

O'REILLY®

Snowflake The Definitive Guide

Architecting, Designing, and Deploying
on the Snowflake Data Cloud

Early
Release

RAW &
UNEDITED



Joyce Kay Avila

Snowflake: The Definitive Guide

*Architecting, Designing, and Deploying on the
Snowflake Data Cloud*

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Joyce Kay Avila

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Snowflake: The Definitive Guide

by Joyce Kay Avila

Copyright © 2022 Joyce Kay Avila. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Acquisitions Editor: Jessica Haberman

Interior Designer: David Futato

Development Editor: Michele Cronin

Cover Designer: Karen Montgomery

Production Editor: Clare Jensen

Illustrator: Kate Dullea

September 2022: First Edition

Revision History for the Early Release

2021-08-04: First Release

2021-09-13: Second Release

2021-10-20: Third Release

2021-12-17: Fourth Release

2022-02-08: Fifth Release

2022-03-17: Sixth Release

2022-04-13: Seventh Release

2022-05-24: Eighth Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781098103828> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Snowflake: The Definitive Guide*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author, and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and Snowflake. See our [statement of editorial independence](#).

978-1-098-10375-0

[LSI]

Table of Contents

1. Creating and Managing Snowflake Architecture.....	1
Traditional Data Platform Architectures	2
Shared-Disk (Scalable) Architecture	2
Shared-Nothing (Scalable) Architecture	3
NoSQL Alternatives	4
Snowflake Architecture	5
Managing the Cloud Services Layer	6
Billing for the Cloud Services Layer	7
Query Processing (Virtual Warehouse) Compute Layer	8
Virtual Warehouse Size	9
Scaling Up a Virtual Warehouse to Process Large Data Volumes and Complex Queries	10
Scaling Out with Multi-Cluster Warehouses to Maximize Concurrency	14
Creating and Using Virtual Warehouses	17
Separation of Workloads and Workload Management	23
Billing for Virtual Warehouse Layer	25
Centralized (Hybrid-Columnar) Database Storage Layer	25
Introduction to Zero Copy Cloning	26
Introduction to Time Travel	26
Billing for Storage Layer	27
Snowflake Caching	27
Query Results Cache	27
Metadata Cache	28
Virtual Warehouse Local Disk Cache	29
Get Ready for Hands-On Learning!	30
Exercises to Test Your Knowledge	31

2. Creating and Managing Snowflake Architecture Objects.....	33
Creating and Managing Snowflake Databases	34
Creating and Managing Snowflake Schemas	40
INFORMATION_SCHEMA and Account Usage	43
Schema Object Hierarchy	47
Introduction to Snowflake Tables	47
Creating and Managing Views	52
Introduction to Snowflake Stages - File Format Included	56
Extending SQL with Stored Procedures and UDFs	59
User Defined Function (UDF) – Task Included	61
Secure SQL UDTF That Returns Tabular Value (Market Basket Analysis Example)	62
Stored Procedures	64
Introduction to Pipes, Streams, and Sequences	71
Code Cleanup	72
Exercises to Test Your Knowledge	72
3. Exploring Snowflake SQL Commands, Data Types, and Functions.....	75
Working with SQL Commands in Snowflake	76
Data Definition Language (DDL) Commands	77
Data Control Language (DCL) Commands	77
Data Manipulation Language (DML) Commands	78
Transaction Control Language (TCL) Commands	78
Data Query Language (DQL) Commands	78
SQL Query Development, Syntax and Operators in Snowflake	79
SQL Development and Management	79
Query Syntax	80
Query Operators	90
Long Running Queries and Query Performance & Optimization	91
Snowflake Query Limits	91
Introduction to Data Types Supported by Snowflake	92
Numeric Data Types	93
String & Binary Data Types	95
Date & Time Input / Output Data Types	96
Semi-structured Data Types	97
Unstructured Data Types	98
Snowflake SQL Functions and Session Variables	98
Using System Defined (Built-In) Functions	98
Creating SQL & Javascript User-defined Functions (UDFs) and using Session Variables	101
External Functions	102
Code Cleanup	103

Summary	103
Exercises to Test Your Knowledge	103
4. Leveraging Snowflake Access Controls.....	105
Creating Securable Objects	109
Snowflake System-Defined Roles	113
Creating Custom Roles	115
Functional-Level Business & IT Roles	116
System-Level Service Account and Object Access Roles	117
Role Hierarchy Assignments: Assign Roles to Other Roles	119
Granting Privileges to Roles	121
Assigning Roles to Users	123
Testing and Validating Our Work	123
User Management	127
Code Cleanup	134
Exercises to Test Your Knowledge	136
5. Managing Snowflake Account Costs.....	137
Prep Work	138
Snowflake Monthly Bill	138
Storage Fees	140
Data Transfer Costs	140
Credits Consumed	140
Monitoring Virtual Warehouse Usage and Reducing Costs	141
Creating and Managing Resource Monitors	141
Using Object Tagging for Cost Centers	153
Querying the Account Usage View	153
Using BI Partner Dashboards to Monitor Snowflake Usage and Costs	154
Snowflake Agile Software Delivery	155
Why Do We Need DevOps?	155
Continuous Data Integration, Continuous Delivery, and Continuous Deployment	156
What is Database Change Management (DCM)?	156
How Zero-Copy Cloning can be used to Support Dev / Test Environments	158
Code Cleanup	162
Summary	162
Exercises to Test Your Knowledge	163
6. Configuring and Managing Secure Data Sharing.....	165
Introduction	165
Snowflake Architecture Supports Data Sharing	166
Data Sharing Use Cases	167

Snowflake Support for Unified ID 2.0	167
Snowflake Secure Data Sharing Approaches	168
Prep Work	169
Snowflake Direct Secure Data Sharing Approach	170
Creating Outbound Shares	171
How Inbound Shares are Used By Snowflake Data Consumers	182
How to List and Shop on the Public Snowflake Data Marketplace	185
Snowflake Data Marketplace for Providers	186
Standard Versus Personalized Data Listings	189
Harnessing the Power of a Snowflake Private Data Exchange	192
Snowflake Data Clean Rooms	193
Important Design, Security and Performance Considerations	194
Share Design Considerations	194
Share Security Considerations	195
Shared Table Performance Considerations	195
Difference Between Database Sharing and Database Cloning	196
Data Shares and Time Travel Considerations	196
Sharing of Data Shares	196
Summary	196
Code Cleanup	197
Knowledge Check	197
7. Visualizing Data for Better Insights.	199
How to Access Snowsight	200
Navigating Snowsight and Data Sampling	203
Improved Productivity	208
Using Contextual Suggestions from Smart Autocomplete	208
Formatting SQL	209
Previewing Data Quickly	210
Using Shortcuts	210
Using Automatic Statistics and Interactive Results	212
Accessing Version History	214
Visualization	215
Creating a Dashboard and Tiles	216
Working with Chart Visualizations	218
Aggregating and Bucketing Data	220
Editing and Deleting Tiles	224
Collaboration	225
Sharing Your Query Results	225
Using a Private Link to Collaborate on Dashboards	225
Exercises to Test Your Knowledge	227

Creating and Managing Snowflake Architecture

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

In the last decade, computer data storage and computer performance had to evolve. Teams, both large and small, near or far apart, often needed access to the same data at the same time. Having access to that data and the ability to generate actionable insights quickly is now an absolutely must. The sheer amount and complexity of today’s data platforms, including data warehouses, had to evolve into incredibly data-intensive applications. Yet, as we’ll discover in the next section, simply making modifications to existing data platform architectures did not solve the scalability problem. Then, Snowflake burst onto the scene with a unique architecture.

Snowflake is an evolutionary modern data platform that solved the scalability problem. Compared to traditional cloud data platform architectures, Snowflake enables data storage and processing that is significantly faster and easier to use and is much more affordable. Snowflake’s Data Cloud provides users with a unique experience by

combining a new SQL query engine with an innovative architecture that was designed and built from the ground up, specifically for the cloud.

Traditional Data Platform Architectures

In this section, we'll briefly review some traditional data platform architectures and how they were designed in an attempt to improve scalability. Scalability is the ability of a system to handle an increasing amount of work. We'll also discuss the limitations of these architectures and we will discover what makes the Snowflake Data Cloud architecture so unique. Afterward, we will learn about each of the three different Snowflake architecture layers in detail: Cloud Services Layer, Query Processing (Virtual Warehouse) Compute Layer, and the Centralized (Hybrid-Columnar) Database Storage Layer.

Shared-Disk (Scalable) Architecture

The shared-disk architecture was an early scaling approach designed to keep data stored in a central storage location, accessible from multiple database cluster nodes ([Figure 1-1](#)). The data accessed by each of the cluster nodes is consistently available because all data modifications are written to the shared disk. This architecture is a traditional database design and is known for the simplicity of its data management. While the approach is simple in theory, it requires complex on-disk locking mechanisms to ensure data consistency which, in turn, causes bottlenecks. Data concurrency, allowing many users to affect multiple transactions within a database, is also a major problem and adding more compute nodes only compounds the problem in a shared-disk architecture. Therefore, the true scalability of this architecture is limited.

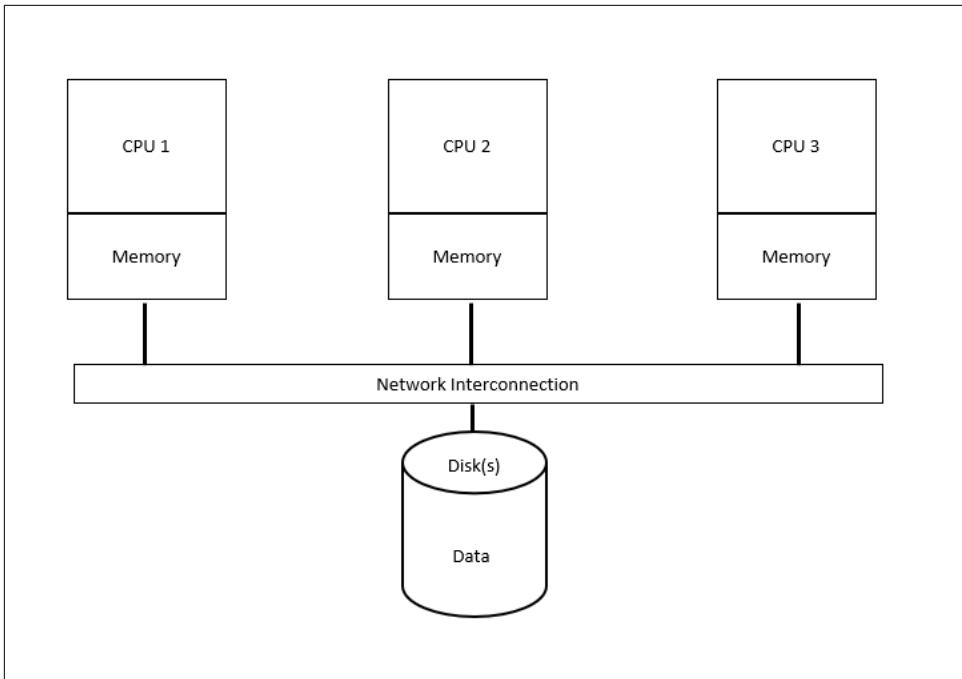


Figure 1-1. Shared-Disk Architecture

Oracle RAC is an example of shared disk architecture.

Shared-Nothing (Scalable) Architecture

The shared-nothing architecture, in which storage and compute is scaled together (Figure 1-2), was designed in response to the bottleneck created by the shared-disk architecture. This evolution in architecture was made possible because storage had become relatively inexpensive. However, distributed cluster nodes along with the associated disk storage, CPU, and memory, requires data to be shuffled between nodes, which adds overhead. Depending on how the data is distributed across the nodes will determine the extent of the additional overhead. Striking the right balance between storage and compute is especially difficult. Even when it is possible to resize a cluster, it takes time to do so. Thus, organizations often overprovision shared-nothing resources, which results in unused, unneeded resources.

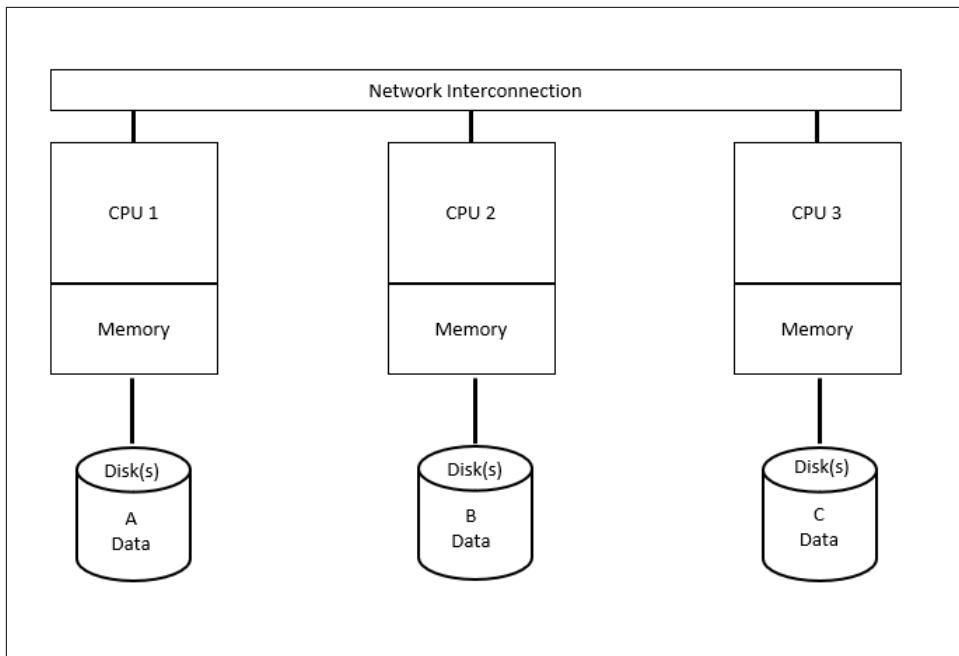


Figure 1-2. Shared-Nothing Architecture

Shared nothing architecture is also known as Massively Parallel Processing (MPP). Examples of a Shared Nothing architecture include IBM DB2, Vertica, and Pivotal Greenplum.

NoSQL Alternatives

Most NOSQL solutions rely on shared-nothing architecture; thus, they have many of the same limitations. However, the benefit of NoSQL solutions is that they can store non-relational data without first requiring transformation of the data. Additionally, most NoSQL systems don't require schemas. NoSQL, a term that implies “Not Only SQL” rather than “NO to SQL”, is a good choice for storing e-mail, web links, social media posts and tweets, road maps, and spatial data.

There are four types of NoSQL databases: Document Stores, Key-Value (KV) Stores, Column Family Data Stores or Wide Column Data Stores, and Graph Databases.

Document-based NoSQL databases such as MongoDB store data in JSON objects where each document has key-value pair like structures. Key-value databases such as DynamoDB are especially useful for capturing customer behavior in a specific session. Cassandra is an example of a column-based database where large numbers of dynamic columns are logically grouped into column families. Graph-based databases, such as Neo4j and Amazon Neptune, work well for recommendation engines and

social networks where they're able to help find patterns or relationships among data points.

A major limitation of NoSQL stores is that they perform poorly when doing calculations involving many records, such as aggregations, window functions, and arbitrary ordering. Thus, NoSQL stores can be great when you need to quickly create, read, update and delete (CRUD) individual entries in a table but aren't recommended for adhoc analysis. Additionally, NoSQL alternative solutions require specialized skill sets and they aren't compatible with most SQL-based tools.

The NoSQL solutions, however, are not database warehouse replacements. While NoSQL alternatives can be useful for data scientists, they do not perform well for analytics.

Snowflake Architecture

Even the improved traditional data platforms, especially those that were implemented on-premise, couldn't adequately address modern data problems or solve the long-standing scalability issue. The Snowflake team made the decision to take a unique approach. Rather than trying to incrementally improve or transform existing software architectures, the Snowflake team's approach was to build an entirely new modern data platform, just for the cloud, that allows multiple users to concurrently share live data.

The unique Snowflake design physically separates but logically integrates storage and compute along with providing services such as security and management. As we explore the many unique Snowflake features throughout the upcoming chapters, you'll be able to see for yourself why the Snowflake architecture is the only architecture that can enable the Data Cloud.

The Snowflake hybrid-model architecture is comprised of three layers ([Figure 1-3](#)) known as the cloud services layer, the compute layer, and the data storage layer. Each of these layers, along with the three Snowflake caches, are discussed in more detail in the following sections.

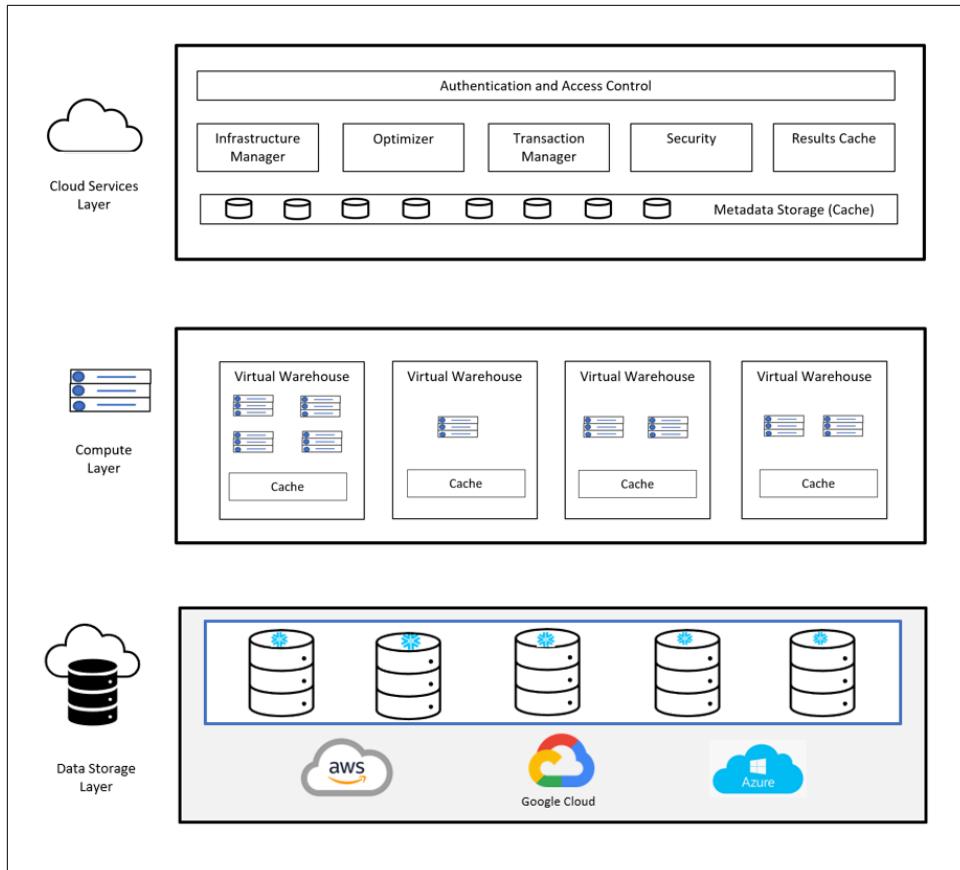


Figure 1-3. Snowflake's hybrid columnar architecture

Snowflake's processing engine is native SQL and, as we will see in later chapters, Snowflake is also able to handle semi-structured and unstructured data.

Managing the Cloud Services Layer

All interactions with data in a Snowflake instance begin in the cloud services layer, also called the global services layer (Figure 1-4). The Snowflake cloud services layer is a collection of services that coordinate activities such as authentication, access control and encryption. It also includes management functions for handling infrastructure and metadata, as well as performing query parsing and optimization, among other features. This global services layer is sometimes referred to as the Snowflake “brain” because all the various service layer components work together to handle user requests that begin from the time a user requests to log in.

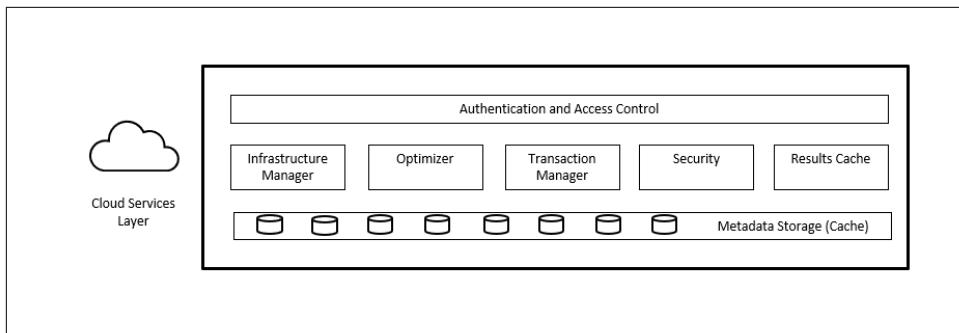


Figure 1-4. Snowflake Cloud Services Layer

Each time a user requests to log in, their request is handled by the services layer. When a user submits a Snowflake query, the SQL query will be sent to the cloud services layer optimizer before being sent to the compute layer for processing. The service layer is what enables the SQL client interface for Data Definition Language (DDL) and Data Manipulation Language (DML) operations on data.

The cloud services layer manages data security including the security for data sharing. The Snowflake cloud services layer runs across multiple availability zones in each cloud provider region and holds the results cache, a cached copy of the executed query results. The metadata required for query optimization or data filtering are also stored in the cloud services layer.



Just like the other Snowflake layers, the cloud services layer will scale independently of the other layers. The scaling of the cloud services layer is an automated process that cannot be directly manipulated by the Snowflake end user.

Billing for the Cloud Services Layer

Most cloud services consumption is already incorporated into Snowflake pricing. However, when cloud services layer usage exceeds 10% of compute usage (calculated daily), they are billed at the normal credit price. Note that daily compute credit usage is calculated daily in UTC time zone.

All queries use a small amount of cloud services resources. Data Definition Language operations are metadata operations and, as such, they use only cloud services. Keeping both facts in mind, we should evaluate some situations where we know the cost will be higher for cloud services to consider whether the benefits will be worth the increased costs.

Increased usage of the cloud services layer will likely occur when using several simple queries, especially queries accessing session information or using session variables.

Increased usage also occurs when using large complex queries with many joins. Single row inserts, rather than bulk or batch loading, will also result in higher cloud services consumption. Finally, you'll consume only cloud services resources if you use `Information_Schema` commands or certain metadata-only commands such as the "Show" command. If you are experiencing higher than expected costs for cloud services, you may want to investigate these situations. Be sure to also investigate any partner tools, including those using the JDBC driver, as there could be opportunities for improvement from these third-party tools.

Even though the cloud services cost for a particular use case is high, sometimes it makes sense either economically and/or strategically to incur those costs. For example, taking advantage of the result cache for queries, especially for large or complex queries, will mean zero compute cost for that query. Thoughtful choices about the right frequency and granularity for DDL commands, especially for use cases such as cloning, help to better balance the costs between cloud services consumption and warehouse costs to achieve an overall lower cost.

Query Processing (Virtual Warehouse) Compute Layer

A Snowflake compute cluster, most often referred to simply as a "virtual warehouse," is a dynamic cluster of compute resources consisting of CPU memory and temporary storage. Creating virtual warehouses in Snowflake makes use of the compute clusters, virtual machines in the cloud, which are provisioned behind the scenes. Snowflake doesn't publish the exact server in use at any given time; it could change as the cloud providers modify their services. The Snowflake compute resources are created and deployed on-demand anytime to a Snowflake user, such as yourself, for whom the process is transparent.

A running virtual warehouse is required for most SQL queries and all DML operations, including loading and unloading data into tables, as well as updating rows in tables. Some SQL queries can be executed without requiring a virtual warehouse and we'll soon see examples of that when we discuss the query results cache later in the chapter.

Snowflake's unique architecture allows for separation of storage and compute which means any virtual warehouse can access the same data as another, without any contention or impact on performance of the other warehouses.

No virtual warehouse has an impact on any other virtual warehouse because each Snowflake virtual warehouse operates independently and does not share compute resources with other virtual warehouses ([Figure 1-5](#)).

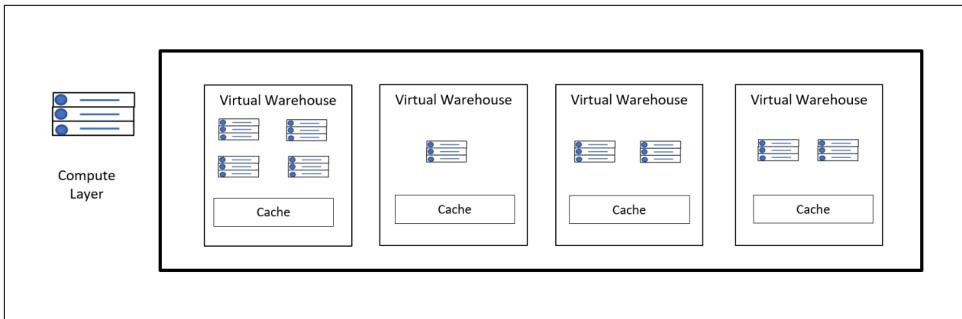


Figure 1-5. Snowflake Compute (Virtual Warehouse) Layer

A virtual warehouse is always consuming credits when it is running in a session. However, Snowflake virtual warehouses can be started and stopped at any time and they can be resized at any time, even while running. Snowflake supports two different ways to scale warehouses. Virtual warehouses can be scaled up by resizing a warehouse and can be scaled out by adding clusters to a warehouse. It is possible to use one or both scaling methods at a time.



Unlike the Snowflake cloud services layer and the data storage layer, the Snowflake virtual warehouse layer (Figure 1-5) is not a multi-tenant architecture. Snowflake pre-determines the CPU, memory, and SSD configurations for each node in a virtual warehouse (Figure 2-6). While these definitions are subject to change, they are consistent in configuration across all three cloud providers.

Virtual Warehouse Size

A Snowflake compute cluster is defined by its size with size corresponding to the number of servers in the virtual warehouse cluster. For each virtual warehouse size increase, the number of servers per cluster increases by a factor of 2 up to size 4X-Large (Figure 2-6). Beyond 4X-Large, a different approach is used to determine the number of servers per cluster. However, the credits per hour does still increase by a factor of 2 for these extremely large virtual warehouses.

Table 1-1. Snowflake Virtual Warehouse Sizes and associated number of servers per cluster

X-Small	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X- Large
1	2	4	8	16	32	64	128

Virtual Warehouse resizing to a larger size, also known as scaling up, is most often undertaken to improve query performance and handle large workloads. This will be discussed in more detail in the next section.



Because Snowflake utilizes per second billing, it can often be cost effective to run larger warehouses because you are able to suspend virtual warehouses when they aren't being used. The exception is when you are running a lot of small or very basic queries on large warehouse sizes. There won't likely be any benefit from adding the additional resources regardless of the number of concurrent queries.

Scaling Up a Virtual Warehouse to Process Large Data Volumes and Complex Queries

In a perfect world (i.e., simple workload, exact same per test), you'd pay the same total cost for using an XS virtual warehouse as using a 4XL virtual warehouse. The only difference would be a decrease in the time to completion. In reality, though, it isn't quite that simple. Many factors affect the performance of a virtual warehouse. The number of concurrent queries, the number of tables being queried, and the size and composition of the data are a few things that should be considered when sizing a Snowflake virtual warehouse.

Sizing appropriately matters. A lack of resources, due to the virtual warehouse being too small, could result in taking too long to complete the query. There could be a negative impact if the query is too small and the virtual warehouse too large.

Resizing a Snowflake virtual warehouse is a manual process and can be done even while queries are running because a virtual warehouse does not have to be stopped or suspended to be resized. However, when a Snowflake virtual warehouse is resized, only subsequent queries will make use of the new size. Any queries already running will finish running while any queued queries will run on the newly sized virtual warehouse. Scaling a virtual warehouse UP will increase the number of servers ([Figure 1-6](#)). An example would be from MEDIUM to LARGE. Scaling a virtual warehouse DOWN will decrease the number of servers.



It is recommended that you experiment with different types of queries and different virtual warehouse sizes to determine the best way to manage your virtual warehouses effectively and efficiently. The queries should be of a certain size and complexity that you would typically expect to complete within no more than 5 to 10 minutes. Additionally, it is recommended that you start small and increase in size as you experience. It is easier to identify an under-sized virtual warehouse than an under-utilized one.

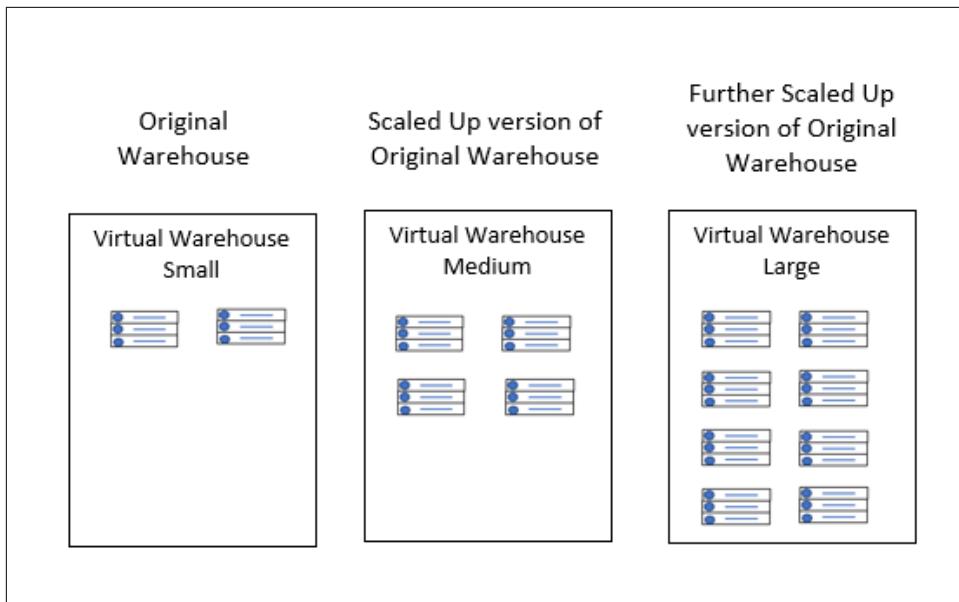


Figure 1-6. Scaling up a Snowflake warehouse increases the size of the cluster

In practice, you'd create a new Original virtual warehouse with the size being defined as small, as shown in [Figure 1-7](#).

New Warehouse

Creating as SYSADMIN

Name	Size ?
Original	Small 2 credits/hour
Comment (optional)	
Multi-cluster Warehouse Scale compute resources as query needs change	
Advanced Warehouse Options	
<input type="button" value="Cancel"/>	<input style="background-color: #0072BD; color: white; border-radius: 5px; padding: 5px; font-weight: bold; font-size: 1em; border: none; width: 150px; height: 30px;" type="button" value="Create Warehouse"/>

Figure 1-7. Creating a new warehouse in the Snowflake Web UI

Then, to scale up the Original virtual warehouse, you would edit the virtual warehouse properties to increase the cluster size to medium ([Figure 1-8](#)).

Edit Warehouse

COMPUTE_WH as SYSADMIN

Name: Original

Size: Medium 4 credits/hour (highlighted with a red box)

Comment (optional):

Multi-cluster Warehouse: Scale compute resources as query needs change (toggle switch)

Mode: Maximized

Clusters: 1

Advanced Warehouse Options:

- Auto Resume: On
- Auto Suspend: On
- Suspend After (min): 10

Cancel | Save Warehouse

Figure 1-8. Increasing the cluster size to Medium in the Snowflake Web UI



Larger virtual warehouses do not necessarily result in better performance for query processing or data loading.

Query processing, in terms of query complexity, is a consideration for choosing a virtual warehouse size because the time it takes for a server to execute a complex query will likely be greater than running a simple query. The amount of data to be loaded or unloaded can greatly affect performance. We'll be diving into data loading and unloading in Chapter 6 and reviewing ways to improve performance in Chapter 9.

Scaling Out with Multi-Cluster Warehouses to Maximize Concurrency

A multi-cluster warehouse operates in much the same way as a single-cluster warehouse. The goal is to find the right balance where the Snowflake system will perform optimally in terms of size and number of clusters. From the previous section, we learned that when there was a queuing problem due to very long-running SQL queries or when there was a large data volume to be loaded or unloaded then scaling up could result in increased performance since the queries could run faster.

If a concurrency problem is due to many users, or connections, then scaling up will not adequately address the problem. Instead, we'll need to scale out by adding clusters ([Figure 1-9](#)), going from a MIN value of 1 to a MAX value of 3, for example. Multi-cluster warehouses can be set to automatically scale if the number of users and/or queries tend to fluctuate.



Multi-cluster warehouses are available on the Enterprise, Business Critical, and Virtual Private Snowflake Editions.

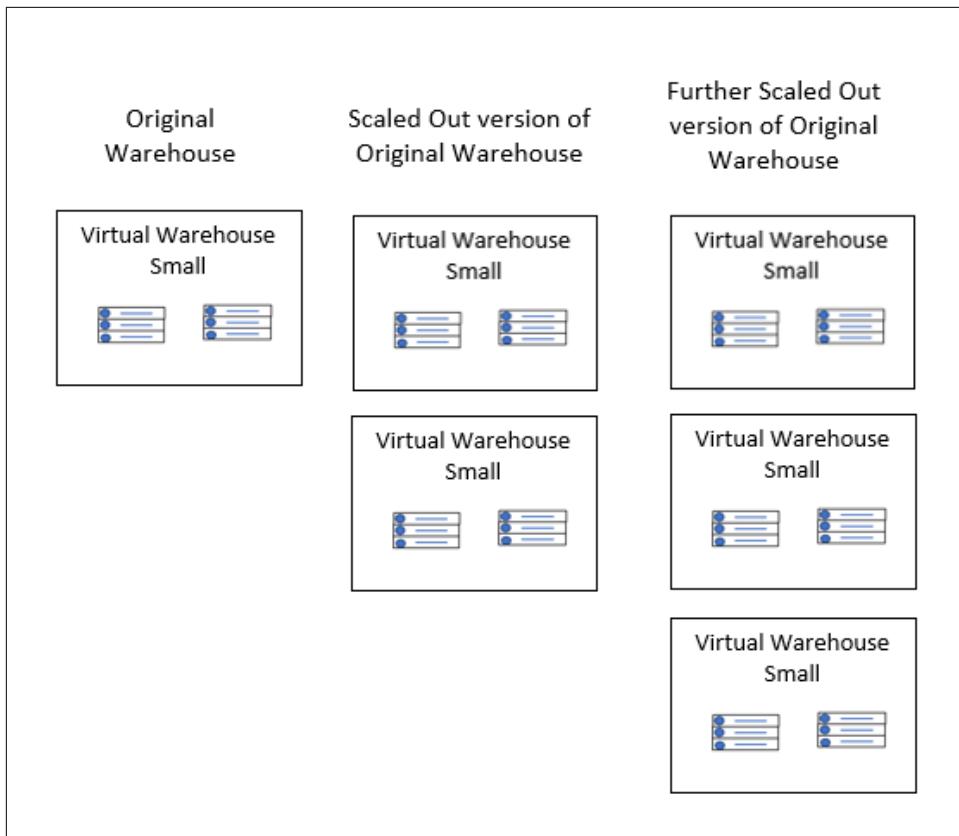


Figure 1-9. Scaling out increases the number of Snowflake compute clusters

Just like single cluster warehouses, multi-cluster warehouses can be created through the web interface or by using SQL for Snowflake instances. Unlike single cluster warehouses where sizing is a manual process, scaling in or out for multi-cluster warehouses is an automated process. An example of how you can edit a Snowflake virtual warehouse to have a minimum of one small cluster and a maximum of three small clusters can be seen in [Figure 1-10](#).

Edit Warehouse

COMPUTE_WH as SYSADMIN

Name	Size ?
Original	Small 2 credits/hour
Comment (optional)	
Multi-cluster Warehouse	
Scale compute resources as query needs change	
Mode	Auto-scale
Min Clusters	1
Max Clusters	3
Scaling Policy	Standard
Advanced Warehouse Options ▾	
<input type="button" value="Cancel"/> <input type="button" value="Save Warehouse"/>	

Figure 1-10. Multi-cluster virtual warehouse in Snowflake Web UI

The two different types of modes that can selected for a multi-cluster warehouse are *auto-scale* and *maximized*. The Snowflake scaling policy, designed to help control the usage credits in the auto-scale mode, can be set to *standard* or *economy*.

Whenever a multi-cluster warehouse is configured with the scaling policy set as standard, the first warehouse immediately starts when a query is queued, or the Snowflake system detects that there is one more query than the currently-running clusters

can execute. Each successive warehouse starts 20 seconds after the prior warehouse has started.

If a multi-cluster warehouse is configured with the scaling policy set as economy, a warehouse starts only if the Snowflake system estimates the query load can keep the warehouse busy for at least six minutes. The goal of the economy scaling policy is to conserve credits by keeping warehouses fully loaded. As a result, queries may end up being queued and could take longer to complete.

It is recommended to set the MAXIMUM value as high as possible, while being aware of the associated costs. For example, if you set the MAXIMUM at 10, keep in mind you could experience a tenfold compute cost for the length of time all 10 clusters are running. A multi-cluster warehouse is “maximized” when the MINIMUM is greater than 1 and both the MINIMUM and MAXIMUM values are equal. We'll see an example of that in the next section.



Compute can be scaled up, down, in, or out. In all cases, there is no effect on storage used.

Creating and Using Virtual Warehouses

Commands for virtual warehouses can be executed in the Web UI or within a worksheet by using SQL. We'll first take a look at creating and managing virtual warehouses with SQL. Next, we'll take a look at the Web UI functionality for virtual warehouses.

Auto-suspend and auto-resume are two options available when creating a Snowflake virtual warehouse. Auto-suspend is the number of seconds that the virtual warehouse will wait if no queries need to be executed before going offline. Auto-resume will restart the virtual warehouse once there is an operation that requires compute resources.

The following SQL script will create a medium virtual warehouse, with four clusters, that will automatically suspend after 5 minutes. The virtual warehouse will immediately resume when queries are executed.

```
USE ROLE SYSADMIN;
CREATE WAREHOUSE WH_CH2 WITH WAREHOUSE_SIZE = MEDIUM Auto_suspend = 300 Auto_resume = true Initially_suspended
```



Unless the Snowflake virtual warehouse is created initially in a suspended state, the initial creation of a Snowflake virtual warehouse could take time to provision compute resources.

Earlier, we discussed how we can scale virtual warehouses up or down and that doing so is a manual process. In order to scale up or down, i.e., change the size of a virtual warehouse, we will use the “Alter” command.

```
USE ROLE SYSADMIN;
ALTER WAREHOUSE WH_CH2
SET
    WAREHOUSE_SIZE = LARGE;
```

Any SQL statements executed in this workbook after creating this virtual warehouse will run on that virtual warehouse. If you prefer to use a certain warehouse to execute a script instead, then you can specify that warehouse in the worksheet.

```
USE WAREHOUSE WH_CH2;
```

Alternatively, you can update the warehouse field in the context menu located on the left by selecting Compute then Warehouse ([Figure 1-11](#)).

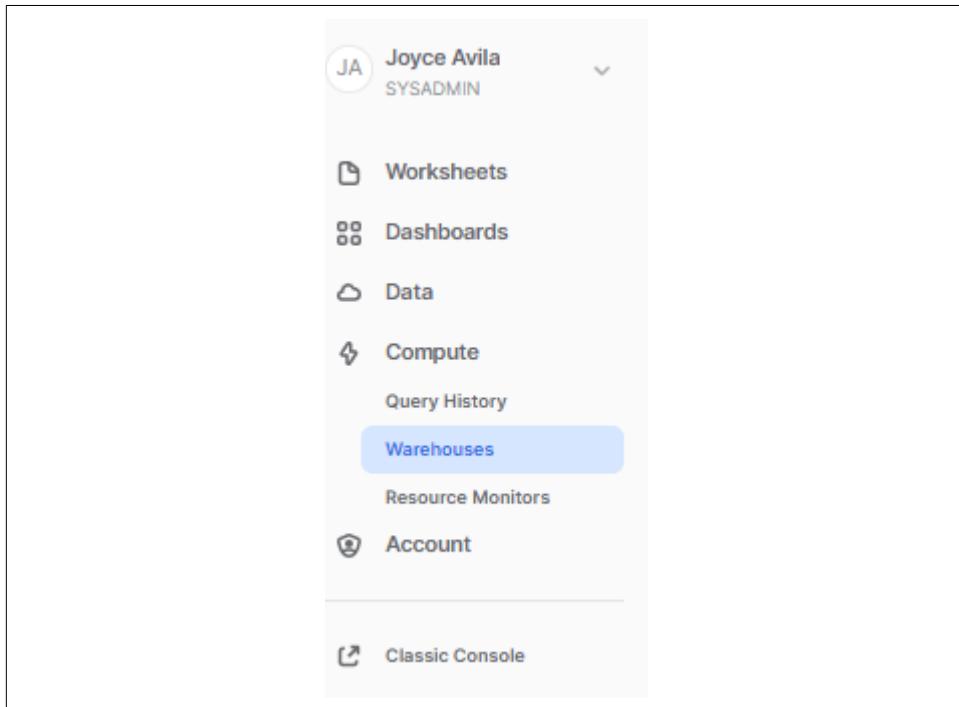


Figure 1-11. Snowflake Web UI Warehouses selection

Once you select Warehouses from the sub-menu, you’ll see the list of available virtual warehouses, their status, size, clusters, and more information ([Figure 1-12](#)).

Warehouses			
1 Warehouse			
NAME ↑	STATUS	SIZE	CLUSTERS
COMPUTE_WH	Suspended	X-Small	min: 1, max: 1

Figure 1-12. Snowflake Web UI detail in the Warehouses selection

Look to the far right, click on the ellipses and then select edit (Figure 1-13)

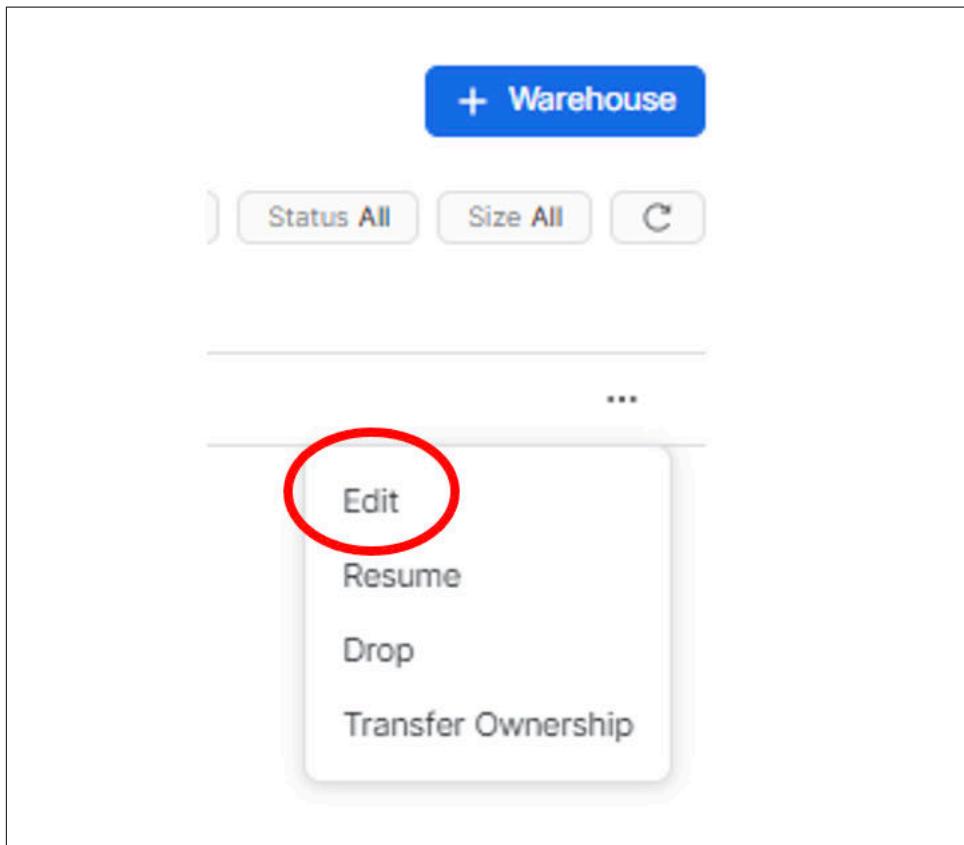


Figure 1-13. Snowflake Web UI Edit Warehouses selection

You should now see the Edit Warehouse screen as shown in Figure 1-14.

Edit Warehouse

COMPUTE_WH as SYSADMIN

Name	Size <small>?</small>
COMPUTE_WH	X-Small 1 credit/hour <small>▼</small>
Comment (optional)	
Multi-cluster Warehouse	
Scale compute resources as query needs change <input checked="" type="checkbox"/>	
Mode	Maximized <small>▼</small>
Clusters	1 <small>▼</small>
Advanced Warehouse Options <small>^</small>	
Auto Resume	<input checked="" type="checkbox"/>
Auto Suspend	<input checked="" type="checkbox"/>
Suspend After (min)	10

Cancel **Save Warehouse**

Figure 1-14. Snowflake Web UI Edit Warehouse

Rather than editing an existing virtual warehouse, you could choose to create a new virtual warehouse. While in the Warehouses sub-section of the Compute menu, you can add a new virtual warehouse by clicking on the button as shown in [Figure 1-15](#).



Figure 1-15. Snowflake Web UI to add a new virtual warehouse

A multi-cluster virtual warehouse can be easily created in the Web UI as well as with SQL code in the Worksheet. Note that for Enterprise Edition, Business Critical Edition, and Virtual Private Snowflake Editions, multi-cluster warehouses are enabled. You are able to add a multi-cluster warehouse by toggling on that option, as shown in [Figure 1-16](#).

New Warehouse

Creating as SYSADMIN

Name	Size
<input type="text"/>	Medium 4 credits/hour
Comment (optional)	<input type="text"/>
Multi-cluster Warehouse	
Scale compute resources as query needs change	
Mode	Auto-scale
Min Clusters	1
Max Clusters	1
Scaling Policy	Standard
Advanced Warehouse Options ^	
Auto Resume	
Auto Suspend	
Suspend After (min)	10

Create Warehouse

Figure 1-16. Creating a Snowflake multi-cluster virtual warehouse

To create a Snowflake multi-cluster virtual warehouse, you'll need to specify the scaling policy as well as the minimum and maximum number of clusters. As stated previ-

ously, the scaling policy, which applies only if the warehouse is running in Auto-scale mode, can be either economy or standard.



