE customer_details ADD COLUMN IF NOT EXISTS alternate_address STRING;
```

```
ALTER TABLE customer_details ADD COLUMN IF NOT EXISTS alternate_address STRING;
```

These are some of the recent updates and new features and enhancements that will be released every week. You can refer to the Snowflake release page for the most recent releases and features.

Understanding new types of tables

Snowflake supports native tables and their types – permanent tables, transient tables, and temporary tables. You can also create materialized views to store pre-calculated data ready to be consumed, and the materialized view service will take care of table updates/data sync. Along with these tables, Snowflake extends the tables features to add to the below set of tables for your application deployments and use cases:

- Dynamic tables
- Event tables
- Hybrid tables
- Iceberg tables

This section focuses on the additional tables supported and relates them with the real-time use cases to understand their implementations.

Dynamic tables

This is a feature in preview as of now on all cloud and available to all accounts. Consider a data engineering use case where data needs to undergo a set of transformation steps to transform the data and make it ready to be consumed. In case of data engineering, you need to define the pipeline, as in, engineer the transformation steps to take care of bulkier transformations and conversions. Snowflake brings in dynamic tables to take care of pipeline engineering and avoid any additional transformational steps. You can define the end state of the table and leave the logic or transformation to the Snowflake.

Dynamic tables are the database objects that materialize the result of the query. You can define the table and specify the query that will be used to transform the data. Snowflake takes care of the conversion or transformation as specified in the query and refreshes the data automatically. As these tables are a result of the query specified, this has structure as per the query result and you cannot change the content of the table using any DML command. You cannot run insert, update, or delete to update the content of the table.

You can use dynamic tables to reduce down the transformation steps and simplify the process. you can use these tables to transform the data and use them for the below use cases or type of operations:

- Use when you do not want to track the data changes and maintain the data dependencies.
- Use it when you want to avoid data transformation complexities using streams and tasks.
- Use it to materialize the result of a query on multiple tables.
- Use it when you need target data and do not want to control the refresh schedule.
- Use it to build multiple tables to transform the data.

Dynamic tables follow an automated process to refresh the data. This follows a process that computes changes of the base tables and merge these changes to dynamic table. This process uses compute resources associated with a dynamic table. You can also specify the target data freshness. This is also referred to as lag to get the refreshed data to the target table. The automated process refreshes the data based on the lag time specified. You can also set a longer duration if you do not need fresh data often.

Dynamic table costing consists of three parts – storage cost, cloud services cost,

and warehouse cost. As dynamic tables run an automated process to refresh data, this needs a dedicated warehouse to run the query and refresh data. The process uses cloud services to identify the changes on base tables since the last refresh. If there are no changes to the base tables, there is no compute required or no compute cost involved until data is refreshed on base tables.

Event tables

As the name suggests, the event table stores the events, event logs and trace events logged from Snowflake objects. Event tables are used to capture the messages and event data generated by event handlers. You can define the event handler using handler code in functions and procedures. Event tables have a set of predefined columns that are used to capture event details. Following are some of the key characteristics of the event tables:

- Table has a predefined format to store log entries and trace events.
- This table can be associated with an account. Only one account can be associated with an event table at a time. This is referred to as an active event table.
- You can also specify the severity of the events to be captured.
- Log messages and event messages are generated by the handler code. This handler code can be embedded with database objects like **stored procedures (SPs)**, UDFs, and UDFTs into an active event table.

You can create an event table using the **CREATE** statement. As you know, event tables have predefined columns and you do not have to define the schema or specify columns while creating the table. You can use the **CREATE** command as shown below:

```
CREATE EVENT TABLE RETAIL_POC_DB.POC.app_events;
```

Now, you can associate the created event table with an account to set it as an active event table. All events and logs are captured in the active event table automatically. You can set the event table as an active event table using the **ALTER** command as shown below:

```
ALTER ACCOUNT SET EVENT_TABLE =
RETAIL_POC_DB.POC.app_events;
```

You can run specific operations on top of the event table. Though this is a table,

you cannot run all DML operations on top of this table as this captures event logs and messages. You can run the following operations on the event table:

- **DROP/UNDROP** table
- **TRUNCATE** table
- **DELETE** table
- **SHOW** event tables
- **DESCRIBE** event table

You can use **DELETE** to delete rows from **event** table and use **TRUNCATE** command to delete all rows from the **event** table. You can also create a stream on top of the **event** table. You can set the **session** parameters to capture the log and trace details using the **SET** command. You can set them using the **ALTER** command as shown below:

```
/* To capture log details, use below command */
ALTER SESSION SET LOG_LEVEL = INFO;

/* To capture trace event details, use below command
*/
ALTER SESSION SET TRACE_LEVEL = ON_EVENT;
```

Refer to the following sample UDF to capture the messages in UDF, SP or UDFTs as shown below:

```
CREATE OR REPLACE FUNCTION app_log_trace_data()
RETURNS VARCHAR
LANGUAGE PYTHON
RUNTIME_VERSION = 3.8
HANDLER = 'run'
AS $$

import logging
logger = logging.getLogger("logger")
```

```
def run():
    logger.info("Logging from Python function.")
    return "SUCCESS"
$$;
```

This logger captures log events and stores in the active **event** table. You can run the **SELECT** query on top of event tables to get log details. Refer to the following query to get log data from the **event** table:

```
SELECT
*
FROM
RETAIL_POC_DB.POC.app_events
WHERE RECORD_TYPE = 'LOG'
AND SCOPE['name'] = 'logger';
```

You can use event tables to capture the log and event data. This feature enables users to automate the logging and tracing using code handlers.

Hybrid tables

Snowflake offers an extended feature to tables that support analytical and transactional data. This is part of the Snowflake's unistore offering where users can use analytical and transactional data together. Hybrid tables are in preview as of now. This table is designed on a principal to support most common transactional capabilities of an application and to support this feature Snowflake has developed a new row-based storage engine.

You can create hybrid tables using the **CREATE** statement and need to specify the **PRIMARY KEY** for uniqueness, as it needs to support the transactional operations as well. Refer to the below sample table DDL:

```
/* Create orders table as retail use case */
CREATE HYBRID TABLE Order_details (
```

```
    Order_id number(38,0) PRIMARY KEY,  
    Customer_id number(38,0),  
    Order_status varchar(25),  
    Total_amount number(38,0),  
    Order_date timestamp_ntz  
);
```

As this supports transactional operations, you can use a set of hybrid tables required for your application and define entity relationship using **Primary Key (PK)** and **Foreign Key (FK)**. You can also define the constraints on the keys while defining the tables using the **CREATE** statement. You can refer to the retail use case and create a set of tables as shown below:

```
/* Create customer table as retail use case */  
  
CREATE HYBRID TABLE customer_details (  
    Customer_id number(38,0) PRIMARY KEY,  
    Cust_name varchar(50),  
    Cust_Address Varchar(50),  
    order_id number(38,0),  
    Order_status varchar(25),  
    Total_amount number(38,0),  
    Order_date timestamp_ntz  
);
```

```
/* Create orders table as retail use case and  
customer_id foreign key reference */  
  
CREATE HYBRID TABLE Order_details (  
    Order_id number(38,0) PRIMARY KEY,
```

```

Customer_id number(38,0),
Order_status varchar(25),
Total_amount number(38,0),
Order_date timestamp_ntz,
CONSTRAINT fk_o_customerkey FOREIGN KEY (Customer_id)
REFERENCES Customer_details(Customer_id)
);

```

You can also run analytics on transactional data. You can run powerful insights on hybrid tables using Snowflake's analytical capabilities. With hybrid tables, you can also break down data silos between transactional and historic data. You can also join hybrid tables with any other tables in Snowflake.

Iceberg tables

These are the brand-new table types in Snowflake and are available for preview as of now. These are designed to resemble Snowflake native tables with the distinguishing features of iceberg. Apache iceberg is an open table format for huge analytic tables. Apache Iceberg offers rich metadata on top of data files that can be used to perform analytical operations, historical search, change management, and tracking using metadata. This is designed to work on Apache Parquet data files. You can use it whenever you have huge data files in Parquet format stored on cloud storage and need them to be accessed for analytical operations. Based on the Apache iceberg open table format features and characteristics, Snowflake has considered extending support to create integration as a table.

Iceberg tables are Snowflake tables that use open format with customer provided storages. These tables work similarly to Snowflake's native tables, with three key differences:

- The table metadata is in iceberg format.
- As iceberg supports Apache Parquet data format, data has to be stored in Parquet file formats.
- The table metadata and data are stored in customer-provided storage.

Earlier, Snowflake also announced the availability of iceberg external tables to

support the Apache iceberg format and data stored on cloud storage. This feature is further extended to get this integrated as a native table format due to similarities between the metadata management between Snowflake and Apache iceberg.

You can create an iceberg table using the **CREATE** command. As you know, the data and metadata are stored on cloud storage. You need to create an external volume first to create a table, as follows:

```
-- Create an External Volume to hold Parquet and
Iceberg data

create or replace external volume app_ext_vol
STORAGE_LOCATIONS =
(
    (
        NAME = 'app-s3-us-west-1'
        STORAGE_PROVIDER = 'S3'
        STORAGE_BASE_URL = 's3://app-s3-
bucket/data/snowflake_vol/'
        STORAGE_AWS_ROLE_ARN = '*****'
    )
);
```

Once volume is created, you can define the table using the **CREATE** command and **ICEBERG** identifier to define the table:

```
-- Create an Iceberg Table using iceberg External
Volume

create or replace iceberg table app_iceberg_table
with EXTERNAL_VOLUME = 'app_ext_vol'
as select order_id,
order_date,order_status,Total_amount from orders;
```

You can use the **iceberg** table as native table and run all types of DML operations to access data. Snowflake has also announced managed and unmanaged iceberg tables in preview. With these types of tables, you can read data from cloud storage as well as write data to the cloud storage. The managed **iceberg** table is when you can perform write operations on the data, and the unmanaged iceberg table is when you can perform read-only operations on the data. Similar to any other native tables, you can also define the dynamic masking and row-level security implementation for iceberg tables.

Introduction to new Snowsight features

Snowsight is Snowflake's web interface. All new users are given Snowsight as the default web UI, and you can access it from <https://app.snowflake.com>. You can log on to Snowsight and start performing your operations. The Snowflake team is working towards making this a one-stop solution to all the features being offered as a data platform. You can refer to the following features added, as well as enhanced support to Snowsight.

Worksheets

You can create worksheets to run your code to query and process data stored within Snowflake or with external tables. You can access these worksheets from Snowsight's UI menu and create new worksheets. With recent releases and features, you can create a SQL or Python worksheet. You can build your Python code to Snowpark for data processing or AI/ML use cases using Python worksheets. You can also create a folder to manage the worksheets. Worksheet management and sharing are easy with folders and sharing features. Worksheets can also collaborate with the team while developing any application.

Data governance

Data governance is a new interface added to the Data tab on Snowsight. You can use this to review the governance implemented. This is a consolidated view of the number of objects with the most commonly used policies and tags. There are two panels to take a deep dive into the governance feature. You can also use the set of filters on top of the tagged object panel to view objects with tags and policies. Refer to *Figure 17.1* which represents a consolidated view of tagged objects, masking policies, tables, and columns secured in dashboard view. You can also get a list of tagged objects in the tagged object tab.

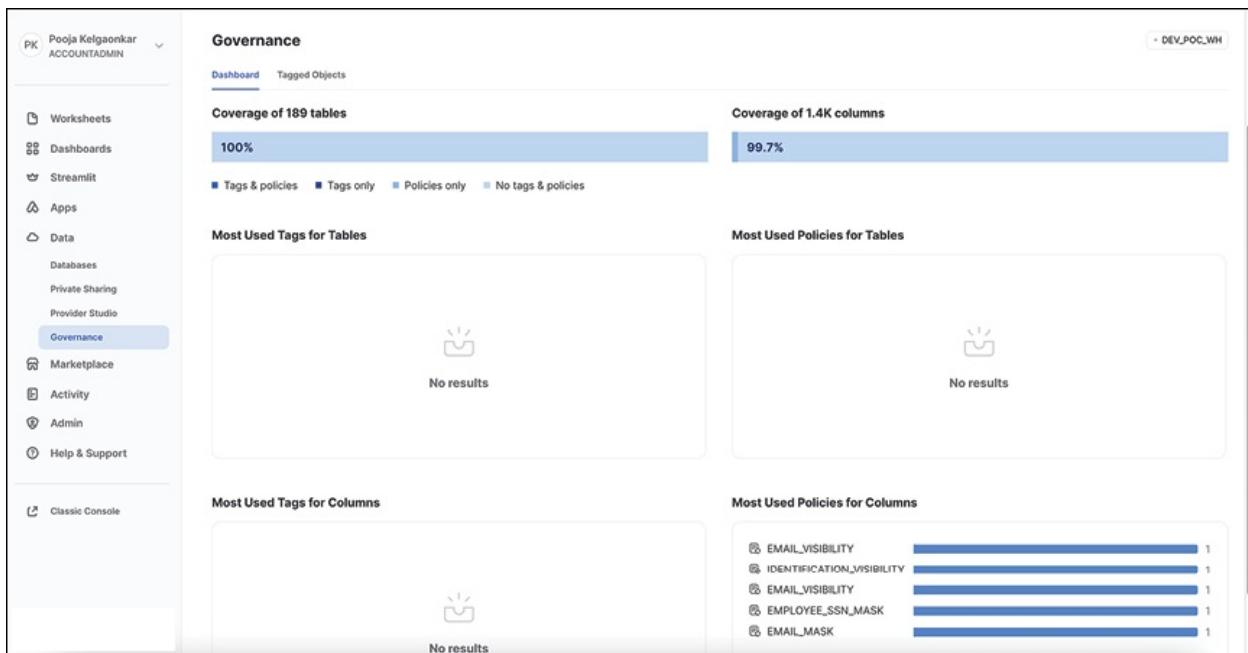


Figure 17.1: Snowsight Data Governance

Native apps

Snowflake announced integration with the native app framework to build data applications leveraging Snowflake's data functionalities and features. You can find a new interface added to Snowsight as Apps and Streamlit. Snowflake has acquired Streamlit and added Snowsight to help developers to build the application and submit it to apps listing. Snowsight's app interface list down the apps that are installed as well as the apps as package to the Snowflake listings. The native app framework is currently available in preview and available for accounts on AWS. You can upload all required files – streamlit, python files, mandatory files like **setup.sql**, **manifest.yml** and **readme.md**. You need to create a schema and stage to upload necessary files. You can upload them using the Snowsight stage feature. You can also use the PUT command to upload these files using **snowsql** CLI. You can create an application package using the SQL command. The application deployed is listed in the APP navigation menu of Snowsight. Refer to the following figure that lists down the existing or used apps in the account:

The screenshot shows the Snowsight interface with the 'Apps' tab selected. On the left, a sidebar lists options like Worksheets, Dashboards, Streamlit, Apps (which is selected), Data, Marketplace, Activity, Admin, and Help & Support. The main area is titled 'Installed Apps' and displays two entries:

TITLE	OWNER	VERSION	INSTALLED
CHAIRLIFT_APP @CHAIRLIFT_PKG	CHAIRLIFT_ADMIN	version 1	...
HELLO_SNOWFLAKE_APP @HELLO_SNOWFLAKE_PACKAGE	ACCOUNTADMIN	UNVERSIONED	...

Figure 17.2: Snowsight App page

You can refer to <https://docs.snowflake.com/en/developer-guide/native-apps/tutorials/getting-started-tutorial> to get started with native app development. You can also refer to the below quickstarts and start building native apps:

- Build native apps to analyze chairlift sensor Data: <https://quickstarts.snowflake.com/guide/native-app-chairlift/index.html?index=..%2F..index#0>
- Data Mapping in Snowflake native app using Streamlit: https://quickstarts.snowflake.com/guide/data_mapping_in_native_app_index=..%2F..index#0

Budgets