A multi-cluster virtual warehouse is said to be maximized when the minimum number of clusters and maximum number of clusters are the same. Additionally, value(s) must be more than one. An example of a maximized multi-cluster virtual warehouse is `MIN_CLUSTER_COUNT = 3 MAX_CLUSTER_COUNT = 3`.

Separation of Workloads and Workload Management

Query processing tends to slow down when the workload reaches full capacity on traditional database systems. In contrast, Snowflake estimates resources needed for each query and as the workload approaches 100%, each new query is suspended in a queue until there are sufficient resources to execute them. Handling the queues can be accomplished in multiple ways. One way is to separate the workloads by assigning different warehouses to different users or groups of users (Figure 1-17). Another way is to take advantage of multi-cluster warehouses and their ability to automatically scale in and out (Figure 1-11). These two approaches are not mutually exclusive.

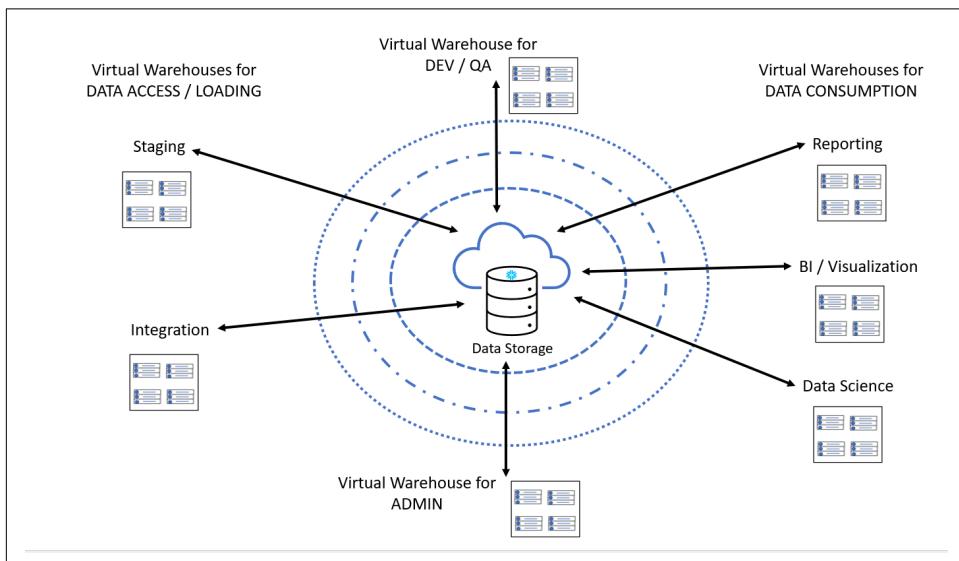


Figure 1-17. Separation of Snowflake workloads by assigning different warehouses to groups of users

Different groups of users can be assigned to different Snowflake virtual warehouses of varying sizes. Thus, users who are querying the data will experience an average query time that is consistent. Marketing and Sales can create and evaluate campaigns while

also capturing sales activities. Accounting and Finance departments can access their reports and delve into the details of the underlying data. Data scientists can run large complex queries on vast amounts of data. And ETL processes can continuously load data.

We learned earlier in the chapter that multi-cluster warehouses can be set to automatically scale to avoid concurrency problems. For an automatically scaling multi-cluster warehouse, we will still need to define the warehouse size and the minimum and maximum number of clusters. Previously, we saw how to create a new warehouse through the Snowflake UI. Now let's use SQL to create a multi-cluster virtual warehouse for our Accounting and Finance, then take a look at an example of how auto-scaling for that warehouse might work.

```
CREATE WAREHOUSE ACCOUNTING WITH Warehouse_Size = MEDIUM MIN_CLUSTER_COUNT = 1 MAX_CLUSTER_COUNT = 6
```

The scaling process occurs automatically once the multi-cluster warehouse is configured. [Figure 1-18](#) illustrates how auto-scaling works when the number of concurrent SQL statements increase.

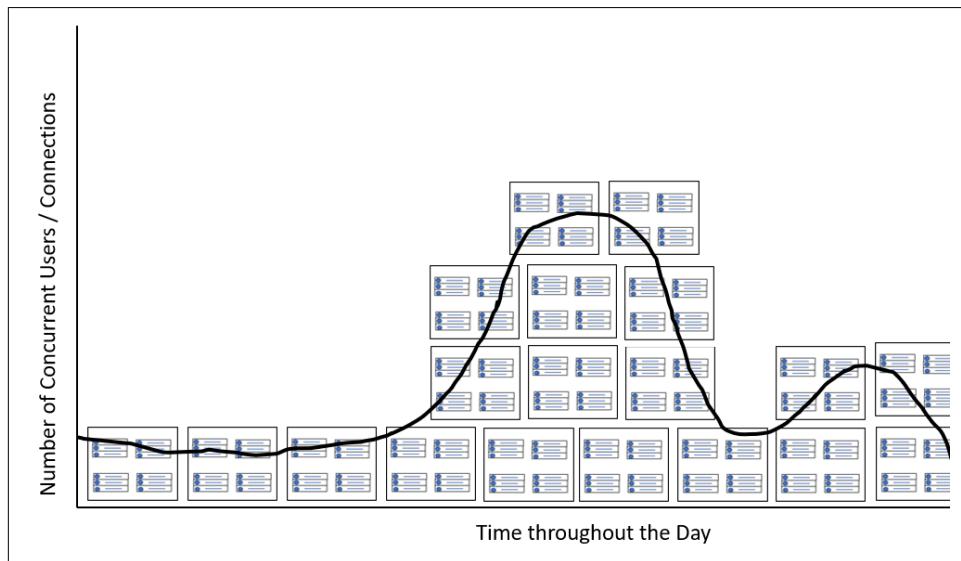


Figure 1-18. Management of Snowflake workloads by using multi-cluster warehouses to scale in and out

You can see that on an hourly basis, the workload is heavier between core working hours for employees. We might also want to investigate to confirm that the daily the workload is heavier overall at the beginning of the month for the Consumption virtual warehouse for Reporting, as the accounting department works to prepare and review the accounting statements for the prior month.

Billing for Virtual Warehouse Layer

Consumption charges for Snowflake virtual warehouses are calculated based on the warehouse size, as determined by the number of servers per cluster, the number of clusters if there are multi-cluster warehouses, and the amount of time each cluster server runs. Snowflake utilizes per-second billing with a 60-second minimum each time a warehouse starts or is resized. When a warehouse is scaled up, credits are billed for one minute of the additional resources that are provisioned. All billing, even though calculated in seconds, is reported in fractions of hours.

When using the ACCOUNTADMIN role, you can view the warehouse credit usage for your account by clicking on Account > Usage in the UI. You can also query the Account Usage view in the SNOWFLAKE shared database to obtain the information. It is recommended that you choose an XS (extra small) warehouse to do so because of the small size of the data set and simplicity of the query.

Centralized (Hybrid-Columnar) Database Storage Layer

Snowflake's centralized database storage layer holds all data, including structured and semi-structured data. As data is loaded into Snowflake, it is optimally reorganized into a compressed, columnar format and stored and maintained in Snowflake databases. Each Snowflake database consists of one or more schemas, which is a logical grouping of database objects such as tables and views. Chapter 3 is entirely devoted to showing you how to create and manage databases and database objects. In Chapter 9, we will learn about Snowflake's physical data storage as we take a deep dive into micro partitions to better understand data clustering.

Data stored in Snowflake databases is always compressed and encrypted. Snowflake takes care of managing every aspect of how the data is stored. Snowflake automatically organizes stored data into micro-partitions, an optimized immutable compressed columnar format, which is encrypted using AES-256 encryption. Snowflake optimizes and compresses data to make metadata extraction and query processing easier and more efficient. We learned earlier in the chapter that whenever a user submits a Snowflake query, that query will be sent to the cloud services optimizer before being sent to the compute layer for processing.

Snowflake's data storage layer is sometimes referred to as the Remote Disk layer. The underlying file system is implemented on Amazon, Microsoft, or Google Cloud ([Figure 1-19](#)). The specific provider used for data storage is the one you selected when you created your Snowflake account. Snowflake doesn't place limits on the amount of data you can store or on the number of databases or database objects that you can create. Snowflake tables can easily store petabytes of data. There is no effect on virtual warehouse sizes as the storage increases or decreases in a Snowflake

account. The two are scaled independently from each other and from the cloud services layer.

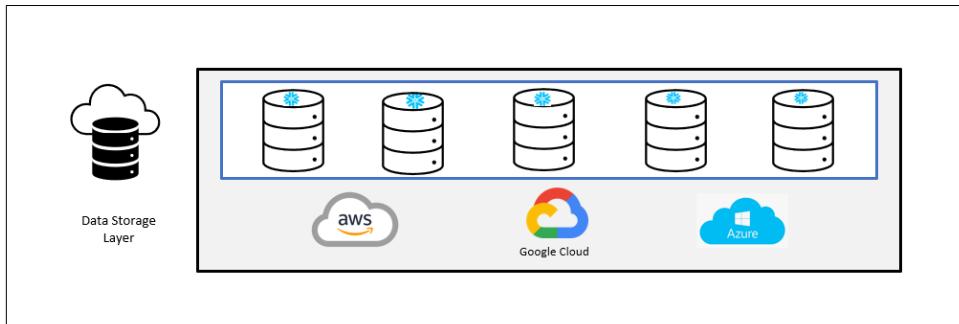


Figure 1-19. Snowflake Data Storage Layer

There are two unique features in the storage layer architecture – time travel and zero-copy cloning. Both very powerful Snowflake features will be introduced in this chapter and will be covered in more detail in later chapters. To prepare for those later chapters, you'll want to have a thorough understanding of these two features.

Introduction to Zero Copy Cloning

Zero-copy cloning offers the user a way to “snapshot” a Snowflake database, schema, or table along with its associated data. There is no additional storage charge until changes are made to the cloned object because zero-copy data cloning is a metadata-only operation. For example, if you clone a database and then add a new table or delete some rows from a cloned table, at that point then there would be storage charges assessed. There are many uses for zero-copy cloning other than creating a backup. Most often, zero-copy clones will be used to support development and test environments. We'll see examples of this in Chapter 8.

Introduction to Time Travel

Time travel allows you to restore a previous version of a database, table, or schema. This is an incredibly helpful feature that gives you an opportunity to fix previous edits incorrectly done or to restore items deleted in error. With time travel, you can also back up data from different points in the past by combining the time travel feature with the clone feature, or you can perform a simple query of a database object that no longer exists. How far back you can go into the past depends on a few different factors. Time travel will be discussed in detail in Chapter 7. For the purposes of this chapter, it is important to note that there will be data storage fees assessed for any data that has been deleted but is still available to restore.

Billing for Storage Layer

Snowflake data storage costs are calculated based on the daily average size of compressed rather than uncompressed, data. Storage costs include the cost of persistent data stored in permanent tables and files staged for bulk data loading and unloading. Fail-safe data and the data retained for data recovery using time travel are also considered in the calculation of data storage costs. Clones of tables referencing data that has been deleted are similarly considered.

Snowflake Caching

When you submit a query, Snowflake checks to see if that query has been previously run and, if so, whether the results are still cached. Snowflake will use the cached result set if it is still available rather than executing the query you just submitted. In addition to retrieving the previous query results from a cache, Snowflake supports other caching techniques. There are three Snowflake caching types; query results cache, virtual warehouse cache, and metadata cache.

Query Results Cache

The fastest way to retrieve data from Snowflake is by using the query results cache. The results of a Snowflake query are cached, or persisted, for 24 hours and then purged. This contrasts with how the warehouse cache and metadata cache work. Neither of those two caches are purged based on a timeline. Even though the results cache only persists for 24 hours, the clock is reset each time the query is re-executed up to a maximum of 31 days from the date and time when the query was first executed. After 31 days, or sooner if the underlying data changes, a new result is generated and cached when the query is submitted again.

The results cache is fully managed by the Snowflake global cloud services (GCS) layer, as shown in [Figure 1-20](#), and is available across all virtual warehouses since virtual warehouses have access to all data. The process for retrieving cached results is managed by GCS. However, once the size of the results exceeds a certain threshold, the results are stored in and retrieved from cloud storage.

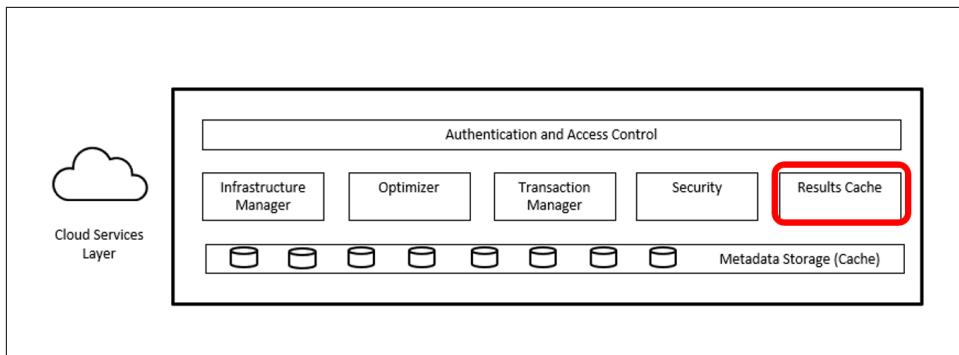


Figure 1-20. Results Cache in the Snowflake Cloud Services Layer

Query results returned to one user are also available to any user who has the necessary access privileges and who executes the same query. Therefore, any user can run a query against the result cache with no running virtual warehouse needed, assuming the query is cached and the underlying data has not changed.

Another unique feature of the query results cache is that it is the only cache that can be disabled by a parameter.

```
ALTER SESSION SET USE_CACHED_RESULT=FALSE;
```

Disabling the result cache is necessary to do before performing A/B testing and it is important to enable query result caching once the testing is complete.

Metadata Cache

The metadata cache is fully managed in the global services layer ([Figure 1-21](#)) where the user does have some control over the metadata but no control over the cache.

Snowflake collects and manages metadata about tables, micro-partitions, and even clustering. For tables, Snowflake stores row count, table size in bytes, file references and table versions. Thus, a running warehouse will not be needed because the count statistics are kept in the metadata cache when running a `SELECT COUNT(*)` on a table.

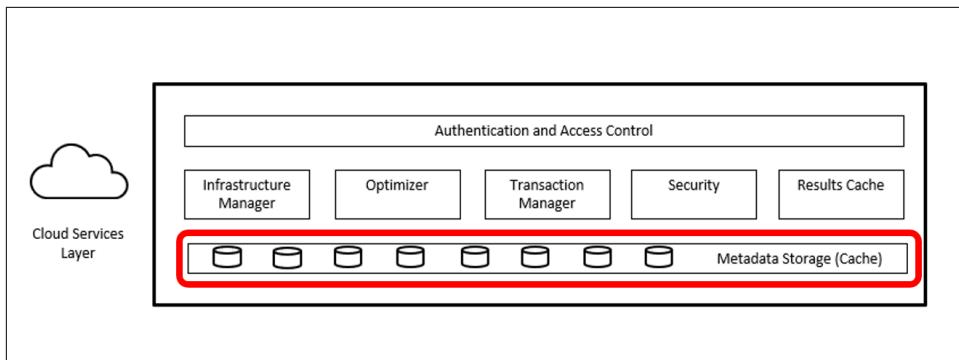


Figure 1-21. Metadata Storage Cache in the Snowflake Cloud Services Layer

The Snowflake metadata repository includes table definitions and references to the micro-partition files for that table. The range of values in terms of MIN and MAX, the NULL count, and the number of distinct values are captured from micro-partitions and stored in Snowflake. As a result, any queries which return the MIN or MAX value, for example, will not need a running warehouse. Snowflake stores the total number of micro-partitions and the depth of overlapping micro-partitions to provide information about clustering.



The information stored in the metadata cache is used to build the query execution plan.

Virtual Warehouse Local Disk Cache

The traditional Snowflake data cache is specific to the virtual warehouse used to process the query. Running virtual warehouses use SSD storage to store the micro-partitions that are pulled from the centralized database storage layer when a query is processed. This is necessary to complete the query requested, whenever a query is executed. The size of the warehouse's SSD cache is determined by the size of the virtual warehouse's compute resources ([Figure 1-22](#)). Whenever a virtual warehouse receives a query to execute, that warehouse will scan the SSD cache first before accessing the Snowflake remote disk storage. Reading from SSD is faster than from the database storage layer but still requires the use of a running virtual warehouse.

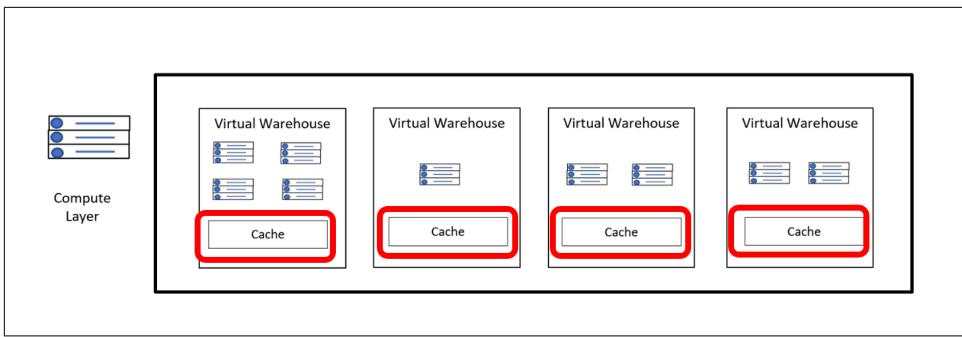


Figure 1-22. Virtual Warehouse Cache in the Snowflake Cloud Services Layer

Although the warehouse cache is implemented in the virtual warehouse layer where each virtual warehouse operates independently, the global services layer handles the overall system data freshness. It does so via the query optimizer which checks the freshness of each data segment of the assigned warehouse and then builds a query plan to update any segment by replacing it with data from the remote disk storage.

Note that the virtual warehouse cache is sometimes referred to as the “raw data cache”, the “SSD cache”, or the “data cache.” This cache is dropped once the virtual warehouse is suspended, so you’ll want to consider the trade-off between the credits that will be consumed by keeping a warehouse running versus the value from maintaining the cache of data from previous queries to improve performance. By default, Snowflake will automatically suspend a virtual warehouse after 10 minutes of idle time, but this can be changed.



Whenever possible, and where it makes sense, assign the same virtual warehouse to users who will be accessing the same data for their queries. This increases the likelihood that they will benefit from the virtual warehouse local disk cache.

Get Ready for Hands-On Learning!

Hopefully these first two chapters have given you an understanding of the power of the Snowflake Data Cloud and its simplicity of use. In the upcoming chapters, we’ll be demonstrating how Snowflake works by deep diving into hands-on learning examples throughout each of the chapters. If you haven’t already done so, now is a good time to sign up for a Snowflake free trial account. Refer to Chapter 1 for more details on getting set up in a trial account.

Exercises to Test Your Knowledge

The following exercises are based on the coverage in this chapter.

1. Name the three Snowflake architecture layers.
2. Which of the three Snowflake layers are multi-tenant?
3. In which of the three Snowflake architecture layers will you find the warehouse cache? the result cache?
4. If you are experiencing higher than expected costs for Snowflake cloud services, what kinds of things might you want to investigate?
5. Explain the difference between scaling up and scaling out.
6. What effect does scaling up or scaling out have on storage used in Snowflake?
7. Shared-nothing architecture evolved from shared-disk architecture. NoSQL alternatives have also been created. What one main problem have they all been trying to solve?
8. In a Snowflake multi-cluster environment, what scaling policies can be selected?
9. What components do you need to configure specifically for multi-cluster warehouses?
10. What are two options to change the warehouse that will be used to run a SQL command within a specific worksheet?

Solutions to these exercises are available in Appendix A.

Creating and Managing Snowflake Architecture Objects

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 3rd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

Within Snowflake, all data is stored in database tables logically structured in collections of rows and columns. This chapter focuses on the logical structure of databases and database objects, such as tables and views.

In this chapter, we will cover topics in specific order because the series of examples in each topic build upon each other. The code is provided for you here in the chapter as well as on Github. These are the topics we will work through together:

1. Databases
2. Schemas
3. Information Schema and Account Usage
4. Tables
5. Views

6. Stages, File Format Included
7. Stored Procedures, Task Included
8. UDFs
9. Pipes, streams, and sequences
10. Code Cleanup and Test Your Knowledge

Our initial discussion here of databases, tables, views, stages, pipes, and streams lay the foundation for following chapters where we do a deeper dive into these objects. We will also conduct a deep dive for the User Defined Function (UDF) and Stored Procedure objects. One advanced deep dive example includes using a file format object and another example uses a task object. Pipes, streams, and sequences are briefly introduced and covered in more detail in later chapters.

Creating and Managing Snowflake Databases

In the relational world, database objects such as tables, views, and more, are maintained within databases. In Snowflake, the database logically groups the data while the schema organizes it. Together, the database and schema comprise the *namespace*. In the examples throughout this chapter, whenever we work with database objects, we'll need to specify a namespace unless the schema and database we want to use are the active context in the workspace. If the database or schema needs to be specified, we'll include the "USE" command. That way, it is clear to Snowflake the location where objects are to be created or which specific object is being referenced in the commands.

There are two main types of databases we can create – permanent (persistent) and transient databases. At the time we create a database, the default will be a permanent database, if we don't specify which of the two types we want to create.



Snowflake is designed so that your data is accessible and recoverable at every stage within the data lifecycle. This is achieved through Continuous Data Protection (CDP), Snowflake's comprehensive set of features that help protect data stored in Snowflake against human error, malicious acts, and software or hardware failure. The important Snowflake CDP features introduced in this chapter are time travel and fail-safe.

Transient databases have a maximum one-day data retention period, aka *time-travel* period, and do not have a *fail-safe* period.

The Snowflake time travel period is the time during which table data within the database can be queried at a point in time. This also enables databases and database objects to be cloned or "undropped" and historical data to be restored. The default

time travel period is one day but can be up to 90 days for permanent databases; or a user could set the time travel period to zero days if no time travel period is desired. Note that Enterprise Edition or up is necessary to take advantage of the 90-day time travel period.

Snowflake's fail-safe data recovery service provides a seven-day period during which data from permanent databases and database objects may be recoverable by Snowflake. The fail-safe data recovery period is the seven-day period *after* the data retention period ends. Unlike time-travel data, which is accessible by Snowflake users, fail-safe data is recoverable *only* by Snowflake employees.

It is important to note that data storage costs are incurred for those seven days. That is one consideration when deciding about the database type you want to create. Another consideration is the ability to recover data from other sources if the data stored in the database is lost after a single data time travel period is up.

These are the basic SQL commands for Snowflake databases that we will be covering in this section:

- CREATE DATABASE
- ALTER DATABASE
- DROP DATABASE
- SHOW DATABASES

CREATE DATABASE is the command used to create a new database, clone an existing database, create a database from a share provided by another Snowflake account, or to create a replica of an existing primary database (i.e., a secondary database).

We can create databases from the User Interface (UI) or by using SQL code in the Snowflake worksheet. We created a database and database objects in Chapter 1 using the Web User Interface. We'll be using SQL commands in the Snowflake Worksheet for this chapter.



For all exercises in this chapter, make sure you have your role set to SYSADMIN throughout the chapter unless otherwise directed.

Let's go ahead and get started. We'll create one permanent database and one transient database.

```
USE ROLE SYSADMIN;  
CREATE  
OR REPLACE DATABASE CHAPTER3_PDB1 Comment = "Permanent Database 1 used for Exercises in Definitive Guide to Snowflake"
```

```
CREATE  
OR REPLACE TRANSIENT DATABASE CHAPTER3_TDB1 Comment = "Transient Database 1 used for Exercises in Definitive Guide"
```

Notice above that we used the words “OR REPLACE” optional keyword in the command. That way, an error would not be returned if the database already exists, though the existing database would be completely overwritten.



Use the CREATE OR REPLACE statement sparingly in production so as not to overwrite an existing database. As an alternative, if you don't want to have an error but also don't want to overwrite an existing database, use the CREATE DATABASE CHAPTER3_PDB1 IF NOT EXISTS; statement instead. The OR REPLACE and IF NOT EXISTS cannot both be used in the same statement.

Whenever you create a new database, that database is automatically set as the active database for the current session. It's the equivalent of using the “USE DATABASE” command. If we needed or wanted to use a database, other than the one we just created, we'd have to include the “USE DATABASE” command in the worksheet to select the appropriate database.

If you navigate to the Databases UI, as shown in [Figure 2-1](#), you will see that it shows the two databases we just created plus three out of the four databases that automatically come with the Snowflake account.

The screenshot shows the Snowflake web interface for managing databases. At the top, there is a navigation bar with icons for Databases, Shares, Data Marketplace, Warehouses, Worksheets, and History. Below the navigation bar, the title 'Databases' is displayed. There are buttons for 'Create...', 'Clone...', 'Drop...', and 'Transfer Ownership'. A search bar shows 'Search Databases' and indicates '5 databases'. A table lists the databases:

Database	Origin	Creation Time	Owner	Comment
CHAPTER3_PDB1		8:37 AM	SYSADMIN	Permanent Database 1 used for Exercises in Definitive Guide
CHAPTER3_TDB1		8:34 AM	SYSADMIN	Transient Database 1 used for Exercises in Definitive Guide
SNOWFLAKE_SAMPLE_DATA	SFC_SAMPLES.SA...	4/10/2021, 10:39 AM	ACCOUNTADMIN	TPC-H, OpenWeatherMap, etc
DEMO_DB		4/10/2021, 10:39 AM	SYSADMIN	demo database
UTIL_DB		4/10/2021, 10:39 AM	SYSADMIN	utility database

Figure 2-1. Web User Interface displaying a list of active databases, based on the user's SYSADMIN role



By default, the SNOWFLAKE database is shown only to those using the ACCOUNTADMIN role. However, that privilege can be granted to other roles.

Let's switch back to the Worksheets and change our role to ACCOUNTADMIN and see if we can view the SNOWFLAKE database. Once in the worksheet:

```
USE ROLE ACCOUNTADMIN;
SHOW DATABASES;
```

Notice in [Figure 2-2](#) that all databases, including the ones we just created, have a 1-day retention time. Data retention time (in days) is the same as the time travel number of days and specifies the number of days for which the underlying data is retained after deletion, and for which “CLONE” and “UNDROP” commands can be performed on the database.

Row	created_on	name	is_default	is_current	origin	owner	comment	options	retention_time
1	2021-04-25 06:37:59.72...	CHAPTER3_PDB1	N	Y		SYSADMIN	Permanent Database 1 use...		1
2	2021-04-25 06:48:57:60...	CHAPTER3_TDB1	N	N		SYSADMIN	Transient Database 1 use...	TRANSIENT	1
3	2021-04-10 08:39:08:54...	DEMO_DB	N	N		SYSADMIN	demo database		1
4	2021-04-10 08:39:09:24...	SNOWFLAKE	N	N	SNOWFLAKE.ACCOUNT...				1
5	2021-04-10 08:39:09:20...	SNOWFLAKE_SAMPLED_D...	N	N	SFC_SAMPLES.SAMPLE_...	ACCOUNTADMIN	TPC-H, OpenWeatherMa...		1
6	2021-04-10 08:39:05:33...	UTIL_DB	N	N		SYSADMIN	utility database		1

Figure 2-2. Worksheet displaying a list of active Databases, based on the user’s ACCOUNTADMIN role

We can change the data retention time for a permanent database but not for a transient one. We can change the retention time up to 90 days for permanent databases. We’ll go ahead and change the retention time for our permanent database to 10 days by using the “ALTER DATABASE” command. Be sure to change your role back to SYSADMIN before issuing the commands.

```
USE ROLE SYSADMIN;
ALTER DATABASE CHAPTER3_PDB1
SET
    DATA_RETENTION_TIME_IN_DAYS = 10;
```

If you attempt to change the data retention time for a transient database, you’ll receive an error telling you that the value “10” is an invalid value for the parameter. That is because a transient database can have a maximum 1-day data retention. You could change the retention time to 0 days but then you wouldn’t be able to “CLONE” or “UNDROP” that database if you do that.

We’ll be covering tables in more detail in a later section, but for now, it is important to mention a few things about tables as they relate to permanent versus transient databases.

Snowflake uses a hybrid approach when it comes to permanent databases but not transient databases. A permanent database type is not limited to the different types of objects that can be stored within them. For example, you can store transient tables within a permanent database but not permanent tables within a transient database. As a reminder, transient tables are designed to hold transitory data that doesn’t need

the same level of protection and recovery as permanent tables but does still need to be maintained beyond a session.

Below is an example of creating a table in our transient database.

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE TABLE "CHAPTER3_TDB1"."PUBLIC"."SUMMARY" (
    CASH_AMT number,
    RECEIVABLES_AMT number,
    CUSTOMER_AMT number
);
```

Notice we didn't specify the type of table as either permanent or transient. By default, Snowflake creates a permanent table unless you indicate otherwise when you are creating the table. The exception would be when you are creating a table within a transient database. In that case, the table would also have to be transient. By default, all tables created in a transient schema are transient. We can see that is the case by using the "SHOW TABLES" command which gives us the following result, shown in [Figure 2-3](#), for the table we just created.

Row	created_on	name	database_name	schema_name	kind
1	2021-04-25 ...	TABLE1	CHAPTER3_TDB1	PUBLIC	TRANSIENT

Figure 2-3. Worksheet results of the "SHOW TABLES" command



There is no limit to the number of database objects, schemas, and databases that can be created within a Snowflake account.

Each Snowflake account also comes with certain databases, schemas, and tables already included as shown in [Figure 2-4](#) below. As shown, there are four databases that initially come with a Snowflake account:

- SNOWFLAKE database
- UTIL_DB database
- DEMO_DB database

- SNOWFLAKE_SAMPLE_DATA database

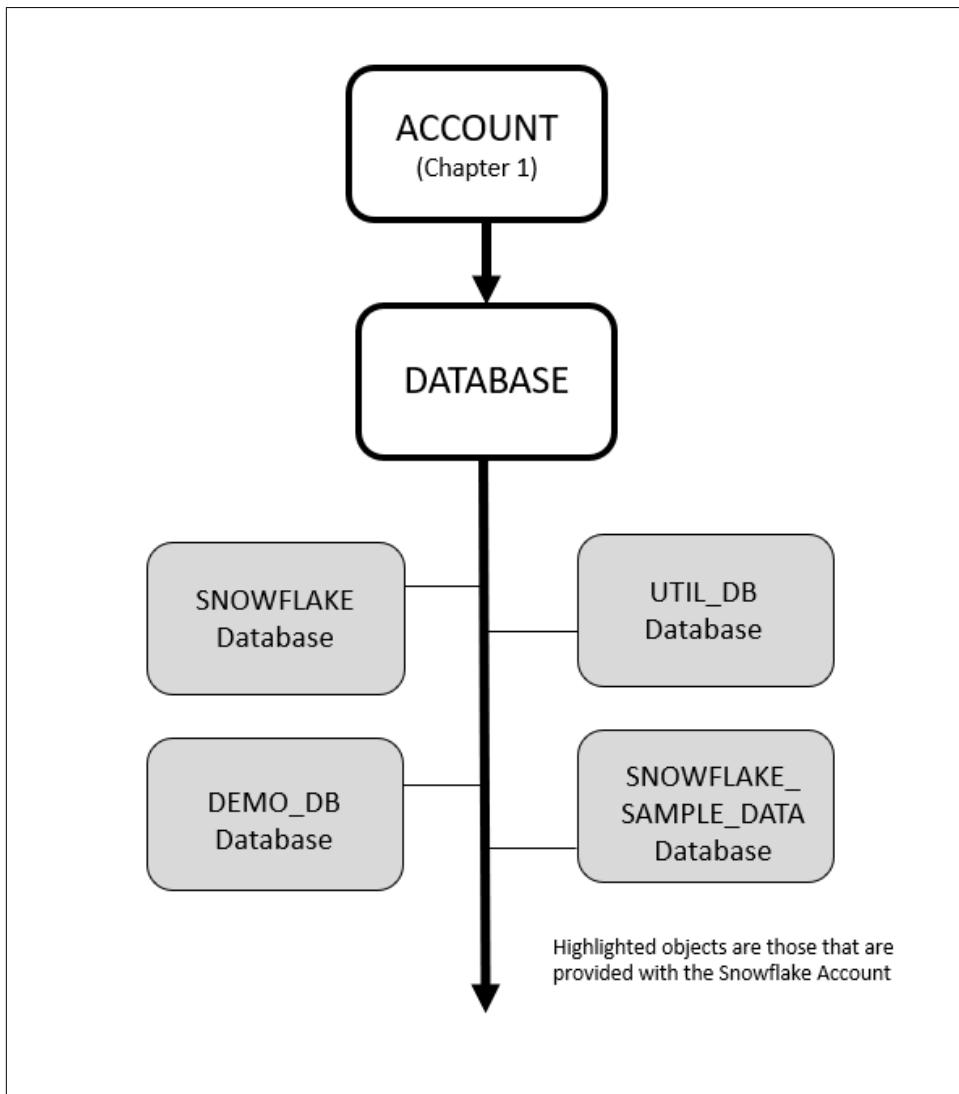


Figure 2-4. Objects Hierarchy for Snowflake Database

The SNOWFLAKE database is owned by Snowflake Inc. and is a system-defined, read-only shared database which provides object metadata and usage metrics about your account. Unlike the other three databases imported into your account at the time of setup, the SNOWFLAKE database cannot be deleted from your account.

The UTIL_DB database and DEMO_DB database, also imported into your account at the time of setup, contains no data and, as such, there are no storage charges for those databases, and they can be dropped at any time.

Upon first look, the SNOWFLAKE_SAMPLE_DATA database appears to be something similar to what you might create in your Snowflake account. However, the sample database is actually one that has been shared from the Snowflake SFC_SAMPLES account and the database is read-only in your account which means that no DDL commands can be issued. In other words, database objects cannot be added, dropped, or altered within the sample database. In addition, no DML commands for actions such as cloning can be performed on the tables. You can, however, view the sample database and execute queries on the tables.

We'll be using the SNOWFLAKE_SAMPLE_DATA database in some of our examples in this chapter. In Chapter 10, we'll be learning about shared databases but, for now, what is important to know is that while we don't incur any storage costs for the shared sample database, we do need a running warehouse to run queries and so there will be an associated compute cost for running those queries on the Snowflake sample database.

Even though it may be obvious, one final important consideration for how to architect your solution and which objects to choose for storing data is that there is a monetary cost for storing data in Snowflake and there are also performance implications.

This chapter is intended to give you the necessary understanding of Snowflake databases and database objects as a foundation for later chapters which will address more complex topics such as improving performance and reducing costs, data recovery, and data loading and unloading.

Creating and Managing Snowflake Schemas

When we created databases, we didn't have to specify the account because we can only operate in one account at a time. But when we create a schema, we need to let Snowflake know which database we want to use. If we don't specify a particular database, then Snowflake will use the one that is active.

Just like databases, schemas can be either permanent or transient with the default being permanent. Just like databases, we have available the same SQL commands. However, for schemas, we have something unique called a managed access schema. In a managed access schema, the schema owner manages grants on the objects within a schema, such as tables and views, but doesn't have any of the USAGE, SELECT, or DROP privileges on the objects.

There are different ways to create a schema that will achieve the same result. Here below are two examples that accomplish the same thing. In this first example, the “USE” command lets Snowflake know for which database the schema will be created.

```
USE ROLE SYSADMIN;
USE DATABASE CHAPTER3_PDB1;
CREATE
OR REPLACE SCHEMA BANKING;
```

In the second example, we simply use the *fully qualified schema name*.

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE SCHEMA CHAPTER3_PDB1.BANKING;
```

If we use the “SHOW SCHEMA” command as demonstrated in [Figure 2-5](#), we notice that the retention time of the new schema also has a retention time of 10 days, just like the database in which it was created, rather than the default 1-day retention.

Row	created_on	name	is_default	is_current	database_name	owner	comment	options	retention_time
1	2021-04-25 09:24:25.191...	INFORMATION_SCHEMA	N	N	CHAPTER3_PDB1	SYSADMIN	Views describing the con...	10	
2	2021-04-25 09:37:59.73...	PUBLIC	N	N	CHAPTER3_PDB1	SYSADMIN		10	
3	2021-04-25 09:22:58.94...	SOHEMA1	N	Y	CHAPTER3_PDB1	SYSADMIN		10	

Figure 2-5. Worksheet results of the “SHOW SCHEMA” command

However, we can always change the retention time to one day for the schema.

```
USE ROLE SYSADMIN;
ALTER SCHEMA CHAPTER3_PDB1.BANKING
SET
    DATA_RETENTION_TIME_IN_DAYS = 1;
```

Now, run the “SHOW SCHEMAS” command again and you’ll see the retention time has been changed for the schema.

If we’d like to create a schema with managed access, we need to add the “WITH MANAGED ACCESS” command.

```
USE ROLE SYSADMIN;
USE DATABASE CHAPTER3_PDB1;
CREATE
OR REPLACE SCHEMA MSCHEMA WITH MANAGED ACCESS;
```

Now, when you run the “SHOW SCHEMAS” command, you’ll notice that “Managed Access” will be displayed under the options column for the schema named MSCHEMA.

As discussed in Chapter 5, for regular schemas the object owner role can grant object access to other roles and can also grant those roles the ability to manage grants for the object. However, in managed access schemas, object owners are unable to issue grant privileges. Instead, only the schema owner or a role with the MANAGE GRANTS privilege assigned to it can manage the grant privileges.



The SECURITYADMIN and ACCOUNTADMIN inherently have the “MANAGE GRANTS” privilege. Thus, both roles can manage the grant privileges on all managed schemas.

There are two database schemas, as shown in [Figure 2-6](#), that are included in every database that is created: INFORMATION_SCHEMA and PUBLIC. The PUBLIC schema is the default schema and can be used to create any other objects whereas the INFORMATION_SCHEMA is a special schema for the system that contains views and table functions that provide access to the metadata for the database and account. The Information Schema will be discussed in the next section.

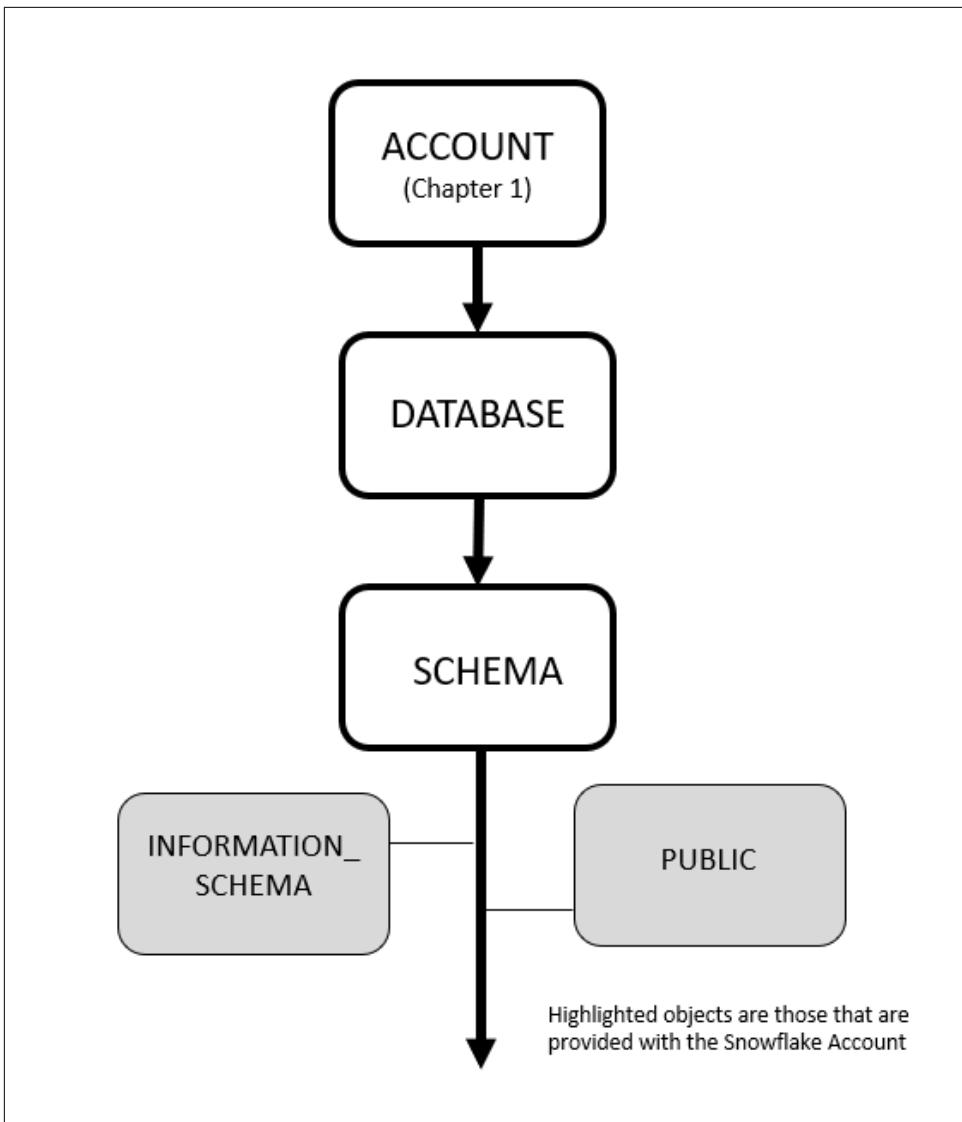


Figure 2-6. Objects Hierarchy for Snowflake Schema

INFORMATION_SCHEMA and Account Usage

As we just learned, the Snowflake INFORMATION_SCHEMA is included within every database created in Snowflake. The Information Schema, also known as the “Data Dictionary”, includes metadata information about the objects within the database as well as account-level objects like roles. In Chapter 8, we’ll explore Information Schema table functions, which can be used to return historical information and

account usage information. For now, we'll be exploring Information Schema Database Views in a little more detail.

More than 20 system-defined views are included in every Information Schema and can be divided between account views and database views.

Information Schema Account Views:

- **Applicable_Roles**: Displays one row for each role grant
- **Databases**: Displays a row for each database defined in your account
- **Enabled_Roles**: Displays a row for each currently-enabled role in the session
- **Information_Schema_Catalog_Name**: the name of the database in which the information_schema resides
- **Load_History**: Displays one row for each file loaded into tables using the COPY INTO <table> command. Returns history for past 14 days except no history for data loaded using Snowpipe,
- **Replication_Databases**: Displays a row for each primary and secondary database (i.e., database for which replication has been enabled) in your organization

You may want to look at what is within each of these views. You'll notice that for some of them, all views in the account contain the same information. Try the SQL statements below. What do you notice?

```
SELECT
  *
FROM
  "SNOWFLAKE_SAMPLE_DATA"."INFORMATION_SCHEMA"."DATABASES";
SELECT
  *
FROM
  "DEMO_DB"."INFORMATION_SCHEMA"."DATABASES";

SELECT
  *
FROM
  "SNOWFLAKE_SAMPLE_DATA"."INFORMATION_SCHEMA"."APPLICABLE_ROLES";

SELECT
  *
FROM
  "DEMO_DB"."INFORMATION_SCHEMA"."APPLICABLE_ROLES";
```

Information Schema Database Views:

- **Columns:** Displays a row for each column in the tables defined in the specified (or current) database
- **External_Tables:** Displays a row for each external table in the specified (or current) database
- **File_Formats:** Displays a row for each file format defined in the specified (or current) database
- **Functions:** Displays a row for each user-defined function (UDF) or external function defined in the specified (or current) database
- **Object_Privileges:** Displays a row for each access privilege granted for all objects defined in your account.
- **Pipes:** Displays a row for each pipe defined in the specified (or current) database
- **Procedures:** Displays a row for each stored procedure defined in the specified (or current) database
- **Referential_Constraints:** Displays a row for each referential integrity constraint defined in the specified (or current) database
- **Schemata:** Displays a row for each schema in the specified (or current) database,
- **Sequences:** Displays a row for each sequence defined in the specified (or current) database
- **Stages:** Displays a row for each stage defined in the specified (or current) database
- **Table_Constraints:** Displays a row for each referential integrity constraint defined for the tables in the specified (or current) database
- **Table_Privileges:** Displays a row for each table privilege that has been granted to each role in the specified (or current) database
- **Table_Storage_Metrics:** Displays table-level storage utilization information, includes table metadata, and displays the number of storage types billed for each table. NOTE: Rows are maintained in this view until the corresponding tables are no longer billed for any storage, regardless of various states that the data in the tables may be in (i.e. active, Time Travel, Fail-safe, or retained for clones)
- **Tables:** Displays a row for each table and view in the specified (or current) database
- **Usage_Privileges:** Displays a row for each privilege defined for sequences in the specified (or current) database
- **Views:** Displays a row for each view in the specified (or current) database

There are different ways to look at some of the metadata in Snowflake, some of which do use the Information Schema. If you try each of the commands below, you'll see

that there are two ways we can get the information about schemas within the Snowflake sample database.

```
SELECT
  *
FROM
  "SNOWFLAKE_SAMPLE_DATA"."INFORMATION_SCHEMA"."SCHEMATA";
SHOW SCHEMAS IN DATABASE "SNOWFLAKE_SAMPLE_DATA";
```

However, one thing you will notice is that the metadata contained with the INFORMATION_SCHEMA is much more complete with several more columns of information than when you simply use the “SHOW” command.

If you try the following SQL statement, what happens?

```
SELECT
  *
FROM
  "DEMO_DB"."INFORMATION_SCHEMA"."TABLE_PRIVILEGES";
```

You will notice that no rows are returned in the results. The reason is because there are no tables in the database, thus, there will be no table privileges.

The INFORMATION_SCHEMA, one of the two schemas that are included with every Snowflake database, has a great many uses. The Information Schema provides a great deal of information about an account’s object metadata and usage metrics. There is also another place within Snowflake where object metadata and usage metrics are stored.

Account Usage Schema

The SNOWFLAKE database, viewable by the ACCOUNTADMIN by default, includes an ACCOUNT_USAGE schema that is very similar to the INFORMATION_SCHEMA but with three differences. First, the SNOWFLAKE database ACCOUNT_USAGE schema includes records for dropped objects whereas the INFORMATION_SCHEMA does not. The ACCOUNT_USAGE schema also has a longer retention time for historical usage data. Whereas the INFORMATION_SCHEMA has data available ranging from seven days to six months, the ACCOUNT_USAGE view retains historical data for one year. Finally, there is no latency when querying the INFORMATION_SCHEMA but the latency time for ACCOUNT_USAGE could range from 45 minutes to three hours.

One of the common uses for the ACCOUNT_USAGE schema is to keep track of credits used over time by each warehouse in your account (month-to-date):

```
USE ROLE ACCOUNTADMIN;
USE DATABASE SNOWFLAKE;
USE SCHEMA ACCOUNT_USAGE;
SELECT
```

```
start_time::date AS USAGE_DATE,
WAREHOUSE_NAME,
SUM(credits_used) AS TOTAL_CREDITS_CONSUMED
FROM
warehouse_metering_history
WHERE
start_time >= date_trunc(Month, current_date)
GROUP BY
1,
2
ORDER BY
2,
1;
```

The SNOWFLAKE database, which includes the Account Usage schema, is only available to the ACCOUNTADMIN role, unless the ACCOUNTADMIN grants imported privileges from the underlying share to another role. We'll explore the account_usage schema in more detail in future chapters, especially in Chapters 8 and 9 when we learn more about improving performance and reducing costs.

Schema Object Hierarchy

In this section, we've learned about the schema object and explored the two schemas that come with each Snowflake database. A look back at [Figure 2-6](#) shows us the schema object hierarchy that exists above the schema object. Next, we'll want to explore the Snowflake objects below the schema in the hierarchy.

Within a Snowflake schema object, there exists many objects including tables, views, stages, policies, stored procedures, user defined functions, and more. In the next sections, we'll take a closer look at several of these Snowflake objects. While we will explore some of these Snowflake objects in detail, the explanations in this chapter are meant to be foundational; we'll dive deeper into these objects throughout many of the subsequent chapters.

Introduction to Snowflake Tables

As previously mentioned, all Snowflake data is stored in tables. In addition to permanent and transient tables, it is also possible to create temporary and external tables as shown in [Figure 2-7](#). Like database and schemas, we can use the CREATE, ALTER, DROP, and SHOW TABLES commands. In addition, we'll need to use INSERT INTO or COPY INTO to place data in a table. For Snowflake tables, we can also use the TRUNCATE or DELETE command to remove data from a table but not remove the table object itself.



TRUNCATE and DELETE are different in that TRUNCATE also clear table load history metadata, while delete retains the metadata .

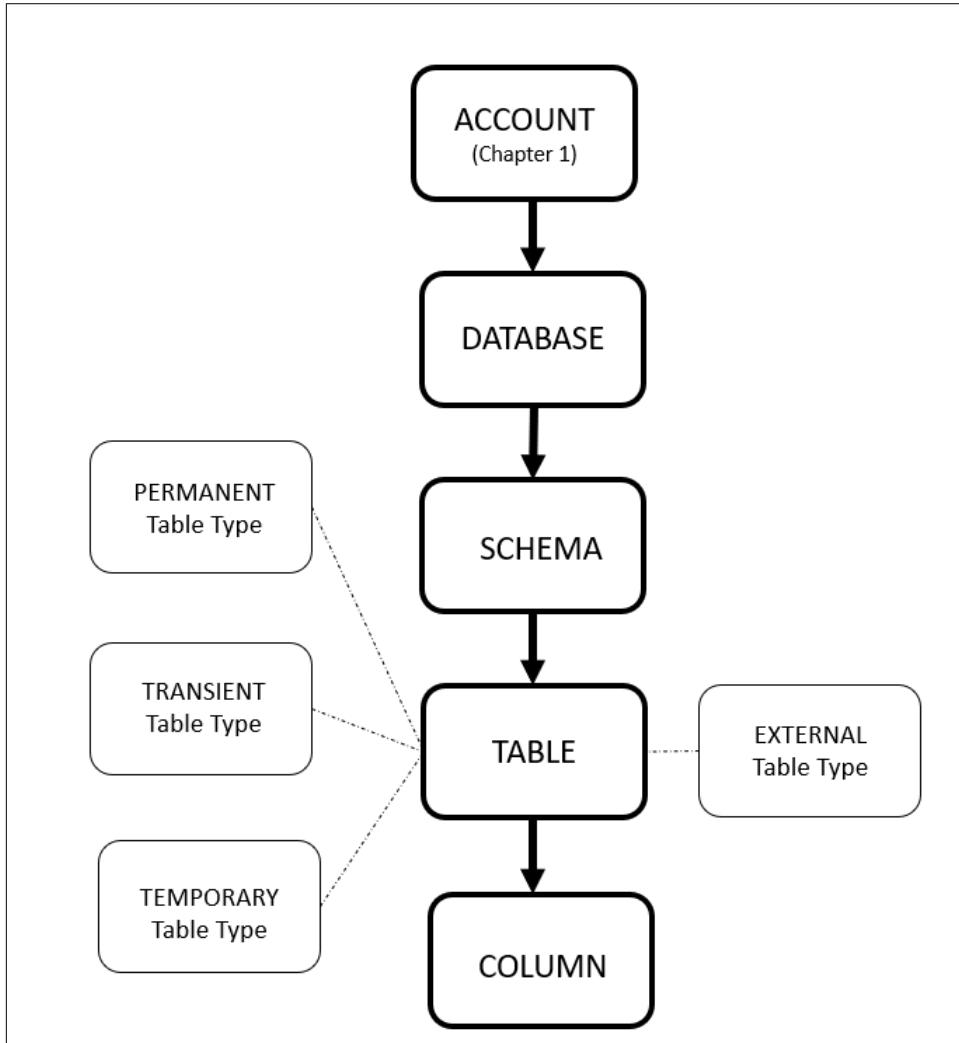


Figure 2-7. Objects Hierarchy for Snowflake Table

As we saw in the database section, Snowflake assumes it should create a permanent table if the table type is not specified, unless the table is created within a transient database.

Transient tables are unique to Snowflake and have characteristics of both permanent and temporary table. Transient tables are designed for transitory data that needs to be maintained beyond a session but doesn't need the same level of data recovery by permanent tables. As a result, the data storage costs for a transient table would be less than a permanent table. One of the biggest differences between transient tables and permanent tables is that the fail-safe service is not provided for transient tables.

It isn't possible to change a permanent table to a transient table by using the ALTER command because the TRANSIENT property is set at table creation time. Likewise, a transient table cannot be converted to a permanent table. If you would like to make a change to a transient or permanent table type, you'll need to create a new table, use the "COPY GRANTS" clause, and then copy the data. Using the COPY GRANTS clause will ensure that the table will inherit any explicit access privileges.

It was mentioned that the default for creating tables is that a permanent table would be created unless otherwise specified. If it makes sense to have new tables automatically created as a transient type by default, you can first create a transient database or schema. As we saw in the databases section, all tables created afterward will be transient rather than permanent.

Transient tables can be accessed by other users who have the necessary permissions. On the other hand, temporary tables exist only within the session in which they are created. This means they are not available to other users and cannot be cloned. Temporary tables have many uses including being used for ETL data and for session-specific data needs.



The temporary table, as well as its data within, is no longer accessible once the session ends. During the time a temporary table exists, it does count toward storage costs; therefore, it is a good practice to drop a temporary table once you no longer need it.



Interestingly, you can create a temporary table that has the same name as an existing table in the same schema since the temporary table is session-based. No errors or warnings will be given. It is also a best practice to give temporary tables unique names to avoid unexpected problems given that the temporary table takes precedence.

Table 3-1 summarizes some characteristics of the different Snowflake tables.

Table 2-1. Snowflake Table Characteristics

Snowflake Tables	Permanent	Transient	Temporary	External
Persistence	Until explicitly dropped	Until explicitly dropped	Remainder of session	Until explicitly dropped

Time Travel Retention (Days)	0 – 90 days*	0 or 1	0 or 1	0
Fail-Safe Period (Days)	7	0	0	0
Cloning Possible	Yes	Yes	Yes	No
Create Views Possible	Yes	Yes	Yes	Yes

*Enterprise Edition and above 0-90 days. Standard Edition 0 or 1 day.

We will now create some tables that we'll use later in the chapter:

```
USE ROLE SYSADMIN;
USE DATABASE CHAPTER3_PDB1;
CREATE
OR REPLACE SCHEMA BANKING;
CREATE
OR REPLACE TABLE CUSTOMER_ACCT (
    Customer_Account int,
    Amount int,
    transaction_ts timestamp
);
CREATE
OR REPLACE TABLE CASH (
    Customer_Account int,
    Amount int,
    transaction_ts timestamp
);
CREATE
OR REPLACE TABLE RECEIVABLES (
    Customer_Account int,
    Amount int,
    transaction_ts timestamp
);
```

After creating this table, the active role is the “SYSADMIN” role, the active database is “CHAPTER3_PDB1” and the active schema is “BANKING”. Thus, a newly created table will be located within the BANKING schema if you create a new table without specifically using a different namespace. Let's try that now:

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE TABLE NEWTABLE (
    Customer_Account int,
    Amount int,
    transaction_ts timestamp
);
```

In [Figure 2-8](#), we can see that NEWTABLE was created in the active namespace.



Figure 2-8. List of the tables in the Banking schema of the CHAPTER3_PDB1 database

Let's now drop the new table we just created. We want to use the fully qualified table name. Type in "DROP TABLE" with a space afterward. Then, double click on the table name and Snowflake will insert the fully qualified table name. Be sure to put a semi-colon at the end and then run the statement.

```
USE ROLE SYSADMIN;
DROP TABLE "CHAPTER3_PDB1"."BANKING"."NEWTABLE";
```

We weren't required to use the fully qualified table name because we were in the active space where we wanted to drop the table. We could have just used the command "DROP TABLE NEWTABLE;". However, it is best practice to use a table's fully qualified name or the "USE" command, which achieves the same goal.

At this point, [Figure 2-9](#) shows what your Account should look like.

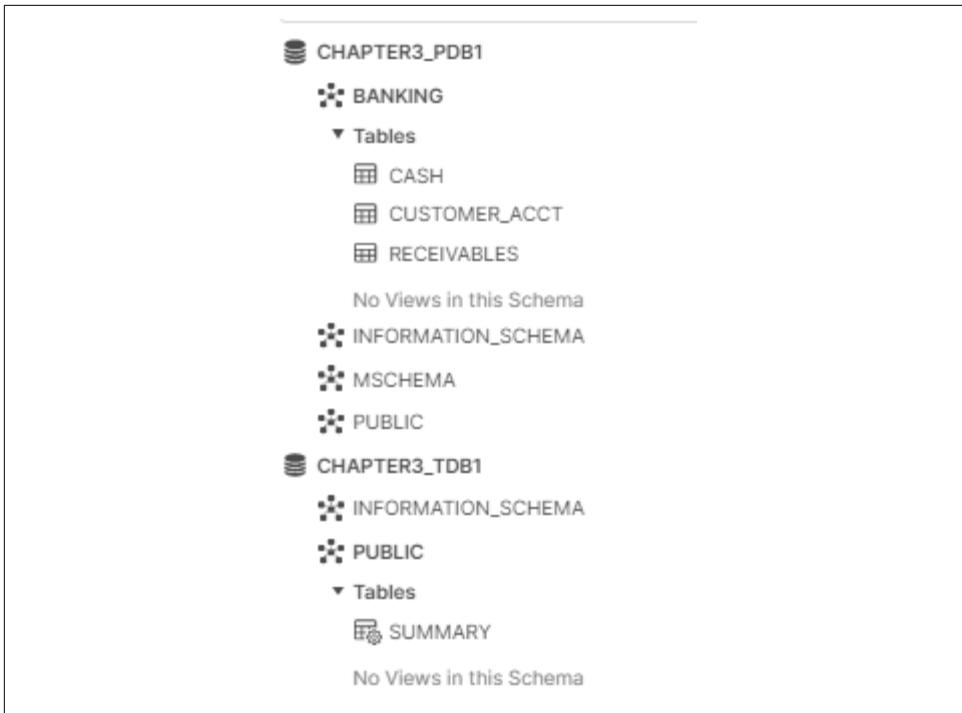


Figure 2-9. Databases and tables created thus far in this chapter

Creating and Managing Views

Along with tables, Snowflake views are the primary objects maintained in database schemas as shown in [Figure 2-10](#). Views are of two types, materialized and non-materialized. Whenever the term “view” is mentioned and the type is not specified, it is understood that it is a non-materialized view.

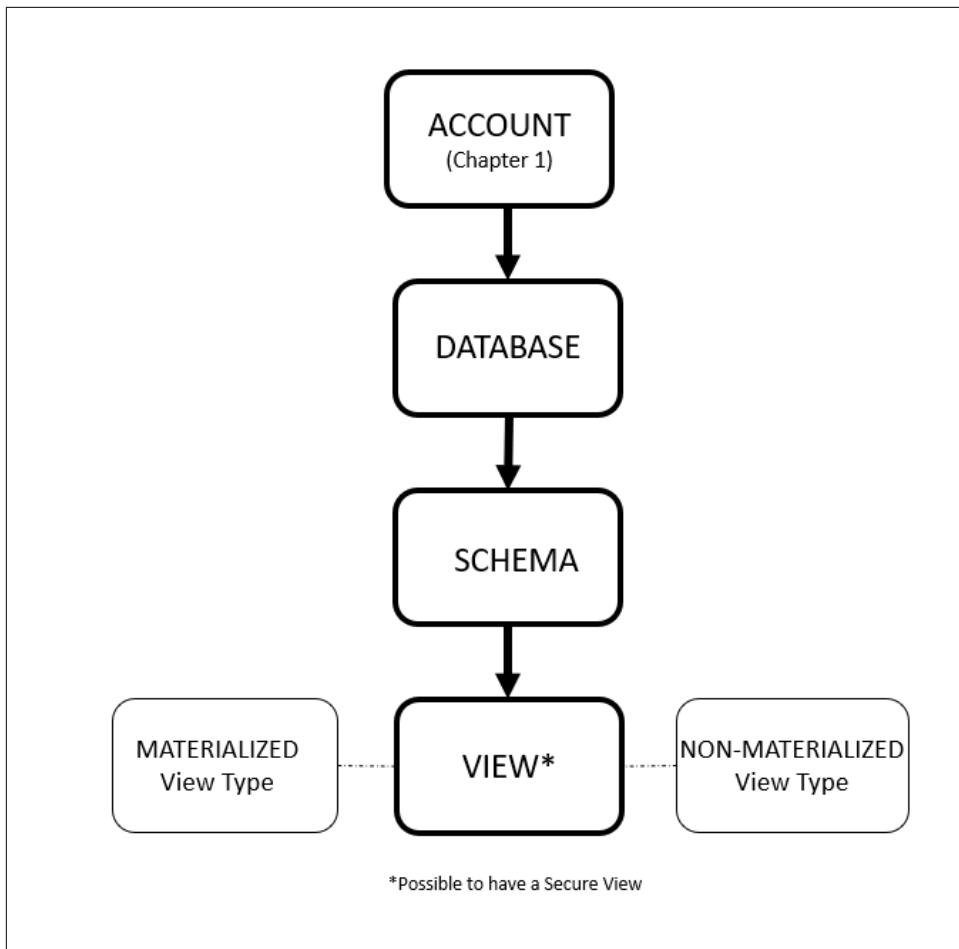


Figure 2-10. Objects Hierarchy for Snowflake View

A *view* is considered to be a virtual table created by a query expression; something like a window into a database. Let's create a new view by selecting one column from the Snowflake sample data.



We've mentioned the importance of using fully qualified names for tables. The importance of using a fully qualified name for the table is even more important when creating views because the connected reference will be invalid if the namespace of the base table is not used and this table or the view is moved to a different schema or database later.

```
CREATE
OR REPLACE VIEW CHAPTER3_TDB1.PUBLIC.NEWVIEW AS
SELECT
    CC_NAME
FROM
(
    SELECT
        *
    FROM
        SNOWFLAKE_SAMPLE_DATA.TPCDS_SF100TCL.CALL_CENTER
);
```

One purpose of views is to display selected rows and columns from one or more tables. This is a way to provide some security by only exposing certain data to specific users. There is the ability for views to provide even more security by creating a specific *secure view* of either a non-materialized or materialized view. We'll dive deeper into secure views when we discuss implementing account security and protections in Chapter 7. We'll also learn more about secure views in Chapter 10, Configuring and managing secure data sharing.

It's important to remember that creating materialized views require Snowflake Enterprise Edition. Let's create a materialized view using the same query as before.

```
CREATE
OR REPLACE MATERIALIZED VIEW CHAPTER3_TDB1.PUBLIC.NEWVIEW_MATERIALIZED AS
SELECT
    CC_NAME
FROM
(
    SELECT
        *
    FROM
        SNOWFLAKE_SAMPLE_DATA.TPCDS_SF100TCL.CALL_CENTER
);
```

You can run a "SHOW VIEWS" command and both views will be returned in the results. If you run a "SHOW MATERIALIZED VIEWS" command, then only the materialized view result will be returned.

If you run a SELECT * command for each of the views, you'll notice that the results are identical. That is because we haven't really used a materialized view for its intended purpose. Unlike a regular view, a materialized view object gets periodically refreshed with the data from the base table. Thus, it is illogical to consider using a materialized view for the Snowflake sample database because the Snowflake sample database cannot be updated with new data.

Also, the query to retrieve a single column is not one typically used for materialized views. Materialized views are generally used to aggregate as well as filter data so that the results of resource-intensive operations can be stored in a materialized view for

improved data performance. Improved data performance is especially good when that same query is frequently used.

The data within a materialized view is always current because Snowflake uses a background service to automatically update materialized views. To see that in action, we're going to create a materialized view. We will revisit the view later in the chapter.

```
CREATE  
OR REPLACE MATERIALIZED VIEW CHAPTER3_TDB1.PUBLIC.BANKINGVIEW_MV AS  
SELECT  
    *  
FROM  
(  
    SELECT  
        *  
    FROM  
        CHAPTER3_TDB1.PUBLIC.SUMMARY  
);
```

As you would expect, views are read-only. Thus, it isn't possible to use the INSERT, UPDATE, or DELETE commands on views. Further, Snowflake doesn't allow users to truncate views. While it is not possible to execute DML commands on a view, you can use a subquery within a DML statement that can be used to update the underlying base table. An example might be something like:

```
DELETE FROM <Base Table> WHERE <Column> > (SELECT AVG <Column> FROM View);
```

Other things to be aware of is that a view definition cannot be updated with the ALTER VIEW command. However, the ALTER MATERIALIZED VIEW command can be used to rename a materialized view, to add or replace comments, to modify the view to be a secure view, and much more. The SHOW and DESCRIBE commands are also available for views.

If you wanted to change something structural about the view, you would have to recreate it with a new definition.



Changes to a source table's structure do not automatically propagate to views. For example, dropping a table column won't drop the column in the view.

There are many considerations when deciding between a regular view and a materialized view. Other considerations beyond whether to create a view or a materialized view is considering whether to use ETL to materialize the data set in a table.

As a general rule, it is best to use a non-materialized view when the results of the view frequently change, the query isn't so complex and expensive to rerun, and the results

of the view often change. Regular views do incur compute costs but not storage costs. The compute cost to refresh the view and the storage cost will need to be weighed against the benefits of a materialized view when the results of a view change often.



Generally, it is beneficial to use a materialized view when the query consumes a lot of resources as well as when the results of the view are used often and the underlying table doesn't change frequently. Also, if a table needs to be clustered multiple ways, a materialized view can be used with a cluster key.

There are some limitations for materialized views, such as a materialized view can query only a single table and joins are not supported. It is recommended that you consult the Snowflake documentation for more detail on materialized view limitations.



One thing to remember is that we are using the SYSADMIN role currently and we're creating the views using that role. For someone who doesn't have the SYSDADMIN role, they will need to have assigned to them the privileges on the schema, the database objects in the underlying table(s), and the view itself if they are to work with the view.

Introduction to Snowflake Stages - File Format Included

There are two types of Snowflake stages: internal and external. Stages are Snowflake objects that point to a storage location, either internal to Snowflake or on external cloud storage. Stage objects can be “named stages” or internal user or table stages. The “temporary” keyword can be used to create a session-based name stage object.

In most cases, the storage is permanent while the stage object, a pointer to the storage location, may be temporary or dropped at any time. Snowflake stages are often used as an intermediate step to load files to Snowflake tables or to unload data from Snowflake tables into files.

Snowflake permanent and internal temporary stages are used to store data files internally, on cloud storage managed by Snowflake whereas external stages reference data files that are stored in a location outside of Snowflake. Outside locations, whether private / protected, or public, like Amazon S3 buckets, Google Cloud Storage buckets, and Microsoft Azure containers, are supported by Snowflake and can be used in external stages.

Each Snowflake user has a stage for storing files which is accessible only by that user. The User stage can be referenced by @~. Likewise, each Snowflake table has a stage allocated for storing files and can be referenced by using @%<name of table>.



Table stages are useful if multiple users need to access the files and those files only need to be copied into a single table whereas a user stage is best when the files only need to be accessed by one user but will need to be copied into multiple tables. [end note]

User and table stages cannot be altered or dropped and neither of these stages support setting the file format, but you can specify the format and copy options at the time the “COPY INTO” command is issued. Additionally, table stages do not support transforming the data while loading it. A table stage is tied to the table itself and is not a separate database object. To perform actions on the table stage, you must have been granted the table ownership role.

The command to list a user stage is `ls@~`; or `LIST @~`. [Figure 2-11](#) shows the results of the command to list a user stage.

The screenshot shows a Snowflake session titled "ls LIST @~". The results pane displays a table with four rows, each representing a user stage. The columns are "Row", "name", "size", "md5", and "last_modified".

Row	name	size	md5	last_modified
1	worksheet_data@f5dc520c-e3c5-48e0-90ce-42890d0284ed	720	c1c014cbcded30f94da205893d44775f	Sun, 18 Apr 2021 14:57:21 GMT
2	worksheet_data@ad07684-1fc0-48a4-43fb-57684590123a	484	6a594c2f22a9fb987a7c898fb848431bd	Sun, 18 Apr 2021 16:40:14 GMT
3	worksheet_data@2692c94e-127c-4a25-a059-fecce7c1b501	1568	d1f735dd8d779e2029a5a8170e751f444	Sun, 25 Apr 2021 16:17:17 GMT
4	worksheet_data@29629855-42f9-48d5-a705-b140e34a5e07	944	fe004f5a2d0f5a409fb839739909f938	Sat, 17 Apr 2021 17:38:24 GMT

Figure 2-11. Results of the command to list a user stage

User stages and table stages, both of which are types of internal stages, are automatically provided for each Snowflake account. In addition to user and table stages, internal named stages can also be created (see [Figure 2-12](#)). Internal named stages are database objects, which means that they can be used not just by one user but by any user who has been granted a role with the appropriate privileges.

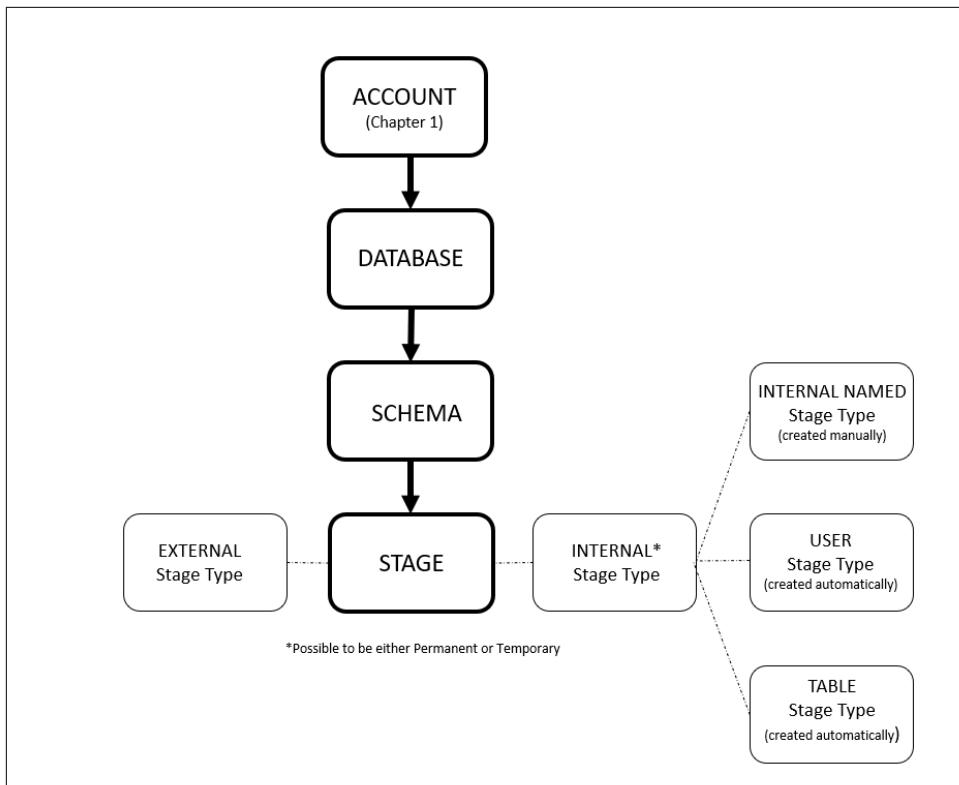


Figure 2-12. Objects Hierarchy for Snowflake Stage

Internal named stages and external stages can be created as either a permanent or a temporary stage. When a temporary external stage is dropped, no data files are removed because those files are stored external to Snowflake. Only the stage object is dropped. For a temporary internal stage, however, the data and stage are both dropped; and the files are not recoverable. It is important to note that the behavior just described is not limited to just temporary stage objects. Both temporary and permanent stages, internal or external, have the same characteristics described.

When using stages, we can use file formats to store all the format information we need for loading data from files to tables. The default file format is CSV. However, you can create file formats for other formats such as JSON, AVRO, ORC, PARQUET, and XML. There are also optional parameters that can be included when you create a file format. We are going to create a file format for loading JSON data. Then, we'll make use of that file format when we create a stage.

```

USE ROLE SYSADMIN;
USE DATABASE CHAPTER3_TDB1;
CREATE
OR REPLACE FILE FORMAT JSON_FILEFORMAT TYPE = JSON;

```

```
USE DATABASE CHAPTER3_TDB1;
USE SCHEMA PUBLIC;
CREATE
OR REPLACE TEMPORARY STAGE TEMP_STAGE FILE_FORMAT = JSON_FILEFORMAT;
```



The data is always in an encrypted state, whether data is in flight between the customer and internal stage or at rest and stored in a Snowflake database table.

File formats have been briefly introduced here and will be explored much more in Chapter 6, Data Loading and Unloading.

Extending SQL with Stored Procedures and UDFs

To extend SQL capabilities in Snowflake, you can create stored procedures and user defined functions (UDFs) to achieve functionalities not possible with Snowflake built-in functions. Both stored procedures and UDFs encapsulate and return a single value (scalar). UDTFs can return multiple values (tabular) whereas stored procedures can return only a single value. You can create stored functions in JavaScript and UDFs in both SQL and JavaScript languages. It is possible to create secure UDFs (see [Figure 2-13](#)).



The return value for stored procedures is scalar but procedures can return multiple values if the return type is a variant.

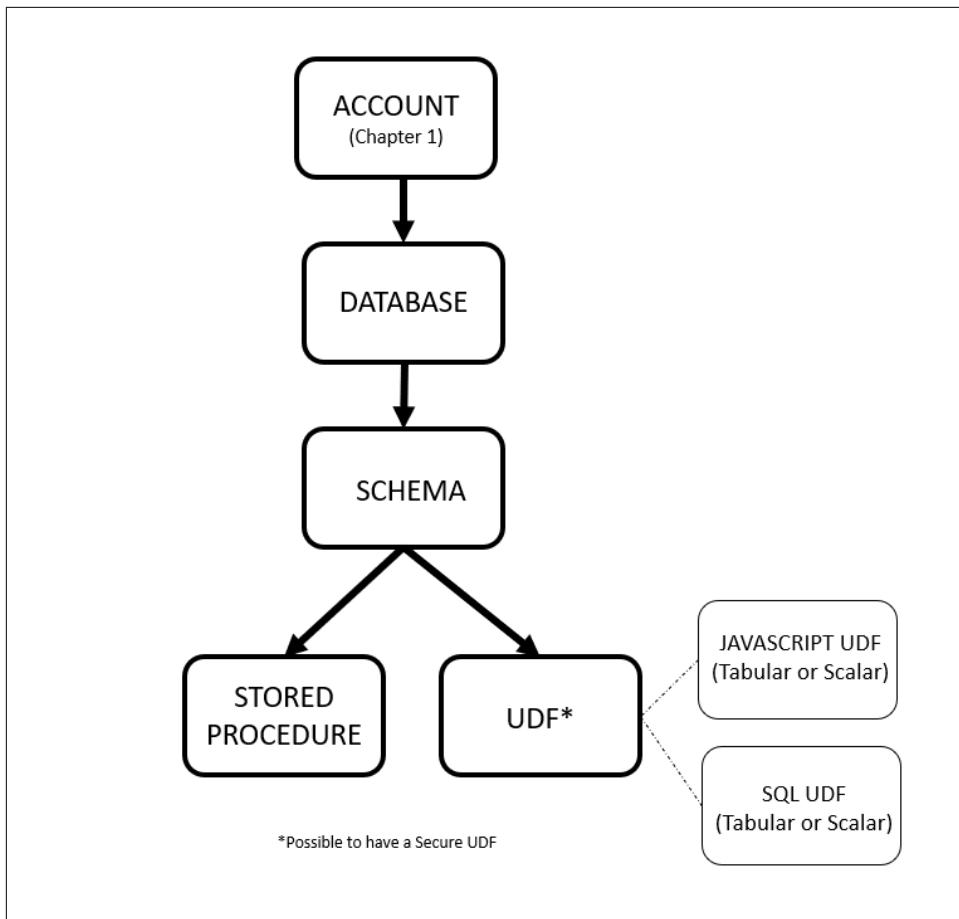


Figure 2-13. Objects Hierarchy for Snowflake Stored Procedures and UDF]

Both stored procedures and UDFs extend SQL capabilities but there are many differences between the two. One of the most important differences is how they are used.



If you need to perform database operations such as SELECT, DELETE, or CREATE, you'll need to use a stored procedure. If you want to use a function as part of the SQL statement or expression or if your output needs to include a value for every input row, then you'll want to use a Snowflake UDF.

A UDF is called as part of the SQL statement, but a stored procedure cannot be called within a SQL statement. Instead, a stored procedure is called as an independent statement using the “CALL” statement. The “CALL” statement can call only one stored procedure per statement.

A UDF is required to return a value and you can use the UDF return value inside your SQL statements. Although not required, a stored procedure is allowed to return a value, but the “CALL” command doesn’t provide a place to store the returned value. The stored procedure also doesn’t provide a way to pass it to another operation.

In the next two sections, there are a total of five examples provided. The first is a simple JavaScript UDF that returns a scalar value and the next is a secure SQL UDF that returns tabular results. Following those are three stored procedures examples. The first is where an argument is directly passed in. The next example is lengthier and demonstrates the basics of an Accounting Information System where each banking transaction has a debit and credit entry. The last example is an advanced example that combines a stored procedure with a task.

User Defined Function (UDF) – Task Included

UDFs allow you to perform some operations that are not available through the built-in, system-defined functions provided by Snowflake. There are two languages for UDFs supported by Snowflake. Both SQL and JavaScript can return either scalar or tabular results.

A SQL UDF evaluates SQL statements and it can refer to other UDFs, although a SQL UDF cannot refer to itself either directly or indirectly.

A JavaScript UDF is useful for manipulating variant and JSON data. A JavaScript UDF expression can refer to itself recursively, although it cannot refer to other UDFs. JavaScript UDFs also have size and depth limitations that don’t apply to SQL UDFs. We’ll see that demonstrated in one of our examples.

JavaScript UDFs have access to the basic standard JavaScript library needed to create arrays, variables, and simple objects. You cannot use math functions or use error handling because Snowflake does not let you import external libraries. The properties that are available for both JavaScript UDFs and JavaScript Procedures can be found by using the commands below.

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE DATABASE DB CHAPTER3;
USE DATABASE DB CHAPTER3;
CREATE
OR REPLACE FUNCTION JS_PROPERTIES() RETURNS string LANGUAGE JAVASCRIPT AS $$
    return Object.getOwnPropertyNames(this);
$$;
SELECT
    JS_PROPERTIES();
```

For our first UDF example, we are going to create a simple JavaScript UDF which returns a scalar result. We mentioned that JavaScript UDFs have size and depth limitations and we'll be able to demonstrate that in this example.

```
USE ROLE SYSADMIN;
USE DATABASE DB CHAPTER3;
CREATE
OR REPLACE FUNCTION FACTORIAL(n variant) RETURNS variant LANGUAGE JAVASCRIPT AS '
    var f=n;
    for (i=n-1; i>0; i--) {
        f=f*i
    }
    return f';
SELECT
    FACTORIAL(5);
```

The result returned is 120 when the number 5 is used. If you use a number greater than 33, you'll receive an error message. Try finding the result of FACTORIAL(50) and see what happens.

Secure UDFs are the same as non-secure UDFs except that they hide the DDL from the consumer of the UDF. Secure UDFs do have limitations on performance functionality due to some optimizations being bypassed. Thus, data privacy versus performance are the considerations because only secure UDFs can be shared. Secure SQL UDFs and JavaScript secure UDFs are both shareable but operate differently and are generally used for different purposes.

JavaScript secure UDFs are often used for data cleansing, address matching, or other data manipulation operations.

Unlike JavaScript secure UDFs, a secure SQL UDF can run queries. For a secure shared UDF, the queries can only be run against the provider's data. When a provider shares a secure UDF with a customer, the cost of data storage for any tables being accessed by the function is paid for by the provider and the compute cost is paid for by the consumer. There may be shared functions that don't incur storage costs.

In this example created for you, there will be no data storage cost incurred because we are using the Snowflake sample data set.

Secure SQL UDTF That Returns Tabular Value (Market Basket Analysis Example)

Market basket analysis is a common use of secure SQL UDFs and that is what we'll be demonstrating. In our example, the output is aggregated data where a consumer would like to see how many times other items were sold with their particular individual item. We wouldn't want the consumer account to have access to our raw sales data, so we'll wrap the SQL statement in a secure UDF and create an input parameter.

Using the secure UDF with an input parameter, the consumer still gets the same results as running the SQL statement directly on the underlying raw data.

We want to use data that is already provided for us in the Snowflake sample database but there are more records than we need for this demonstration. Let's create a new table and select 100,000 rows from the sample database. That will be more than enough.

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE DATABASE DB_UDF;
CREATE
OR REPLACE SCHEMA SCHEMA1;
CREATE
OR REPLACE TABLE DB_UDF.SCHEMA1.SALES AS (
    SELECT
        *
    FROM
        "SNOWFLAKE_SAMPLE_DATA"."TPCDS_SF100TCL"."WEB_SALES"
)
LIMIT
    100000;
```

Next, we'll run our query directly on the new table. In this instance, we are interested in the product with the SK of 87. We want to know the different items that were sold with this product in the same transaction. We can use this query to find that information.

```
SELECT
    87 AS INPUT_ITEM,
    WS_WEB_SITE_SK AS BASKET_ITEM,
    COUNT (DISTINCT WS_ORDER_NUMBER) BASKETS
FROM
    DB_UDF.SCHEMA1.SALES
WHERE
    WS_ORDER_NUMBER IN (
        SELECT
            WS_ORDER_NUMBER
        FROM
            DB_UDF.SCHEMA1.SALES
        WHERE
            WS_WEB_SITE_SK = 87
    )
GROUP BY
    WS_WEB_SITE_SK
ORDER BY
    3 DESC,
    2;
```

The results of this query shows us the product with SK of 87 was sold in 152 unique transactions within our records. The item with SK of 9 and the item with SK of 72

were each sold in the same transaction as the item with SK of 87 a total of 34 times. For the manufacturers of product with SK of 9 and 72, this could be valuable information. We might be willing to share those details but we wouldn't want to allow access to the underlying sales data. Thus, we'll want to create a secure SQL UDF function.

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE SECURE FUNCTION DB_UDF.SCHEMA1.GET_MKTBASKET(INPUT_WEB_SITE_SK number(38)) RETURNS TABLE (
    INPUT_ITEM NUMBER(38, 0),
    BASKET_ITEM NUMBER(38, 0),
    BASKETS NUMBER(38, 0)
) AS 'SELECT input_web_site_sk, WS_WEB_SITE_SK as BASKET_ITEM, COUNT(DISTINCT
WS_ORDER_NUMBER) BASKETS
FROM DB_UDF.SCHEMA1.SALES
WHERE WS_ORDER_NUMBER IN (SELECT WS_ORDER_NUMBER FROM DB_UDF.SCHEMA1.SALES WHERE WS_WEB_SITE_SK = inpu
GROUP BY ws_web_site_sk
ORDER BY 3 DESC, 2';
```

If we were to share with another user this secure UDF through a Snowflake consumer account, that user could run the secure UDF without seeing any of the underlying data, table structures, or the SQL code. This is the command that the owner of the consumer account would use to obtain the results. You can try this command with the number 87 that we used earlier when we queried the table directly and you'll get the same exact result with the UDF function. You can also try other product SKs.

```
SELECT
*
FROM
TABLE(DB_UDF.SCHEMA1.GET_MKTBASKET(9));
```



Using Secure UDFs should be specifically used for instances where data privacy is of concern, and you want to limit access to sensitive data. It is important to consider the purpose and necessity of creating a secure UDF and weigh that against the decreased query performance that is likely to result from using a secure UDF.

Stored Procedures

Stored procedures are similar to functions in that they are created once and can be executed many times. Stored procedures allow you to extend Snowflake SQL by combining it with JavaScript in order to include branching and looping, as well as error handling. Stored procedures, which must be written in JavaScript, can be used to automate tasks that require multiple SQL statements performed frequently. Although stored procedures can only be written in JavaScript at this time, SQL stored procedures are now in private preview.

While you can “SELECT” statements inside a stored procedure, the results must be used within the stored procedure. If not, then only a single value result can be returned. Stored procedures are great for batch actions because a stored procedure runs by itself and, similar to a trigger, can be conditionally tied to database events.

In the first stored procedures example, we are going to pass in an argument to the stored procedure.

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE DATABASE DB_CHAPTER3;
CREATE
OR REPLACE PROCEDURE STOREDPROC1(ARGUMENT1 VARCHAR)
returns string not null
language javascript
as
$$
var INPUT_ARGUMENT1 = ARGUMENT1;
var result = `${INPUT_ARGUMENT1}`;
return result;
$$;
CALL STOREDPROC1('I really love Snowflake ');
```



Javascript is case sensitive so make sure you pay close attention when creating the stored procedure.

Figure 2-14 shows the results of the passing in an argument to the stored procedure.

Results		Data Preview
✓	Query ID	SQL 4.02s 1 rows
Filter result...		
Row		STOREDPROC1
1	I really love Snowflake	

Figure 2-14. Results of passing in an argument to the stored procedure created in the chapter

You can use the INFORMATION_SCHEMA to take a look at the information for Snowflake Procedures in the database.

```
SELECT
  *
FROM
  "DB CHAPTER3"."INFORMATION_SCHEMA"."PROCEDURES";
```

The next stored procedure example is one where we are demonstrating the very basics of an Accounting Information System where tabular results are returned for banking transactions that record debits and credits. If you remember, earlier in the chapter we created the tables needed for this example so we can just dive right into creating the stored procedure.

We are recording the accounting transactions for a bank that deals with customer's giving cash to the bank and getting cash from the bank. We'll keep the accounting portion of the example limited to three types of transactions – deposits, withdrawals, and loan payments.

First, we'll want to create a stored procedure for a deposit transaction where the bank's cash account is debited and the customer account on the bank's balance sheet is credited whenever a customer gives the bank cash to add to their account.

```
USE DATABASE CHAPTER3_PDB1;
USE ROLE SYSADMIN;
CREATE
OR REPLACE PROCEDURE deposit(PARAM_ACCT FLOAT, PARAM_AMT FLOAT) returns STRING LANGUAGE javascript AS
  var ret_val = "";
  var cmd_debit = "";
  var cmd_credit = "";

  // INSERT data into tables
  cmd_debit = "INSERT INTO CHAPTER3_PDB1.BANKING.CASH VALUES (" + PARAM_ACCT + "," + PARAM_AMT + ",";
  cmd_credit = "INSERT INTO CHAPTER3_PDB1.BANKING.CUSTOMER_ACCT VALUES (" + PARAM_ACCT + "," + PARAM_AMT + ")";

  // BEGIN transaction
  snowflake.execute ({sqlText: cmd_debit});
  snowflake.execute ({sqlText: cmd_credit});
  ret_val = "Deposit Transaction Succeeded";
  return ret_val;
$$;
```

Next, we'll create the transaction for a withdrawal. In this case, it is just the reverse of what happens for a deposit.

```
USE ROLE SYSADMIN;
CREATE OR REPLACE PROCEDURE withdrawal (PARAM_ACCT FLOAT,PARAM_AMT FLOAT)
  returns STRING
  LANGUAGE javascript
AS
$$
```

```

var ret_val = "";
var cmd_debit = "";
var cmd_credit = "";

// INSERT data into tables
cmd_debit = "INSERT INTO CHAPTER3_PDB1.BANKING.CUSTOMER_ACCT VALUES (" + PARAM_ACCT + "," + (-PARAM_AMT) +
cmd_credit = "INSERT INTO CHAPTER3_PDB1.BANKING.CASH VALUES (" + PARAM_ACCT + "," + (-PARAM_AMT) + 

// BEGIN transaction
snowflake.execute ({sqlText: cmd_debit});
snowflake.execute ({sqlText: cmd_credit});
    ret_val = "Withdrawal Transaction Succeeded";
return ret_val;
$$;

```

Finally, the transaction for the loan payment is one where the bank's cash account is debited and the receivables account is credit.

```

USE ROLE SYSADMIN;
CREATE OR REPLACE PROCEDURE loan_payment (PARAM_ACCT FLOAT,PARAM_AMT FLOAT)
returns STRING
LANGUAGE javascript
AS
$$
var ret_val = "";
var cmd_debit = "";
var cmd_credit = "";

// INSERT data into the tables
cmd_debit = "INSERT INTO CHAPTER3_PDB1.BANKING.CASH VALUES (" + PARAM_ACCT + "," + PARAM_AMT + ",";
cmd_credit = "INSERT INTO CHAPTER3_PDB1.BANKING.RECEIVABLES VALUES (" + PARAM_ACCT + "," + (-PARAM_AMT) +
//BEGIN transaction
snowflake.execute ({sqlText: cmd_debit});
snowflake.execute ({sqlText: cmd_credit});
    ret_val = "Loan Payment Transaction Succeeded";
return ret_val;
$$;

```

Now, let's run a few quick tests to see if the procedures are working.

```

CALL withdrawal(21, 100);
CALL loan_payment(21, 100);
CALL deposit(21, 100);

```

We can tell that the procedures are working if we run some SELECT commands. For example:

```
SELECT CUSTOMER_ACCOUNT, AMOUNT FROM "CHAPTER3_PDB1"."BANKING"."CASH";
```

After using the “CALL” command followed by the SELECT command to test a few transactions, we feel confident that the procedure is working as intended.

So now we can truncate the tables, leaving the tables intact but removing the data.

```
USE ROLE SYSADMIN;
USE DATABASE CHAPTER3_PDB1;
USE SCHEMA BANKING;
TRUNCATE TABLE "CHAPTER3_PDB1"."BANKING"."CUSTOMER_ACCT";
TRUNCATE TABLE "CHAPTER3_PDB1"."BANKING"."CASH";
TRUNCATE TABLE "CHAPTER3_PDB1"."BANKING"."RECEIVABLES";
```

Note that if we run the previous SELECT statements again now, we'll be able to see that there is no data in the tables. Now, we'll go ahead and input some transactions.

```
USE ROLE SYSADMIN;
CALL deposit(21, 10000);
CALL deposit(21, 400);
CALL loan_payment(14, 1000);
CALL withdrawal(21, 500);
CALL deposit(72, 4000);
CALL withdrawal(21, 250);
```

We'd like to see a transaction summary, so we'll create one final stored procedure.

```
USE ROLE SYSADMIN;
USE DATABASE CHAPTER3_TDB1;
USE SCHEMA PUBLIC;
CREATE OR REPLACE PROCEDURE Transactions_Summary()
    returns STRING
    LANGUAGE javascript
    AS
$$
var cmd_truncate = `TRUNCATE TABLE IF EXISTS CHAPTER3_TDB1.PUBLIC.SUMMARY;`;
var sql = snowflake.createStatement({sqlText: cmd_truncate});
var cmd_cash = `Insert into CHAPTER3_TDB1.PUBLIC.SUMMARY (CASH_AMT) select sum(AMOUNT) from "CHAPTER3_TDB1.PUBLIC.CASH";`;
var sql = snowflake.createStatement({sqlText: cmd_cash});
var cmd_receivables = `Insert into CHAPTER3_TDB1.PUBLIC.SUMMARY (RECEIVABLES_AMT) select sum(AMOUNT) from "CHAPTER3_TDB1.PUBLIC.RECEIVABLES";`;
var sql = snowflake.createStatement({sqlText: cmd_receivables});
var cmd_customer = `Insert into CHAPTER3_TDB1.PUBLIC.SUMMARY (CUSTOMER_AMT) select sum(AMOUNT) from "CHAPTER3_TDB1.PUBLIC.CUSTOMER";`;
var sql = snowflake.createStatement({sqlText: cmd_customer});

//BEGIN transaction
snowflake.execute ({sqlText: cmd_truncate});
snowflake.execute ({sqlText: cmd_cash});
snowflake.execute ({sqlText: cmd_receivables});
snowflake.execute ({sqlText: cmd_customer});
ret_val = "Transactions Successfully Summarized";
return ret_val;
$$;
```

Now, we can see the Transaction Summary. We'll call the stored procedure so that all the debit and credit transactions are summarized and then we'll take a look at the table.