Snowflake has recently announced budgets available for preview. This is available for all Snowflake accounts except the trial accounts or accounts in government regions. Budgets are nothing but the account level monitoring and notification of Snowflake's credit usage for specific objects as a group. You can define the budget as a monthly limit for the compute cost. You can view the budgets in the preview on Snowsight. Go to the **Admin | Usage** interface. You will find two panels, namely, **Consumption** and **Budgets**. You can also set up the account budget using the top left button. Once you set up the account budget, you can also set the custom budgets using the + button on the top left of the budget interface. You can specify the name of the custom budget, define your budget, and activate notifications to send mail alerts. You can also attach an existing resource monitor to track defined budgets. You can also use Snowsight to monitor the budgets defined.

Data loading

With new releases, you can use Snowsight to load the data to the tables using console features. You can load up to 50MB of data to the table using Snowsight. You can use the load data option from the data interface and table menu panel. You can load data from the local table and specify the stage to load files from external stages.

Stage options

Snowflake announced the new feature to access, create, and edit named stages from Snowsight directly without using SQL commands. Until this announcement, users were using SQL commands to create the named stages. With new releases, named stages are made available to be created as a UI menu option. You can access it from the data interface and select the database and schema where the named stage is to be created. Once the schema is selected, click on the create option in the top right corner to create named stages – internal as well as external stages. Refer to [Figure 17.3](#) to get a glimpse of stage files uploaded to the named stage created using Snowsight UI options. You can click on the **+Files** button on the top right corner and upload files to the stage from Web UI:

The screenshot shows the Snowsight interface with the 'Data' section selected in the sidebar. In the main area, the 'Stages' section is expanded, and the 'SOURCE' stage under the 'CHAIRLIFT_PKG / CODE / SOURCE' database is selected. The stage details show it is an Internal Stage for the CHAIRLIFT_PROVIDER schema, created 1 week ago. The 'Stage Files' tab is selected, displaying a list of files in the SOURCE stage:

NAME	SIZE	LAST MODIFIED	... (More Options)
ui	22.3KB	1 week ago	...
setup.sql	11.2KB	1 week ago	...
README.md	224.0B	1 week ago	...
manifest.yml	1.6KB	1 week ago	...

Figure 17.3: Snowsight Stage View

Snowflake releases new features and patches every week and month. You can refer to the Snowflake documentation and latest release page to get new release

updates.

Introduction to new LLM capabilities

Large Language Models (LLMs) are designed to work with human language. These are nothing but algorithms developed using artificial neural network that are trained on millions to billions of different parameters. These are designed to be trained or learn like humans, they are usually trained on the enormous amount of data in the form of books, articles, internet, and content available. This is used to generate content, and is therefore referred as Generative AI that is used to generate the content in the form of text, documents, images, and so on. LLMs are the subset of Gen AI that are designed to focus on generating content. These can also be used to translate, summarize, and search with text as human language or natural language processing.

Snowflake, as a data platform, continuously works on enhancing the features to extend support to the LLMs. Snowflake's extended support with Snowpark capabilities allows users to bring their own models to the platform. You can bring your LLM or model to train and run on top of your data stored in Snowflake. Typically, to run the AI or Gen AI use cases, data needs to be moved to the environment where machine learning models can be run or have the required infrastructure to run the machine learning models. In the case of Snowflake, users do not need to move their data outside the platform as Snowflake announced support to AI and extensions to LLMs.

Snowflake's features – scalability, flexibility, and performance offer a powerful foundation for LLM-enabled machine learning applications. Snowflake's **Bring Your Own Model (BYOM)** allows users to bring their models to run LLM capabilities. You can also integrate the LLM with your application development using Streamlit and build a conversational AI or bot. Users can build and implement data apps quickly with the integration of Streamlit and foundational models or AI models.

Snowflake announced exciting news during Summit 2023. With Streamlit being available in the same environment and data platform, you can build interactive applications using Streamlit. The native apps framework announced is in private preview. There are ML-powered functions announced that can be used for forecasting, anomaly detection, and contribution exploration. These three analytical functions are powered by machine learning algorithms and are available in Private Preview.

Snowflake's Snowpark ML is a set of Python tools required to build machine-learning models within Snowflake. This feature is available in preview and consists of popular libraries used for machine learning, like – **scikit-learn**, **xgboost** and **lightgbm**. Snowpark ML works with Snowpark Python to use data frames to store data used to train or store results.