```

CALL Transactions_Summary();

SELECT
    *
FROM
    CHAPTER3_TDB1.PUBLIC.SUMMARY;

Let's also take a look now at the materialized view we created earlier.
USE ROLE SYSADMIN;
USE DATABASE CHAPTER3_TDB1;
USE SCHEMA PUBLIC;
SELECT
    *
FROM
    "CHAPTER3_TDB1"."PUBLIC"."BANKINGVIEW_MV";

```

Now, we have our final stored procedural example. It's an advanced example because we are going to add a task to execute this stored procedure.

Just a quick side note here. Tasks are incredibly powerful and an important topic. We'll cover tasks in more detail in chapter 12 as we see how tasks can be combined with table streams. For now, it's enough to understand that a task can call a stored procedure, as we are going to see in this example, or it can execute a single SQL statement.

We will be creating a stored procedure that will delete a database. Thus, it's important that we create the stored procedure in a different database than the one we'll want to delete using the stored procedure.

```

USE ROLE SYSADMIN;
USE DATABASE DB_CHAPTER3;
CREATE
OR REPLACE PROCEDURE drop_db() RETURNS STRING NOT NULL LANGUAGE javascript AS $$%
    var cmd = `DROP DATABASE "CHAPTER3_PDB1";`;
    var sql = snowflake.createStatement({sqlText: cmd});
    var result = sql.execute();
    return 'Database has been successfully dropped';
$$;

```

Now we can call the stored procedure and the database will be dropped. This is part of our cleanup for this chapter.

```
CALL drop_db();
```

Now that you've seen how this stored procedure works, we'll modify it so that it will drop a different database, and we'll add a task so that the database will be dropped 15 minutes later. That way you can see how a task works.

Let's go ahead and change our drop_db procedure so that we'll be dropping the database CHAPTER3_TDB1.

```

USE ROLE SYSADMIN;
CREATE
OR REPLACE PROCEDURE drop_db() RETURNS STRING NOT NULL LANGUAGE javascript AS $$ 
    var cmd = `DROP DATABASE "CHAPTER3_TDB1";` 
    var sql = snowflake.createStatement({sqlText: cmd}); 
    var result = sql.execute(); 
    return 'Database has been successfully dropped'; 
$$;

```

Next, we'll want to create the task that will delay the stored procedure by 15 minutes.

```

USE ROLE SYSADMIN;
CREATE
OR REPLACE TASK wait_15_task WAREHOUSE = COMPUTE_WH SCHEDULE = '15 MINUTE' AS CALL drop_db();

```

The SYSADMIN role is going to need some privileges to execute the task so be sure to use the ACCOUNTADMIN role for this command. Even though the SYSADMIN role is the task owner, an elevated account level privilege to execute tasks is required.

```

USE ROLE ACCOUNTADMIN;
GRANT EXECUTE TASK ON ACCOUNT TO ROLE SYSADMIN;

```

Because tasks are always created in a suspended state, they'll need to be resumed.

```

USE ROLE SYSADMIN;
ALTER TASK IF EXISTS wait_15_task RESUME;

```

Now our task is in a scheduled state. We'll be able to see that by using this query against the INFORMATION_SCHEMA TASK_HISTORY table function.

```

SELECT
  *
FROM
  table(
    information_schema.task_history(
      task_name => 'wait_15_task',
      scheduled_time_range_start => dateadd('hour', -1, current_timestamp())
    )
  );

```

[Figure 2-15](#) shows the results of the query.

SCHEDULED_TIME	QUERY_START_	NEXT_SCHEDULED_TIME..
2021-04-18 09:13:38.188 -0700	NULL	2021-04-18 09:28:38.1...

Figure 2-15. Query results to see the current status of the task named “wait_15_task”

You can use the next 15 minutes to answer the questions at the end of the chapter to test your knowledge and then come back. Once you see that the task has been completed and the database has been dropped (you can refresh your screen, or you can run the query again to see the state of the task) then you can suspend the task.

```
USE ROLE SYSADMIN;
ALTER TASK IF EXISTS wait_15_task SUSPEND;
```

After you suspend the task, you can run the query one last time and you'll see that the task succeeded and that there are no tasks scheduled.

There is so much more to explore than we can cover in this chapter! But, don't worry, you'll learn more about stored procedures, Java/Python UDFs, external functions, and Snowpark in Chapter 12.

Introduction to Pipes, Streams, and Sequences

Pipes, streams, and sequences are Snowflake objects we've not yet covered. *Pipes* are objects that contain a "COPY" statement that is used by Snowpipe. Snowpipe is used for continuous, serverless loading of data into a Snowflake target table. Snowflake table *streams*, aka change data capture (CDC), keep track of certain changes made to a table including inserts, updates, and deletes. Streams have many useful purposes including recording changes made in a staging table which are used to update another table.

A *sequence* object is used to generate unique numbers. Often, sequences are used as surrogate keys for primary key values. Here is how you can generate a sequence that begins with the number one and increments by one.

```
USE ROLE SYSADMIN;
USE DATABASE DB CHAPTER3;
CREATE
OR REPLACE SEQUENCE SEQ_01 START = 1 INCREMENT = 1;
CREATE
OR REPLACE TABLE SEQUENCE_TEST_TABLE(i integer);
```

You can use the select command three or four times to see how the NEXTVAL increments by one every time.

```
SELECT
SEQ_01.NEXTVAL;
```

What happens when you try this?

```
USE ROLE SYSADMIN;
USE DATABASE DB CHAPTER3;
CREATE
OR REPLACE SEQUENCE SEQ_02 START = 1 INCREMENT = 2;
CREATE
```

```
OR REPLACE TABLE SEQUENCE_TEST_TABLE(i integer);
SELECT
    SEQ_02.NEXTVAL a,
    SEQ_02.NEXTVAL b,
    SEQ_02.NEXTVAL c,
    SEQ_02.NEXTVAL d;
```

You should see that the value of A is 1, the value of B is 3, the value of C is 5 and the value of D is 7.

Some important things to remember about sequences is that the first value in a sequence cannot be changed after the sequence is created and sequence values, although unique, are not necessarily gap-free. Because sequences are user-defined database objects, the sequence value can be shared by multiple tables because sequences are not tied to any specific table. An alternative to creating a sequence is to use identity columns where you would generate auto-incrementing numbers, often used as a primary key, in one specific table.



A consideration when using sequences is that they may not be appropriate for situations such as a secure UDF case. The reason is that, in some circumstances a consumer may be able to use the difference between sequence numbers to infer information about the number of records. In that case, one option is to exclude the sequence column from the consumer results or to use a unique string ID instead of a sequence.

We'll explore Snowpipe in Chapter 6, Data Loading and Unloading. Streams will be discussed in-depth in Chapter 12 as we discover how to use streams for change data capture. We'll also see how to combine the power of streams and tasks in Chapter 12.

Code Cleanup

Let's perform a code cleanup to remove the objects in your Snowflake account in preparation for working on another chapter example.

Note that there is no need to drop objects in the hierarchy below the database before dropping the databases. If you've been following along, you dropped all databases except DB CHAPTER3 so you'll just need to drop that database now.

Exercises to Test Your Knowledge

The following exercises are based on the coverage in this chapter about databases and database objects such as schemas, tables, views, stages, stored procedures and UDFs.

1. What are the different types of databases, schemas, and tables that can be created? If a particular type is not specifically stated at the time of creation, what is the default type for each?
2. What is the difference between scalar and tablular UDFs?
3. What kinds of things can you do with a stored procedure that you cannot do with a UDF?
4. What would happen if we used the “CREATE DATABASE” command and the database we want to create already exists? What if we used the “CREATE OR REPLACE DATABASE” command?
5. What is the default retention time for a database? Can the database retention time be changed? Can the database default retention time be changed?
6. Why might you choose to use the “TRUNCATE TABLE” command rather than the “DROP TABLE” command?
7. Are there any storage or compute costs associated with views?
8. What is the difference between a *fully* and a *partially qualified name*?
9. When using stages, what is the default file format? What other file formats does Snowflake support?
10. What is unique about the SNOWFLAKE database that comes with every Snowflake account?

Solutions to these exercises are available in Appendix A.

Exploring Snowflake SQL Commands, Data Types, and Functions

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 4th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

The standard programming language for relational database management is the Structured Query Language (SQL). In addition to SQL support for structured data, Snowflake offers native support for semi-structured data formats such as JSON and XML. Snowflake support for unstructured data is available as well.

The main focus of this chapter is on learning the fundamentals of how to use Snowflake worksheets to execute SQL commands. Other than using worksheets in the Snowflake web UI, it is possible to use a Snowflake-native command line client, known as SnowSQL, to create and execute SQL commands. More detail about SnowSQL will be provided in future chapters.

Besides connecting to Snowflake via the web UI or SnowSQL, you can use ODBC and JDBC drivers to access Snowflake data through external applications such as Tableau and Looker. We’ll explore connections to Tableau and Looker in Chapter 11. Native

connectors such as Python or Spark can also be used to develop applications for connecting to Snowflake.

In Chapter 12, we'll take a deep dive into Snowpark and learn how to query and process data in a pipeline. To help you prepare for mastering advanced topics in the upcoming chapters, we'll want to focus first on learning the basics of Snowflake SQL commands, data types, and Snowflake functions.

Working with SQL Commands in Snowflake

The Structured Query Language (SQL) can be divided into five different language command types. To create Snowflake objects, you need to use Data Definition Language (DDL) commands. Giving access to those objects requires the Data Control Language (DCL). Next, you'll use Data Manipulation Language (DML) commands to manipulate the data into and out of Snowflake. Transaction Control Language (TCL) commands enable you to manage transaction blocks. Data Query Language (DQL) statements are then used to actually query the data. Included below is a list of common SQL commands, by type.

- Data Definition Language (DDL) commands
 - CREATE
 - ALTER
 - TRUNCATE
 - RENAME
 - DROP
 - DESCRIBE
 - SHOW
 - USE
 - SET / UNSET
 - COMMENT
- Data Control Language (DCL) commands
 - GRANT
 - REVOKE
- Data Manipulation Language (DML) commands
 - INSERT
 - MERGE
 - UPDATE

- DELETE
- COPY INTO
- PUT
- GET
- LIST
- VALIDATE
- REMOVE
- Transaction Control Language (TCL)
 - BEGIN
 - COMMIT
 - ROLLBACK
 - CREATE
- Data Query Language (DQL)
 - SELECT

Each of these five different command language types, and their associated commands, will be discussed briefly in the following sections. A comprehensive list of all Snowflake SQL commands can be found in the Snowflake online documentation at <https://docs.snowflake.com/en/sql-reference/sql-all.html>.

Data Definition Language (DDL) Commands

Data Definition Language (DDL) commands are the SQL commands used to define the database schema. The commands create, modify, and delete database structures. In addition, DDL commands can be used to perform account-level session operations, such as setting parameters, as we'll see later in the chapter when we discuss the SET / UNSET command. DDL Commands include CREATE, ALTER, TRUNCATE, RENAME, DROP, DESCRIBE, SHOW, USE, and COMMENT. With the exception of the COMMENT command, each DDL command takes an object type and identifier.

Snowflake DDL commands manipulate objects such as databases, virtual warehouses, schemas, tables, and views; however, they do not manipulate data. Chapter 3, *Creating and Managing Snowflake Architecture Objects*, is entirely devoted to demonstrating Snowflake DDL commands with in-depth explanations and many hands-on examples.

Data Control Language (DCL) Commands

Data Control Language (DCL) commands are the SQL commands used to enable access control. Examples of DCL commands include GRANT and REVOKE. Chapter

5, *Leveraging Snowflake Access Controls*, will take you through a complete detailed series of examples using DCL commands to show you how to secure Snowflake objects.

Data Manipulation Language (DML) Commands

Data Manipulation Language (DML) commands are the SQL commands used to manipulate the data. The traditional DML commands such as INSERT, MERGE, UPDATE, and DELETE, exist to be used for general data manipulation. For data loading and unloading, Snowflake provides COPY INTO <table> and COPY INTO <location> commands. Additionally, Snowflake's DML commands include some commands that do not perform any actual data manipulation but are used to stage and manage files stored in Snowflake locations. Some examples include VALIDATE, PUT, GET, LIST, and REMOVE. Chapter 6, *Data Loading and Unloading*, will explore many of Snowflake's DML commands.

Transaction Control Language (TCL) Commands

Transaction Control Language (TCL) commands are the SQL commands used to manage transaction blocks within Snowflake. Commands, such as BEGIN, COMMIT, and ROLLBACK can be used for multi-statement transactions in a session. A Snowflake transaction is a set of read and write SQL statements that are processed together as one unit. By default and upon query success, a DML statement that is run separately will be committed individually or will be rolled back at the end of the statement, if the query fails.

Data Query Language (DQL) Commands

The Data Query Language (DQL) command is the SQL command used as either a statement or a clause to retrieve data that meet the criteria specified in the SELECT command. Note that the SELECT command is the only DQL command and it is used to retrieve data and does so by specifying the location and then using the WHERE statement to include attributes necessary for data selection inclusion.

Snowflake SELECT command works on external tables and can be used to query historical data. In certain situations, using the SELECT statement will not require a running virtual warehouse to return results; this is because of Snowflake caching, as described in Chapter 2, *Creating and Managing Snowflake Architecture*. Examples of the SELECT statement, the most common SQL statement, can be found throughout most of the chapters in this book. The following section provides details on how to make the most of the SELECT command.

SQL Query Development, Syntax and Operators in Snowflake

Within Snowflake, SQL development can be undertaken natively, using Snowflake UI worksheets or SnowSQL, as well as via the many third-party SQL tools.

Query syntax is how Snowflake SQL queries are structured, or built. Often there are many different ways to write a SQL query that will yield the desired result. It is important to consider how to optimize the query for the best database performance and lowest cost. Chapter 9 includes a section devoted to discussing the topic of analyzing query performance and optimization techniques.

Query operators include words reserved to specify conditions in a SQL query statement and are most often used in the WHERE clause. They can also be used as conjunctions for multiple conditions in a statement. We'll explore query operators later in this section.

SQL Development and Management

There are two native Snowflake options for developing and querying data. It is easy to get started with Snowflake SQL development using the worksheets browser-based SQL editor within the Snowflake interface. Using Snowflake worksheets requires no installation or configuration.

An alternative to worksheets would be SnowSQL, a Python-based client that can be downloaded from the Snowflake client repository and used to perform Snowflake tasks such as querying or executing DDL and DML commands. SnowSQL is frequently used for loading and unloading of data.

Snowflake provides SnowSQL versions for Linux, macOS, and Microsoft Windows. Executable SnowSQL can be run as an interactive shell or in batch mode. Snowflake provides complete instructions on how to download and install SnowSQL for all supported platforms. That documentation can be viewed at <https://docs.snowflake.com/en/user-guide/snowsql-install-config.html>.

You can view the recently used client versions, including the SnowSQL version, in your Snowflake account by using the Snowflake Query History. To access that information, click on the History tab, in the classic web interface. Alternatively, click on Computer > Query History if you are using the new web interface. From there, you can view the Client Info column. Interestingly, the Client Info column in the classic web interface includes an icon to indicate whether the client version is support, unsupported, or nearing the end of support ([Figure 3-1](#))

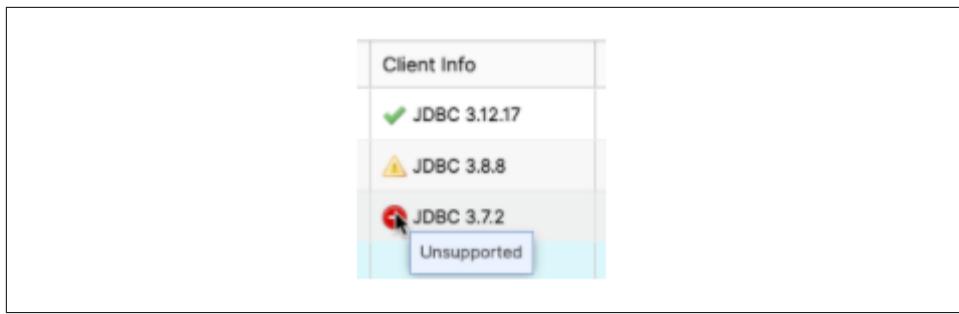


Figure 3-1. Client Info column from the History tab (classic web interface)

In addition to native Snowflake tools, there exists a wide variety of third-party SQL tools available for modeling, developing, and deployment SQL code in Snowflake applications. Some of these third-party tools, such as DataOps and SqlDBM, are available for a free trial by using the Snowflake Partner Connect. You can visit <https://docs.snowflake.com/en/user-guide/ecosystem-editors.html> for a more comprehensive list of third-party SQL tools available for use with Snowflake.



For drivers and connectors that support sending a SQL statement for preparation before execution, Snowflake will prepare DML Commands, SHOW<objects>, and SELECT SQL statements received from drivers and connectors that support that functionality. Other types of SQL statements received from drivers and connectors will be executed by Snowflake without preparation.

Query Syntax

Snowflake SQL queries begin with either the WITH clause or the SELECT command. The WITH clause, an optional clause that precedes the SELECT statement, is used to define common table expressions (CTEs) which are referenced in the FROM clause. Most queries, however, begin with the SELECT command and syntax which appears afterward. The other syntax, described in Table 4-1, are evaluated in the following order:

- From
- Where
- Group by
- Having
- Window
- Qualify
- Order By

- Limit

Table 3-1. Snowflake Query Syntax

Query Syntax	Comments
WITH TOP<n>	Optional clause that precedes the body of the 'SELECT' statement Contains the maximum number of rows returned, recommended to include 'ORDER BY'
FROM AT BEFORE, CHANGES, CONNECT BY, JOIN, MATCH_RECOGNIZE, PIVOT / UNPIVOT, SAMPLE / TABLESAMPLE_VALUE	Specifies the tables, views, or table functions to use in a 'SELECT' statement
WHERE	Specifies a condition that matches a subset of rows; can filter the result of the 'FROM' clause; can specify which rows to operate on in an 'UPDATE', 'MERGE', or 'DELETE'
GROUP BY GROUP BY CUBE, GROUP BY GROUPING SETS, GROUP BY ROLLUP, HAVING	Groups rows with the same group-by-item expressions and computes aggregate functions for resulting group; can be a column name, a number which references a position in the 'SELECT' list, or a general expression
QUALIFY	Filters the results of window functions
ORDER BY	Specifies an ordering of the rows of the result table from a 'SELECT' list
LIMIT / FETCH	Constrains the maximum number of rows returned; recommended to include 'ORDER BY'

Note that QUALIFY is evaluated after a Window function; QUALIFY works with Window functions much in the same way as HAVING does with the aggregate functions and GROUP BY clauses. More information about Window functions can be found later in this chapter.

Subqueries, Derived Columns, and Common Table Expressions (CTEs)

A subquery is a query within another query and can be used to compute values that are returned in a SELECT list, grouped in a GROUP BY clause, or compared with other expressions in the WHERE or HAVING clause

A Snowflake subquery is a nested SELECT statement supported as a block in one or more of the following Snowflake SQL statements:

- CREATE TABLE AS
- SELECT
- INSERT
- INSERT INTO
- UPDATE

- **DELETE**

To prepare for our hands-on exercises for subqueries and derived columns, we need to create a few simple tables and insert some values into those tables. We'll create one database for this chapter. We'll also create a schema and table for our subqueries and derived column examples.

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE DATABASE CH4_DB;
CREATE
OR REPLACE SCHEMA SUBQUERIES;
CREATE
OR REPLACE TABLE CH4_DB.SUBQUERIES.DERIVED (ID integer, Amt integer, Total integer);
INSERT INTO
    DERIVED (ID, Amt, Total)
VALUES
    (1, 1000, 4000),
    (2, 2000, 3500),
    (3, 3000, 9900),
    (4, 4000, 3000),
    (5, 5000, 3700),
    (6, 6000, 2222);
```

We'll need a second table in the Subqueries schema.

```
CREATE
OR REPLACE TABLE CH4_DB.SUBQUERIES.TABLE2 (ID integer, Amt integer, Total integer);
INSERT INTO
    TABLE2 (ID, Amt, Total)
VALUES
    (1, 1000, 8300),
    (2, 1001, 1900),
    (3, 3000, 4400),
    (4, 1010, 3535),
    (5, 1200, 3232),
    (6, 1000, 2222);
```

Having now created both tables, we can write an uncorrelated subquery.

```
SELECT
    ID,
    Amt
FROM
    CH4_DB.SUBQUERIES.DERIVED
WHERE
    Amt = (
        SELECT
            MAX(Amt)
        FROM
            CH4_DB.SUBQUERIES.TABLE2
    );
```

You'll notice that an uncorrelated subquery is an independent query, one where the value returned doesn't depend on any column of the outer query. An uncorrelated subquery returns a single result that is used by the outer query only once. On the other hand, a correlated subquery references one or more external columns. A correlated subquery is evaluated on each row of the outer query table and returns one result per row that is evaluated.

Let's try executing a correlated subquery now.

```
SELECT
    ID,
    Amt
FROM
    CH4_DB.SUBQUERIES.DERIVED
WHERE
    Amt = (
        SELECT
            Amt
        FROM
            CH4_DB.SUBQUERIES.TABLE2
        WHERE
            ID = ID
    );
```

We receive an error message telling us that a single-row subquery returns more than one row ([Figure 3-2](#)). This probably isn't what you expected.

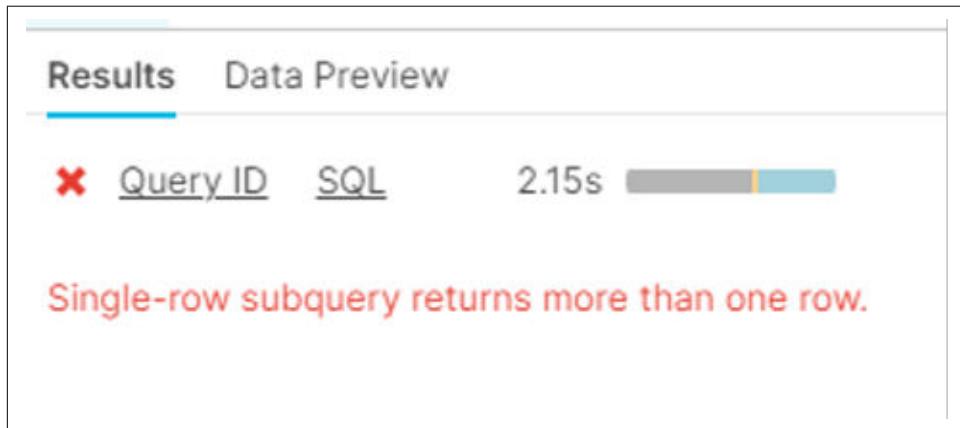


Figure 3-2. Error message received when executing a correlated subquery without an aggregate

Logically, we know that there is only one row per ID; so, the subquery won't be returning more than one row in the result set. However, the server can't know that. We must use a MIN, MAX, or AVG function so that the server can know for certain that only one row will be returned each time the subquery is executed.

Let's go ahead and add "MAX" to the statement to see for ourselves how this works.

```
SELECT
    ID,
    Amt
FROM
    CH4_DB.SUBQUERIES.DERIVED
WHERE
    Amt = (
        SELECT
            MAX(Amt)
        FROM
            CH4_DB.SUBQUERIES.TABLE2
        WHERE
            ID = ID
    );
```

Success! We get a result set of one row with the ID equal to the value of 3. What happens if we change the equal sign to a greater than sign?

```
SELECT
    ID,
    Amt
FROM
    CH4_DB.SUBQUERIES.DERIVED
WHERE
    Amt > (
        SELECT
            MAX(Amt)
        FROM
            CH4_DB.SUBQUERIES.TABLE2
        WHERE
            ID = ID
    );
```

Now we get a result set with three values ([Figure 3-3](#))

The screenshot shows a 'Results' tab in a query editor. At the top, it displays 'Query ID' and 'SQL' with a duration of '473ms' and '3 rows'. Below this is a search bar labeled 'Filter result...' and buttons for 'Copy' and download. The main area is a table with three rows, each containing a 'Row' number, an 'ID', and an 'AMT'. The data is as follows:

Row	ID	AMT
1	4	4000
2	5	5000
3	6	6000

Figure 3-3. Correlated query result set with three values

Now what happens if we change MAX to AVG?

```

SELECT
    ID,
    Amt
FROM
    CH4_DB.SUBQUERIES.DERIVED
WHERE
    Amt > (
        SELECT
            AVG(Amt)
        FROM
            CH4_DB.SUBQUERIES.TABLE2
        WHERE
            ID = ID
    );

```

You may want to try different operators in the WHERE clause and different aggregators in the SELECT clause to see for yourself how correlated subqueries actually work.

Correlated subqueries are used infrequently because they result in one query per each row which is probably not the best scalable approach for most use cases.

Subqueries can be used for multiple purposes, one of which is to calculate or derive values which are then used in a variety of different ways. Derived columns can also be

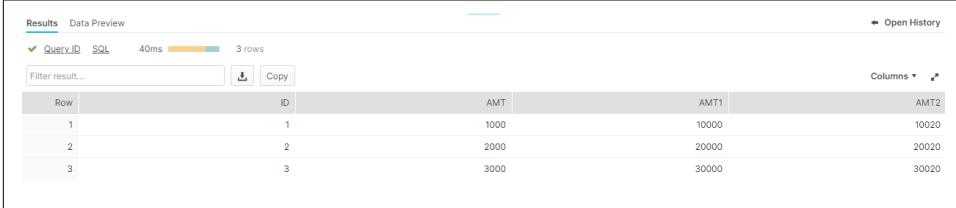
used in Snowflake to calculate another derived column, can be consumed by the outer SELECT query, or can be used as part of the WITH clause. These derived column values, sometimes called computed column values or virtual column values, are not physically stored in a table but are instead recalculated each time they are referenced in a query.

Our next example demonstrates how a derived column can be used in Snowflake to calculate another derived column. We'll also discover how we can use derived columns in one query, in subqueries, and with common table expressions.

Let's create a derived column, "Amt1", from the Amt column and then directly use the first derived column to create the second derived column, "Amt2".

```
SELECT
    ID,
    Amt,
    Amt * 10 as Amt1,
    Amt1 + 20 as Amt2
FROM
    CH4_DB.SUBQUERIES.DERIVED;
```

The result of running that query can be seen in [Figure 3-4](#).



Row	ID	AMT	AMT1	AMT2
1	1	1000	10000	10020
2	2	2000	20000	20020
3	3	3000	30000	30020

Figure 3-4. Result of query with two derived columns

We can achieve the same results by creating a derived column, "Amt1", which can then be consumed by an outer SELECT query. The subquery in our example is a Snowflake uncorrelated scalar subquery. As a reminder, the subquery is considered to be an uncorrelated subquery because the value returned doesn't depend on any outer query column.

```
SELECT
    sub.ID,
    sub.Amt,
    sub.Amt1 + 20 as Amt2
FROM
(
    SELECT
        ID,
        Amt,
        Amt * 10 as Amt1
    FROM
        CH4_DB.SUBQUERIES.DERIVED)
```

```
    CH4_DB.SUBQUERIES.DERIVED  
) as sub;
```

Lastly, we get the same results by using a derived column as part of the WITH clause. You'll notice that we've included a common table expression (CTE) subquery which could help increase modularity and simplify maintenance. The CTE defines a temporary view name which is "CTE1" in our example. Included in the CTE are the column names and a query expression, the result of which is basically a table.

```
WITH CTE1 AS (  
    SELECT  
        ID,  
        Amt,  
        Amt * 10 as Amt2  
    FROM  
        CH4_DB.SUBQUERIES.DERIVED  
)  
SELECT  
    a.ID, b.Amt,  
    b.Amt2 + 20 as Amt2  
FROM  
    CH4_DB.SUBQUERIES.DERIVED a  
JOIN CTE1 b ON(a.ID = b.ID);
```

A major benefit of using a CTE is that it can make your code more readable. With a CTE, you can define a temporary table once and refer to it whenever you need it instead of having to declare the same subquery every place you need it. While not demonstrated here, a CTE can also be recursive. A recursive CTE can join a table to itself many times to process hierarchical data.



Whenever the same names exist for a CTE and a table or view, the CTE will take precedence. Therefore, it is recommended to always choose a unique name for your common table expressions.

Caution about Multi-row Inserts

Now is a good time for a quick pause to learn a little more about multi-row inserts. One or more rows of data can be inserted using a select query or inserted as explicitly stated values in a comma-separated list. To keep things simple, we've been inserting values in comma-separated lists in this chapter.

There is one important thing to be aware of regarding multi-row inserts. When inserting multiple rows of data into a varchar data type, each of the data types being inserted into varchar columns must be the same or else the insert will fail. A varchar data type can accept data values such as 'one' or 1 but never both types of values in the same insert statement. We can best see this with some examples.

We'll first create a new schema and table to do some multi-row insert testing. In the first example, we'll insert 'one' into the varchar Dept column.

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE SCHEMA CH4_DB.TEST;
CREATE
OR REPLACE TABLE CH4_DB.TEST.TEST1 (ID integer, Dept Varchar);
INSERT INTO
    TEST1 (ID, Dept)
VALUES
    (1, 'one');
SELECT
    *
FROM
    CH4_DB.TEST.TEST1;
```

As expected, the value was successfully entered. What happens if we instead insert a numerical value into the varchar column?

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE SCHEMA CH4_DB.TEST;
CREATE
OR REPLACE TABLE CH4_DB.TEST.TEST1 (ID integer, Dept Varchar);
INSERT INTO
    TEST1 (ID, Dept)
VALUES
    (1, 1);
SELECT
    *
FROM
    CH4_DB.TEST.TEST1;
```

Again, the value was successfully entered. What happens, though, if we try inserting both types into the column within the same insert statement?

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE SCHEMA CH4_DB.TEST;
CREATE
OR REPLACE TABLE CH4_DB.TEST.TEST1 (ID integer, Dept Varchar);
INSERT INTO
    TEST1 (ID, Dept)
VALUES
    (1, 'one'),
    (2, 2);
```

When we try to insert two different data types into the varchar column at the same time, we experience an error, as shown in [Figure 3-5](#).

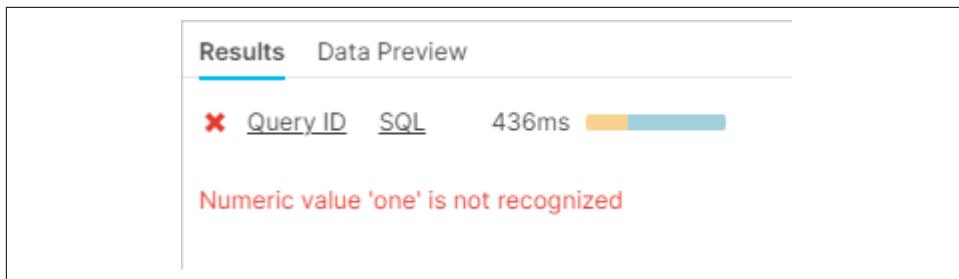


Figure 3-5. Error message received when attempting to insert two different data types into a varchar column in one multi-row insert statement

Let's try again but, this time, insert two values with the same data type.

```
USE ROLE SYSADMIN;
CREATE
OR REPLACE SCHEMA CH4_DB.TEST;
CREATE
OR REPLACE TABLE CH4_DB.TEST.TEST1 (ID integer, Dept Varchar);
INSERT INTO
    TEST1 (ID, Dept)
VALUES
    (1, 'one'),
    (2, 'two');
SELECT
    *
FROM
    CH4_DB.TEST.TEST1;
```

We're also successful if we insert two numerical values into the varchar column.

```
INSERT INTO
    TEST1 (ID, Dept)
VALUES
    (3, 3),
    (4, 4);
SELECT
    *
FROM
    CH4_DB.TEST.TEST1;
```

You'll notice that we are able to successfully load two different data types into the varchar column, but not at the same time. And once we have two different data types in the varchar column, we can still add additional values.

```
INSERT INTO
    TEST1 (ID, Dept)
VALUES
    (5, 'five');
SELECT
```

```
*  
FROM  
    CH4_DB.TEST.TEST1;
```

Multi-row inserts are one way of getting data into Snowflake. Chapter 6 is devoted entirely to data loading and unloading, including in-depth discussion of bulk data loading options and continuous data loading options.

Query Operators

There are several different types of query operators including arithmetic, comparison, logical, subquery, and set operators.

Arithmetic operators, including +, -, *, /, and %, produce a numeric output from one or more inputs. The scale and precision of the output depends on the scale and precision of the input(s). Note that subtraction is the only arithmetic operation allowed on DATE expressions.

Comparison operators, typically appearing in a WHERE clause, are used to test the equality of two inputs. Comparison operators include the following:

- Equal (=)
- Not Equal (!=) or (<>)
- Less Than (<)
- Less Than or Equal (<=)
- Greater Than (>)
- Great Than or Equal (>=)



Remember that TIMESTAMP_TZ values are compared based on their times in UTC, which does not account for daylight saving time. This is important because, at the moment of creation, TIMESTAMP_TZ stores the offset of a given time zone, not the actual time zone.

Logical operators can only be used in the WHERE clause. The order of precedence of these operators is NOT then AND then OR. *Subquery operators* include [NOT] EXISTS, ANY or ALL, and [NOT] IN.

Queries can be combined when using *set operators* such as INTERSECT, MINUS or EXCEPT, UNION and UNION ALL.

The default set operator order of preference is INTERSECT as the highest precedence, followed by EXCEPT, MINUS and UNION, and finally UNION ALL as the

lowest precedence. Of course, you can always use parenthesis to override the default. Note that the UNION set operation is costly because it needs to sort the records to eliminate duplicate rows.



When using set operators, make sure that each query selects the same number of columns and that the data type of each of the columns is consistent although explicit type cast can be used, if the data types are inconsistent.

Long Running Queries and Query Performance & Optimization

The Snowflake system will cancel long running queries. The default duration for long running queries is two days but you can always set the STATEMENT_TIMEOUT_IN_SECONDS duration value at an account, session, object, or warehouse level.

During the Snowflake SQL query process, one of the things that happens is the optimization engines find the most efficient execution plan for a specific query. In Chapter 9, we'll learn more about analyzing query performance and optimization techniques as well as how to use Snowflake's query profiler.

Snowflake Query Limits

SQL statements submitted through Snowflake clients have a query text size limit of one megabyte. Included in that limit are literals, including both string and binary literals. The query text size limit applies to the compressed size of the query. However, because compression ratio for data widely varies, it is recommended to keep the uncompressed query text size below one megabyte.

Additionally, Snowflake limits the number of expressions allowed in a query to 2^{14} . There are ways to resolve this type of error depending on what are you trying to do with your SQL query statement.

If you're attempting to insert data when you receive the error, try breaking up the statement into smaller queries ([Figure 3-6](#)).

Query Run Error

The query you are trying to run is too long. Consider breaking it down into smaller queries.

Close

Figure 3-6. Query Run Error received with the number of expressions exceeds 16,384 values

However, an even better choice would probably be to use the COPY INTO command instead of the INSERT command.

Another type of query limit error occurs when using a SELECT statement with an IN clause that has greater than 16,384 values .

```
SELECT
    column_1
FROM
    table_1
WHERE
    column_2 IN (1, 2, 3, 4, 5,...);
```

One solution would be to use a JOIN or UNION command after placing those values in a second table.

```
SELECT
    column_1
FROM
    table_1 a
JOIN table_2 b ON a.column_2 = b.column_2;
```

Introduction to Data Types Supported by Snowflake

Snowflake supports the basic SQL data types including Geospatial data types and a Boolean logical data type which provides for ternary logic. Snowflake's BOOLEAN data type can have an "unknown" value, or a TRUE or FALSE value. If the Boolean is used in an expression, such as a SELECT statement, then an "unknown" value returns a NULL. If the Boolean is used as a predicate, such as in a WHERE clause, then the "unknown" results will evaluate to FALSE. There are a few data types not supported by Snowflake such as Large Object (LOB), including BLOB and CLOB, as well as ENUM and user-defined data types.

In this section, we'll take a deeper dive into several Snowflake data types such as numeric, string and binary, date and time, semi-structured, and unstructured.

Numeric Data Types

Snowflake's numeric data types include fixed-point numbers and floating-point numbers, as detailed in Table 4-2. Included in the table is information about each numeric data types' precision and scale. Precision, the total number of digits, impacts storage, whereas scale, the number of digits following the decimal point, does not. However, processing numeric data values with a larger scale could cause slower processing.

Table 3-2. Snowflake Numeric Data Types



It is a known issue that DOUBLE, DOUBLE PRECISION, and REAL columns are stored as DOUBLE but displayed as FLOAT.

Fixed-point numbers are exact numeric values and, as such, are often used for natural numbers and exact decimal values such as monetary amounts. In contrast, floating-point data types are used most often for mathematics and science.

You can see how fixed-point numbers vary based on the data type by trying the following example.

```
CREATE  
OR REPLACE DATABASE CH4_DB;  
CREATE  
OR REPLACE TABLE NUMFIXED (  
    NUM NUMBER,  
    NUM12 NUMBER(12, 0),  
    DECIMAL DECIMAL (10, 2),  
    INT INT,  
    INTEGER INTEGER  
);
```

To see what was created, you can run the “DESC TABLE NUMFIXED;” command and get the results in Figure 4-7.

	name	type	kind	null?
1	NUM	NUMBER(38,0)	COLUMN	Y
2	NUM12	NUMBER(12,0)	COLUMN	Y
3	DECIMAL	NUMBER(10,2)	COLUMN	Y
4	INT	NUMBER(38,0)	COLUMN	Y
5	INTEGER	NUMBER(38,0)	COLUMN	Y

Figure 3-7. Results showing Snowflake Fixed-Point Numbers Data Types

Now you can compare fixed-point numbers to floating-point numbers by using this next example.

```
DESC TABLE NUMFIXED;
USE DATABASE CH4_DB;
CREATE
OR REPLACE TABLE NUMFLOAT (
    FLOAT FLOAT,
    DOUBLE DOUBLE,
    DP DOUBLE PRECISION,
    REAL REAL
);
```

Once again, use the Describe command to see the results as shown in Figure 3-8.

```
DESC TABLE NUMFLOAT;
```

	name	type	kind
1	FLOAT	FLOAT	COLUMN
2	DOUBLE	FLOAT	COLUMN
3	DP	FLOAT	COLUMN
4	REAL	FLOAT	COLUMN

Figure 3-8. Results showing Snowflake Floating-Point Number Data Types

In traditional computing, float data types are known to be faster for computation. But, is that still an accurate statement about float data types in modern warehouses such as Snowflake? Not necessarily. It is important to consider that integer values can be stored in a compressed format in Snowflake, whereas float data types are not. This

results in less storage space and cost for integers. Querying rows for an integer table type also takes significantly less time.



Because of the inexact nature of floating-point data types, floating-point operations could have small rounding errors and those errors can accumulate, especially when using aggregate functions to process a large number of rows.

Snowflake's numeric data types are supported by numeric constants. Constants, also referred to as literals, represent fixed data values. Numeric digits 0 through 9 can be prefaced by a positive or negative sign. Exponents, indicated by e or E, are also supported in Snowflake numeric constants.

String & Binary Data Types

Snowflake supports both text and binary string data types, the details of which can be seen in Table 4-3.

Table 3-3. Snowflake Text and Binary String Data Types

TEXT STRINGS	Comments
Data Type	
VARCHAR	Optional parameter (N), max number of characters
CHAR, CHARACTERS	Synonymous with VARCHAR; length is CHAR(1) if not specified
STRING, TEXT	Synonymous with VARCHAR
BINARY	Has no notion of Unicode characters so length is always measured in bytes; if length not specified then the default is 8 MB, the max length
VARBINARY	Synonymous with BINARY

You can see how the various text string data types vary by attempting the following example which creates the text string fields and then describes the table.

```
USE DATABASE CH4_DB;
CREATE
OR REPLACE TABLE TEXTSTRING(
    VARCHAR VARCHAR,
    V100 VARCHAR(100),
    CHAR CHAR,
    C100 CHAR(100),
    STRING STRING,
    S100 STRING(100),
    TEXT TEXT,
    T100 TEXT(100)
```

```
);
DESC TABLE TEXTSTRING;
```

If you followed along with the example, you should see the output in [Figure 3-9](#).

	name	type	kind	null?
1	VARCHAR	VARCHAR(16777216)	COLUMN	Y
2	V100	VARCHAR(100)	COLUMN	Y
3	CHAR	VARCHAR(1)	COLUMN	Y
4	C100	VARCHAR(100)	COLUMN	Y
5	STRING	VARCHAR(16777216)	COLUMN	Y
6	S100	VARCHAR(100)	COLUMN	Y
7	TEXT	VARCHAR(16777216)	COLUMN	Y
8	T100	VARCHAR(100)	COLUMN	Y

Figure 3-9. Results of creating a TEXTSTRING table

Snowflake's string data types are supported by string constants which are always enclosed between delimiters, either single quotes or dollar signs. Using dollar sign symbols as delimiters is especially useful when the string contains many quote characters.

Date & Time Input / Output Data Types

Snowflake uses the Gregorian Calendar, rather than the Julian Calendar, for all dates and timestamps. The Snowflake date and time data types are summarized in Table 4-4.

Table 3-4. Snowflake Date and Time Data Types

DATE & TIME		Comments
Data Type		
DATE		Single DATE type; most common date forms are accepted; all accepted timestamps are valid inputs with TIME truncated; the associated time is assumed to be midnight
DATETIME		Alias for TIMESTAMP_NTZ
TIME		Single TIME type in the form HH:MI:SS, internally stored as "wallclock" time; time zones not taken into consideration
TIMESTAMP	Default is TIMESTAMP_NTZ	User-specified alias of one of the three TIMESTAMP_variations
TIMESTAMP_LTZ		Internally UTC time with a specified precision; TIMESTAMP with local time zone
TIMESTAMP_NTZ		Internally "wallclock" time; TIMESTAMP without time zone
TIMESTAMP_TZ		Internally UTC time with a time zone offset; TIMESTAMP with time zone

Snowflake's data and time data types are supported by interval constants as well as date and time constants. Interval constants can be used to add or subtract a specific period of time to or from a date, time, or timestamp. The interval is not a data type and can be used only in date, time, or timestamp arithmetic and will represent seconds if the date or time portion is not specified.



The order of interval increments is important because increments are added or subtracted in the order in which they are listed. This could be important for calculations affected by leap years.

Semi-structured Data Types

Structured data, known as quantitative data, can be easily stored in a database table as rows and columns whereas semi-structured data, such as XML data, is not schema dependent which makes it more difficult to store in a database. In some situations, however, semi-structured data can be stored in a relational database.

Snowflake supports data types for importing and operating on semi-structured data such as JSON, Avro, ORC, Parquet, and XML data. Snowflake does so through its universal data type VARIANT, a special column type which allows you to store semi-structured data. See Table 4-5 for more information about Snowflake semi-structured data types. Note that it is possible for a VARIANT value to be missing, which is considered to be different from a true null value.

Table 3-5. Snowflake Semi-Structured Data Types

SEMI-STRUCTURED Data Type	Comments
VARIANT	Can store Object and Array Stores values of any other type up to a max of 16 MB uncompressed; internally stored in compressed columnar binary representation
OBJECT	Represents collections of key-value pairs with the key as a non-empty string and the value of VARIANT type
ARRAY	Represents arrays of arbitrary size whose index is a non-negative integer and values have VARIANT type



When loaded into a VARIANT column, non-native values such as dates and timestamps are stored as strings. As such, storing values this way will likely cause operations to be slower and to consume more than space as compared to storing date and timestamp values in a relational column with the corresponding data type.

Unstructured Data Types

There are many advantages to using unstructured data, known as qualitative data, to gain insight. Media logs, medical images, audio files of call center recordings, document images and many other types of unstructured data can be used for analytical purposes and for the purpose of sentiment analysis. Storing and governing unstructured data is not easy. Unstructured data is not organized in a predefined manner which means it is not well-suited for relational databases. Typically, unstructured data has been stored in blob storage locations which has several inherent disadvantages, making it difficult and time-consuming to search for files.

To improve searchability of unstructured data, Snowflake recently launched built-in directory tables. Using a tabular file catalog for searches of unstructured data is now as simple as using a `SELECT *` command on the directory table. Users can also build a table stream on top of a directory table which makes it possible to create pipelines for processing unstructured data. Additionally, Snowflake users can create secure views on directory tables and, thus, are also able to share those secure views with others.

Snowflake SQL Functions and Session Variables

Snowflake offers users the ability to create user-defined functions and to use external functions, as well as to access many different built-in functions. Session variables also extend Snowflake SQL capabilities.

Using System Defined (Built-In) Functions

Examples of Snowflake built-in functions include scalar, aggregate, window, table, and system functions.

Scalar functions accept a single row or value as an input and then returns one value as a result whereas *aggregate functions* also return a single value but they accept multiple rows or values as inputs.

Scalar Functions

Some scalar functions operate on a string or binary input value. Examples include `CONCAT`, `LEN`, `SPLIT`, `TRIM`, `UPPER` and `LOWER` case conversion, and `REPLACE`. Other scalar file functions, such as `GET_STAGE_LOCATION`, enable you to access files staged in Snowflake cloud storage.

Additionally, there are many things that you can do in Snowflake with date and time data types which include the following examples of scalar date & time functions and data generation functions:

- Construct / Deconstruct (Extract) using month, day, and year components
- Truncate or “Round up” dates to a higher level
- Parse and Format dates using strings
- Add / Subtract to find and use date differences
- Generate system dates or a table of dates

Aggregate Functions

A Snowflake aggregate function will always return one row even when the input contains no rows. The returned row from an aggregate function where the input contains zero rows could be a zero, an empty string, or some other value. Aggregate functions can be of a general nature, such as MIN, MAX, MEDIAN, MODE, and SUM. Aggregate functions also include linear regression, statistics and probability, frequency estimation, percentile estimation and much more.

Snowflake *window functions* are a special type of aggregate function that can operate on a subset of rows. This subset of related rows is called a “window”. Unlike aggregate functions which return a single value for a group of rows, a window function will return an output row for each input row. The output depends not only on the individual row passed to the function but also on the values of the other rows in the window passed to the function.

Window functions are commonly used for finding the year-over-year percentage change, a moving average, running or cumulative total and rank rows by groupings or custom criteria.

Let’s compare an aggregate function with a window function. In this first example, we’ll create an aggregate function by using the vowels in the alphabet and their corresponding locations.