Snowflake has also announced many new features with SnowDay 2023. The most exciting feature is the Cortex. Snowflake Cortex is available in **Private Preview** and allows users a fully managed intelligent service to access industry-leading models. This also allows users to run the search functionality to access data and build AI applications. Snowflake also announced functions as part of Cortex functions that support LLM features like summarization, search, text to SQL, so on. Snowflake also announced the availability of a co-pilot to help developers develop applications easily with prompt engineering.

Snowflake also offers Document AI to process the documents and build applications to run analytics on top of document data. You can use document AI to extract data from documents, generate data, and store them as a part of Snowflake tables or datasets.

Snowflake continues to work on enhancing the platform to help users as well as provide a single platform that caters to all types of data needs – data engineering, data science, and data analytics. This is one umbrella that enables users to access, process, transform, store, and analyze their data as per their business needs without worrying about cross-cloud access, heterogeneous integrations, complex analytics, and improved AI capabilities.

You can refer to some of the quickstarts to get started with your version of ML implementation within Snowflake:

- Machine learning with Snowpark ML for Python -
https://quickstarts.snowflake.com/guide/intro_to_machine_learning_w_index=..%2F..index#0
- Image Recognition app using Snowpark, Pytorch, Streamlit and OpenAI -
https://quickstarts.snowflake.com/guide/image_recognition_snowpark_index=..%2F..index#0

You can also use <https://quickstarts.snowflake.com/> to view topics that match your areas of interest and start practicing. These quickstarts help to learn Snowflake features as well as get started with all new advanced features.

Conclusion

This chapter provided a summary of Snowflake's new features, SQL extensions and support to Gen AI implementations. You can leverage Snowflake native features to implement advanced capabilities. You have also learned features to implement AI capabilities with the recent Snowflake releases.

Points to remember

Following are the key takeaways from this chapter:

- Snowflake releases new features, enhancements, and patches every week and month.
- Snowflake extends its features and support to handle varying data needs as per the business requirements.
- You can implement all types of data solutions with Snowflake: OLAP, OLTP and hybrid.
- Snowflake offers extensions to the existing SQL functions to help users to process and transform data seamlessly.
- Snowflake's native app framework allows users to build , manage and deploy apps easily.
- Snowflake's support and new features to extend Gen AI capabilities makes the platform different from other data platform.
- Snowflake's BYOM allows users to use a single platform for all data needs.
- Integration of LLM capabilities and new native functions powered by LLM allows users to solve many business use cases quickly.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



Index

A

access hierarchy 174, 175
access privileges
 validating 173, 174
ACCOUNTADMIN 166, 167
ACCOUNT_USAGE 211, 212
 account usage functions 214
 account usage views 213, 214
 data retention 213
 dropped object information 213
 latency of detail 213
 reader account usage views 214
 roles and permissions 215
Active Directory (AD) integration 71
advanced features 344
 new features 344
 schema features 345
 Snowflake releases 344
 SQL extensions 344
 SQL features 345, 346
 tables 346
application dev team 29

B

binary data types 50
BI team 29
Bring Your Own Model (BYOM) 356
Building Reports (BI) 113
bulk unloading
 from table or query 116, 117
bulk unloading, to one or multiple files 117
COPY INTO options 118
 data, exporting to named external stages 120
 data exporting, to named internal stages 118, 119
 data, unloading to named external stages 120, 121
 file naming, in case of multiple file exports 117
GET command 120
PARTITION, using 117, 118

C

cache [41](#)
 metadata cache [41, 42](#)
 query result cache [42](#)
 warehouse cache [43](#)

Call Detail Records (CDRs) [333](#)

Change Data Capture (CDC) [128-130](#)

cloned objects
 usage [298](#)

cloud services layer [25](#)
 authentication and access control [26](#)
 availability [27](#)
 metadata storage and management [26](#)
 optimizer [26](#)
 result cache [26](#)
 security [26](#)
 uses [27](#)

cloud services optimization [262, 263](#)

Command Line Interface (CLI) [11, 317](#)

component costing [246, 247](#)
 cloud services cost [247](#)
 compute cost [247, 248](#)
 data transfer cost [249, 250](#)
 serverless cost [249](#)
 storage cost [248](#)

compute optimization
 auto suspend and auto resume, setting up [262](#)
 query queue time limit, setting up [262](#)
 query time limit, setting up [261](#)
 spend limits, enforcing [262](#)
 storage optimization [262](#)
 store only required data [262](#)
 warehouse access, setting up [261](#)