```

SELECT
    LETTER,
    SUM(LOCATION) as AGGREGATE
FROM
(
    SELECT
        'A' as LETTER,
        1 as LOCATION
    UNION ALL
    (
        SELECT
            'A' as LETTER,
            1 as LOCATION
    )
    UNION ALL
    (
        SELECT

```

```

        'E' as LETTER,
        5 as LOCATION
    )
) as AGG_TABLE
GROUP BY
    LETTER;

```

The result of this query is shown in [Figure 3-10](#)

	LETTER	AGGREGATE
1	A	2
2	E	5

Figure 3-10. Result of an aggregate function

Next, we'll create a window function using the same logic.

```

SELECT
    LETTER,
    SUM(LOCATION) OVER (PARTITION BY LETTER) as WINDOW_FUNCTION
FROM
(
    SELECT
        'A' as LETTER,
        1 as LOCATION
    UNION ALL
    (
        SELECT
            'A' as LETTER,
            1 as LOCATION
    )
    UNION ALL
    (
        SELECT
            'E' as LETTER,
            5 as LOCATION
    )
) as WINDOW_TABLE;

```

Notice, in [Figure](#), how the Letter 'A' has the same sum value in the window function as in the aggregate function but repeats in the results because the input has two separate 'A' listings.

	LETTER	WINDOW_FUNCTION
1	A	2
2	A	2
3	E	5

Table Functions

Table functions, often called *tabular functions*, return results in a tabular format with one or more columns and none, one, or many rows. Most Snowflake table functions are 1-to-N functions where each input row generates N output rows but there exist some M-to-N table functions where a group of M input rows produces a group of N output rows. Table functions can be system defined or user defined. Some examples of system defined table functions include VALIDATE, GENERATOR, FLATTEN, RESULT_SCAN, LOGIN_HISTORY, and TASK_HISTORY.

System Functions

Built-in *system functions* return system-level information, query information, or perform control operations.

One frequently used system information function is the SYSTEM\$CLUSTERING_INFORMATION function which returns clustering information, including the average clustering depth, about one or more columns in a table.

System control functions allow you to execute actions in the system. One example of a control function is SYSTEM\$CANCEL_ALL_QUERIES and requires the session ID. You can obtain the session ID by logging in as the ACCOUNTADMIN and go to Account > Sessions.

```
SELECT SYSTEM$CANCEL_ALL_QUERIES(<session_id>);
```

If you need to cancel queries for a specific warehouse or user rather than the session, you'll want to use the ALTER command along with ABORT ALL QUERIES instead of a system control function.

Creating SQL & Javascript User-defined Functions (UDFs) and using Session Variables

SQL functionality can be extended by SQL user Functions (UDF), Javascript UDFs, and session variables. We took a deep dive into both SQL and Javascript User Defined Functions in the previous chapter, *Creating and Managing Snowflake Architecture Objects*, so we'll focus on learning more about session variables in this section.

Snowflake supports SQL variables declared by the user, using the SET command. These session variables exist while a Snowflake session is active. Variables are distinguished in a Snowflake SQL statement by a \$ sign prefix and can also contain identifier names when used with objects. You must wrap a variable inside the identifier, such as IDENTIFIER(\$Variable), to use a variable as an identifier. Alternatively, you can wrap the variable inside of an object in the context of a FROM clause.

To see all the variables defined in the current session, use the SHOW VARIABLES command.

Some examples of session variable functions include:

- SYS_CONTEXT and SET_SYS_CONTEXT
- SESSION_CONTEXT and SET_SESSION_CONTEXT
- GETVARIABLE and SETVARIABLE

All variables created during a session are dropped when a Snowflake session is closed. If you want to destroy a variable during a session, you can use the UNSET command.

External Functions

An *external function* is a type of user defined function that calls code which is stored and executed outside of Snowflake. Snowflake supports scalar external functions which means the remote service must return exactly one row for each row received. Within Snowflake, the external function is stored as a database object that Snowflake uses to call the remote service.

It is important to note that rather than calling a remote service directly, Snowflake most often calls a proxy service to relay the data to the remote service. Amazon API Gateway and Microsoft Azure API management service are two examples of proxy services that can be used. A remote service can be implemented as an AWS Lambda function, a Microsoft Azure function, or an HTTPS server (e.g. Node.js) running on an EC2 instance.

Any charges by providers of remote services will be billed separately. Snowflake charges normal costs associated with the data transfer and warehouse usage when using external functions.

There are many advantages of using external functions. External functions can be created to be called from other software programs in addition to being called from within Snowflake. Also, the code for the remote services can be written in languages such as Go or C#; languages that cannot be used within other Snowflake UDFs. One of the biggest advantages is that the remote services for Snowflake external functions can be interfaced with commercially available third-party libraries, such as machine-learning scoring libraries.

Code Cleanup

Code cleanup for this chapter is simple. You can use the following command to drop the database we created earlier.

```
DROP DATABASE CH4_DB;
```

Notice that we don't have to remove all the tables first because dropping the database will automatically drop the associated tables.

Summary

In this chapter, we created and executed all of our Snowflake queries using the SYSADMIN role. This was done intentionally so that we could focus on learning the basics of Snowflake SQL commands, functions, and statements without adding in the complexity of needing to navigate Snowflake access controls . Now it's time to build on that foundational knowledge of this chapter, along with what we learned in the previous chapter about creating and managing architecture objects.

In the next chapter, we'll take a deep dive into leveraging Snowflake access controls. If you expect to be assigned administrator responsibilities for one of the core admin roles, the next chapter will likely be one of the most important chapters for you in your Snowflake journey of learning. Even if you never expect to perform administrator duties, you'll still need to know how to leverage the full functionality of Snowflake within the permissions you are assigned. Also, even if you are not assigned a Snowflake admin role, it's still likely that you will be given access to perform some functions once reserved only for administrators.

Snowflake has taken great care to design and build access controls that address some of the weaknesses of other platforms. One example of this is that Snowflake has purposely designed an access control system that removes the concept of a “super user”, a major risk of many platforms. That said, it is important to recognize that there is still much you can learn about Snowflake’s unique access controls even if you have experience with access controls built for other platforms.

Exercises to Test Your Knowledge

1. What can be used to make a line of text be a comment rather than it being treated as code?
2. Snowflake’s string data types are supported by string constants. What delimiters can be used to enclose strings?
3. What are some advantages of using external functions?

4. What is the default duration Snowflake uses to determine when to cancel long running queries? Can you change that duration and, if so, how would you do that?
5. What are the risks of using floating-point number data types?
6. How does a window function differ from an aggregate function?
7. Does Snowflake support unstructured data types?
8. What semi-structured data types does Snowflake support?
9. Do Snowflake's TIMESTAMP data type support local time zones and daylight savings time? Explain.
10. What are derived columns and how can they be used in Snowflake?

Solutions to these exercises are available in Appendix A.

Leveraging Snowflake Access Controls

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 5th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

An organization’s data repositories are filled with information that is valuable business data and data that is confidential. Securing and protecting this data is therefore essential and often a regulatory or a statutory requirement. Though data security is critical, it must be reasonably balanced against the needs for data access to conduct business. In other words, data access should not be so restrictive as to hobble a business’ ability to operate effectively and efficiently for the sake of security. Rather, the necessary data must be made available to just the right users at just the right time without sacrificing security. Developing a good security model involves planning and needs to involve the appropriate stakeholders.

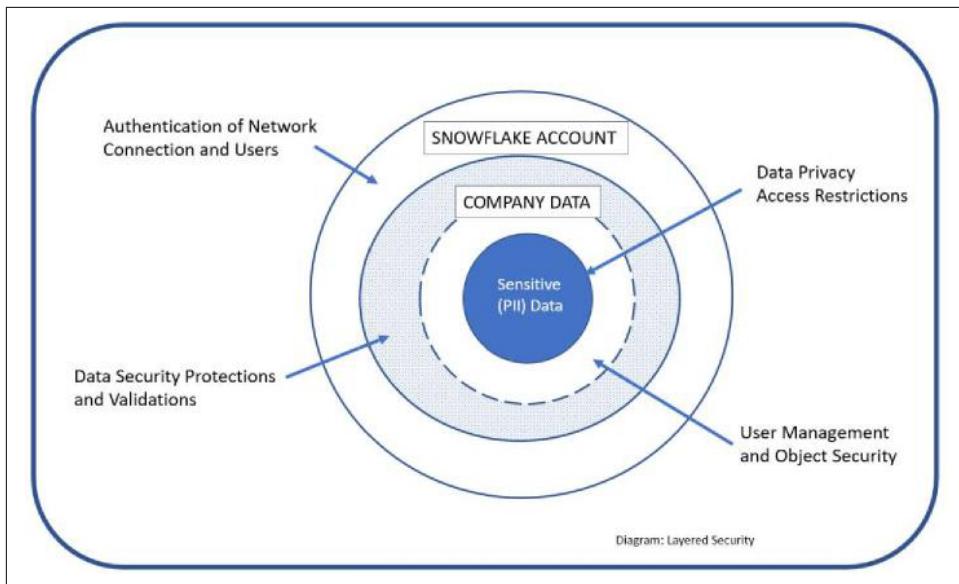


Figure 4-1. Layered Security

Creating multiple layers of security that take advantage of Snowflake's built-in security options is among the best practices for managing security. Per Diagram 5-1, the outer security layer as depicted by the outermost concentric circle, relies on network policies, key pair authentication, MFA and Secure Private Networks as options to ensure that only authenticated users gain access to a Snowflake account. Data security protections are provided by Snowflake in the form of having all data within Snowflake encrypted. Both of these outer layers of security are discussed in more detail in Chapter 7. In this chapter, I'll be explaining the next layer, User Management and Object Security. This type of security uses Role Based Access Control (RBAC) and Discretionary Access Control (DAC) to open up the access to certain data for users. Beyond that, data privacy access restrictions are also covered in Chapter 7 and are those things that restrict access such as dynamic data masking, row level security, and use of secure views and secure UDFs to provide limited access to private data.

In this chapter, you will learn how access control is used to secure Snowflake objects by going through a series of examples which build upon each other. Here are the steps you will work through:

1. Learn about Snowflake's access control model.
2. Create securable objects.
3. Create custom roles.
4. Assign role hierarchies.

5. Grant privileges to roles.
6. Assign roles to users.
7. Test and validate the work.
8. Code cleanup.
9. Test your knowledge.
10. Using Access Controls to Secure Snowflake Objects

Snowflake's hybrid access control provides for access at a granular level. A combination of discretionary access control and role-based access control approaches determine who can access what object and perform operations on those objects as well as who can create or alter the access control policies themselves. *Discretionary access control* is a security model where each object has an owner who has control over that object. *Role-based access control* is an approach where access privileges are assigned to roles and roles are then assigned to one or more users.

In the Snowflake hybrid approach, all securable objects are owned by a role rather than a user. Further, each securable object is owned by only one role which is usually the role used to create the object. Note that because a role can be assigned to more than one user, every user granted the same role also has inherent privileges in a shared controlled fashion. Object ownership can be transferred from one role to another. The Snowflake access control key concepts are described by these definitions and [Figure 4-2](#).

- **Securable object:** Entity such as a database or table. Access to a securable object is denied unless specifically granted.
- **Role:** Roles receive privileges to access and perform operations on an object or to create or alter the access control policies themselves. Roles are assigned to users. Roles can also be assigned to other roles, creating a role hierarchy.
- **Privileges:** Inherent, assigned, or inherited access to an object.
- **User:** A person, service account, or program that Snowflake recognizes.

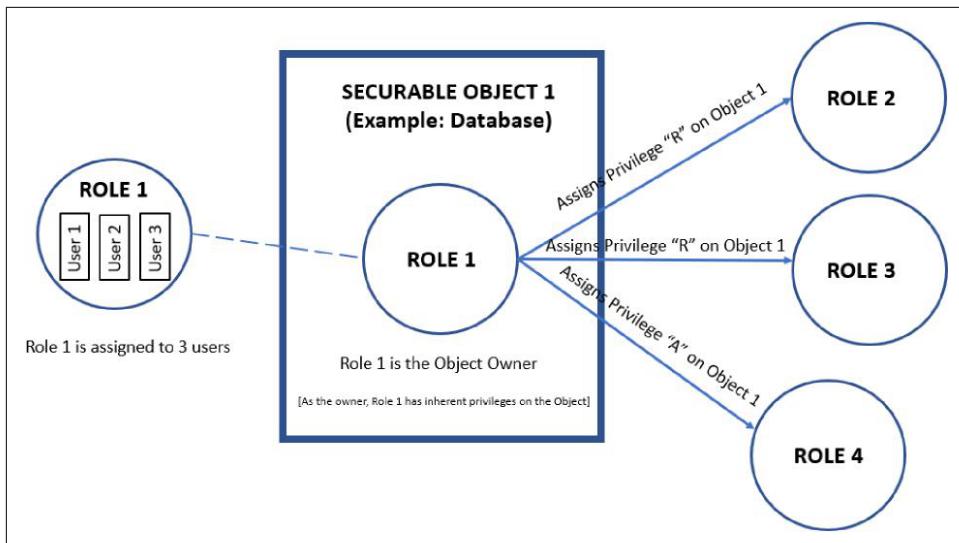


Figure 4-2. Access Control Key Concepts

Every securable object resides in a logical container within a hierarchy with customer Account as the top container. Figure 4-3 shows the hierarchy of Snowflake securable objects along with the system-defined role that inherently owns each type of securable object.

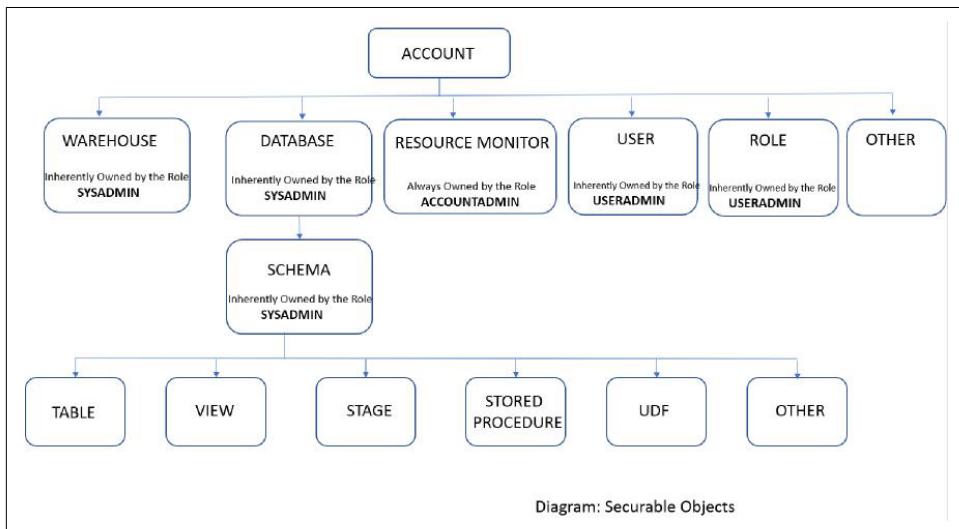


Figure 4-3. Securable Objects



Resource Monitors are *always* owned by the ACCOUNTADMIN role. Only the ACCOUNTADMIN can create or drop this object and cannot grant this privilege to any other role. The ACCOUNTADMIN can grant another role the ability to modify a resource monitor already created.

Creating Securable Objects

To illustrate how Snowflake access controls work, it is necessary to have securable objects in your Snowflake account on which to practice. Let's start by creating a few virtual warehouses which are needed to perform SELECT statements and DML operations. First, the SYSADMIN role is used to create new warehouses. The SHOW command, demonstrated in [Figure 4-4](#), can then be used to confirm that the warehouses were created:

```
USE ROLE SYSADMIN;
CREATE OR REPLACE WAREHOUSE WH_WAREHOUSE1 WITH WAREHOUSE_SIZE='X-SMALL'; CREATE OR REPLACE WAREHOUSE W...
```

Status	Warehouse Name	Size	Clusters	Scaling Poll...	Runn...	Que...	Auto Suspe...	Auto Resume	Created On	Resumed On	Owner
Started	WH_WAREHOUSE3	X-Small	1 active (min: 1, max: 1)	Standard	0	0	10 minutes	Yes	9:43:05 AM	9:43:05 AM	SYSADMIN
Started	WH_WAREHOUSE2	X-Small	1 active (min: 1, max: 1)	Standard	0	0	10 minutes	Yes	9:43:05 AM	9:43:05 AM	SYSADMIN
Started	WH_WAREHOUSE1	X-Small	1 active (min: 1, max: 1)	Standard	0	0	10 minutes	Yes	9:43:04 AM	9:43:05 AM	SYSADMIN
Suspended	COMPUTE_WH	X-Small	min: 1, max: 1	Standard	0	0	10 minutes	Yes	3/13/2021, 1:47:53 ...	3/14/2021, 2:17:04 ...	SYSADMIN

Figure 4-4. Newly Created Virtual Warehouses

Next, the SYSADMIN role is used again to create two databases and three schemas.

Before creating these objects, see how the SHOW DATABASES command returns different results depending on the role that is used to issue the command. The difference occurs because the results are meant to return only the databases to which the role has access.

Three databases are included and made available to every role, including the PUBLIC role, at the time when a Snowflake account is created. Therefore, using role SYSADMIN and role PUBLIC to view the list of databases will return the same results, as seen in [Figure 4-5](#) and [Figure 4-6](#):

```
USE ROLE SYSADMIN;
SHOW DATABASES;
```

Results Data Preview

✓ Query ID SQL 46ms 3 rows

Row	created_on	name	is_default	is_current	origin	owner
1	2021-03-13 ...	DEMO_DB	N	N		SYSADMIN
2	2021-03-13 ...	SNOWFLAKE_SAMPLE_DATA	N	N	SFC_SAMPLES.SAMPLE_DATA	ACCOUNTADMIN
3	2021-03-13 ...	UTIL_DB	N	N		SYSADMIN

Figure 4-5. Existing Snowflake Databases Available to the SYSADMIN Role

```
USE ROLE PUBLIC;
SHOW DATABASES;
```

Results Data Preview

✓ Query ID SQL 46ms 3 rows

Row	created_on	name	is_default	is_current	origin	owner
1	2021-03-13 ...	DEMO_DB	N	N		SYSADMIN
2	2021-03-13 ...	SNOWFLAKE_SAMPLE_DATA	N	N	SFC_SAMPLES.SAMPLE_DATA	ACCOUNTADMIN
3	2021-03-13 ...	UTIL_DB	N	N		SYSADMIN

Figure 4-6. Existing Snowflake Databases Available to the PUBLIC Role

However, a user assigned the ACCOUNTADMIN role will observe a fourth database listed, shown in [Figure 4-7](#). The additional database is a system-defined and read-only shared database provided by Snowflake.

```
USE ROLE ACCOUNTADMIN;
SHOW DATABASES;
```

Results Data Preview

✓ Query ID SQL 41ms 4 rows

Row	created_on	name	is_default	is_current	origin	owner
1	2021-03-13 ...	DEMO_DB	N	N		SYSADMIN
2	2021-03-13 ...	SNOWFLAKE	N	N	SNOWFLAKE.ACCOUNT_USAGE	
3	2021-03-13 ...	SNOWFLAKE_SAMPLE_DATA	N	N	SFC_SAMPLES.SAMPLE_DATA	ACCOUNTADMIN
4	2021-03-13 ...	UTIL_DB	N	N		SYSADMIN

Figure 4-7. Existing Snowflake Databases Available to the ACCOUNTADMIN role

Let's use the SYSADMIN role to create the databases and schemas.



The order in which the SQL commands are written to create databases and schemas is important. Create schemas immediately after creating the database or use the fully qualified object names for creating the schema.

For DB1, we'll create the schema at the end using the fully qualified object name and for DB2 we'll create the schemas right after creating the database. The results are shown in Figure 5-8.

```
USE ROLE SYSADMIN;
CREATE OR REPLACE DATABASE DB1;
CREATE OR REPLACE DATABASE DB2;
CREATE OR REPLACE SCHEMA DB2_SCHEMA1;
CREATE OR REPLACE SCHEMA DB2_SCHEMA2;
CREATE OR REPLACE SCHEMA DB1.DB1_SCHEMA1;
SHOW DATABASES;
```

A screenshot of the Snowflake Data Preview interface. The table shows five rows of database information. The columns are Row, created_on, name, is_default, is_current, origin, and owner. The data is as follows:

Row	created_on	name	is_default	is_current	origin	owner
1	2021-03-19 07:59:11.26...	DB1	N	N		SYSADMIN
2	2021-03-19 07:59:11.97...	DB2	N	Y		SYSADMIN
3	2021-03-13 11:45:43.97...	DEMO_DB	N	N		SYSADMIN
4	2021-03-13 11:45:44.58...	SNOWFLAKE_SAMPLE_DB	N	N	SFC_SAMPLES.SAMPLE_DATA	ACCOUNTADMIN
5	2021-03-13 11:45:41.183...	UTIL_DB	N	N		SYSADMIN

Figure 4-8. Existing Snowflake Databases Available to the SYSADMIN

Resource Monitors will be described in more detail in Chapter 8. Only an Account Administrator can create a Resource Monitor. The results of creating the Resource Monitor can be seen in Figure 4-9.

```
USE ROLE ACCOUNTADMIN;
CREATE OR REPLACE RESOURCE MONITOR RM_MONITOR1 WITH CREDIT_QUOTA=10000
TRIGGERS ON 75 PERCENT DO NOTIFY
ON 98 PERCENT DO SUSPEND
```

```
ON 105 PERCENT DO SUSPEND_IMMEDIATE;  
SHOW RESOURCE MONITORS;
```

Figure 4-9. Resource Monitor Created by the SYSADMIN Role

A screenshot of a Snowflake query results page. The table has columns: Row, name, credit_quota, used_credits, remaining_credits, level, frequency, start_time, end_time, notify_at, suspend_at, and suspend_immediately_at. There is one row with values: 1, RM_MONITOR1, 10000.00, 0.00, 10000.00, NULL, MONTHLY, 2021-02-28 16:00:00, NULL, 75%, 98%, 105%.

Row	name	credit_quota	used_credits	remaining_credits	level	frequency	start_time	end_time	notify_at	suspend_at	suspend_immediately_at
1	RM_MONITOR1	10000.00	0.00	10000.00	NULL	MONTHLY	2021-02-28 16:00:00	NULL	75%	98%	105%

Later in this chapter, we'll explore Snowflake User Management in detail. For now, let's create a few users with minimum information:

```
USE ROLE USERADMIN;  
CREATE OR REPLACE USER USER1  
LOGIN_NAME=ARNOLD; CREATE OR REPLACE USER USER2  
LOGIN_NAME=BEATRICE; CREATE OR REPLACE USER USER3  
LOGIN_NAME=COLLIN;  
CREATE OR REPLACE USER USER4 LOGIN_NAME=DIEDRE;
```

Our first inclination to view a listing of all users would be to use the same role used to create the users, the USERADMIN role. An error is received, shown in Figure 4-10, when we make this attempt because the USERADMIN role does not inherently have sufficient privileges to view a list of users:

```
USE ROLE USERADMIN;  
SHOW USERS;
```

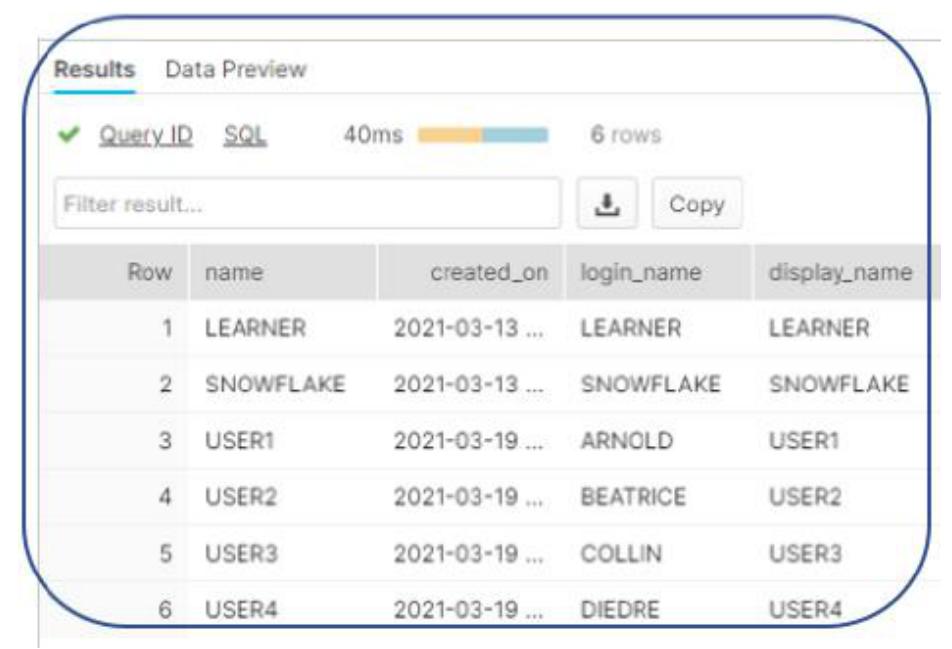
A screenshot of a Snowflake query results page. The SQL query failed with the error: "SQL access control error: Insufficient privileges to operate on account 'QD11374'".

Results	Data Preview
	Query ID SQL 55ms
SQL access control error: Insufficient privileges to operate on account 'QD11374'	

Figure 4-10. Error Displayed Because USERADMIN Role has Insufficient Privileges

However, the listing of all users can be viewed by using the role in the hierarchy just above the USERADMIN role, as shown in Figure 4-11.

```
USE ROLE SECURITYADMIN;  
SHOW USERS;
```



The screenshot shows a 'Results' tab in a Snowflake interface. At the top, it displays 'Query ID' and 'SQL' with a status of '40ms' and '6 rows'. Below this is a search bar labeled 'Filter result...' and buttons for 'Copy' and a download icon. The main area is a table with the following data:

Row	name	created_on	login_name	display_name
1	LEARNER	2021-03-13 ...	LEARNER	LEARNER
2	SNOWFLAKE	2021-03-13 ...	SNOWFLAKE	SNOWFLAKE
3	USER1	2021-03-19 ...	ARNOLD	USER1
4	USER2	2021-03-19 ...	BEATRICE	USER2
5	USER3	2021-03-19 ...	COLLIN	USER3
6	USER4	2021-03-19 ...	DIEDRE	USER4

Figure 4-11. Listing of Snowflake Users

Snowflake System-Defined Roles

A Snowflake role, which is assigned to a user, is an entity to which privileges can be assigned. Assigned privileges do not come with the GRANT option, the ability to grant the assigned privilege to another role, unless specifically assigned. Roles can also be granted to other roles which creates a *role hierarchy*. With this role hierarchy structure, privileges are also inherited by all roles above a particular role in the hierarchy. A user can have multiple roles and can switch between roles, though only one role can be active in a current Snowflake session. As illustrated in the diagram, there are a small number of system-defined roles in a Snowflake account.

As shown in Figure 4-12, the Account Administrator (ACCOUNTADMIN) is at the top level of system-defined roles and can view and operate on all objects in the account with one exception. The Account Administrator will have no access to an object when the object is created by a custom role without an assigned system-defined role. The ACCOUNTADMIN role can stop any running SQL statements and can view and manage Snowflake billing. Privileges for Resource Monitors are unique to the ACCOUNTADMIN role. None of the inherent privileges that come with the Resource Monitor privileges come with the GRANT option but the ACCOUNTADMIN can assign the ALTER Resource Monitor privilege to another role. It is a best

practice to limit the ACCOUNTADMIN role to the minimum number of users required to maintain control of the Snowflake account, but to no less than two users.



Snowflake has no concept of a “Super User” or a “Super Role”. All access to securable objects, even by the Account Administrator, requires access privileges granted explicitly when the ACCOUNTADMIN creates objects itself, or implicitly by being in a higher hierarchy role. As such, if a custom role is created that without assignment to another role in a hierarchy that ultimately leads to the ACCOUNTADMIN role, then any securable objects created by that role would be inaccessible by the Account Administrator.

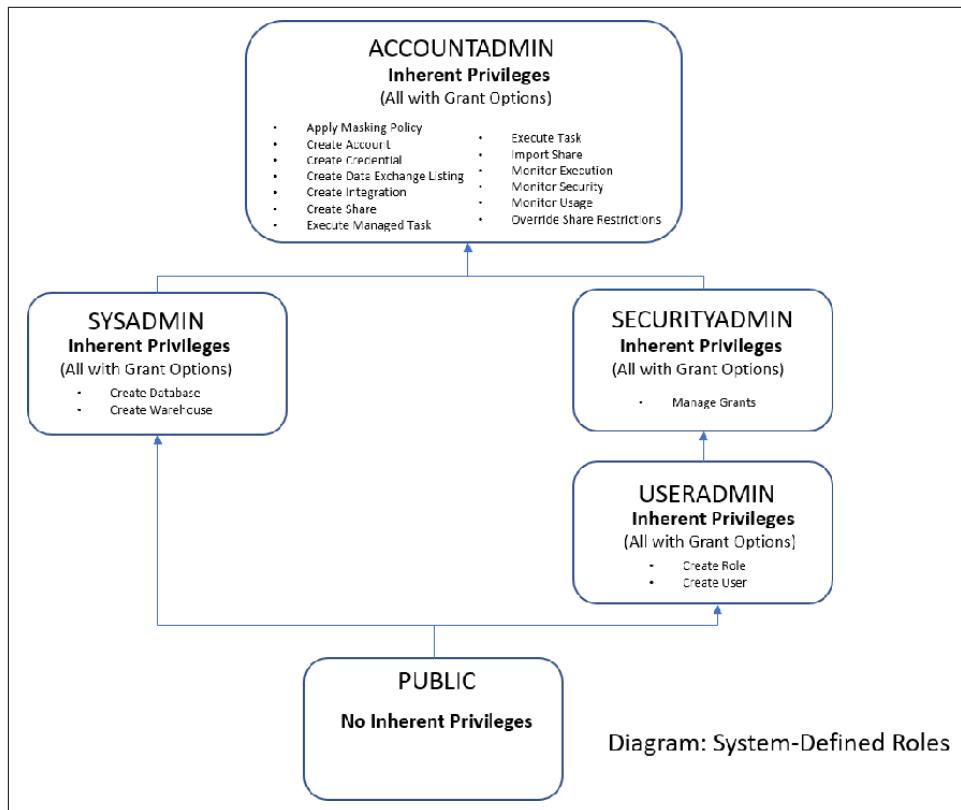


Figure 4-12. System-Defined Roles

In addition to the Account Administrator, other system-defined roles can manage security for objects in the system by using the inherent privileges assigned to them. Those privileges can be granted to other roles in Snowflake, including custom roles, to assign the responsibility for managing security in the system. An example of when

the ACCOUNTADMIN assigns the “Apply Masking Policy” privilege to a custom role will be covered later.

The Security Administrator (SECURITYADMIN) is inherently given the MANAGE GRANTS privilege as well as inheriting all the privileges of the USERADMIN role. The User Administrator (USERADMIN) is responsible for creating and managing users and roles.

The System Administrator (SYSADMIN) role is a system-defined role with privileges to create warehouses, databases, and other objects in the Snowflake account. The most common roles created by the USERADMIN are assigned to the SYSADMIN role, thus enabling the System or Account Administrator to manage any of the objects created by custom roles.

The PUBLIC role is automatically granted to every role and every user in the Snowflake account. Just like any other role, the PUBLIC role can own securable objects.



It is important to remember that any privileges or object access provided to the PUBLIC role is then available to all roles and all users in the account because of the system role hierarchy.

Creating Custom Roles

Creating and managing custom roles are one of the most important functions in Snowflake. Before creating custom roles, planning should be undertaken to design a custom role architecture that will secure and protect sensitive data but not be unnecessarily restrictive. Involved in the planning discussion should be the data stewards, the governance committee, the staff who understand the business needs, and the IT professionals. There is an example in this chapter of an approach for creating custom roles. The approach I’ve taken is to divide the custom roles into *functional-level custom roles*, which include Business and IT roles, and *system-level custom roles*, which include service account roles and object access roles.

Let’s look at the roles that exist in the Snowflake account we created in preparation for creating new custom roles. The command to show roles will return a list of the role names as well as some additional information. In [Figure 4-13](#), for the USERADMIN role, we see the “is_current” status is Y and the “is_inherited” is Y for the PUBLIC role. This means that the USERADMIN role is the role currently in use and above the public role in the hierarchy, thus, it will inherit any privileges assigned to the PUBLIC role:

```
USE ROLE USERADMIN;
SHOW ROLES;
```

Results Data Preview

✓ Query_ID SQL 43ms 5 rows

Row	created_on	name	is_default	is_current	is_inherited
1	2021-03-13 11:45:38.7...	ACCOUNTADMIN	N	N	N
2	2021-03-13 11:45:38.7...	PUBLIC	N	N	Y
3	2021-03-13 11:45:38.8...	SECURITYADMIN	N	N	N
4	2021-03-13 11:45:38.8...	SYSADMIN	N	N	N
5	2021-03-13 11:45:38.8...	USERADMIN	N	Y	N

Figure 4-13. Showing USERADMIN Role as the Current User

As shown in Figure 4-14, the Security Administrator role inherits privileges from the USERADMIN and the PUBLIC role:

```
USE ROLE SECURITYADMIN;
SHOW ROLES;
```

Results Data Preview

✓ Query_ID SQL 59ms 5 rows

Row	created_on	name	is_default	is_current	is_inherited
1	2021-03-13 11:45:38.7...	ACCOUNTADMIN	N	N	N
2	2021-03-13 11:45:38.7...	PUBLIC	N	N	Y
3	2021-03-13 11:45:38.8...	SECURITYADMIN	N	Y	N
4	2021-03-13 11:45:38.8...	SYSADMIN	N	N	N
5	2021-03-13 11:45:38.8...	USERADMIN	N	N	Y

Figure 4-14. Showing SECURITYADMIN as the Current User

Functional-Level Business & IT Roles

You'll notice that in Figure 4-15, I chose to use suffixes to differentiate between levels, such as Sr for Senior Analyst and Jr for Junior Analyst. I chose to use prefixes for

roles where different environments might be needed, such as SBX for Sandbox, DEV for Development, and PRD for Production Environment.



Figure 4-15. Functional-Level Custom Roles

We use the USERADMIN role to create our 10 functional custom roles:

```
USE ROLE USERADMIN;
CREATE OR REPLACE ROLE DATA_SCIENTIST;
CREATE OR REPLACE ROLE ANALYST_SR;
CREATE OR REPLACE ROLE ANALYST_JR;
CREATE OR REPLACE ROLE DATA_EXCHANGE_ASST;
CREATE OR REPLACE ROLE ACCOUNTANT_SR;
CREATE OR REPLACE ROLE ACCOUNTANT_JR;
CREATE OR REPLACE ROLE PRD_DBA;
CREATE OR REPLACE ROLE DATA_ENGINEER;
CREATE OR REPLACE ROLE DEVELOPER_SR;
CREATE OR REPLACE ROLE DEVELOPER_JR;
SHOW ROLES;
```

By using the SHOW command, you'll find that none of the custom roles have been granted to other roles. It is otherwise important to have all custom roles assigned to another role in the hierarchy with the top custom role being assigned to either the SYSADMIN role or the ACCOUNTADMIN role, unless there is a business need to isolate a custom role. Later in this chapter, we'll complete the hierarchy of custom roles by assigning the custom roles to a system-defined role.

System-Level Service Account and Object Access Roles

For System-Level custom roles, as shown in [Figure 4-16](#), I've created two different types of roles. The service account roles are typically those roles used for data loading

or connecting to visualization tools. The object access roles are roles for which data access privileges, such as the ability to view the data in a particular schema or to insert data into a table, will be granted. These object access roles then will be assigned to other roles higher up in the hierarchy.

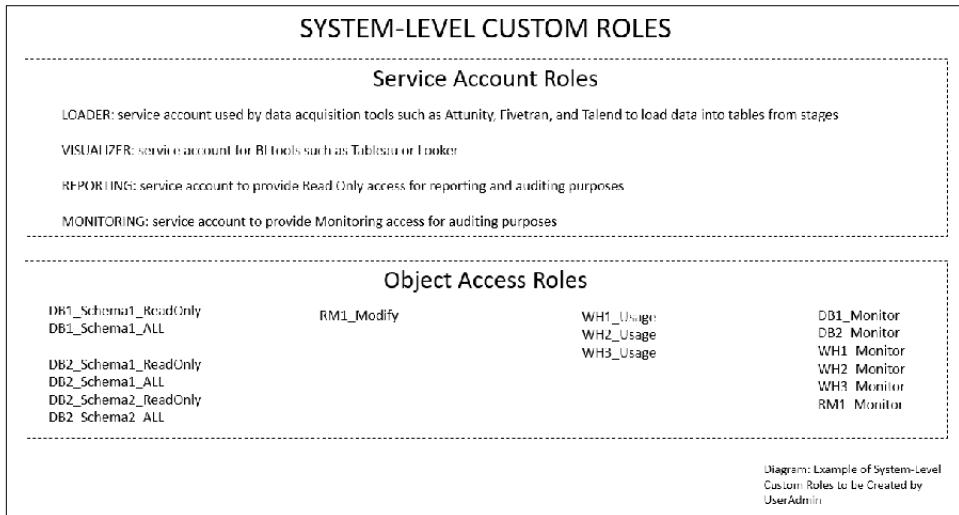


Figure 4-16. System-Level Custom Roles

First, let's use the USERADMIN role to create system service account roles:

```
USE ROLE USERADMIN;
CREATE OR REPLACE ROLE LOADER;
CREATE OR REPLACE ROLE VISUALIZER;
CREATE OR REPLACE ROLE REPORTING;
CREATE OR REPLACE ROLE MONITORING;
```

Next, create the system object access roles:

```
USE ROLE USERADMIN;
CREATE OR REPLACE ROLE DB1_SCHEMA1_READONLY;
CREATE OR REPLACE ROLE DB1_SCHEMA1_ALL;
CREATE OR REPLACE ROLE DB2_SCHEMA1_READONLY;
CREATE OR REPLACE ROLE DB2_SCHEMA1_ALL;
CREATE OR REPLACE ROLE DB2_SCHEMA2_READONLY;
CREATE OR REPLACE ROLE DB2_SCHEMA2_ALL;
CREATE OR REPLACE ROLE RM1_MODIFY;
CREATE OR REPLACE ROLE WH1_USAGE;
CREATE OR REPLACE ROLE WH2_USAGE;
CREATE OR REPLACE ROLE WH3_USAGE;
CREATE OR REPLACE ROLE DB1_MONITOR; CREATE OR REPLACE ROLE DB2_MONITOR; CREATE OR REPLACE ROLE WH1_MONITOR;
```

Role Hierarchy Assignments: Assign Roles to Other Roles

As I covered earlier, we want to assign the object access roles to other roles in the role hierarchy. As part of [Figure 4-17](#), you'll see that it is necessary to also assign a warehouse to any role that we expect would need the ability to run queries.

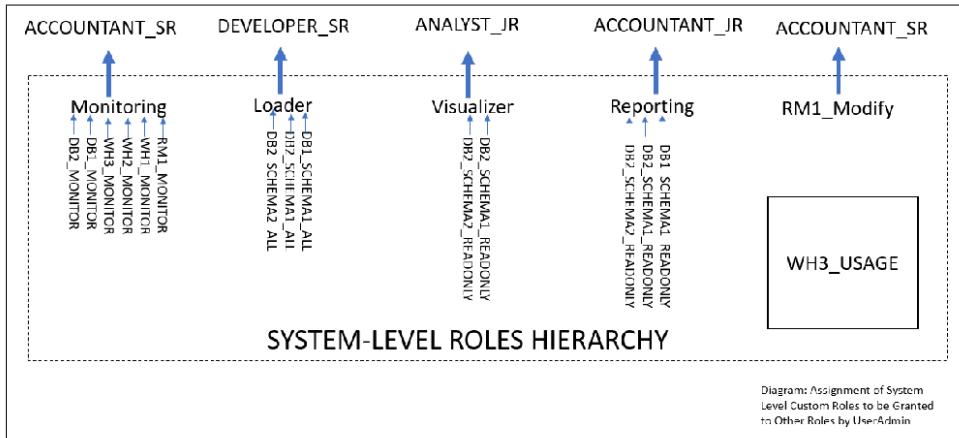


Figure 4-17. System-Level Roles Hierarchy

We'll now complete the system-Level role hierarchy assignments:

```
USE ROLE USERADMIN;
GRANT ROLE RM1_MONITOR TO ROLE MONITORING; GRANT ROLE WH1_MONITOR TO ROLE MONITORING; GRANT ROLE WH2_MONITOR TO ROLE MONITORING;
GRANT ROLE DB1_SCHEMA1_ALL TO ROLE LOADER; GRANT ROLE DB2_SCHEMA1_ALL TO ROLE LOADER; GRANT ROLE DB2_SCHEMA2_ALL TO ROLE LOADER;
GRANT ROLE DB2_SCHEMA1_READONLY TO ROLE VISUALIZER; GRANT ROLE DB2_SCHEMA2_READONLY TO ROLE VISUALIZER;
GRANT ROLE DB1_SCHEMA1_READONLY TO ROLE REPORTING; GRANT ROLE DB2_SCHEMA1_READONLY TO ROLE REPORTING;
GRANT ROLE MONITORING TO ROLE ACCOUNTANT_SR;
GRANT ROLE LOADER TO ROLE DEVELOPER_SR;
GRANT ROLE VISUALIZER TO ROLE ANALYST_JR;
GRANT ROLE REPORTING TO ROLE ACCOUNTANT_JR;
GRANT ROLE RM1_MODIFY TO ROLE ACCOUNTANT_SR;
```

Completing the functional-level role hierarchy assignments means that we'll also want to assign the top-level custom role to either the SYSADMIN or the ACCOUNTADMIN role as a final step in the hierarchy assignment process. As before, we'll also want to assign a warehouse to any role that we expect would need the ability to run queries. This is demonstrated in [Figure 5-18](#).

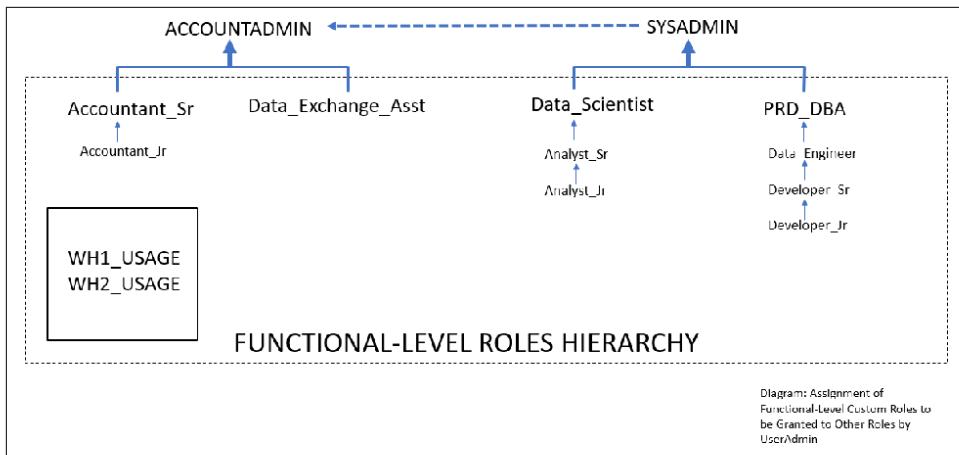


Figure 4-18. Functional-Level Roles Hierarchy

The USERADMIN role is used to complete the functional role hierarchy assignments:

```

USE ROLE USERADMIN;
GRANT ROLE ACCOUNTANT_JR TO ROLE ACCOUNTANT_SR;
GRANT ROLE ANALYST_JR TO ROLE ANALYST_SR;
GRANT ROLE ANALYST_SR TO ROLE DATA_SCIENTIST;
GRANT ROLE DEVELOPER_JR TO ROLE DEVELOPER_SR;
GRANT ROLE DEVELOPER_SR TO ROLE DATA_ENGINEER;
GRANT ROLE DATA_ENGINEER TO ROLE PRD_DBA;
GRANT ROLE ACCOUNTANT_SR TO ROLE ACCOUNTADMIN;
GRANT ROLE DATA_EXCHANGE_ASST TO ROLE ACCOUNTADMIN;
GRANT ROLE DATA_SCIENTIST TO ROLE SYSADMIN;
GRANT ROLE PRD_DBA TO ROLE SYSADMIN;

```

Next, be sure to grant usage of Warehouse1 directly to IT roles and usage of Warehouse2 to business

roles:

```

GRANT ROLE WH1_USAGE TO ROLE DEVELOPER_JR;
GRANT ROLE WH1_USAGE TO ROLE DEVELOPER_SR;
GRANT ROLE WH1_USAGE TO ROLE DATA_ENGINEER;
GRANT ROLE WH1_USAGE TO ROLE PRD_DBA;
GRANT ROLE WH2_USAGE TO ROLE ACCOUNTANT_JR;
GRANT ROLE WH2_USAGE TO ROLE ACCOUNTANT_SR;
GRANT ROLE WH2_USAGE TO ROLE DATA_EXCHANGE_ASST;
GRANT ROLE WH2_USAGE TO ROLE ANALYST_JR;
GRANT ROLE WH2_USAGE TO ROLE ANALYST_SR;
GRANT ROLE WH2_USAGE TO ROLE DATA_SCIENTIST;

```

Granting Privileges to Roles

Snowflake privileges are a defined level of access to a securable object. The granularity of access can be granted by using different distinct privileges and privileges can also be revoked if necessary. Each securable object has a specific set of privileges which can be granted on it and for existing objects, these privileges must be granted on the individual object. To make grant management more flexible and simpler, future grants can be assigned on objects created in a schema. I've elected here to assign the following privileges to the roles shown in Figure 5-19. These particular privileges are privileges that only the Account Administrator can assign.

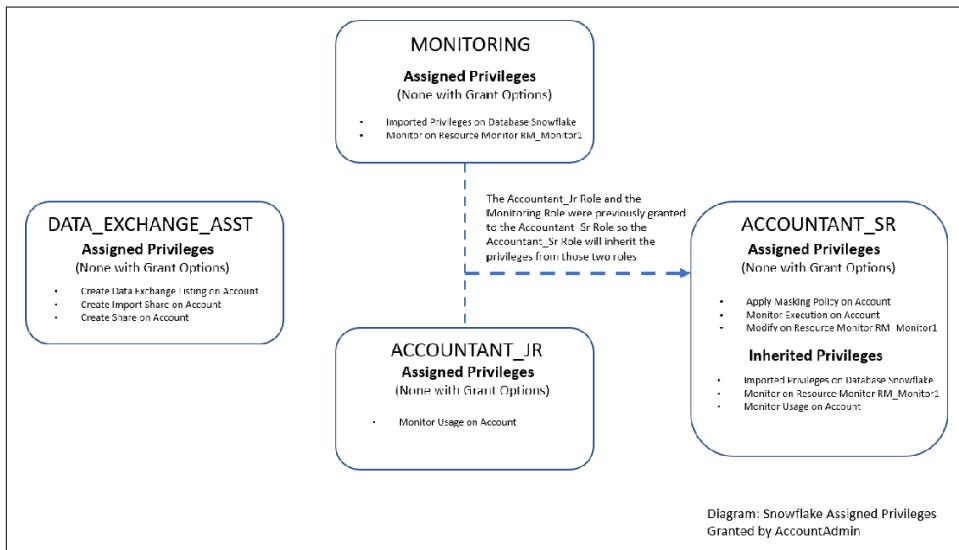


Figure 4-19. Snowflake Assigned Privileges Granted by ACCOUNTADMIN Role

The ACCOUNTADMIN role is required to grant direct global privileges to the functional Roles. We also will need to use the ACCOUNTADMIN role to grant privileges to custom roles that only the Account Administrator can grant:

```
USE ROLE ACCOUNTADMIN;
GRANT CREATE DATA EXCHANGE LISTING ON ACCOUNT TO ROLE DATA_EXCHANGE_ASST;
GRANT IMPORT SHARE ON ACCOUNT TO ROLE DATA_EXCHANGE_ASST;
GRANT CREATE SHARE ON ACCOUNT TO ROLE DATA_EXCHANGE_ASST;
GRANT IMPORTED PRIVILEGES ON DATABASE SNOWFLAKE TO ROLE MONITORING;
GRANT MONITOR ON RESOURCE MONITOR RM_MONITOR1 TO ROLE MONITORING;
GRANT MONITOR USAGE ON ACCOUNT TO ROLE ACCOUNTANT_JR;
GRANT APPLY MASKING POLICY ON ACCOUNT TO ROLE ACCOUNTANT_SR;
GRANT MONITOR EXECUTION ON ACCOUNT TO ROLE ACCOUNTANT_SR;
GRANT MODIFY ON RESOURCE MONITOR RM_MONITOR1 TO ROLE ACCOUNTANT_SR;
```

A number of different custom roles need privileges to interact with data in objects and need the ability to use a warehouse to make that interaction. For the ability to view data in a table, a role needs privileges to use the database and schema in which the table resides as well as the ability to use the SELECT command on the table. The privileges will be assigned for any existing objects in the schema when these privileges are granted. We'll also want to consider assigning FUTURE GRANT privileges so that the role can access tables created in the future. Future grants can only be assigned by the ACCOUNTADMIN, therefore we'll have to assign future grant access in a later step. Notice in [Figure 4-20](#) that the object monitoring privilege is set at the database level, thus, the role will be able to monitor both databases we created and all objects below the databases in the hierarchy.



Figure 4-20. Assigned Privileges Granted by SYSADMIN to Access Object Roles

Notice that the System Administrator role, not the User Administrator role, is required to grant privileges. Use the SECURITYADMIN Role to Grant Direct assigned privileges to system-level object access roles:

```
USE ROLE SYSADMIN;
GRANT USAGE ON DATABASE DB1 TO ROLE DB1_SCHEMA1_READONLY;
GRANT USAGE ON DATABASE DB2 TO ROLE DB2_SCHEMA1_READONLY;
GRANT USAGE ON DATABASE DB2 TO ROLE DB2_SCHEMA2_READONLY;
GRANT USAGE ON SCHEMA DB1.DB1_SCHEMA1 TO ROLE DB1_SCHEMA1_READONLY; GRANT USAGE ON SCHEMA DB2.DB2_SCHEMA1 TO ROLE DB1_SCHEMA1_READONLY;
GRANT SELECT ON ALL TABLES IN SCHEMA DB1.DB1_SCHEMA1 TO ROLE DB1_SCHEMA1_READONLY;
GRANT SELECT ON ALL TABLES IN SCHEMA DB2.DB2_SCHEMA1 TO ROLE DB2_SCHEMA1_READONLY;
GRANT SELECT ON ALL TABLES IN SCHEMA DB2.DB2_SCHEMA2 TO ROLE DB1_SCHEMA1_READONLY;
```

```
GRANT ALL ON SCHEMA DB1.DB1_SCHEMA1 TO ROLE DB1_SCHEMA1_ALL; GRANT ALL ON SCHEMA DB2.DB2_SCHEMA1 TO ROLE DB2_SCHEMA1_ALL;
GRANT MONITOR ON DATABASE DB1 TO ROLE DB1_MONITOR;
GRANT MONITOR ON DATABASE DB2 TO ROLE DB2_MONITOR;
GRANT MONITOR ON WAREHOUSE WH_WAREHOUSE1 TO ROLE WH1_MONITOR; GRANT MONITOR ON WAREHOUSE WH_WAREHOUSE2 TO ROLE WH2_MONITOR;
GRANT USAGE ON WAREHOUSE WH_WAREHOUSE1 TO WH1_USAGE;
GRANT USAGE ON WAREHOUSE WH_WAREHOUSE2 TO WH2_USAGE;
GRANT USAGE ON WAREHOUSE WH_WAREHOUSE3 TO WH3_USAGE;
```

Use the ACCOUNTADMIN Role to Grant FUTURE Direct Assigned Privileges:

```
USE ROLE ACCOUNTADMIN;
GRANT SELECT ON FUTURE TABLES IN SCHEMA DB1.DB1_SCHEMA1 TO ROLE DB1_SCHEMA1_READONLY;
GRANT SELECT ON FUTURE TABLES IN SCHEMA DB2.DB2_SCHEMA1 TO ROLE DB2_SCHEMA1_READONLY;
GRANT SELECT ON FUTURE TABLES IN SCHEMA DB2.DB2_SCHEMA2 TO ROLE DB2_SCHEMA2_READONLY;
GRANT SELECT ON FUTURE TABLES IN SCHEMA DB1.DB1_SCHEMA1 TO ROLE DB1_SCHEMA1_ALL;
GRANT SELECT ON FUTURE TABLES IN SCHEMA DB2.DB2_SCHEMA1 TO ROLE DB2_SCHEMA1_ALL;
GRANT SELECT ON FUTURE TABLES IN SCHEMA DB2.DB2_SCHEMA2 TO ROLE DB2_SCHEMA2_ALL;
```

Assigning Roles to Users

We created four users earlier in the chapter. Now, let's assign roles to each of those four users. It is possible to assign more than one role to each user but only one role can be used at any given time (i.e., assigned roles cannot be 'layered' or combined). Remember, that for any role that is assigned for which there are roles below in the hierarchy, the user has already inherited that role. For example, the Data Scientist role inherits the Analyst Sr and Analyst Jr role and will see those roles in their account. Accordingly, it would be redundant to assign either of those two roles to a user who is assigned the Data Scientist role.

```
USE ROLE USERADMIN;
GRANT ROLE DATA_EXCHANGE_ASST TO USER USER1;
GRANT ROLE DATA_SCIENTIST TO USER USER2;
GRANT ROLE ACCOUNTANT_SR TO USER USER3;
GRANT ROLE PRD_DBA TO USER USER4;
```

Testing and Validating Our Work

Now is when we get to test and validate the work we completed to establish our access control security.



When running any of the queries using any of the custom roles, you'll need to have a running warehouse to complete the queries. If, at any time, you receive an error message that there is no running warehouse, then you can always use the SHOW command to find a list of available warehouses for that role and the USE command to get the warehouse running.

For example, the role PRD_DBA has available both Warehouse1, which was assigned to the role, and Warehouse3, which was inherited. This is evidenced in [Figure 4-21](#).

```
USE ROLE PRD_DBA;
SHOW WAREHOUSES;
```

The screenshot shows a 'Results Data Preview' window with the following details:

- Query ID: 53ms
- SQL: SHOW WAREHOUSES;
- Number of rows: 2
- Table Headers: Row, name, state, type, size, min_cluster_size, max_cluster_size, started_clusters, running, queued, is_default, is_current.
- Data Rows:

Row	name	state	type	size	min_cluster_size	max_cluster_size	started_clusters	running	queued	is_default	is_current
1	WH_WAREHOUSE1	SUSPENDED	STANDARD	X-Small	1	1	0	0	0	N	N
2	WH_WAREHOUSE3	SUSPENDED	STANDARD	X-Small	1	1	0	0	0	N	Y

Figure 4-21. Current Snowflake Warehouses Available to Role PRD_DBA

So, to run any query, the PRD_DBA role use Warehouse3:

```
USE WAREHOUSE WH_WAREHOUSE3;
```

or, alternatively, can use Warehouse1:

```
USE WAREHOUSE WH_WAREHOUSE1;
```

Access to the SNOWFLAKE database should be granted to a minimum number of people. The ACCOUNTADMIN role has access to the Snowflake table and earlier we used the ACCOUNTADMIN role to grant access to the Senior Accountant. Therefore, we would expect an error returned if the Junior Accountant attempted to access the table. This is confirmed in [Figure 4-22](#).

```
USE ROLE ACCOUNTANT_JR;
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY WHERE QUERY_TYPE =
'GRANT';
```

The screenshot shows a 'Results Data Preview' window with the following details:

- Query ID: 29ms
- SQL: SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY WHERE QUERY_TYPE = 'GRANT';
- Error Message: SQL compilation error: Shared database is no longer available for use. It will need to be re-created if and when the publisher makes it available again.

Figure 4-22. Error because Junior Accountant Role has not been granted access.

If the Senior Accountant ran the same query, the Sr Accountant would receive this query results shown in [Figure 4-23](#)

```
USE ROLE ACCOUNTANT_SR;
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY WHERE QUERY_TYPE =
'GRANT';
```

The screenshot shows a Snowflake interface with the 'Results' tab selected. The query executed was 'SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY WHERE QUERY_TYPE = 'GRANT''. The results show 337 rows, with the first few rows listed below:

Row	QUERY_ID	QUERY_TEXT	DATABASE_ID	DATABASE_NAME
1	019ae281-00d0-1e92-0000-00000096d118	GRANT USAGE ON FILE FORMAT csv_dq T...	2	UTIL_DB
2	019ae281-0079-2d72-0000-000096d11051	GRANT IMPORTED PRIVILEGES ON DATABASE...	1	SNOWFLAKE_SAMPLE...
3	019ae281-004b-ca9d-0000-00000096d128	GRANT USAGE ON FILE FORMAT psv_dq T...	2	UTIL_DB
4	019ae281-00c0-a4a3-0000-00000096d134	GRANT USAGE ON FILE FORMAT tsv_dq T...	2	UTIL_DB
5	019ae281-00bb-4152-0000-00000096d130	GRANT USAGE ON FILE FORMAT soft_dq T...	2	UTIL_DB
6	019ae281-0038-20f4-0000-000096d11045	GRANT USAGE ON DATABASE demo_db T...	3	DEMO_DB

Figure 4-23. Results Shown because Senior Account has been granted access

Right now, our Snowflake account has no tables created by us. Whenever the SYSADMIN role creates a new table, it must assign other roles the necessary privileges for that table or else they can't access the table. However, we used a future grants option earlier to grant access to any future objects, like tables, that we created in the three schemas. Therefore, no action is needed to assign privileges on a newly created table. Let's test to see if the future grants privilege we assigned will work as intended.

First, you can see that no tables currently exist in the DB1 Schema:

```
USE ROLE SYSADMIN;
SHOW DATABASES;
USE SCHEMA DB1_SCHEMA1;
SHOW TABLES;
```

Now, we'll create a simple table and confirm that the table was created, as shown in figure 5-24.

```
CREATE OR REPLACE TABLE DB1.DB1_SCHEMA1.TABLE1 (a varchar);
INSERT INTO TABLE1 VALUES ('A');
SHOW TABLES;
```

The screenshot shows the Snowflake UI interface. At the top, there are tabs for 'Results' (which is selected) and 'Data Preview'. Below the tabs, there is a green checkmark icon next to 'Query ID', followed by 'SQL', '65ms', a progress bar, and '1 rows'. There is also a 'Filter result...' input field, a download icon, and a 'Copy' button. A table below the controls has columns: Row, created_on, name, database_name, schema_name, and kind. The data row shows: Row 1, created_on 2021-03-19 ..., name TABLE1, database_name DB1, schema_name DB1_SCHEM1, and kind TABLE.

Row	created_on	name	database_name	schema_name	kind
1	2021-03-19 ...	TABLE1	DB1	DB1_SCHEM1	TABLE

Figure 4-24. Confirmation that Table was Created

Next, we'll test to see if the REPORTING role can access the table we just created. Based on the future grants privileges we assigned to the role that was then assigned to the REPORTING role, we expect that the REPORTING role should be able to access the table. This is confirmed in [Figure 4-25](#).

```
USE ROLE REPORTING;
SELECT * FROM DB1.DB1_SCHEM1.TABLE1;
```

The screenshot shows the Snowflake UI interface. At the top, there are tabs for 'Results' (selected) and 'Data Preview'. Below the tabs, there is a green checkmark icon next to 'Query ID', followed by 'SQL', '292ms', a progress bar, and '1 rows'. There is also a 'Filter result...' input field, a download icon, and a 'Copy' button. A table below the controls has columns: Row and A. The data row shows: Row 1, A.

Row	A
1	A

Figure 4-25. Confirmation that the REPORTING Role can access the newly created table

We did not grant access to DB1 to the VISUALIZER role so that role will not be able to see the table. This is evidenced in [Figure 4-26](#).

```
USE ROLE VISUALIZER;
SELECT * FROM DB1.DB1_SCHEM1.TABLE1;
```

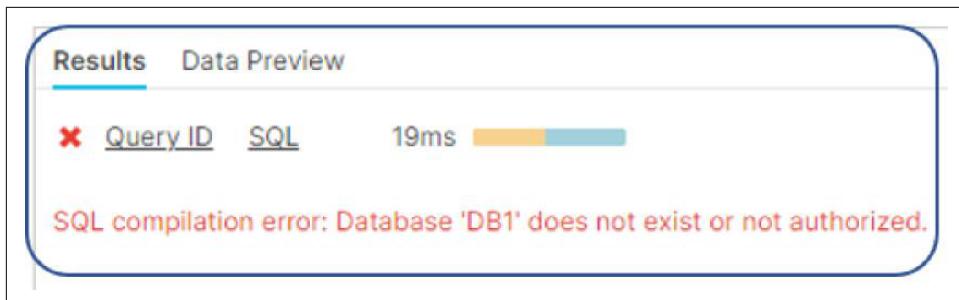


Figure 4-26. VISUALIZER Role does not have access to the newly created table.

Here are some additional queries you can try on your own that will give you some useful information:

```
USE ROLE ACCOUNTANT_SR;
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.GRANTS_TO_USERS;
SHOW GRANTS ON ACCOUNT;
SHOW GRANTS ON DATABASE DB1;
SHOW GRANTS OF ROLE ANALYST_SR;
SHOW FUTURE GRANTS IN DATABASE DB1;
SHOW FUTURE GRANTS IN SCHEMA DB1.DB1_SCHEMA1;
```

Any of the global privileges, privileges for account objects, and privileges for schemas can be revoked from a role. As an example, we'll have the Account Administrator grant a role to the junior analyst and then the USERADMIN will revoke that role.

```
USE ROLE ACCOUNTADMIN;
GRANT MONITOR USAGE ON ACCOUNT TO ROLE ANALYST_JR;
USE ROLE USERADMIN;
REVOKE MONITOR USAGE ON ACCOUNT FROM ROLE ANALYST_JR;
```

During a session, a user can change their role if they are assigned more than one role. When a user attempts to execute an action on an object, Snowflake compares the privileges required to complete the action against any privileges that the current role was inherently given, assigned, or was inherited. The action is allowed if the role in the session has the necessary privileges.

User Management

A *User object* in Snowflake stores all the information about a user, including their login name, password, and defaults. A Snowflake user can be a person or a program. From previous discussion, we know that Snowflake users are created and managed by the USERADMIN system-defined role. The user name is required and should be unique when creating a user. Even though all other properties are optional, it is a best practice to include many of them. At a minimum, include some basic details and

assign an initial password which you require the user to change it at the next login. As an example:

```
USE ROLE USERADMIN;
CREATE OR REPLACE USER USER10
PASSWORD='123'
LOGIN_NAME = ABARNETT
DISPLAY_NAME = AMY
FIRST_NAME = AMY
LAST_NAME = BARNETT
EMAIL = 'ABARNETT@COMPANY.COM'
MUST_CHANGE_PASSWORD=TRUE;
```

You can add new properties or change existing user properties by using the ALTER command. For example, you may have a user that you want to grant temporary access to your Snowflake account. That can be accomplished by setting up an expiration time for the user:

```
USE ROLE USERADMIN;
ALTER USER USER10 SET DAYS_TO_EXPIRY = 30;
```

Adding defaults for the user makes it easier for the user. You can make it so that the user has a default role or default warehouse, for example.



Adding a default warehouse for the user does not verify that the warehouse exists. We never created a Warehouse52, yet the code to assign that warehouse as a default will be executed successfully with no errors or warning:

```
USE ROLE USERADMIN;
ALTER USER USER10 SET DEFAULT_WAREHOUSE=WH_WAREHOUSE52;
```



We can attempt to assign a default role that does exist to a user. However, if we did not previously grant that role to the user then we cannot make it the default role. In other words, setting a default role does not assign the role to the user. The same thing applies for access to a database. When we attempt to assign defaults that have not previously been assigned to a user, no warning is given but the command is not actually executed successfully:

```
USE ROLE USERADMIN;
ALTER USER USER10 SET DEFAULT_ROLE=ANALYST_JR;
ALTER USER USER10 SET DEFAULT_NAMESPACE=DB1.DB1_SCHEMA;
```

To see for yourself, log out of your Snowflake account and log in as User10. There, you'll observe that only the Public role is available to User10, there are no warehouses

available to select, and only the three databases given to everyone are accessible to User10. This is shown in [Figure 4-27](#).

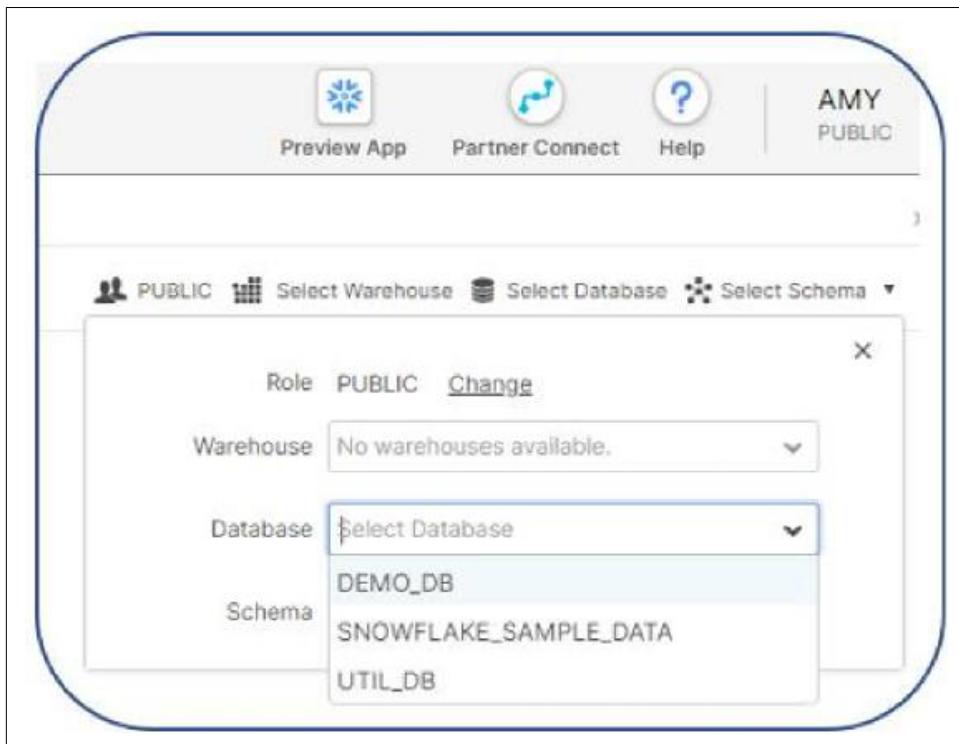


Figure 4-27. Warehouses currently available to User10

Be sure to log out and log back in as the Account administrator once you're done taking look around as User10.

Now we'll set some defaults for User10 that are correct, as shown in [Figure 4-28](#). We'll log back out of our admin account and log back in as user10 once we're done.

```
USE ROLE USERADMIN;
GRANT ROLE ACCOUNTANT_SR TO USER USER10;
ALTER USER USER10 SET DEFAULT_NAMESPACE=SNOWFLAKE.ACOUNT_USAGE;
ALTER USER USER10 SET DEFAULT_WAREHOUSE=WH_WAREHOUSE2;
ALTER USER USER10 SET DEFAULT_ROLE = ACCOUNTANT_SR;
```

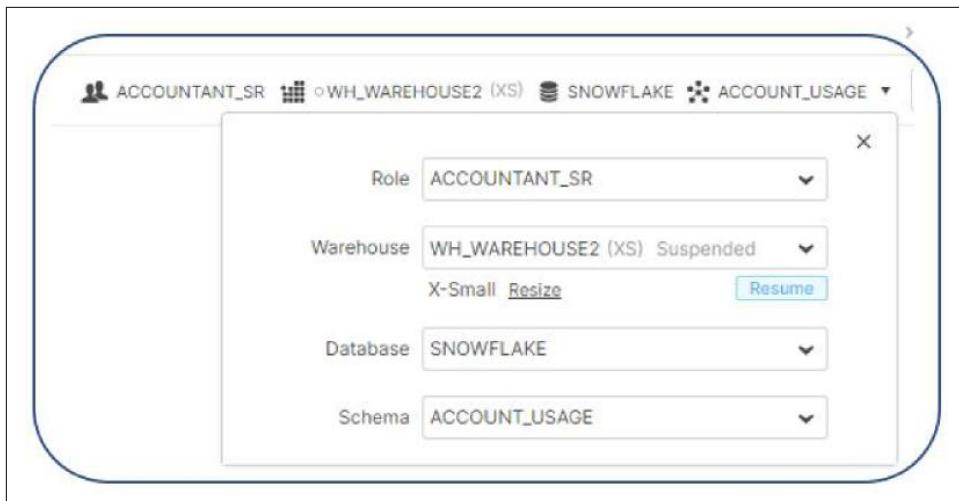


Figure 4-28. User10 has been granted the role of ACCOUNTANT_SR and assigned certain defaults

A common user management problem is users who are unable to log in to their account. The Snowflake system will lock out a user who failed to login successfully after five consecutive attempts. The Snowflake system will automatically clear the lock after 15 minutes. If the user cannot wait, the timer can immediately be reset:

```
USE ROLE USERADMIN;
ALTER USER USER10 SET MINS_TO_UNLOCK=0;
```

If the user forgot the password and needs for it to be reset, reset it this way.



The original way we used to create the password will not work because '123' is not an acceptable password. If we try to use the '123' password, an error will be returned, as shown in [Figure 4-29](#).

```
USE ROLE USERADMIN;
ALTER USER USER10 SET PASSWORD = '123'
MUST_CHANGE_PASSWORD = TRUE;
```



Figure 4-29. Password rejected due to not achieving minimum length

The password submitted by the USERADMIN to reset a forgotten password must be at least 8 characters in length, and contain at least 1 digit, 1 uppercase letter and 1 lowercase letter:

```
USE ROLE USERADMIN;
ALTER USER USER10 SET PASSWORD = '123456Aa'
MUST_CHANGE_PASSWORD = TRUE;
```

Keep in mind that because the USERADMIN role rolls up to the SECURITYADMIN role which itself rolls up to the ACCOUNTADMIN role, the two higher roles can also run any command that the USERADMIN role can run. As an example:

```
USE ROLE SECURITYADMIN;
ALTER USER USER10 SET PASSWORD = '123456Bb'
MUST_CHANGE_PASSWORD = TRUE;
```

There may be a time when it is necessary to abort a user's currently running queries and to prevent the person from running any new queries. To accomplish this and to immediately lock the user out of Snowflake, use the following command:

```
USE ROLE USERADMIN;
ALTER USER USER10 SET DISABLED = TRUE;
```

To describe an individual user and get a listing of all the user's property values and default values, use the DESCRIBE command. The results are shown in Figure 4-30.

```
USE ROLE USERADMIN;
DESC USER USER10;
```

The screenshot shows a 'Results' tab in a Snowflake interface. At the top, it displays 'Query ID' and 'SQL' with a duration of '40ms'. Below this is a progress bar indicating '25 rows' have been processed. There are buttons for 'Filter result...', 'Copy', and a download icon. The main area is a table with the following data:

Row	property	value	default
16	DEFAULT_WAREHOUSE	WH_WAREHOUSE2	null
17	DEFAULT_NAMESPACE	SNOWFLAKE.ACCOUNT_USAGE	null
18	DEFAULT_ROLE	ACCOUNTANT_SR	null
19	EXT_AUTHN_DUO	false	false
20	EXT_AUTHN_UID	null	null
21	MINS_TO_BYPASS_MFA	null	null

Figure 4-30. Description of User10

To reset one of the user's property values back to the default value, you can use this command:

```
USE ROLE USERADMIN;
ALTER USER USER10 SET DEFAULT_WAREHOUSE = DEFAULT;
USE ROLE USERADMIN;
DESC USER USER10;
```

We discovered earlier that the SECURITYADMIN role, not the USERADMIN role, has inherent privileges to see a list of all users. See [Figure 4-31](#) for the results.

```
USE ROLE SECURITYADMIN;
SHOW USERS;
```

Results Data Preview

✓ [Query ID](#) [SQL](#) 41ms 7 rows

Filter result... [Download](#) [Copy](#)

Row	name	created_on	login_name	display_name
2	SNOWFLAKE	2021-03-13 ...	SNOWFLAKE	SNOWFLAKE
3	USER1	2021-03-21 ...	ARNOLD	USER1
4	USER10	2021-03-21 ...	ABARNETT	AMY
5	USER2	2021-03-21 ...	BEATRICE	USER2
6	USER3	2021-03-21 ...	COLLIN	USER3
7	USER4	2021-03-21 ...	DIEDRE	USER4

Figure 4-31. List of Users in the Snowflake Account

You'll notice that the list of users includes the initial user that set up the account and all the users that you created. In addition, there is one more user that came with the account. The "SNOWFLAKE" user is a special user that is only used by Snowflake support with the permission of the ACCOUNTADMIN when there is a need to troubleshoot account issues.



It is possible to delete this "SNOWFLAKE" user, but once deleted, you can't simply create another "SNOWFLAKE" user that can be used for support. Therefore, it is highly recommended that you do not delete this user.

Wildcards are supported when using the SHOW command. You can use the LIKE command with the underscore to match any single character and the percentage character to match any sequence of 0 or more characters. As an example:

```
USE ROLE SECURITYADMIN;
SHOW USERS LIKE 'USER%';
```

Just as users can be created, they can also be dropped. Notice that nothing needs to be done to anything prior to dropping the user, such as revoking their role. Just issue the DROP command:

```
USE ROLE USERADMIN;
DROP USER USER10;
```



The Account Administrator has access to a list of all users by querying the SNOWFLAKE.ACOUNT_USAGE.USERS table, including those who have been deleted.

Code Cleanup

Let's perform a code cleanup so that you can remove the objects in your Snowflake account in preparation for working on another chapter example.

Note that there is no need to drop objects in the hierarchy below the database before dropping the databases or to revoke the warehouse usage privileges before dropping the warehouses:

```
USE ROLE SYSADMIN;
DROP DATABASE DB1;
DROP DATABASE DB2;
SHOW DATABASES;
DROP WAREHOUSE WH_WAREHOUSE1; DROP WAREHOUSE WH_WAREHOUSE2; DROP WAREHOUSE WH_WAREHOUSE3; SHOW WAREHO
```

Just as the ACCOUNTADMIN had to create Resource Monitors, the same role must be used to drop Resource Monitors:

```
USE ROLE ACCOUNTADMIN;
DROP RESOURCE MONITOR RM_MONITOR1;
SHOW RESOURCE MONITORS;
```

Next, the USERADMIN role will want to drop all the roles that were created. However, not all of the roles can be dropped by the USERADMIN role. Below is a list of those roles which can be dropped by the User Administrator role:

```
USE ROLE USERADMIN;
DROP ROLE DATA_SCIENTIST;
DROP ROLE DATA_EXHANGE_ASST;
DROP ROLE ANALYST_SR;
DROP ROLE ANALYST_JR;
DROP ROLE DATA_EXCHANGE_ASST;
DROP ROLE ACCOUNTANT_SR;
DROP ROLE ACCOUNTANT_JR
DROP ROLE PRD_DBA;
DROP ROLE DATA_ENGINEER;
DROP ROLE DEVELOPER_SR;
DROP ROLE DEVELOPER_JR;
DROP ROLE LOADER;
DROP ROLE VISUALIZER;
DROP ROLE REPORTING;
```

```
DROP ROLE MONITORING;
DROP ROLE RM1 MODIFY;
DROP ROLE WH1 USAGE;
DROP ROLE WH2 USAGE;
DROP ROLE WH3 USAGE;
DROP ROLE DB1_MONITOR;
DROP ROLE DB2_MONITOR;
DROP ROLE WH1_MONITOR;
DROP ROLE WH2_MONITOR;
DROP ROLE WH3_MONITOR;
DROP ROLE RM1_MONITOR;
SHOW ROLES;
```

While we used the USERADMIN role to drop most of the custom roles, I mentioned that we cannot drop all of the custom roles. Let's see what happens if we attempt to drop this custom role using the USERADMIN role.

```
USE ROLE USERADMIN;
DROP ROLE DB1_SCHEMA1_READONLY;
```

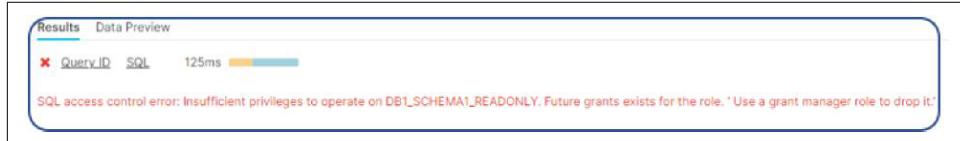


Figure 4-32. USERADMIN Role does not have privileges necessary to drop roles where future grants have been assigned

We see that for any object for which future grants were assigned, the USERADMIN role did not have the necessary privileges to drop the role, as shown in Figure 4-32. Instead, a SECURITYADMIN role or higher is required to drop the roles. Let's use the SECURITYADMIN role to finish dropping the roles and confirm that all custom roles have now been dropped:

```
USE ROLE SECURITYADMIN;
DROP ROLE DB1_SCHEMA1_READONLY;
DROP ROLE DB1_SCHEMA1_ALL;
DROP ROLE DB2_SCHEMA1_READONLY;
DROP ROLE DB2_SCHEMA1_ALL;
DROP ROLE DB2_SCHEMA2_READONLY;
DROP ROLE DB2_SCHEMA2_ALL;
SHOW ROLES;

And, now, we are able to drop the users we created:
USE ROLE USERADMIN;
DROP USER USER1;
DROP USER USER2;
DROP USER USER3;
DROP USER USER4;
```

We can confirm that all the users have now been dropped:

```
USE ROLE SECURITYADMIN;  
SHOW USERS;
```

Exercises to Test Your Knowledge

The following exercises are based on the coverage in this chapter about user management and object security.

1. How would you define inherent privileges? Give a few examples.
2. Can you name all the Snowflake system-defined roles? To which of the system-defined roles should most custom roles be assigned to in the role hierarchy?
3. What are some of the important considerations to keep in mind when assigning defaults, such as default role or default warehouse, to a user?
4. What is one thing that an Account Administrator can do that no other role can ever do?
5. Can you name the privileges a role needs in order to view the contents of a table?
6. What is unique about the SNOWFLAKE database that is included in the account?

Solutions to these exercises are available in Appendix A.

Managing Snowflake Account Costs

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 8th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

Organizations create business plans and budgets using the most current and accurate information possible. Knowing the amount of on-premises warehouse costs is relatively easy because, for the most part, the costs are fixed. A usage-based cost approach, such as Snowflake’s current pricing model, may initially seem intimidating because the actual costs are unknown. Indeed, understanding any usage-based cost model requires making sure you clearly understand in advance how costs are computed so you don’t experience runaway costs. Going beyond basic understanding of usage-based pricing, and positioning your organization to take advantage of it, can yield many positive results.

One of the greatest benefits of usage-based costing is that it provides an opportunity to understand how your organization is charged for storing and using data. This could lead to changes you can make that will instantly lower costs. Snowflake’s usage-based billing and Snowflake object tagging also makes it easy to assign appropriate costs to the appropriate accounts or departments. This means that there is more cost accountability. With more accountability at a granular level, there will likely be more

consideration about how resources are consumed and, thus, more action will be taken to become more efficient.

Snowflake's pricing model is straightforward and easy to understand. In addition to the many benefits that are possible with any cloud technology usage-based cost approach, Snowflake has unique functionality which can result in decreased costs. One example is zero-copy cloning which offers the ability to duplicate an object without actually creating a physical copy. This means there are no storage costs assessed for the clone. There also exists native functionality which will allow you to set limits on your credit consumption costs. These Snowflake resource monitors can be set to take actions such as notify or suspend consumption.

In conjunction with taking advantage of Snowflake's usage-based cost approach features to better manage costs, consideration of software delivery operations should be made from the outset. Fortunately, Snowflake has several ways to alleviate the unique challenges of database change management and to reduce software development costs.

Prep Work

Later in this chapter, we're going to create some Snowflake resource monitors. Resource monitors help to monitor the usage of Snowflake virtual warehouses. To prepare for the resource monitors section, we'll need to create some Snowflake virtual warehouses. A best practice is to use the SYSADMIN role to create the new virtual warehouses, which we do first.

```
USE ROLE SYSADMIN;
CREATE OR REPLACE WAREHOUSE WH_VW2 WITH WAREHOUSE_SIZE = MEDIUM Auto_suspend = 300 Auto_resume = true
Initially_suspended=true;
CREATE OR REPLACE WAREHOUSE WH_VW3 WITH WAREHOUSE_SIZE = SMALL Auto_suspend = 300 Auto_resume = true
Initially_suspended=true;
CREATE OR REPLACE WAREHOUSE WH_VW4 WITH WAREHOUSE_SIZE = MEDIUM Auto_suspend = 300 Auto_resume = true
Initially_suspended=true;
CREATE OR REPLACE WAREHOUSE WH_VW5 WITH WAREHOUSE_SIZE = SMALL Auto_suspend = 300 Auto_resume = true
Initially_suspended=true;
USE ROLE SYSADMIN;
CREATE OR REPLACE WAREHOUSE WH_VW6 WITH WAREHOUSE_SIZE = MEDIUM Auto_suspend = 300 Auto_resume = true
Initially_suspended=true;
```

Snowflake Monthly Bill

Multiple Snowflake accounts can be created and managed by an organization. Each of the Snowflake accounts incurs fees that are summarized in an organization's monthly bill. Your total Snowflake monthly bill includes three distinct charges per account: storage fees, data transfer costs and credits consumed. We'll review an explanation of

each of these types of charges and you can also follow along by looking in your Snowflake org to see your current charges.

Once in Snowflake, with the role set to ACCOUNTADMIN, click on *Account* then on *Usage*, as shown in [Figure 5-1](#).

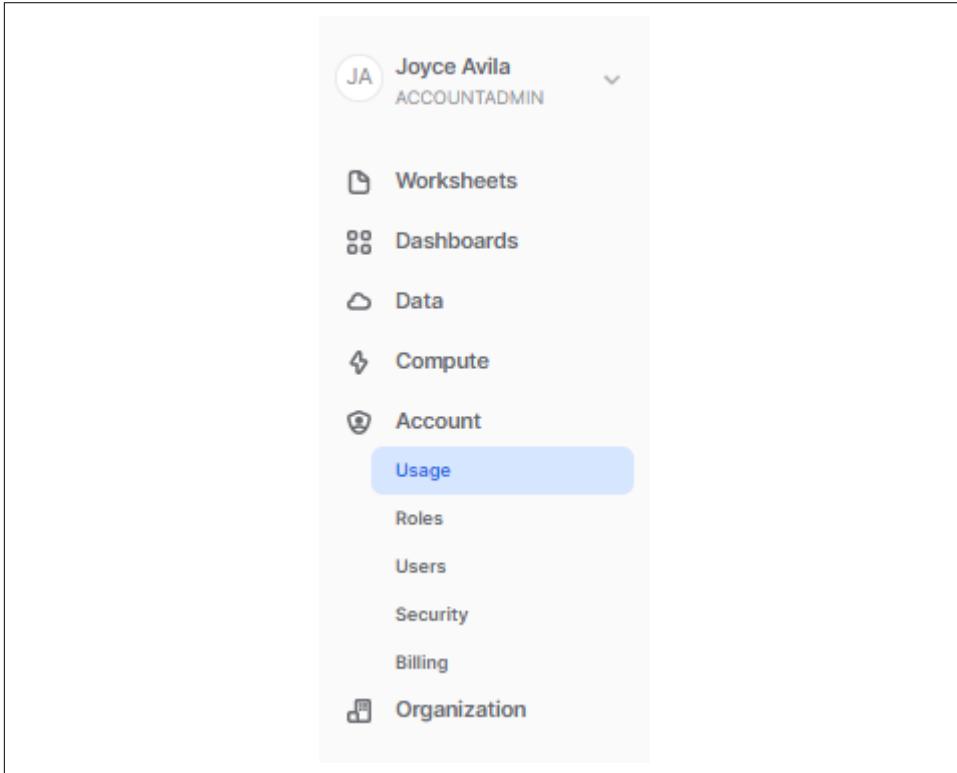


Figure 5-1. Snowflake Menu option for Account > Usage

In the Account Usage section, you'll have the option to select Organization, Consumption, Storage, or Transfers. The Organization information applies if you currently have more than one Snowflake account.

In this section going forward, we'll be focusing on billing at the account level. For account-level billing, we're going to start getting an understanding of storage fees and then review transfer costs. Finally, we'll take an in-depth look at Snowflake credits. For the most current Snowflake service consumption table pricing, you can visit this link at <https://www.snowflake.com/legal-files/CreditConsumptionTable.pdf>

Storage Fees

Snowflake monthly storage fees are calculated based on the average terabytes per month used, based on periodic snapshots of all data stored in an account. You'll notice that capacity pricing offers steep discounts for upfront payments.

Note that while storage pricing is a consideration of which region to select for your Snowflake account, you'll also want to consider which region could minimize latency and could give you access to any required features you may need. It's important to note that if you decide later to move your data to a different region, data transfer costs will be incurred.



Try to make more use of transient tables as they are not maintained in the history tables which in turn reduces the data storage costs for history tables. More information about transient tables can be found in Chapter 3.

Data Transfer Costs

The next type of Snowflake costs is data transfer costs. These are assessed when data stored in Snowflake is transferred from one region to another, within the same cloud platform, or when data is transferred out of the cloud.



Snowflake does not charge *data ingress* fees. The *Data egress* fees you pay are a reimbursement to Snowflake coming from the three cloud data providers applying data egress charges when data is transferred.

Data egress charges apply only when unloading data from Snowflake, when external functions are used, or when replicating data to a Snowflake account in a region or cloud platform different from where your primary Snowflake account is hosted.

Credits Consumed

The consumption of resources on Snowflake is paid for with Snowflake credits, a unit of measure. That measure is the equivalent of one hour of usage by a single compute node. You can purchase credits as you go, on demand. The price of on demand credits is dependent on which cloud provider, region, and service edition is selected. Alternatively, you can pre-purchase credits at a negotiated discounted rate.

A snowflake credit is consumed only when you use resources. Snowflake resources are used, or consumed, in one of three ways: running virtual warehouses, when the cloud services layer is performing work, or using serverless features such as automatic clustering or maintaining materialized views.

The credits consumed for Snowflake running virtual warehouses are based on the size of the warehouse. For example, the credit consumption for an X-Small virtual warehouse is billed at one Snowflake credit per hour of usage.



Optimally managing the operational status of your compute clusters with features such as auto suspension and auto resume can help manage the Snowflake consumption credits billed. More information about creating and managing virtual warehouses can be found in Chapter 2.

The credits consumed for the cloud services layer are free up to 10% of consumed daily virtual warehouse compute. Anything above the 10% daily threshold for cloud services layer consumption is billed and appears in your Snowflake account as a charge to the CLOUD_SERVICES_ONLY virtual warehouse.

The credits consumed for using serverless features are billed at a distinctly different rate, depending on the serverless feature.

Snowflake's many native features, such as micro-partitions and automatic compute clustering, lessen the need for credit consumption, which in turn lessens overall costs. Snowflake's highly elastic compute and per-second billing model also helps keep credit consumption charges low. These features come to us from Snowflake out of the box; however, there are also things we can do to take on a more proactive role in understanding and managing our Snowflake consumption credit usage.

Monitoring Virtual Warehouse Usage and Reducing Costs

Snowflake's simple usage-based pricing is easy to understand. It's straightforward and you pay only for what you use. One goal, then, would be to monitor the amount you use.

Creating and Managing Resource Monitors

A Snowflake virtual warehouse consumes credits when it's in use. The number of Snowflake credits consumed by a virtual warehouse depends on two factors – the size of the virtual warehouse and how long it runs.

There is a one-to-one relationship between the number of servers in a virtual warehouse cluster and the number of credits billed for each full hour the virtual warehouse runs. We previously learned in Chapter 2 that virtual warehouse sizes increase by a factor of two up to the 4X-Large size. As such, an X-small virtual warehouse consumes one Snowflake credit per hour whereas a 4X-Large virtual warehouse consumers 128 credits for each hour the virtual warehouse runs.



Virtual warehouse consumption is billed on a per-second basis, with a 60 second minimum.

Monitoring credit consumption will help prevent unexpected credit usage, and as such, provides a means with which to control compute costs. Snowflake provides the necessary tools, known as resource monitors, to help automate the task of monitoring credit consumption and to trigger actions based on preset thresholds. A resource monitor is a first-class Snowflake object that can be applied to both user-managed virtual warehouses and virtual warehouses used by cloud services.

A Snowflake resource monitor has the following properties:

- Credit Quota: number of Snowflake credits
- Credit Usage: total Snowflake credits consumed during billing period
 - Monitor Level: Account or specific virtual warehouse(s)
 - Schedule
 - Frequency: daily, weekly, monthly, yearly, never
 - Start and End, although it is uncommon to have an end value
- Triggers (Actions)
 - Notify
 - Notify & Suspend
 - Notify & Suspend Immediately

Conceptually, resource monitor planning looks like what is shown in [Figure 5-2](#).

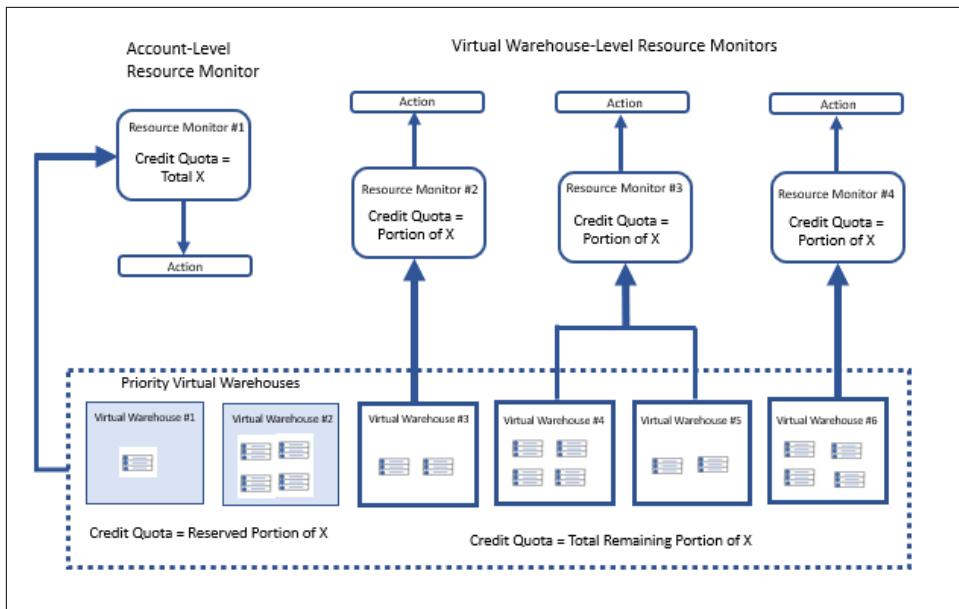


Figure 5-2. Resource Monitor Planning

The recommended steps for resource monitor planning are as follows:

1. Determine the overall account level credit budget that will be used to establish an account-level resource monitor. In [Figure 5-2](#), this is the *Total X* value.
2. Define priority virtual warehouses and assign a portion of the credit budget, referred to as the *Reserved Portion of X*, for all priority virtual warehouses. Priority virtual warehouses are those warehouses that you would not want to be suspended.
3. Assign the remaining portion of the budget to all other non-priority virtual warehouse by way of creating warehouse-level resource monitor(s). Doing so will limit their individual credit usage to ensure there is enough budget reserved for the priority virtual warehouses. Note that there isn't a limit to the number of resource monitors you can create. If you want, you can assign a resource monitor to each individual virtual warehouse.
4. Determine the schedule, if you are not going to use default settings.
5. Decide on the actions for each of the resource monitors. Most likely, you'll want to establish the account-level resource monitor with a Notify-only action so that the priority virtual warehouses do not get suspended.

Resource Monitor Credit Quota

An account-level resource monitor credit quota, stated in terms of Snowflake credits, is based on the entire amount of cloud service usage. However, the account-level credit quota does not consider the daily 10% adjustment for cloud services. Only an account-level resource monitor can monitor credit usage for virtual warehouses that provide cloud services. Resource monitors, at the virtual warehouse-level, can only be created for user-created and managed virtual warehouses.



Whenever a credit quota threshold is reached for a virtual warehouse whose resource monitor action is set to Suspend Immediately, additional Snowflake credits might still be consumed. Thus, you might want to consider utilizing buffers in quota thresholds for trigger actions if you want to strictly enforce your quotes. For example, set your threshold for 95% instead of 100% to ensure that your credit usage won't exceed your actual quota.

Resource Monitor Credit Usage

As stated, resource monitors can be created at the account-level or the virtual warehouse-level.



Account-level resource monitors only control the credit usage of virtual warehouses created in your account; not Snowflake-provided warehouses for serverless features such as those used for Snowpipe, automatic clustering, or materialized views.

Snowflake resource monitors can be created to impose limits on the number of Snowflake credits that are consumed, either for a specified interval or for a date range.

For a default schedule, (1) a resource monitor starts tracking the assigned warehouse as soon as it is created and (2) the credit usage tracking resets to 0 at the beginning of each calendar month.

Customizing the schedule so that the resource monitor resets at a specific interval, such as daily or weekly, requires that you select *both* a frequency and a start time. Selecting an end time is optional.

Resource Monitor Notifications and Other Actions

Resource monitors can be created to trigger specific actions, such as sending alert notifications and/or suspending a virtual warehouse.



There is a limit of five *Notify* actions, one *Suspend* action, and one *Suspend Immediately* action for each resource monitor.

By default, resource monitor notifications are not enabled. ACCOUNTADMIN role users can receive notifications only if they provide a *verified* valid email address and enable notifications.

To enable notifications via the web interface, make sure your role is set to ACCOUNTADMIN. You can select Preferences and then Notifications. You'll need to choose your notification preference as Web, Email, or All. If you select Email or All, you'll be prompted to enter a valid email address.

The difference between the *Suspend* action and the *Suspend Immediately* action is whether statements being executed by the warehouse(s) will be allowed to complete or not.

When a trigger causes a virtual warehouse to suspend, the suspended warehouse cannot be resumed until one of the following conditions occurs:

- The resource monitor is no longer assigned to the warehouse or is dropped
- Either the credit quota for the monitor or the credit threshold for the trigger is increased
- A new monthly billing cycle starts

Resource Monitor Rules for Assignments

Resource monitors rules for assignments can be summarized as follows.

- An account-level resource monitor can be set at the account level to control credit usage for all warehouses in an account; however, there can be only one account-level resource monitor.
- There is a one-to-many relationship between virtual warehouses and warehouse-level resource monitors. Each virtual warehouse can be assigned to only one warehouse-level resource monitor; one resource monitor can be assigned to one or more virtual warehouses.
- The credit quota and credit usage properties are both required to create a resource monitor. If a schedule is not explicitly stated, then the default schedule applies. By default, the credit quota resets at the beginning of each calendar month. Setting triggers is optional.

- Before it can perform monitoring actions, a resource monitor must be assigned to a warehouse or account after it is created.

Let's use what we've learned thus far about resource monitors to create a resource monitor plan (Figure 5-3) for a large enterprise with about 10,000 employees. We'll implement the resource monitor plan in the next section.

Assume we want to limit our credit consumption to 5,000 credits. We'll need to create an account-level resource monitor where the credit quota equals 5,000 and we'll want to be notified when that credit quota has been reached. We'll elect to use the default schedule.

Out of the six virtual warehouses in our account, we've identified two priority virtual warehouses for which we want to allocate half our credit budget. As such, that is 2,500 credits being reserved from the 5,000 total credits. The other four virtual warehouses will need to split the remaining 2,500 credits and we'll create virtual warehouse-level resource monitors to notify and suspend whenever the credit quota has been reached. Additionally, we've decided to be notified when the second of our priority virtual warehouses reaches a credit quota of 1,500.

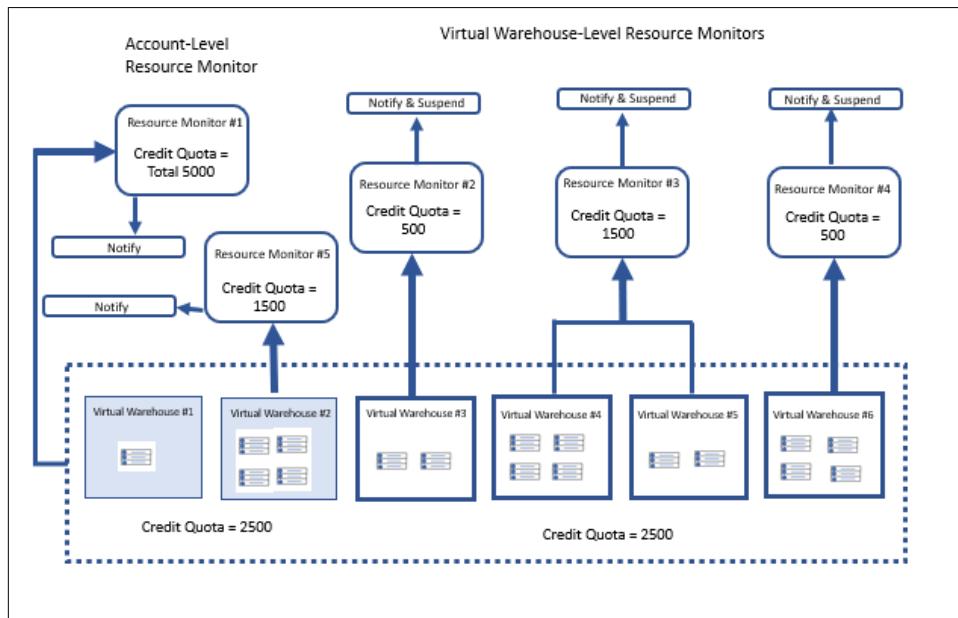


Figure 5-3. Example of an Implementation Plan of Resource Monitoring

If applicable, a virtual warehouse will be suspended if either the account-level resource monitor or the virtual warehouse-level resource monitor reaches its threshold and there are suspended actions associated with the resource monitors. In our example, though, we have established a Notify-only action for the account-level

resource monitor. We've also created a resource monitor associated with virtual warehouse #2, one of the priority virtual warehouses, that has a Notify-only action. Therefore, the first two virtual warehouses will never suspend as a result of either of the resource monitors reaching their threshold. The remaining virtual warehouses will suspend if / when their associated resource monitor reaches its threshold.



Virtual warehouses that share the same resource monitor also share the same thresholds. An example would be virtual warehouses three and four in [Figure 5-3](#). This could result in the credit usage for one virtual warehouse impacting the other assigned virtual warehouse(s).

DDL Commands for Creating and Managing Resource Monitors

By default, Snowflake resource monitors can only be created, viewed, and maintained by the ACCOUNTADMIN role. Although the creation of Snowflake resource monitors is exclusive to the ACCOUNTADMIN role, a Snowflake account administrator can enable other roles to view and modify resource monitors.

If not an ACCOUNTADMIN role, the privileges needed to view and edit resource monitors:

- MODIFY
- MONITOR

Snowflake resource monitors can be created in the web interface or by using SQL in a worksheet. For our examples in this chapter, we'll be using the SQL option to create and manage resource monitors.

Snowflake DDL commands for creating and managing resource monitors:

- CREATE RESOURCE MONITOR to assign warehouse(s) to a resource monitor
- ALTER RESOURCE MONITOR to modify an existing resource monitor
- SHOW RESOURCE MONITOR to view an existing resource monitor
- DROP RESOURCE MONITOR to delete an existing resource monitor

To create a resource monitor, you need to assign at least one warehouse to the resource monitor, unless you are setting the monitor at the account level. You are required to assign the warehouse at the time of creation if you create the resource monitor in the web interface. If you use SQL to create the resource monitor, you'll have to create the resource monitor first and then use the ALTER WAREHOUSE command to assign one or more warehouses to the resource monitor.

Let's first create an account level resource monitor.