 warehouse size, setting up [261](#)

consumer application team [29](#)

context
 setting up [75](#)
 setting up, with SQL command [76](#)

Continuous Data Protection (CDP) [40](#)

COPY, for unloading data [115](#)

COPY INTO [115](#)
 compression supported [116](#)
 COPY for local system [103](#)
 COPY syntax [97, 98](#)
 COPY, with transformation syntax [98, 99](#)
 encryption supported [116](#)
 exported feeds [115](#)

formats [115](#)
locations [115](#)
options [100-102](#)
transformations supported [98](#)
using, for bulk loads [102](#)
using, for bulk upload [97](#)
using, to load data [97](#)
COPY optional parameters [99, 100](#)
cost dashboards
 implementing [263-266](#)
costing components [258](#)
 cloud services layer [259](#)
 compute or virtual warehouse layer [258](#)
 sample use case [259, 260](#)
 serverless services [259](#)
 storage layer [259](#)
cost optimization
 cloud services optimization [262](#)
 compute optimization [261](#)
 implementing [261](#)
cost optimizations [257](#)
CREATE CLONE command [295](#)
custom roles [171](#)
 access, granting to [171](#)
 access, testing [173](#)
 access, validating [173](#)
 creating [170](#)
 DATABASES access, granting to [171](#)
 Dev custom role access, granting to in QA environment [171](#)
 QA custom role access, granting to [171](#)
 using [173](#)
 WAREHOUSES access, granting to [171](#)

D

data access control [183](#)
database
 creating, with SQL commands [75](#)
 creating, with Web Console [75](#)
database feed [114](#)
database objects
 creating [70, 71](#)
 roles, creating with SQL commands [71](#)
 roles, creating with Web Console [72](#)
 users and roles, creating [71](#)
 users, creating with SQL commands [72](#)
 users, creating with Web Console [73](#)
data capture, with Streams

ALTER STREAM 133
CREATE STREAM 132, 133
DESCRIBE STREAM 133
DROP STREAM 133
SHOW STREAM 134
Data Center (DC) 198
data cloning 291
 considerations 295, 296
 implementation 296
 implementing 294
 needs 292, 293
 syntax of CREATE 296, 297
 use cases and examples 297, 298
 zero copy cloning 293, 294
data definition language 60
 account and session DDL 60, 61
 database, schema and share DDL 64
 data loading / unloading DDL 66
 data pipeline DDL 68
 table, view and sequence DDL 65
 user and security DDL 62
 user-defined functions, external functions, and stored procedures DDL 67
 warehouse and resource monitor DDL 63
data encryption 183
data engineering use cases 154, 155
Data Lake 330
 challenges 20
data lake reference use case 333
 business requirements 334
 pipeline design 335, 336
 platform design 336
 proposed solution design 334, 335
 reference architecture 335
 technical requirements 334
data load
 needs 86, 87
data loading and unloading
 file formats, creating 121
 LOAD commands, creating 122
 load, from local using COPY 122
 stages, creating 121
 storage integrations, creating 122
 unload, from local using COPY 123
data manipulation language 69
 data loading/unloading DML 69
 file staging commands 70
 standard DML 69
data masking 183

Data Mesh 330
data mesh reference use case 336
 business requirements 337
 platform design 338
 proposed solution design 337
 reference architecture 337
 technical requirements 336
data protection 182
data recovery 183
data recovery options 190
 time travel 190, 191
 time travel data retention 191, 192
 time travel setup 192
 time travel SQL extensions 191
 time travel storage cost 192
 time travel storage, for temporary and transient tables 192, 193
data replication 183, 198
 business continuity 201
 database replication 199
 implementing 198
 replication group 200
 replication schedule 200, 201
 share replication 199
data sharing 269
 implementing 272, 273
 needs 270-272
 working 273
data sharing options
 consumer task 286
 data exchange 284, 285
 listing 283
 providers task 285
DATA_SHARING_USAGE 212
data sharing, with non-Snowflake consumers
 implementing 280, 281
 reader account, creating 281-283
data sharing, with Snowflake consumers
 considerations 274
 data share, creating 276, 277
 data share, creating with multiple databases 277-279
 data share, using 279, 280
 implementing 274
 pre-requisites 275
 supported commands 274, 275
data solutions, with Snowflake
 data lake reference use case 333
 data mesh reference use case 336
 data warehouse reference use case 331

- implementing 329, 330
- Snowflake features 329
- use cases and architecture 331

data types 48

- binary data types 50
- date and time data types 51
- geospatial data types 59
- logical data types 50, 51
- numeric data types 48
- semi-structured data types 56
 - string data types 49, 50

data unloading 113, 114

Data Warehouse 18, 330

- challenges 18, 19

data warehouse reference use case 331

- business requirements 331
- pipeline design 333
- platform design 333
- proposed solution design 332
- reference architecture 332, 333
- technical requirements 331

date and time data types 51-53

- constants 54
- intervals 54
 - supported date and time arithmetic operations 55
 - supported date and time interval 54

DEV custom role 170

Directed Acyclic Graph (DAG) 137

Disaster Recovery (DR) 190

Discretionary Access Control (DAC) 160

drivers 319

- GO driver 319
- JDBC driver 319