```
USE ROLE ACCOUNTADMIN;
CREATE OR REPLACE RESOURCE MONITOR RM_1 WITH CREDIT_QUOTA = 5000
    TRIGGERS on 50 percent do notify
        on 75 percent do notify
        on 100 percent do notify
        on 110 percent do notify
        on 125 percent do notify;
```

You'll notice that we created five different notify actions and no suspend actions, following along with the plan we created earlier. After we create the resource monitor, we'll need to assign to our Snowflake account.

```
USE ROLE ACCOUNTADMIN;
ALTER ACCOUNT SET RESOURCE_MONITOR = RM_1;
```

Next, we'll create a warehouse-level monitor. We want to create the resource monitor for one of our priority virtual warehouses and we'll use the same notify actions for the priority virtual warehouse as we did for the account-level resource monitor. After we create the resource monitor, we'll assign it to our priority virtual warehouse.

```
USE ROLE ACCOUNTADMIN;
CREATE OR REPLACE RESOURCE MONITOR RM_5 WITH CREDIT_QUOTA = 1500
    TRIGGERS on 50 percent do notify
        on 75 percent do notify
        on 100 percent do notify
        on 110 percent do notify
        on 125 percent do notify;
```



```
ALTER WAREHOUSE WH_VW2 SET RESOURCE_MONITOR = RM_5;
```

Now, let's use the *SHOW RESOURCE MONITORS* command to see the resource monitors we've created thus far.

```
USE ROLE ACCOUNTADMIN;
SHOW RESOURCE MONITORS;
```

In [Figure 5-4](#), you'll notice that we have one account-level and one virtual warehouse-level resource monitor. Both of them have a monthly frequency stated, as expected since we didn't explicitly state a different schedule.

The screenshot shows a database results interface with a 'Results' tab selected. At the top, it displays 'Query ID', 'SQL', '39ms', and '2 rows'. Below this is a search bar labeled 'Filter result...' and buttons for 'Copy' and download. The main area is a table with the following data:

Row	name	credit_quota	used_credits	remaining_credit	level	frequency
1	RM_1	5000.00	0.00	5000.00	ACCOUNT	MONTHLY
2	RM_5	1500.00	0.00	1500.00	WAREHOUSE	MONTHLY

Figure 5-4. Results after creating two resource monitors

We're ready to create another virtual warehouse-level resource monitor.

```
USE ROLE ACCOUNTADMIN;
CREATE OR REPLACE RESOURCE MONITOR RM_2 WITH CREDIT_QUOTA = 500
    TRIGGERS on 50 percent do notify
        on 75 percent do notify
        on 100 percent do notify
        on 100 percent do suspend
        on 110 percent do notify
        on 110 percent do suspend_immediate;

ALTER WAREHOUSE WH_VW3 SET RESOURCE_MONITOR = RM_2;
```

Everything is good thus far. Now, let's purposely make a mistake to see what happens. We'll create a resource monitor and assign it to a virtual warehouse then we'll create another resource monitor and assign it to the same warehouse. We know that a virtual warehouse can only be assigned to one resource monitor, so we expect that the second time we assign a resource monitor to the same virtual warehouse will result in a failed command or the second resource monitor assignment will override the first assignment. Let's see what happens.

```
USE ROLE ACCOUNTADMIN;
CREATE OR REPLACE RESOURCE MONITOR RM_6 WITH CREDIT_QUOTA = 500
    TRIGGERS on 50 percent do notify
        on 75 percent do notify
        on 100 percent do notify
        on 100 percent do suspend
        on 110 percent do notify
        on 110 percent do suspend_immediate;

ALTER WAREHOUSE WH_VW6 SET RESOURCE_MONITOR = RM_6;
```

You'll notice from our planning diagram that we don't have a sixth resource monitor planned, so it was a mistake to create the sixth resource monitor. No worries, though. We'll delete the unneeded resource in a little while. For now, let's create the correct resource monitor that should be assigned to the virtual warehouse.

```

USE ROLE ACCOUNTADMIN;
CREATE OR REPLACE RESOURCE MONITOR RM_4 WITH CREDIT_QUOTA = 500
    TRIGGERS on 50 percent do notify
        on 75 percent do notify
        on 100 percent do notify
        on 100 percent do suspend
        on 110 percent do notify
        on 110 percent do suspend_immediate;

ALTER WAREHOUSE WH_VW6 SET RESOURCE_MONITOR = RM_4;

```

The statement was executed successfully. So, let's run a *SHOW RESOURCE MONITORS* command and see the results, as shown in [Figure 5-5](#).

```

USE ROLE ACCOUNTADMIN;
SHOW RESOURCE MONITORS;

```

Row	name	credit_quota	used_credits	remaining_credit	level	frequency
1	RM_1	5000.00	0.00	5000.00	ACCOUNT	MONTHLY
2	RM_2	500.00	0.00	500.00	WAREHOUSE	MONTHLY
3	RM_4	500.00	0.00	500.00	WAREHOUSE	MONTHLY
4	RM_5	1500.00	0.00	1500.00	WAREHOUSE	MONTHLY
5	RM_6	500.00	0.00	500.00	NULL	MONTHLY

Figure 5-5. Result of the SHOW RESOURCE MONITORS command

We can see that the sixth resource monitor appears to have been nullified when we assigned the fourth resource monitor to the virtual warehouse. We can confirm that by using the *SHOW WAREHOUSES* command.

```

USE ROLE ACCOUNTADMIN;
SHOW WAREHOUSES;

```

[Figure 5-6](#) shows the results of our statement. Just as we suspected, assigning the resource monitor to the virtual warehouse overrode the previous assignment.

Results Data Preview

✓ Query ID SQL

Filter result...

Row	name	
2	WH_VW2	resource_monitor
3	WH_VW3	RM_5
4	WH_VW4	RM_2
5	WH_VW5	null
6	WH_VW6	null
		RM_4

Figure 5-6. Results of the SHOW WAREHOUSES command

We can now get rid of the resource monitor we created in error.

```
DROP RESOURCE MONITOR RM_6;
```

Now we can create the last resource monitor from our planning diagram. We'll assign the resource monitor to two virtual warehouses.

```
USE ROLE ACCOUNTADMIN;
CREATE OR REPLACE RESOURCE MONITOR RM_3 WITH CREDIT_QUOTA = 1500
    TRIGGERS on 50 percent do notify
        on 75 percent do notify
        on 100 percent do notify
        on 100 percent do suspend
        on 110 percent do notify
        on 110 percent do suspend_immediate;
```

```
ALTER WAREHOUSE WH_VW4 SET RESOURCE_MONITOR = RM_3;
ALTER WAREHOUSE WH_VW5 SET RESOURCE_MONITOR = RM_3;
```

You can once again take a look at all the resource monitors in your Snowflake account.

```
USE ROLE ACCOUNTADMIN;  
SHOW RESOURCE MONITORS;
```

You'll notice that the dropped resource monitor does not appear in the list of resource monitors. At this point, we can run another *SHOW WAREHOUSES* statement. We can also take it a step further to get a list of any of our virtual warehouses that have not been assigned to a virtual warehouse-level resource monitor.

```
SHOW WAREHOUSES;  
SELECT "name", "size"  
FROM TABLE (RESULT_SCAN(LAST_QUERY_ID()))  
WHERE "resource_monitor" = 'null';
```

We can see that our COMPUTE_WH, the virtual warehouse that came with our Snowflake account and what we are referring to as virtual warehouse #1, has not been assigned to a resource monitor (Figure 5-7). If we review our planning diagram, we see that is as we would expect.

The screenshot shows a Snowflake worksheet interface. The top navigation bar has 'Results' selected and 'Data Preview' as an option. Below the bar, there's a status message: '✓ Query ID SQL 586ms 1 rows'. There are also 'Filter result...', 'Download', and 'Copy' buttons. The main area displays a table with two columns: 'Row' and 'name'. A single row is shown, with 'Row' value '1' and 'name' value 'COMPUTE_WH'. To the right of the name column, under the heading 'size', is the value 'X-Small'. The table has a light gray header row and white data rows.

Row	name	size
1	COMPUTE_WH	X-Small

Figure 5-7. Result showing virtual warehouse not assigned to a resource monitor

We've created and managed our resource monitors using SQL in a worksheet. Alternatively, we could have created and managed our resources in the Web UI. We'd need to make sure we are using the ACCOUNTADMIN role. Then, we would be able to view the resource monitors we just finished creating and we'd be able to create any new resources from the Web UI, as is shown in Figure 5-8

Resource Monitors						+ Resource Monitor				
6 Resource Monitors						Search	Level All	Warehouses All	Frequency All	C
Name	Quota Used	Level	Warehouses	Frequency	Start Time					
RM_1	0.03%	Account	—	Monthly	12/31/2021, 6:00 PM	...				
RM_2	0.00%	Warehouse	Wh_LVW3	Monthly	12/31/2021, 6:00 PM	...				
RM_3	0.00%	Warehouse	Wh_LVW4, Wh_LVW5	Monthly	12/31/2021, 6:00 PM	...				
RM_4	0.00%	Warehouse	Wh_LVW6	Monthly	12/31/2021, 6:00 PM	...				
RM_5	0.00%	Warehouse	Wh_LVW2	Monthly	12/31/2021, 6:00 PM	...				
RM_6	0.00%	—	—	Monthly	12/31/2021, 6:00 PM	...				

Figure 5-8. Snowflake Web UI for ACCOUNTADMIN role showing Resources Monitors

One final note about resource monitors is that for Snowflake provider accounts that have created reader accounts, there exists a RESOURCE_MONITORS view which can be used to query the resource monitor usage of the reader account. You can find more information about Snowflake provider and reader accounts in Chapter 10, Configuring and Managing Secure Data Sharing.

Using Object Tagging for Cost Centers

Object tagging was introduced in chapter 7 when we created tags for data governance purposes. In the examples from Chapter 7, we learned that Snowflake object tags can be assigned to tables, views, or columns to help keep track of the type of data being stored and whether that data included sensitive information.

In addition to using object tags for security reasons, Snowflake tags can be created for objects to help track resource usage. For example, cost center tags could have values such as accounting, finance, marketing, and engineering. In that case, cost center object tagging would allow for more granular insight, at a department-level . Additionally, tags can be used for tracking and analyzing resource usage by specific short-term or long-term projects.

Querying the Account Usage View

To get more granular detail about virtual warehouse costs, we can leverage queries on the Snowflake Account Usage view.



Access to the Account Usage view is given only to the ACCOUNTADMIN role by default. Thus, we will use that role in our examples for querying the view.

This query provides the individual details of cost, based on virtual warehouse start times.

```

USE ROLE ACCOUNTADMIN;
SET CREDIT_PRICE = 3.00;
USE DATABASE SNOWFLAKE;
USE SCHEMA ACCOUNT_USAGE;
SELECT WAREHOUSE_NAME
    ,START_TIME
    ,END_TIME
    ,CREDITS_USED
    ,($CREDIT_PRICE*CREDITS_USED) AS DOLLARS_USED
FROM SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY
ORDER BY START_TIME DESC;

```

This summarizes the costs of each virtual warehouse for the past 30 days.

```

SELECT WAREHOUSE_NAME
    ,SUM(CREDITS_USED_COMPUTE) AS CREDITS_USED_COMPUTE_30DAYS
    ,($CREDIT_PRICE*CREDITS_USED_COMPUTE_30DAYS) AS DOLLARS_USED_30DAYS
FROM ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY
WHERE START_TIME >= DATEADD(DAY, -30, CURRENT_TIMESTAMP())
GROUP BY 1
ORDER BY 2 DESC;

```

The previous code samples provided an example of how you can query the Account Usage view to monitor the virtual warehouse metering for compute warehouse credits consumed. Similar queries can be created to monitor compute credits consumed from Snowpipe, clustering, materialized views, search optimization, and replication. There are also some good code examples that can be found in the Snowflake quick-start for *Resource Optimization: Billing Metrics* at <https://quickstarts.snowflake.com/>.

Using BI Partner Dashboards to Monitor Snowflake Usage and Costs

If you have access to the Snowflake BI and analytics partners tools, you'll be able to access one or more plug-and-play dashboards to help you monitor your Snowflake usage. Here is a list of some pre-built dashboards currently available.

- Tableau Dashboards: Compute Cost Overview
- Sigma: Compute Cost, Storage Cost, Snowflake Cost (Reader Accounts)
- Looker: Snowflake Cost and Usage Analysis
- Microsoft Power BI: Snowflake Usage Report
- Qlik: Snowflake Usage Dashboard

Snowflake Agile Software Delivery

Managing Snowflake costs is not just limited to reviewing the Snowflake monthly bill. Current Snowflake charges are the result of past decisions including architecture choices, decisions about what data to store, and deciding how to construct certain queries to access the data. But, organizations and their needs are always changing. Thus, it is expected that an organization's Snowflake environment will change as well.

There are costs associated with changing any software, including making changes to your Snowflake software-as-a-service account environment. It takes employees' time to architect and build new software and/or enhance existing software. Any disruptions to the business also have a cost associated with them. Minimizing software development costs and disruption to the business while working as quickly as possible to make improvements requires an established set of practices and an agile software delivery mindset.

DevOps is a term used to describe the set of practices that are established with the goals of high software quality and helping software developers and IT operations to shorten the systems development lifecycle and provide continuous delivery. Database Change Management (DCM) is a subset of DevOps which specifically focuses on the changes made to databases. Database change management has unique challenges, but the good news is that Snowflake features, such as zero-copy cloning and time travel, make it considerably easier for teams to use an agile approach for DevOps.

Why Do We Need DevOps?

It is crucial that an audit trail of software and application changes be maintained because of regulatory and compliance requirements applicable to an organization's software systems and its data. For many reasons, it is also important that any bug fixes be resolved quickly. New features and requested enhancements should also be released frequently to give organizations a competitive advantage in today's quick-paced environment, especially given the vast number of powerful software application development tools available. Achieving these things requires careful consideration. And most importantly, it requires automation.

Automation of software development, testing, and release is the most likely way to innovate faster, deliver improved applications in greater numbers, adhere to regulatory requirements, and obtain greater return on investment. It is the objective of DevOps to minimize the manual aspects of software development, testing, and deployment.

Continuous Data Integration, Continuous Delivery, and Continuous Deployment

Achieving continuous integration and continuous delivery (CI/CD) through the ongoing automation of development, testing, and deployment occurs in a connected way through a CI/CD pipeline (Figure 8-9).

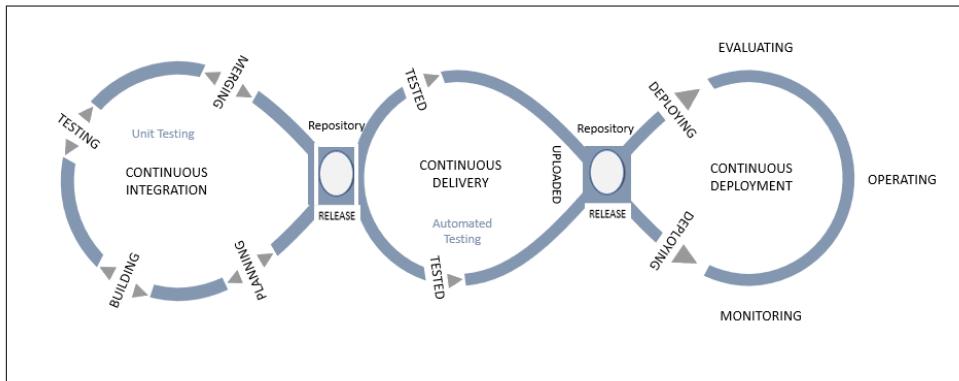


Figure 5-9. Figure 8-9 CI/CD Pipeline

In general terms, continuous integration is a DevOps software development practice whereby developers merge their code regularly into a central repository. Continuous delivery occurs when application changes are automatically tested and uploaded to a source control repository such as Github where the operations team can then deploy them to production. Continuous deployment automates the next stage in the pipeline by automatically releasing changes from the repository to production.

What the terms continuous integration, continuous delivery, and continuous deployment mean for each specific organization is dependent upon how much automation has been built into their CI/CD pipeline.

What is Database Change Management (DCM)?

Database DevOps is often referred to as Database Change Management (DCM). Database DevOps, especially for virtual warehouses, have some unique challenges as compared to the difficulties of DevOps for software application. Snowflake's unique architecture solves many of these problems. There also exists specific database change management tools that work easily with Snowflake.

Overcoming the Unique Challenges with Database Changes

Making changes to databases present their own unique set of challenges, especially for traditional on-premises databases which have to deal with the problems of down time for hardware upgrades and explicitly having to back up databases beforehand so

that they will be available for restoration, if needed. In the event a rollback is needed, an operations freeze with no work happening during the restoration process is required. In addition, there is the problem of drift whereby developers could be working on a different version of the database than the version in production, and there is also the problem of load balancing.

With Snowflake, these top challenges no longer exist for users. Hardware fixes and upgrades are handled by the Snowflake team with no impact on users. For rollback and version control, Snowflake has time travel and fail-safe which eliminates the need to make traditional data backups. Time travel and fail-safe are part of Snowflake's continuous data protection lifecycle.

To alleviate the problem of drift, Snowflake users can utilize zero-copy cloning that was described in the previous section to simplify the loading and seeding of databases in different environments.

Given all this and because most everything in Snowflake can be automated via SQL, the database change management process could be handled quite simply in Snowflake by creating a lightweight DevOps framework for small deployments. For a more heavyweight Snowflake DevOps framework, however, there are database change management tools that can be used.

DCM Tools for a Heavyweight Snowflake DevOps Framework

Database Change Management (DCM) refers to a set of processes and tools used to manage objects within a database. The most important tool to consider for Snowflake database change management is a source control repository and pipeline tool. Examples of hosted options for database change management include AWS CodeCommit and CodePipeline, Azure DevOps, GitHub Actions, or BitBucket.

In addition to a source control repository and pipeline tool, you'll need database migration tools to deploy the changes to the database. Several database change management tools that can be used with Snowflake exist. Some are primarily a SaaS tool or data transformation tool and many of them are community-developed. Examples of tools that can be used as a database migration tool include Sqitch, Flyway, Snowchange, Schemachange, SqlDBM, Jenkins, and dbt.

Lastly, a testing framework such as Junit, pytest, or DbFit is needed.

Using DCM tools which automate our data integration and delivery pipeline can provide all the DevOps benefits we previously mentioned and enable us to achieve continuous delivery and deployment.

How Zero-Copy Cloning can be used to Support Dev / Test Environments

Zero-copy cloning was introduced in Chapter 2, *Creating and Managing Snowflake Architecture*. Because zero-copy cloning is a metadata-only operation, there is no additional storage charge for Snowflake cloned object unless and until changes are made.

Zero-copy clones are frequently used to support working in development and test environments as part of the development lifecycle. We are going to demonstrate concepts in this section that show how zero-copy cloning can be used in Snowflake agile development. Changes to a Snowflake production environment can take the form of new development. Creating a new table is one example. It could also be in the form of enhancements to an existing object in production. We'll look in-depth at how zero-copy cloning can be used for new development.

In our example, we'll use some roles-based access control best practices that we learned about in Chapter 5. However, in a real-world setting, you'd likely need to have different roles and privileges to set up to properly manage the CI/CD process. The purpose in this chapter, as it relates to role-based access, is to give you a little more hands-on practice.

In our example, let's assume that Table A already exists in our production environment and we want to add an additional table in production. What we'll do first is create a clone off the production database in which we can develop. That is where we'll create the new table. We'll also want to create a clone of the development database in which we can perform our testing, or quality assurance activities. See [Figure 5-10](#) which illustrates our first steps.

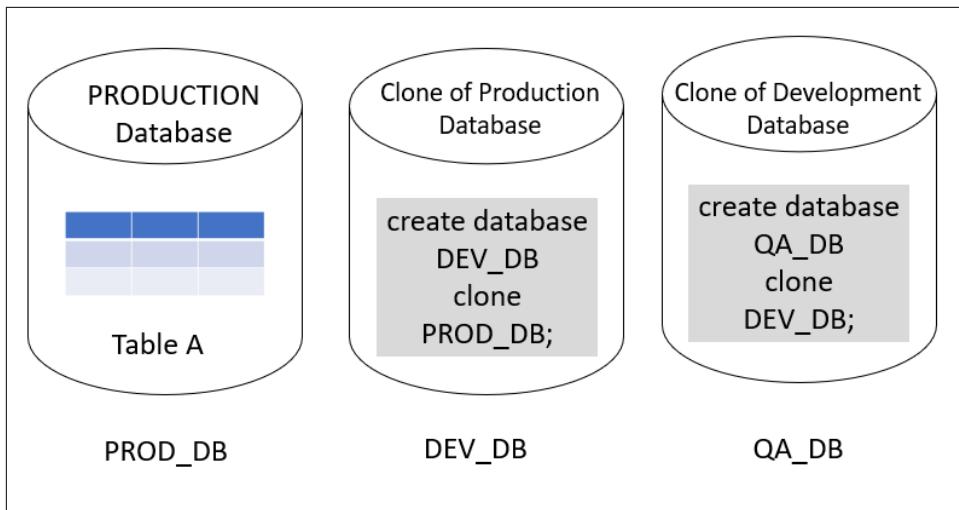


Figure 5-10. Agile development using zero-copy cloning

Let's first create some specific roles for our production, development, and QA environments. As a best practice, we'll assign the new custom roles back to the SYSADMIN role.

```
USE ROLE SECURITYADMIN;
CREATE OR REPLACE ROLE PROD_ADMIN;
CREATE OR REPLACE ROLE DEV_ADMIN;
CREATE OR REPLACE ROLE QA_ADMIN;
GRANT ROLE PROD_ADMIN TO ROLE SYSADMIN;
GRANT ROLE DEV_ADMIN TO ROLE SYSADMIN;
GRANT ROLE QA_ADMIN TO ROLE SYSADMIN;
```

Next, we'll want to give some privileges to the new roles so that they can create databases on the account.

```
USE ROLE SYSADMIN;
GRANT CREATE DATABASE ON ACCOUNT TO ROLE PROD_ADMIN;
GRANT CREATE DATABASE ON ACCOUNT TO ROLE DEV_ADMIN;
GRANT CREATE DATABASE ON ACCOUNT TO ROLE QA_ADMIN;
```

Let's assume that we want to assign a separate virtual warehouse to each of the administrators. We can rename three of the virtual warehouses that we created earlier.

```
USE ROLE SYSADMIN;
ALTER WAREHOUSE IF EXISTS WH_VW2 RENAME TO WH_PROD;
ALTER WAREHOUSE IF EXISTS WH_VW3 RENAME TO WH_DEV;
ALTER WAREHOUSE IF EXISTS WH_VW4 RENAME TO WH_QA;
SHOW WAREHOUSES;
```

You can see from the SHOW command that the warehouse have now been renamed. Let's grant usage for these virtual warehouses to each of the associated roles.

```
GRANT USAGE ON WAREHOUSE WH_PROD TO ROLE PROD_ADMIN;
GRANT USAGE ON WAREHOUSE WH_DEV TO ROLE DEV_ADMIN;
GRANT USAGE ON WAREHOUSE WH_QA TO ROLE QA_ADMIN;
```

We need to create a production environment database, schema, and table. We'll also grant usage on the database to the development administrator.

```
USE ROLE PROD_ADMIN;
USE WAREHOUSE WH_PROD;
CREATE OR REPLACE DATABASE PROD_DB;
CREATE OR REPLACE SCHEMA CH9_SCHEMA;
CREATE OR REPLACE TABLE TABLE_A (Customer_Account int, Amount int, transaction_ts timestamp);
GRANT USAGE ON DATABASE PROD_DB TO ROLE DEV_ADMIN;
```

Now the development administrator can create a clone of the production database and grant usage of the cloned database to the QA administrator role.

```
USE ROLE DEV_ADMIN;
USE WAREHOUSE WH_DEV;
CREATE OR REPLACE DATABASE DEV_DB CLONE PROD_DB;
GRANT USAGE ON DATABASE DEV_DB TO ROLE QA_ADMIN;
```

The QA administrator now creates a clone of the development database and grants usage to the production administrator role.

```
USE ROLE QA_ADMIN;
USE WAREHOUSE WH_QA;
CREATE OR REPLACE DATABASE QA_DB CLONE DEV_DB;
GRANT USAGE ON DATABASE QA_DB TO ROLE PROD_ADMIN;
```

Now we're ready for the development work. Back in the development database, we use the DEV_ADMIN role to create a new development schema and a new table.

```
USE ROLE DEV_ADMIN;
USE WAREHOUSE WH_DEV;
USE DATABASE DEV_DB;
CREATE OR REPLACE SCHEMA DEVELOPMENT;
CREATE OR REPLACE TABLE TABLE_B (Vendor_Account int, Amount int, transaction_ts timestamp);
GRANT USAGE ON SCHEMA DEVELOPMENT TO ROLE QA_ADMIN;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA DEVELOPMENT TO ROLE QA_ADMIN;
```

Once development is done, the QA team can do their testing by creating a new table in the testing environment from the newly created table in the development environment.



The new table in the QA environment could be created using either the *Create Table As Select*, also known as CTAS, or by using the *Create Table LIKE* command. The difference between the two is that the latter creates an empty copy of the existing table whereas the CTAS creates a populated table.

```
USE ROLE QA_ADMIN;
USE WAREHOUSE WH_QA;
USE DATABASE QA_DB;
CREATE OR REPLACE SCHEMA TEST;
CREATE OR REPLACE TABLE QA_DB.TEST.TABLE_B AS SELECT * FROM DEV_DB.DEVELOPMENT.TABLE_B;
GRANT USAGE ON SCHEMA TEST TO ROLE PROD_ADMIN;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA TEST TO ROLE PROD_ADMIN;
```

When the QA team has completed their testing, the production administrator can then copy the table into production.

```
USE ROLE PROD_ADMIN;
USE WAREHOUSE WH_PROD;
USE DATABASE PROD_DB;
USE SCHEMA CH9_SCHEMA;
CREATE OR REPLACE TABLE TABLE_B AS SELECT * FROM QA_DB.TEST.TABLE_B;
```

If you'd like to get a complete idea of what was created, you can use the ACCOUNTADMIN role. With that role, you can see what was created in the development, production, and testing environments ([Figure 5-11](#))

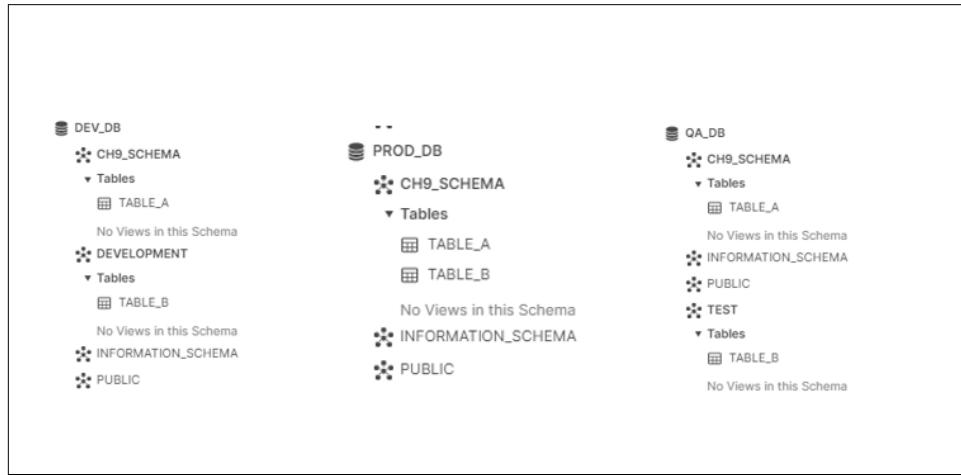


Figure 5-11. The results of creating the databases, schemas, and tables in this section

For development that involves enhancing an existing object, you'll most likely want to use a combination of zero-copy cloning and time travel. You can clone the production database as it was a certain number of days prior, make the enhancements to the

tables using scripts, and then rerun the data loads for the prior number of days. Comparing the results between the two should show that, other than the enhancements, there should be no difference.

If you'd like to see how much compute you used for the exercises in this example, you can rerun the code we saw earlier in the chapter which summarized the cost of each virtual warehouse for the past 30 days.

Code Cleanup

We can clean up our Snowflake environment in just a few quick steps. We can drop our databases and the new custom roles we created.

```
USE ROLE ACCOUNTADMIN;
DROP DATABASE DEV_DB;
DROP DATABASE PROD_DB;
DROP DATABASE QA_DB;
DROP ROLE PROD_ADMIN;
DROP ROLE DEV_ADMIN;
DROP ROLE QA_ADMIN;
```

Let's drop our resource monitors next.

```
DROP RESOURCE MONITOR RM_1;
DROP RESOURCE MONITOR RM_2;
DROP RESOURCE MONITOR RM_3;
DROP RESOURCE MONITOR RM_4;
DROP RESOURCE MONITOR RM_5;
DROP RESOURCE MONITOR RM_6;
```

And then finally, let's drop our virtual warehouses. Note that we will drop the virtual warehouses we created but will not attempt to drop the COMPUTE_WH.

```
DROP WAREHOUSE WH_PROD;
DROP WAREHOUSE WH_DEV;
DROP WAREHOUSE WH_QA;
DROP WAREHOUSE WH_VW5;
DROP WAREHOUSE WH_VW6;
```

Summary

Throughout this chapter, we've focused on Snowflake costs and how to manage them. Managing costs can be achieved by understanding Snowflake's usage-based pricing, by using Snowflake resource monitors and object tagging for cost centers, by querying the Account Usage view, and using BI partner dashboards. We've also considered the cost of making changes in a Snowflake environment. Managing costs are important. Just as important as managing costs is having a solution that works well. Effective and efficient Snowflake performance is critical for success. In the next chapter,

we'll learn about Snowflake's unique micro-partitions, how to take advantage of clustering, and how to improve query performance.

Exercises to Test Your Knowledge

1. What are the three major categories of a Snowflake monthly bill?
2. The region selected is an important factor in determining storage costs. Other than storage costs considerations, what other things are important to think about when selecting a region?
3. The price of on demand Snowflake credits is dependent on what three things?
4. What are the three properties of a Snowflake resource monitor? Which of those properties are required to be included in the SQL statement in order to create a resource monitor?
5. Is there a limit to the number of Snowflake resource monitors that can be created?
6. Is there a limit to the number of Snowflake resource monitor actions?
7. What are some of the unique challenges associated with database changes?
8. What are some unique characteristics of Snowflake's zero-copy cloning?
9. What is the difference between the Create Table As Select, also known as CTAS, and the Create Table LIKE command?
10. What happens if you assign a virtual warehouse resource monitor to a virtual warehouse which is already assigned to a resource monitor?

Solutions to these exercises are available in Appendix A.

Configuring and Managing Secure Data Sharing

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 10th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

Introduction

Data sharing supports collaboration between business partners, provides opportunities to inform customers, and gives you a way to obtain real-time information from vendors and suppliers. Data sharing also provides opportunities for you to monetize your data. However, there are impediments to effective and efficient data sharing when using traditional data sharing options. For one thing, traditional data sharing options, such as FTP transfers, API calls, sending or receiving CSV files, and ETL tools and processes, often require building complex infrastructure.

Similarly, traditional data sharing approaches, which involve transferring data copies and reconstructing and storing duplicate data, are expensive. There is often no single source of truth with traditional data sharing options and fewer actionable insights result from having slow access to stale and sometimes incomplete data. Overcoming

many of these sharing challenges, however, can be accomplished by using Snowflake's Secure Data Sharing technology.

Secure Data Sharing makes it possible for data to be accessible via a live connection so that updated data is automatically available to data consumers in real-time. The business logic, along with the data, can also be shared. In addition to being secure, efficient, and a great way to monetize data, Snowflake data sharing comes with the peace of mind knowing you can revoke access at any time.

These powerful Secure Data Sharing capabilities are made possible because of Snowflake's unique architecture.

Snowflake Architecture Supports Data Sharing

In Chapter 2, *Creating and Managing Snowflake Architecture*, we learned how traditional data platform architectures have a fixed ratio of compute and storage. This is in contrast to how Snowflake's multi-cluster shared data architecture has storage capability which can expand and contract automatically and has independent compute clusters that can read and write at the same time and instantly resize.

When you separate compute from storage and data access control is shared, as Snowflake does, you can have multiple warehouses working on the same data at the same time. As such, Snowflake's architecture supports the ability to have one single source of truth.

If you share data within the same cloud provider and region, there is no need to copy the data to share it. If you share data to a different region or different cloud provider, Snowflake replicates the data. However, it does so via auto-fulfillment, so you don't need to do any manual work. *Snowgrid* is the term to describe data replication for the purpose of facilitating data sharing.

The Power of Snowgrid

Snowgrid spans globally, seamlessly connecting Snowflake users who may be separated by region or cloud. Snowgrid achieves this through replication via auto-fulfillment. Even when sharing data across clouds and regions, the shares are *transactionally consistent*, meaning the source of truth is still maintained. Within Snowgrid, there exists all the native cross-cloud governance controls that serve as the foundational building blocks for enabling federated governance. For more details about the specifics of database replication, refer back to Chapter 7, *Implementing Account Security, Protections and Data Recovery*.

With the power of Secure Data Sharing and Snowgrid, Snowflake providers can create and share any number of data shares with data consumers worldwide. Each share encapsulates the necessary access control, so it is secure. Once a provider's outbound share appears as an inbound share in the data consumer's account, the consumer is

then able to create a database from that share. The metadata points back to the original data source so the data continues to exist only in the provider account.

Data Sharing Use Cases

Many industries have begun adopting the features from Snowflake's data sharing capabilities. For example, financial services and insurance companies leverage Snowflake's secure storage and data sharing capabilities to efficiently ingest IoT data as well as transactional and clickstream data. Enriching that data with third-party Snowflake data, such as Equifax data, provides a more complete view of their customers. Having a complete 360-view of the customer provides organizations with the opportunity to identify high-value customers and ensure they have a good experience at every touch-point.

Healthcare and life sciences organizations can take advantage of Snowflake data sharing capabilities to replace decades old methods of data sharing such as FTP, DVDs or other physical media. In fact, payers storing petabytes of PHI and PII data that need to be shared with insurance brokers, providers, and other vendors are turning to Snowflake as a solution because data sharing in the healthcare industry is mandated and regulated. Healthcare providers that own massive amounts of data often collaborate with healthcare consortiums or are affiliated with universities; thus, sharing data in real time, rather than in batches, is absolutely essential.

Trucking companies that can optimize fleet routing by using IoT data and retailers that are able to improve the supply chain by sharing data with logistics partners are two other examples of industries who incorporate Snowflake's live data sharing capabilities. Also, some enterprise companies use different Snowflake accounts for development, testing, and production purposes. Data sharing provides the ability to incorporate strong audit controls and move data in a controlled manner rather than having to prepare flat files and reload them into a different environment.

Publishers, advertising agencies, and brands need to develop a new data sharing paradigm quickly as we are quickly approaching a world without the existence of third-party cookies. As third-party cookies are being deprecated, marketers will need to find new ways of identifying people online so that they can optimize marketing campaigns and continue to personalize messages.

Snowflake Support for Unified ID 2.0

Cookies, developed in 1994 as a tiny piece of software code installed on your web browser when you visit a website to facilitate revisiting and tracking, is an outdated technology. Today, internet usage occurs more frequently in mobile apps and on connected TV devices where cookies is mostly irrelevant. Cookies often contain a wealth of personal data that could potentially identify you without your consent; thus, they are subject to the European Union's General Data Protection Regulation (GDPR) and

the ePrivacy Directive (EPD). The EPD will eventually be replaced by the ePrivacy Regulation (EPR). Major platforms such as Apple, Firefox, and Google have begun limiting the use of third-party cookies. In the near future, changes to internet platforms' privacy policies could even render cookies, the internet's first universal identifier, obsolete. You can visit <https://cms.law/en/deu/insight/e-privacy> if you are interested in learning more about ePrivacy.

Concerns over user privacy and more regulation have necessitated new approaches be considered. Unified ID 2.0, developed by The Trade Desk as an open-source industry-governed identity solution to provide anonymization, greater transparency, and better control for users. A person's UID 2.0 is a random string of numbers and letters generated from an email address that cannot be reverse engineered to an email address or any other form of identification. Currently, Unified 2.0 is accepted by The Washington Post, Oracle, Nielsen, Tubi TV, among others.

Snowflake supports Unified ID 2.0 for use with secure data sharing, most notably with data clean rooms which are discussed in more detail later. As such, Snowflake customers can join first-party data and third-party directly in the Snowflake platform in a more privacy-conscious way. The power of the Snowflake Data Cloud and Unified ID 2.0 provides for a single view of the customer.

Snowflake Secure Data Sharing Approaches

There are four different ways to approach Snowflake's Secure Data Sharing. The simplest way to use Snowflake data sharing is to engage in account-to-account sharing of data where you can share data directly with another account and have your data appear in their Snowflake account without having to move or copy it. The other account could belong to a different internal business unit, for example, or to a different organization altogether.

You can use Snowflake's Data Marketplace, a public Secure Data Sharing approach, which connects global providers of data with consumers throughout the world.

You can even create your own Snowflake private Data Exchange to collaborate with others; you'll get to control who can join the data exchange and which members can provide data, consume data, or both.

Alternatively, you can use a data clean room, a framework for sharing data between two or more parties in a privacy protected way, that allows multiple companies to bring data together under specific guidelines so that personally identifiable information (PII) is anonymized, processed, and stored in a compliant way. Figure 10-1 summarizes the four main Snowflake Secure Data Sharing approaches.

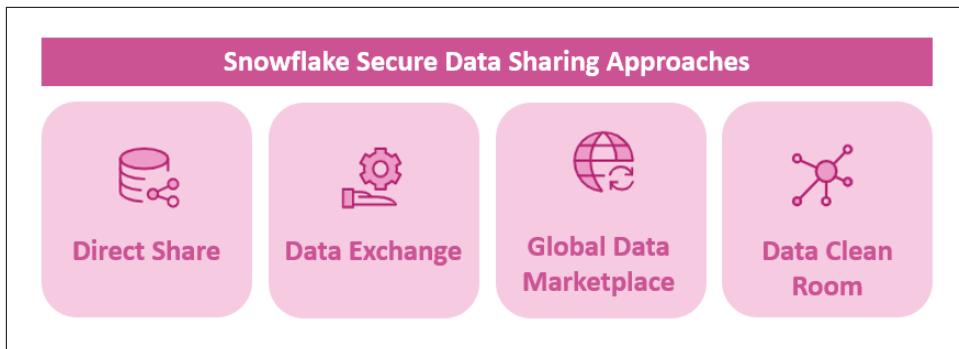


Figure 6-1.

Figure 10-1 Four Snowflake Secure Data Sharing Approaches

We'll now take care of some preparation work and then explore each of the four Snowflake Secure Data Sharing approaches.

Prep Work

If you are working in a Snowflake free trial account, you'll notice that when you first log in you'll see that you've been assigned the SYSADMIN role. Working with data shares requires certain privileges to be assigned to your role unless you are using the Account Administrator role. For the lessons in this chapter, we'll be using the ACCOUNTADMIN role so make sure to set your role appropriately. As a reminder, you can change your role using the drop-down or you can use a SQL statement in the worksheet.

We're ready to create a new folder titled *Chapter10* and a new worksheet titled *Data Sharing*. Refer to the *Navigating Snowsight Worksheets* section in Chapter 1 if you need help creating a new folder and worksheet. Next, we'll want to create a new database and table for the hands-on examples in this chapter. In your new *Data Sharing* worksheet, execute the following commands.