- ODBC driver 319

dynamic masking 183

- custom role, creating 188
- custom role, granting to user 188
- data querying 189
- implementing 183
- masking policies, validating 189
- masking policy, applying to database objects 189
- masking policy, creating 188
 - using 187, 188

dynamic tables 347

E

Enterprise Edition and higher [191](#)
event tables [348](#), [349](#)
extended SQL objects implementation [77](#)
 external functions [81](#)
 SQL UDFT function [80](#)
 tabular functions [79](#)
 UDF example (JavaScript) [78](#)
 UDF example (SQL) [78](#)
Extract, Load and Transform (ELT) [327](#), [328](#)
Extract, Transform and Load (ETL) [326](#)

F

failsafe [193](#)
 storage cost [194](#)
file feed [114](#)

G

General availability (GA) [344](#)
geospatial data types [59](#)
 geometry [59](#)
GO driver [319](#)

H

Hadoop [21](#)
Hadoop ecosystem [21](#)
 extended support, to NoSQL [22](#), [23](#)
hybrid tables [350](#), [351](#)

I

iceberg tables [351](#), [352](#)
implementation commands, dynamic making
 alter masking policy [186](#)
 conditional masking policy [185](#)
 describe masking policy [187](#)
 drop masking policy [187](#)
 masking policy, creating [183](#), [184](#)
 normal masking policy [184](#), [185](#)
 show masking policy [187](#)
INFORMATION_SCHEMA [210](#), [211](#), [216](#)
 functions [217](#), [218](#)
 views [216](#)

J

JDBC driver [319](#)

L

Large Language Models (LLMs) [155](#)

capabilities [356, 357](#)

legacy data warehouse

challenges [19](#)

LIST command [93](#)

listing [283, 284](#)

free listing [284](#)

marketplace listing [284](#)

paid listing [284](#)

personalized listing [284](#)

private listing [284](#)

logical data types [50, 51](#)

M

Massively Parallel Processing (MPP) [18, 339](#)

metadata cache [41, 42](#)

N

near real time data

loading [111](#)

Network Attached Storage (NAS) [87, 114, 270](#)

numeric data types

fixed point data types [48](#)

floating data types [49](#)

O

ODBC driver [319](#)

ORGADMIN [166](#)

P

performance measures

calculating [219-221](#)

performance optimization [228](#)

implementing [227](#)

query performance optimization [228](#)

storage performance optimization [237](#)

warehouse performance optimization [233](#)

platform needs

ELT [327, 328](#)

ETL [326, 327](#)

private preview (Pr Pr) [344](#)

programming connectors [318](#)

Snowflake connector, for Kafka [318](#)

Snowflake connector, for Python [318](#)

Snowflake connector, for Spark 318
public preview (Pu Pr) 344
PUT command 103

Q

Query Acceleration Service 224
enabling 226
query acceleration billing 227
QUERY_ACCELERATION_ELIGIBLE view 225, 226
query acceleration usage 226, 227
SYSTEM\$ESTIMATE_QUERY_ACCELERATION function 225
QUERY_HISTORY 221-224
query performance optimization 228
historical performance, viewing in Snowsight 228-230
queries, listing 231, 232
query profile Snowsight, viewing 230, 231
query result cache 42

R

RBAC implementation, with Snowflake 167, 168
access privileges, validating 173
custom roles access, granting 171
custom roles, assigning to users 172
custom roles, creating 170
custom roles, using 173, 174
DEV custom role, creating 170
ML databases and schemas, creating 170
Snowflake resources, creating 168, 169
users, creating 172
warehouses, creating for users and use cases 170
real time streaming data
loading 109
Snowflake connector for Kafka 109, 110
Snowpipe 109
Snowpipe streaming API 110, 111
Snowpipe streaming, versus Snowpipe 111
streaming cost 111
Role Based Access Control (RBAC) 71, 159-165
role types
account roles 162
active roles 162
custom roles 162
database roles 162
system-defined roles 162

S

- scaling policies
 - economy policy [33](#)
 - standard policy [33](#)
- schemas
 - creating, with SQL commands [75](#)
- SECURITYADMIN** [167](#)
- semi-structured data types [56](#)
 - ARRAY** [58](#)
 - ARRAY considerations [59](#)
 - ARRAY constant [58](#)
 - ARRAY elements [59](#)
 - OBJECT** [57](#)
 - OBJECT considerations [58](#)
 - OBJECT constant [57](#)
 - OBJECT elements [57](#)
 - VARIANT** [56, 57](#)
- Service Level Agreements (SLAs) [227](#)
- Service Level Objectives (SLOs) [227](#)
- shared data architecture [25](#)
- Slowly Changing Dimensions (SCD) [130](#)
- Snowflake [1](#)
 - advanced features [344](#)
 - architecture [23-25](#)
 - certifications [3, 4](#)
 - community [4, 5](#)
 - connecting to [7, 12-14](#)
 - default roles [166, 167](#)
 - features [2](#)
 - history [3](#)
 - securable objects [163](#)
 - trial account, setting up [5-7](#)
- Snowflake access control overview [160](#)
 - role and users [161](#)
 - types, of roles [162](#)
- Snowflake architecture [23-25](#)
 - cloud services layer [25](#)
 - storage layer [39](#)
 - virtual warehouse layer [28, 29](#)
- Snowflake clones
 - creating [298, 299](#)
- Snowflake costing [244](#)
 - control [245](#)
 - optimization [246](#)
 - visibility [245](#)
- Snowflake costs
 - monitoring [253-255](#)
- Snowflake load objects
 - creating [87](#)

data, loading from Web UI 94-96
source files, loading from file locations 87-94
source files, loading from local system 94
sources files 87
Snowflake metadata objects 210
 ACCOUNT_USAGE 211
 cloud services usage, viewing 251
 compute usage, viewing 250, 251
 DATA_SHARING_USAGE 212
 INFORMATION_SCHEMA 210, 211
 ORGANIZATION_USAGE 212
 serverless usage, viewing 252, 253
 using 250
Snowflake performance 208
 query performance 208-210
Snowflake policies
 creating 202, 203
Snowflake recommendations 339
 load recommendations 339, 340
 storage recommendations 339
 transformation recommendations 340
 usage recommendations 340
 warehouse recommendations 339
Snowflake shares
 account setup 287
 creating 286
 data sharing 287
 scenario 286
 technical requirement 287
Snowflake Standard Edition 191
Snowflake Streams 130
 append-only stream 131
 billing 132
 change logs validity 132
 creating 142
 examples 134, 135
 insert-only streams 131
 standard streams 131
Snowflake Tasks
 creating 142
Snowpark 145
 benefits over Spark Connector 150
 engineering use cases 155, 156
 features 146-149
 setting up 155
 Snowpark-ML 150
 Snowpark-optimized warehouse 149
 Snowpark-optimized warehouse billing 149

- use cases 155
- Snowpark implementation 150
 - data engineering use cases 154
 - environment setup 151
 - for ML 153
 - Python and developing code, using 151
 - Python dataframe API 152, 153
 - use cases 153
- Snowpipe 104
 - commands 106
 - features 106
 - implementing 104, 112, 113
 - load, automating with cloud messaging 104, 105
 - recommendations 106
 - REST endpoint, using 105
 - SQL commands 106
 - versus bulk load 105
- Snowsight 302
 - activity page 306
 - admin page 307
 - budgets 354, 355
 - dashboard page 305, 306
 - dashboards, implementing with 307-313
 - data governance 353
 - data loading 355
 - data page 306
 - data pages 305
 - interface 302, 303
 - marketplace page 306
 - native app 354
 - new features 352
 - stage options 355
 - worksheet pages 303, 304
 - worksheets 353
- Snowsight dashboard
 - creating 313
 - implementing 307-313
 - initial setup 314
 - key asks 314
 - use case 313
- Snowsql 317-320
 - connecting to 320
 - downloading 11
 - installing 320
 - password, connecting with 321
 - setting up 12
- Software as a Solution (SaaS) 23
- SQL commands

- data definition language [60](#)
- data manipulation language [69](#)
- using [60](#)
- SQL commands, Snowpipe
 - ALTER PIPE [107, 108](#)
 - CREATE PIPE [106, 107](#)
 - DESCRIBE PIPE [108](#)
 - DROP PIPE [108](#)
 - SNOW PIPE [108](#)
- SQL UDF function [80](#)
 - creating [80](#)
 - using [81](#)
- storage layer [39](#)
 - data cloning [40](#)
 - failsafe [40](#)
 - time-travel [40](#)
 - usage and billing [40, 41](#)
- storage performance optimization [237](#)
 - automatic clustering [237](#)
 - materialized views [237](#)
 - performance considerations [238, 239](#)
 - right strategy, selecting [238](#)
 - search optimization service [238](#)
- Stored Procedure (SP) [151](#)
- streaming loads [104](#)
- string data types [49, 50](#)
- SYSADMIN [167](#)

T

- tables [346](#)
 - creating, with SQL command [76, 77](#)
 - dynamic tables [347](#)
 - event tables [348, 349](#)
 - hybrid tables [350, 351](#)
 - iceberg tables [351, 352](#)
- tabular functions
 - creating [79](#)
 - sample function [79](#)
 - using [79](#)
- tagging objects
 - creating [202, 203](#)
- tasks [135, 136](#)
 - scheduling tasks [137](#)
 - user-managed tasks [136](#)
- TASKs
 - scheduling implementation [137](#)
- TASKs commands [137](#)

ALTER TASK [140](#)
CREATE TASK [138, 139](#)
DROP TASK [141](#)
EXECUTE TASK [141](#)
SHOW TASKS [141](#)
time travel implementation [194](#)
 commands, to access dropped and restored objects [196](#)
 dropped objects, restoring [196, 197](#)
 historical data access [195](#)
 list dropped objects [196](#)
 objects cloning [195](#)
 reference use case [197, 198](#)
 setting up, at create table [194](#)

U

Unified Data Platform (UDP) [18](#)
USERADMIN [167](#)
User Defined Functions (UDFs) [77, 151](#)
User Defined Table Functions (UDFT) [79, 151](#)
users
 creating [172](#)
 custom roles, assigning to [172](#)
users and roles management [175](#)
 ALTER command [176](#)
 CREATE command [176](#)
 DESCRIBE command [176](#)
 DROP command [176](#)
 SHOW command [176](#)
 user commands [176](#)

V

VALIDATE command [99](#)
virtual warehouse layer [28, 29](#)
 billing and usage [38, 39](#)
 multi-cluster warehouses [30](#)
 single and multi-cluster [29](#)
 single cluster warehouses [30](#)
SQL commands, using [37](#)
virtual warehouse properties [30](#)
virtual warehouse sizes [31](#)
warehouse commands [38](#)
warehouse, creating [37](#)
warehouse scaling [31, 33](#)
warehouse setup, using console [33-37](#)

W

warehouse cache [43](#)
warehouse performance optimization [233](#)
 cache, optimizing [235, 236](#)
 concurrent queries, limiting [236](#)
 memory spillage, resolving [234, 235](#)
 query acceleration service, using [236, 237](#)
 queue time, reducing [233, 234](#)
 warehouse size, changing [235](#)
warehouses
 creating [73](#)
 creating, with SQL command [74](#)
 creating, with Web Console [74](#)
warehouse scaling [31](#)
 scaling modes [32](#)
 scaling out [32](#)
 scaling policies [33](#)
 scaling up [31, 32](#)
Web User Interface
 classic console [8](#)
 context setting [9](#)
 Snowsight [10, 11](#)

Z

zero copy cloning [293, 294](#)