```
USE ROLE ACCOUNTADMIN;
USE WAREHOUSE COMPUTE_WH;
CREATE OR REPLACE DATABASE DEMO10_DB;
USE SCHEMA DEMO10_DB.PUBLIC;
CREATE OR REPLACE TABLE SHARINGDATA (i integer);
```

Let's go ahead and take a look to see what data has been shared with us. Navigate back to the main menu but clicking on the home icon and then click on the *Data >> Private Sharing* option in the menu. You can see there in [Figure 6-2](#) that I currently have access to data from two direct data shares.

The screenshot shows the Snowflake Data Sharing interface. On the left, there's a sidebar with a user profile for Joyce Avila (ACCOUNTADMIN) and a navigation menu with options like Worksheets, Dashboards, Data, Databases, Private Sharing (which is selected), Marketplace, Activity, Admin, Help & Support, and Classic Console. At the top right, there are three tabs: Shared With Me (selected), Shared By My Account, and Reader Accounts. Below the tabs is a search bar. The main area is titled "Direct Shares" and displays two shares: "ACCOUNT_USAGE" from "SNOWFLAKE" and "SAMPLE_DATA" from "SFC_SAMPLES". Both shares were shared 2 years ago. The "ACCOUNT_USAGE" share is associated with the "SNOWFLAKE" account.

Figure 6-2. Direct Data Shares that are Shared With Me as the Account Administrator

Throughout the chapters, we have used the SAMPLE_DATA share for examples. We've also discussed the Snowflake ACCOUNT_USAGE share. We'll review details about this special ACCOUNT_USAGE share in a later section.

The default for the *Shared Data* menu option is *Shared With Me* which you can see at the top middle of the screen. To see any of your outbound data shares, you can click on *Shared By My Account* as shown in [Figure 6-3](#).

This screenshot shows the same Snowflake interface as Figure 6-2, but with the "Shared By My Account" tab selected. The main area displays a message: "No Shared Data" with a small icon of a document with arrows. Below the message, it says "Data that has been shared by your account will appear here." The rest of the interface is identical to Figure 6-2, including the sidebar and other tabs.

Figure 6-3. Data I've Shared With Others

Snowflake Direct Secure Data Sharing Approach

Direct sharing, Snowflake's simplest form of data sharing, results when a Snowflake provider creates an outbound data share which then becomes an inbound share in a consumer's Snowflake account. A provider can share a Snowflake database table, a secure view, or a secure UDF via a share object. It is important to realize that the

share is not created *from* objects but rather the share object is granted privileges to the database, schema, table, view or UDF.

Creating Outbound Shares

When you are creating outbound shares, your account is the Snowflake *provider* account whereas the account to which you are sharing your data is the Snowflake *consumer* account. It's important to know that a share can contain only a single database.



To include multiple databases in a share, you can create a secure view which can be shared. This is possible provided the objects referenced in the share reside in the same Snowflake account.

To create a share object, click on the *Share Data* button on the top right of the screen then click on the *Select Data* button. Be sure to select the DEMO10_DB database, PUBLIC schema, and the SHARINGDATA table. It is important to check the box next to the table in order to select it to be included in the share (Figure 6-4). Click on *Done*.

A screenshot of the Snowflake UI showing the 'Share Data' dialog box. The 'Select Data' section is open, showing the 'SHARINGDATA' table under 'DEM010_DB.PUBLIC'. The 'SHARINGDATA' table has checkboxes next to it, both of which are checked. A 'Done' button is at the bottom right of the dialog.

Figure 6-4. Selecting Data When Creating a Share Object

You'll see that the Secure Share Identifier now is now filled (Figure 6-5) but you can change the identifier. It's important to note that the Secure Share identifier must be

unique for the account in which the share is created. The identifier is case sensitive. Also notice that you could share the object with a specific consumer by enter the accounts in your region by name.

The screenshot shows the 'Share Data' dialog box. At the top, it says 'Sharing as ACCOUNTADMIN'. Below that, there's a section for 'Secure Share Identifier' with a blue circular icon containing a cube. It displays 'DEMO10_DB' and '1 Table'. A note says 'A secure share that packages the data you selected will be created.' with a link to 'Learn More.'. The identifier field contains 'DEMO10_DB_SNOWFLAKE_SECURE_SHARE_1652043031528'. Below it, a note says 'Allowed characters: A-Z, 0-9, \$, _'. There's also a 'Description (optional)' field with an empty input box. On the left, there's a grey circular icon with a shield. To its right, a text input box says 'Add accounts in your region by name'. At the bottom, there are 'Cancel' and 'Create Share' buttons, with 'Create Share' being blue and bold.

Figure 6-5. Naming the Secure Share and Adding Consumers to a Share Object

Let's change the identifier that was automatically generated and let's skip adding new consumers at this time. In the Secure Share Identifier field, put DEMO10_SHARE and then click on the *Create Share* button. Once we do so, we'll get confirmation that the share was created (Figure 10-6)

The screenshot shows the 'Shared With' section of a 'Share Object Successfully Created' page. It includes a 'Shared With' summary, an 'Add Consumers' button, a 'Description' section with an 'Add' button, and a 'Data' section listing a single table named 'SHARINGDATA' under the schema 'PUBLIC'. The table details are: NAME: SHARINGDATA, TYPE: Table, SCHEMA: PUBLIC.

Figure 6-6.

Figure 10-6 Share Object Successfully Created

Another way to create an outbound share is to use SQL commands within a worksheet. Navigate back to the *Data Sharing* worksheet.

```
USE ROLE ACCOUNTADMIN;
CREATE OR REPLACE SHARE DEMO10_SHARE2;
GRANT USAGE ON DATABASE DEMO10_DB TO SHARE DEMO10_SHARE2;
GRANT USAGE ON SCHEMA DEMO10_DB.PUBLIC TO SHARE DEMO10_SHARE2;
GRANT SELECT ON TABLE DEMO10_DB.PUBLIC.SHARINGDATA TO SHARE DEMO10_SHARE2;
```

One of the things you'll notice is that you are assigning access privileges to the shares that will be used by the data consumers. This should look familiar. The way in which you grant access to objects for data shares is part of the same role-based access control examples we went through together in Chapter 5.

Now, if you go back to the menu option for *Data >> Private Sharing >> Shared By My Account*, you'll see both shares we just created (Figure 6-7). If both shares don't show up immediately, try refreshing the page.

Shared With Me Shared By My Account Reader Accounts				Share Data
<input type="text" value="Search"/> Shared with All All Types C				
TITLE	SHARED WITH	CREATED ↑	DATA	
 DEMO10_SHARE2	—	just now	 DEMO10_DB	
 DEMO10_SHARE	—	3 minutes ago	 DEMO10_DB	

Figure 6-7. A Listing of All Shares Within An Account

Data Providers Role in Creating and Managing Shares

Snowflake data providers create an outbound share object and assign privileges on one or more database tables, secure views, or secure UDFs using data stored in the provider's account. As such, the data provider is responsible for bearing the cost of data storage. The consumer is not billed for any data storage costs because the data share, and objects accessed via the share, are metadata only.



A Snowflake provider can share an outbound share with a virtually unlimited number of accounts and there is never a cost for the act of sharing data across accounts within the same cloud provider region. The Snowflake data provider can also set terms, through a contract, to limit the resharing of the data.

If you are ready to give consumers access to the DEMO10_SHARE, you can click on the share listing and then on the *Add Consumers* button on the right (Figure 10-8).

The screenshot shows the Snowflake UI for managing data shares. A modal window titled 'Add Consumers' is open, prompting the user to 'Share With Snowflake Accounts' by entering account names. Below the modal, the main interface shows a 'Shared' data share named 'DEMO10_SHARE'. It includes sections for 'Description' (with a summary and an 'Add' button) and 'Data' (listing 'DEMO10_DB' with 1 table, 'SHARINGDATA', which is a Table type under the PUBLIC schema). There is also an 'Edit' button for the data section.

Figure 6-8.

Figure 10-8 Adding Data Consumers To An Existing Share

We'll not be adding any consumers here in the UI so go ahead and hit the *Cancel* button. In addition to the using the UI, a Snowflake provider can use a SQL command in the worksheet to add one or more consumers who have their own Snowflake accounts. That command would look like this:

```
ALTER SHARE <name_of_share> ADD ACCOUNTS = <name_of_consumer_account>;
```



You can add multiple consumer accounts by separating names with a comma.

Remember, of course, that there are ways you can restrict certain data from being accessible either by excluding the data altogether or by masking the data that can be seen. Some examples are dynamic data masking and row-level security which we discussed in Chapter 7.

What if you want to share specific records within a data share with different consumers? One way to achieve the goal is to separate the storage of those records and then create many different data shares. That is time-consuming and prone to errors. For-

tunately, there is an easier and better way. A provider can maintain one data share and, using Snowflake's *Current Account* capability, can provide access to custom slices of the data to the current account. It's also possible to provide specific slices of the data to each consumer account external to the current account.

Let's see this in action. For this example, we'll have different countries with data, and we'll associate the countries to a specific region. We want this regional data to be accessible to different accounts. We need to create the table and insert the values for that base table in our Snowflake instance. We'll designate the ID as the region. Navigate back to the *Data Sharing* worksheet.

```
USE ROLE ACCOUNTADMIN;
USE DATABASE DEMO10_DB;
CREATE OR REPLACE SCHEMA PRIVATE;
CREATE OR REPLACE TABLE DEMO10_DB.PRIVATE.SENSITIVE_DATA
    (nation string,
     price float,
     size int,
     id string);

INSERT INTO DEMO10_DB.PRIVATE.SENSITIVE_DATA
    values('USA', 123.5, 10,'REGION1'),
          ('USA', 89.2, 14, 'REGION1'),
          ('CAN', 99.0, 35, 'REGION2'),
          ('CAN', 58.0, 22, 'REGION2'),
          ('MEX', 112.6,18, 'REGION2'),
          ('MEX', 144.2,15, 'REGION2'),
          ('IRE', 96.8, 22, 'REGION3'),
          ('IRE', 107.4,19, 'REGION3');
```

Next, we'll create a table that will hold the mapping to individual accounts.

```
CREATE OR REPLACE TABLE DEMO10_DB.PRIVATE.SHARING_ACCESS
    (id string,
     snowflake_account string);
```

Let's give our current account the access to the data in REGION1, which includes the USA data, by inserting those details into the SHARING_ACCESS mapping table.

```
INSERT INTO SHARING_ACCESS values('REGION1', current_account());
```

Now, we'll assign REGION2 and REGION3 the values associated with their respective accounts. Afterward, let's take a look at the SHARING_ACCESS table contents.

```
INSERT INTO SHARING_ACCESS values('REGION2', 'ACCT2');
INSERT INTO SHARING_ACCESS values('REGION3', 'ACCT3');
SELECT * FROM SHARING_ACCESS;
```

You'll notice in [Figure 6-9](#) that we have each region associated with a Snowflake account. You can see in the results that my Snowflake trial account identifier is

DX58224; your Snowflake account is different and will be unique to your organization.

	ID	SNOWFLAKE_ACCOUNT
1	REGION1	AE74983
2	REGION2	ACCT2
3	REGION3	ACCT3

Figure 6-9. Results of SELECT * On The SHARING_ACCESS Table

In our example, we've used ACCT2 and ACCT3 but if you were to create this SHARING_ACCESS table in your production Snowflake org you'd want to substitute actual Snowflake account identifiers for ACCT2 and ACCT3.

Our next action is to create a secure view where we'll join all the data in the SENSITIVE_DATA base table with the SHARING_ACCESS mapping table.

```
CREATE OR REPLACE SECURE VIEW DEMO10_DB.PUBLIC.PAID_SENSITIVE_DATA AS
    SELECT nation, price, size
    FROM DEMO10_DB.PRIVATE.SENSITIVE_DATA sd
    JOIN DEMO10_DB.PRIVATE.SHARING_ACCESS sa on sd.id = sa.id
    AND sa.snowflake_account = current_account();
```

We'll want to grant the SELECT privilege to the secure view to all roles which we can do by granting it to the PUBLIC role.

```
GRANT SELECT ON DEMO10_DB.PUBLIC.PAID_SENSITIVE_DATA TO PUBLIC;
```

Let's take a look at the SENSITIVE_DATA base table. Remember that this table holds all eight records.

```
SELECT * FROM DEMO10_DB.PRIVATE.SENSITIVE_DATA;
```

As expected, Figure 6-10 shows that we're able to view all eight of the records.

	NATION	...	PRICE		SIZE	ID
1	USA		123.5		10	REGION1
2	USA		89.2		14	REGION1
3	CAN		99		35	REGION2
4	CAN		58		22	REGION2
5	MEX		112.6		18	REGION2
6	MEX		144.2		15	REGION2
7	IRE		96.8		22	REGION3
8	IRE		107.4		19	REGION3

Figure 6-10. All Records in the SENSITIVE_DATA Table

Now, let's find out what records we are able to see in the secure view.

```
SELECT * FROM DEMO10_DB.PUBLIC.PAID_SENSITIVE_DATA;
```

Our current Snowflake account is mapped to Region1 which includes the USA nation. As expected, we see both USA records (Figure 6-11).

	NATION	PRICE	...	SIZE
1	USA	123.5		10
2	USA	89.2		14

Figure 6-11. Records in the PAID_SENSITIVE_DATA Secure View Accessible To Our Account

Let's use a session variable to simulate our Snowflake account being ACCT2 and take a look at what we'll be able to see in the secure view.

```
ALTER SESSION SET simulated_data_sharing_consumer='ACCT2';
SELECT * FROM DEMO10_DB.PUBLIC.PAID_SENSITIVE_DATA;
```

As ACCT2, we can see all of REGION2 data which includes the nations of Canada and Mexico (Figure 6-12)

	NATION	PRICE	...	SIZE
1	CAN	99		35
2	CAN	58		22
3	MEX	112.6		18
4	MEX	144.2		15

Figure 6-12. Records in the PAID_SENSITIVE_DATA Secure View Accessible to ACCT2

And, finally, let's see what is shown to us if set our Snowflake account to ACCT3.

```
ALTER SESSION SET simulated_data_sharing_consumer='ACCT3';
SELECT * FROM DEMO10_DB.PUBLIC.PAID_SENSITIVE_DATA;
```

As Figure 6-13 shows, we can see the nation of Ireland which is part of Region3.

	NATION	...	PRICE	SIZE
1	IRE		96.8	22
2	IRE		107.4	19

Figure 6-13. Records in the PAID_SENSITIVE_DATA Secure View Accessible to ACCT3

We want to return our Snowflake account to our original account.

```
ALTER SESSION UNSET simulated_data_sharing_consumer;
```

Now that we have successfully tested how our data will be shared with different accounts, we are ready to create a new share.

```
USE ROLE ACCOUNTADMIN;
USE DATABASE DEMO10_DB;
USE SCHEMA DEMO10_DB.PUBLIC;
CREATE OR REPLACE SHARE NATIONS_SHARED;
SHOW SHARES;
```

You'll notice that this new share will be included in the list of outbound shares. But we are not done yet. We need to grant privileges to the new share.

```
GRANT USAGE ON DATABASE DEMO10_DB TO SHARE NATIONS_SHARED;
GRANT USAGE ON SCHEMA DEMO10_DB.PUBLIC TO SHARE NATIONS_SHARED;
GRANT SELECT ON DEMO10_DB.PUBLIC.PAID_SENSITIVE_DATA TO SHARE NATIONS_SHARED;
```

You can use the SHOW GRANTS statement to confirm the contents of the share.

```
SHOW GRANTS TO SHARE NATIONS_SHARED;
```

Again, you'll notice that the grantee name has your current account information (Figure 6-14).

privilege	granted_on	name	granted_to	grantee_name	grant_option	granted_by
USAGE	DATABASE	DEMO10_DB	SHARE	AE74983.NATIONS_SHARED	false	ACCOUNTADMIN
USAGE	SCHEMA	DEMO10_DB.PUBLIC	SHARE	AE74983.NATIONS_SHARED	false	ACCOUNTADMIN
SELECT	VIEW	DEMO10_DB.PUBLIC.PAID_SENSITIVE_DATA	SHARE	AE74983.NATIONS_SHARED	false	ACCOUNTADMIN

Figure 6-14. Privileges Granted To the NATIONS_SHARED Share

Now, all that is left to be done from the provider perspective is to add accounts to the share.

```
//This is pseudocode; need to replace with actual Snowflake accounts  
ALTER SHARE NATIONS_SHARED ADD ACCOUNTS = ACCT2, ACCT3;
```

If you'd like to retrieve a list of all the Snowflake consumer accounts who have created a database from a share, you can use the following command:

```
SHOW GRANTS OF SHARE NATIONS_SHARED;
```

It is important to remember that Provider accounts pay for data storage costs of the shared data and consumer accounts pay for the virtual warehouse cost of compute for querying the data. This assumes that both the data provider and data consumer each have their own Snowflake account.

A situation could arise where a data provider would like to share data with a consumer who does not currently have a Snowflake account. In that case, the provider can establish and manage a Snowflake Reader Account. In situations where a Provider establishes a Reader Account for the consumer, the Reader Account will offer read-only capability for the consumer.

Setting Up a Reader Account

In the previous section, we navigated to the *Private Sharing* section of the *Data* main menu and clicked on the *Shared By My Account*. We then clicked on the specific share for which we wanted to add a consumer. If, instead, we want to create a Reader Account, we would click on the *Reader Accounts* section at the top of the screen and then on the *New* button. We would then be given a dialog page where we can create a Reader Account ([Figure 6-15](#))

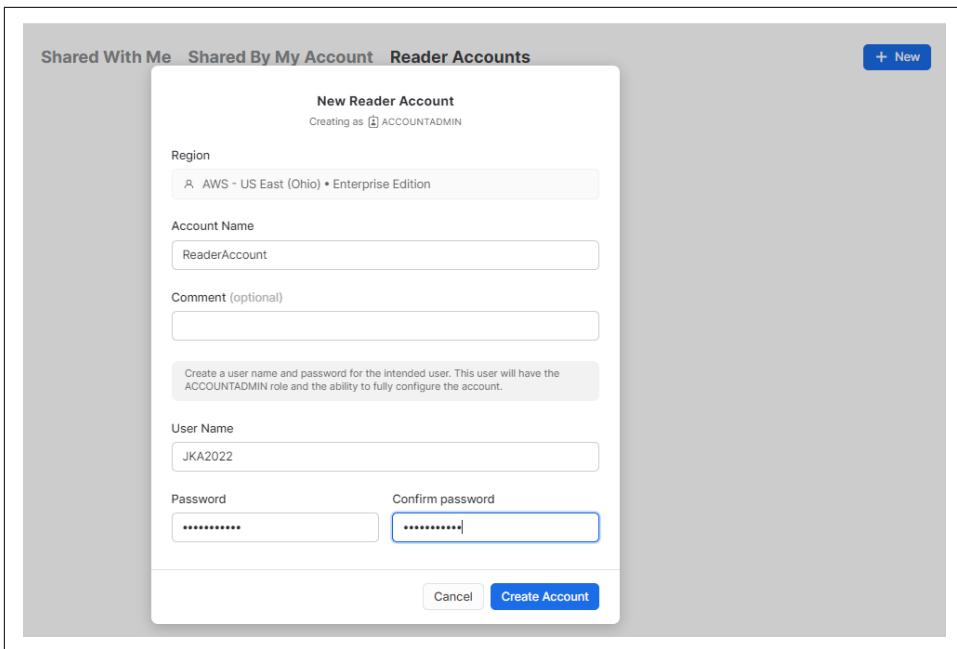


Figure 6-15. Creating A New Reader Account

The dialog page asks you to provide the Reader Account Name and the User Name and password for the person who will be given Admin privileges.



The reader account created will be the same edition, region, and cloud as your own Snowflake account and the Admin Login information is what you will use to manage the reader account.

You'll notice that after you create the Reader Account, if you click on the ellipsis on the right side, the option to *Drop* the Reader Account appears (Figure 6-16)

Shared With Me Shared By My Account Reader Accounts						+ New
1 Reader Account ⓘ						C
NAME ↑	CLOUD	REGION	LOCATOR	LOCATOR URL	CREATED	...
READERACCOUNT Pending	AWS	US East (Ohio)	YC36032	⌚ just now		...
						Drop

Figure 6-16. How To Drop A Reader Account

Once you click on it, you'll also notice the Locator URL for the Reader Account will copy the URL to your clipboard. You can then use that URL and the login information you created to log in to the Reader Account.

Once you are logged into the Reader Account as the Data Provider Admin, you'll want to create a database from an inbound share and set up a Data Consumer user account to assign to the person you want to have access to the Snowflake Reader Account.



[[TIP]] As the data provider, you are responsible for the compute charges incurred when the data is queried so you may want to consider setting up a resource monitor to limit the total costs or to be notified when a specified number of credits have been consumed.

How Inbound Shares are Used By Snowflake Data Consumers

Using the ACCOUNTADMIN role, a data consumer who is the owner of a Snowflake account can create a database from the provider's outbound share which is now the consumer's inbound share.

In the Snowflake worksheet, here is the statement that a consumer would use to create the shared database. Notice that in order for this statement to execute you'll need to include the database name and inbound share name. The statement would look something like this.

```
CREATE DATABASE <name_of_new_database> FROM SHARE <name_of_inbound_share>;
```

As previously discussed, it is not possible to edit shared data. Shared databases are read-only; thus, the data cannot be updated and no new objects within the database can be created by the data consumer. There are some additional limitations for this newly created database. One unique property of the shared database is that the comments cannot be edited.

The data consumer is unable to clone a shared database or the objects within it. However, it is possible to copy the shared data into a new table. While it is technically possible to make a copy of the shared database, it might be a violation of the terms of the contract if both parties entered into such an agreement.

For data consumers who are the owner of a Snowflake account where the database resides, there is no limit on the number of times that the data can be queried. It is important to keep in mind, though, that data consumers bear the compute cost of querying the data.

Understanding Consumer Accounts: Reader Accounts versus Full Accounts

Both reader accounts and full accounts are types of consumer accounts used in Snowflake data sharing. Reader accounts pay none of the compute costs for querying the data. Those costs are paid for by the provider account that created the reader account. As such, providers may choose to limit the queries of the reader accounts by using resource monitors. Reader accounts can be useful as an intermediate step for partners who are not yet ready to transition from their current data processes. Setting up and maintaining reader accounts for their partners' use does require additional effort by providers.



Full consumer accounts also have the ability to join their data with the shared data directly in Snowflake, while a Reader Consumer Account can only view the data shared with them in Snowflake.

How Account Usage Share is Different From All Other Inbound Shares

The Account Usage share, a type of inbound share, was first introduced in Chapter 3 when we compared the ACCOUNT_USAGE Share from the SNOWFLAKE Database to the INFORMATION_SCHEMA, a schema that is provided for each database. The SNOWFLAKE database, viewable only by the ACCOUNTADMIN by default, is similar to the INFORMATION_SCHEMA in that they both provide information about an account's object metadata and usage metrics. There are three major differences. The ACCOUNT_USAGE schema, with multiple views, includes records for dropped objects. It also has a longer retention time than the INFORMATION_SCHEMA. However, there is no latency when querying the INFORMATION_SCHEMA, unlike the ACCOUNT_USAGE share which has an average latency of about two hours.

Share privileges are imported and most inbound shares allow you to create and rename a database as well as drop the database from which you created from the inbound share. However, the Account Usage Share is different. You cannot create, rename, or drop the database associated with the Account Usage Share. You are not even able to add a comment. The Account Usage share is managed differently because it is a way Snowflake communicates with you about your account.

Throughout this chapter, when we refer to *inbound* shares, we are referring to all inbound shares other than the Account Usage share which is unique. Unlike the Account Usage share, inbound shares can be removed, or a new database can be created with a different name from the inbound share name.

Comparison Between Databases on Inbound Shares and Regular Databases

As we've seen, share privileges are imported and shared data is read-only. As shown in Table 10-1, inbound shares allow for creating and dropping a database for the inbound share but not to change the structure of the database.

	<i>CREATE, RENAME, or DROP Database</i>	<i>CREATE, ALTER or DROP Schema, Table, Stage, and/or View in the Existing Database</i>	<i>GRANT / REVOKE ALL, MONITOR or CREATE Schema in the Existing Database</i>
Inbound Share	YES*	NO	NO
Regular Database	YES	YES	YES

*except for the Account Usage share

Figure 6-17. Table 10-1 Table showing the difference between an Inbound Share and a Regular Database

One important note is that, unlike most traditional databases, Snowflake supports cross database joins, including joins with databases built from inbound shares (see Figure 6-18). Consumers, other than those with Reader Accounts, can also create views in their own database which can combine their own data with data from the shared database. This provides great benefit because the consumer can enrich their own data to make it even more valuable.

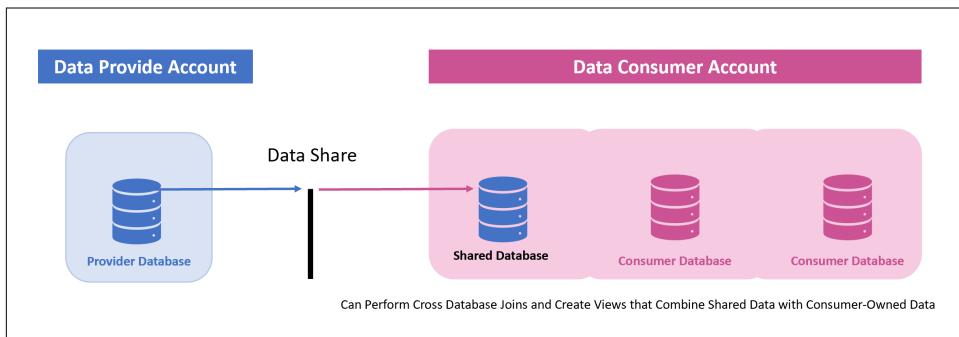


Figure 6-18. Shared Data can be used in combination with Consumer-Owned Data

When new records are created within the provider's Snowflake account as part of the shared database, the records are instantly available in a data consumer's inbound shared database. However, when new objects are created within the shared database by the provider, those objects are not automatically shared with the data consumer. The provider has to grant authorization to the data share before the records in those objects are viewable by the consumer.

How to List and Shop on the Public Snowflake Data Marketplace

Data providers can be publicly connected with data consumers globally via the Snowflake data marketplace. As a potential consumer of third-party data, you can access the data marketplace directly from the Snowflake user interface.

First, make sure you have your role set to ACCOUNTADMIN and then click on the Marketplace option, as shown in Figure 10-18. If your role is not currently set to ACCOUNTADMIN, be sure to click on the drop-down arrow next to your name and change your role.

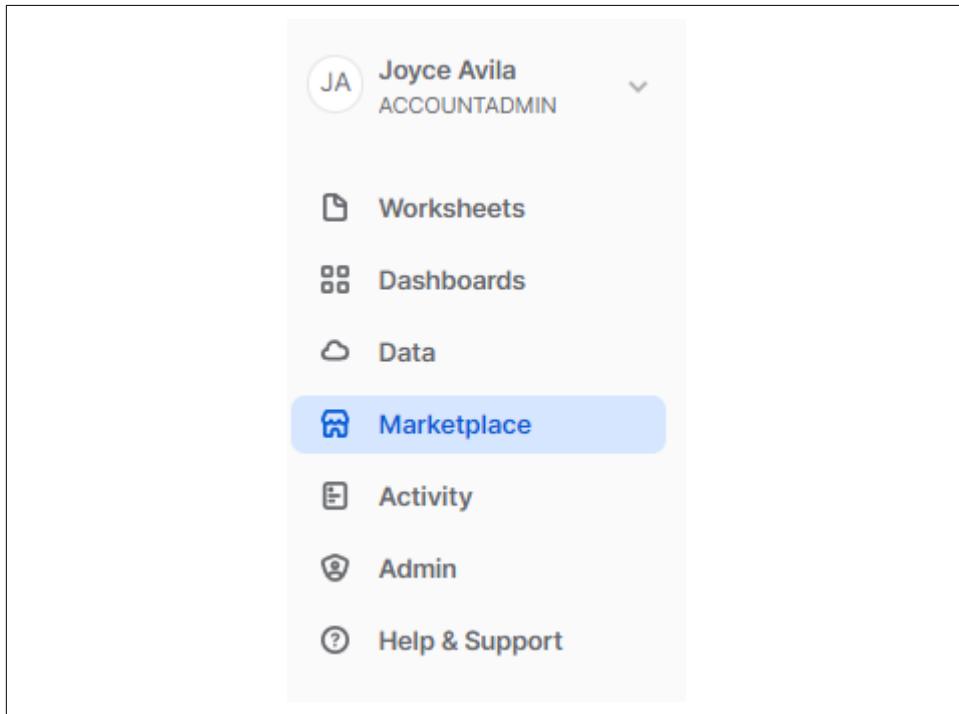


Figure 6-19.

Figure 10-18 Accessing Data Marketplace directly from the user interface

From your Snowflake UI, you can now search the Data Marketplace (see Figure 10-19). When you search the marketplace, you'll find standard listings, which are available for you to access immediately, and personalized listings for which you can send a request to access.

The screenshot shows the Snowflake Data Marketplace homepage. At the top, there's a search bar with placeholder text "Search Snowflake Marketplace". Below it are navigation links for "Categories", "Business Needs", "Providers", and "My Requests". A horizontal menu bar includes filters like "Ready to Query", "Free", "Weather", "Financial", "360-Degree Customer View", "Advertising Optimization", "Japan", and "Last month".

Featured Providers:

- Funnel
- Dun & Bradstreet
- Equifax
- ADP, Inc.

Most Recent:

- Snowflake Inc. - Quarterly Financial Statements, Snowflake Financial Data
- Spatial Risk Systems - Spatial Risk Factors - (Demo/Sample), Location based Risk Scores - Quantifies Real-World Impact on Assets, Municipalities and Organizations.
- Gravy - Sample Personas, Over 350 consumer personas associated with a U.S. mobile device ID
- Sonarverse - BNB Chain (Blockchain), Complete BNB chain dataset of blocks, transactions, token transfers, tokens, logs, receipts, and contracts. (Personalized)

Financial:

- Quantify Inc. - SEC Form 4 Insider Transaction Activity, Query insider transaction activity
- SafeGraph - Transactions at Individual POI - US, Transactions at individual points of interest
- GlobalData - Job Analytics Data Solutions, Daily updated database of 13m jobs covering 20+ sectors across 200 countries with thematic tagging
- Token Flow Insights - Ethereum Data Warehouse beta, Full detailed history of the Ethereum blockchain.

Figure 6-20.

Figure 10-19 Snowflake Data Marketplace

Some third-party data providers offer data access for free or sometimes charge you a fee to access their data. They may also require a contract to ensure that you'll not reshare their data. Whether the access to the data is free or the provider charges you a fee, that arrangement between you and the data provider is separate from your arrangement with Snowflake. You'll be billed by Snowflake only for the compute you use to query the shared data. The data storage charges are the responsibility of the data provider.

Snowflake Data Marketplace for Providers

For data providers, the Snowflake Data Marketplace provides an opportunity to monetize data in ways never before possible. You can provide secure, personalized data views using your own customized business model. Snowflake isn't a party to the transaction between a data provider and data consumer. There is no cost for the act of

sharing data on the Data Cloud. Instead, Snowflake enables data sharing on its platform and earns revenue when data consumers use virtual warehouses to query the data.

To become an approved Snowflake data provider, you'll need to submit a request and sign the Snowflake Provider agreement and also agree to operate within the Snowflake Provider Policies. Importantly, your data will have to meet certain requirements.

One requirement is that your data must be fresh and non-static, meaning that the historical data you'll provide must be relatively recent and that you have a plan to continue updating the data going forward.

Another requirement is that the data cannot be sample, mocked-up data. To be clear, sample data can be provided to help consumers determine if they want to use the full data set or not. The full data set itself, though, cannot be only sample or mocked-up data.

Finally, you must also possess the right to make the data available and ensure that the data doesn't include private data. For each of your listings, object identifiers for tables, columns, and share names must be *upper case* and use only *alphanumeric* characters. You'll also need to provide at least one usage example for each of your listings. The Zoominfo usage example is shown in [Figure 6-21](#).

Usage Examples

ZoomInfo SQL Query

Select all companies and their core firmographics + advanced attributes including employee range, revenue range, primary industry

```
1  SELECT "ZoomInfo Company ID",
2  "Company Name",
3  "Website",
4  "Company HQ Phone",
5  "Fax",
6  "Ticker",
7  "Revenue (in 000s)",
8  "Revenue Range",
9  "Employees",
10 "Employee Range",
11 "SIC Code 1",
12 "SIC Code 2",
13 "SIC Codes",
14 "NAICS Code 1",
15 "NAICS Code 2",
16 "NAICS Codes",
17 "Primary Industry",
18 "Primary Sub-Industry",
19 "All Industries",
20 "All Sub-Industries",
21 "Industry Hierarchical Category",
22 "Secondary Industry Hierarchical Category",
23 "ZoomInfo Company Profile URL",
24 "Ownership Type",
25 "Business Model",
26 "Certified Active Company",
27 "Company Street Address",
28 "Company City",
29 "Company State",
30 "Company Zip Code",
31 "Company Country",
32 "Company Is Acquired",
33 "Company ID (Ultimate Parent)",
34 "Entity Name (Ultimate Parent)",
35 "Company ID (Immediate Parent)",
36 "Entity Name (Immediate Parent)",
37 "Relationship (Immediate Parent)"
38 FROM "PROD_MASTER_DB"."COMPANY"."COMPANY_DATABRICK_TABLE"
```

Show less ^

Copy

Figure 6-21. Zoominfo Usage Example for the Snowflake Data Marketplace

Provider Studio

Provider Studio offers the ability to perform required actions to get listings approved and to review data request from consumers. Additionally, Provider Studio makes it possible for Snowflake providers to view key analytics relating to Marketplace listings by selecting the Analytics tab. Access to Provider Studio is available only to Snowflake data providers.

Standard Versus Personalized Data Listings

All Snowflake Data Marketplace listings include details about the shared data as well as sample queries and usage examples. Listings also include information about the data provider. The difference between the two types of listings, standard and personalized, is whether the shared data is instantly accessible or not.

A Snowflake Data Marketplace standard listing which provides instant access to a published data set is best used for generic, aggregated, or non-customer specific data. A personalized data listing is one where a request to access to data sets must be made. Once the request is made, the data provider is notified and then contacts the consumer.

An example of a standard listing is shown in [Figure 6-22](#).

The screenshot shows a standard listing for "US Open Census Data & Neighborhood Insights" by SafeGraph. The listing includes a thumbnail icon, the title, and a brief description: "SafeGraph's Open Census Data contains 7500+ demographic attributes (like income, age, education, etc.) available at the Census Block Group level. All data from the American Community Survey is available in bulk with a clean schema and joined with Census Block Group (CBG) geometry. This dataset is also available for download in all regions." It indicates the listing is "Free" with "Unlimited queries" and features a prominent blue "Get Data" button. Below the main description, there's a section titled "Primary Fields Included:" with a "Show More" dropdown. To the right, there's a sidebar with the SafeGraph logo and a brief company description: "SafeGraph is a data company that seeks to unlock understanding of the physical world and power innovation through open access to geospatial data. SafeGraph has built the source of

Figure 6-22. Snowflake Data Marketplace Standard Listing Example

You can tell that it is a standard listing because *Personalized* wasn't included on the listing's main page and when you click on the listing, you'll see the *Get Data* button. When you click on that button, you'll see the details about the listing and one final button you'll need to click ([Figure 6-23](#)). Notice that you can add more roles, in addition to the ACCOUNTADMIN role, to give more roles access to the shared database.

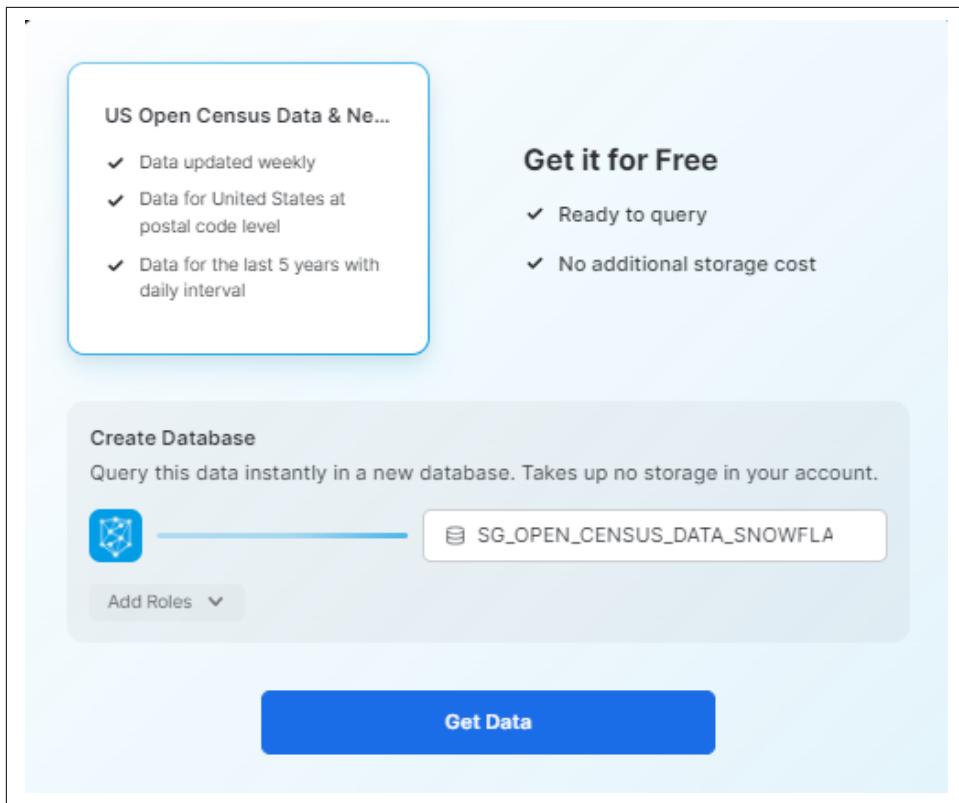


Figure 6-23. Ability to Add Additional Roles Before Getting the Data From Data Marketplace Standard Listing

When you click on the Get Data button, you'll see that the data is now ready to query (Figure 6-24)

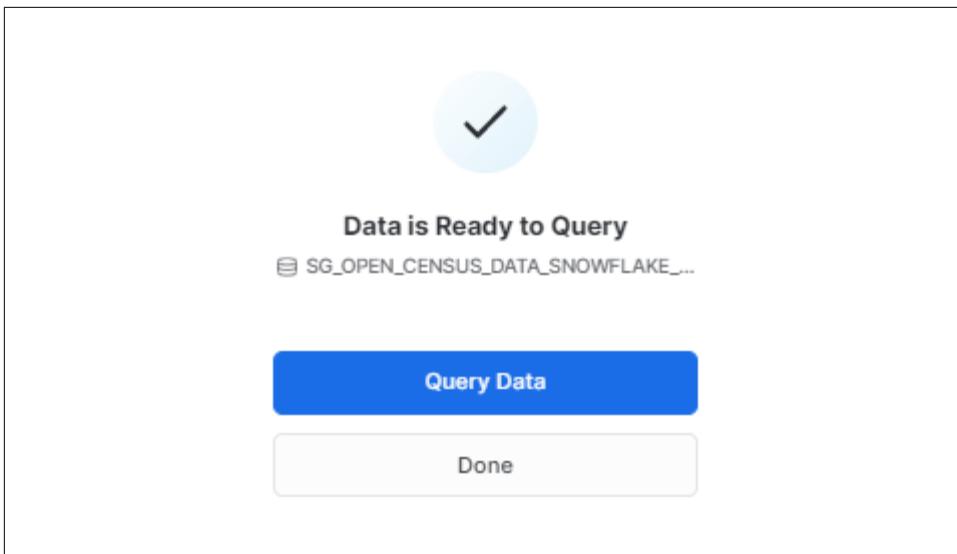


Figure 6-24. Confirmation That The Data From Data Marketplace Standard Listing is Ready to Query

Figure 6-25 shows Zoominfo's example of a personalized listing in the Snowflake Data Marketplace. You would click on the *Request Data* button to request access from Zoominfo.

The screenshot shows a detailed listing for ZoomInfo Data-as-a-Service. The main content includes:

- ZoomInfo Data-as-a-Service**: Industry-leading company and business contact data for B2B intelligence.
- Use Cases:** B2B Marketing, Operations, Credit and risk management, Advanced analytics, Data science and machine learning, Master Data Management, Data quality.
- Available Data Sets:** Show More ▾
- Business Needs**: 360-Degree Customer View, Contact Data Enrichment.
- Personalized Data**: Available on request, Request Data button.
- Contact Information**: ZoomInfo, More >, Contact, Documentation, Support, Terms of Service.

Figure 6-25. Zoominfo Personalized listing on the Snowflake Data Marketplace

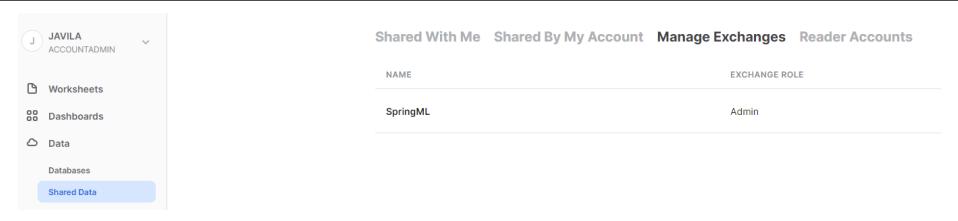
Today, there are hundreds of providers who participate on the Snowflake Marketplace and hundreds of listings that include many well-known data providers. The latest counts and details of all Snowflake Marketplace data sets can be found at <https://www.snowflake.com/data-marketplace>.

Harnessing the Power of a Snowflake Private Data Exchange

A Snowflake Data Exchange enables organizations to privately share data sets with a select group of participants. In this way, organizations can create their own private version of a Data Marketplace but with control over what data can be listed and who can access the listings. A Snowflake Data Exchange is a great option for a company with different divisions, each of which have their own Snowflake account and want to have control over which data is shared organization wide. A Snowflake Data Exchange could also be used by a retailer who wants to share data with their suppliers.

Getting a Snowflake Data Exchange set up is a fairly easy process and generally only takes one or two business days. You will want to navigate to the Snowflake support page and submit a case. You'll need to provide a business case justification, a unique name for the new data exchange as well as the data exchange display name, your account URL, and your organization ID, if you are using the organization Snowflake feature.

Once you have set up your Data Exchange, you can access it by selecting *Data >> Shared Data* and then selecting *Manage Exchanges*. You'll then see the private Data Exchange. In this example, I've created a Data Exchange with the name *SpringML* (Figure 6-26).



The screenshot shows the Snowflake interface for managing data exchanges. On the left, there's a sidebar with a user icon and the name 'JAVILA ACCOUNTADMIN'. Below this are links for 'Worksheets', 'Dashboards', 'Data', 'Databases', and 'Shared Data', with 'Shared Data' being highlighted. The main area has tabs: 'Shared With Me', 'Shared By My Account', 'Manage Exchanges' (which is selected and highlighted in blue), and 'Reader Accounts'. Below these tabs is a table with two columns: 'NAME' and 'EXCHANGE ROLE'. There is one row in the table with the name 'SpringML' and the role 'Admin'.

NAME	EXCHANGE ROLE
SpringML	Admin

Figure 6-26. Managing A Private Snowflake Data Exchange

Once you click on the name of the exchange, you'll see the details of the Data Exchange and you'll have the ability to add a member to your Data Exchange (Figure 6-27).

The screenshot shows the Snowflake interface for managing a Private Data Exchange. On the left, a sidebar menu includes options like Worksheets, Dashboards, Data, Databases, Shared Data (which is selected and highlighted in blue), and Marketplace. The main content area is titled 'SPRINGML' and shows a table of members. The table has columns for ACCOUNT NAME, REGION, and EXCHANGE ROLE. One member is listed: 'YQ77172' from 'GCP - US Central (Iowa)' with the role 'Admin'. At the top right of this section is a blue 'Add Member' button.

Figure 6-27. Details Of A Snowflake Private Data Exchange

If you click on the Add Member button, you'll have the option to add a new account and give them the ability to participate in the Data Exchange as a Consumer, Provider, or both (Figure 6-28).

The screenshot shows the 'Add Member' dialog box. It has a title 'Add Member' at the top. Under the 'Account' section, there is a text input field containing the URL 'https://accountname.snowflakecomputing.com'. Below the input field is a placeholder text: 'Enter a Snowflake account URL to add as a member.' Under the 'Role' section, there are two checkboxes: 'Consumer' (checked) with the description 'Discover and get data from listings.' and 'Provider' (checked) with the description 'Publish listings and review requests from consumers.' At the bottom right of the dialog are two buttons: 'Cancel' and 'Add' (in a blue box).

Figure 6-28. Adding Members To a Private Data Exchange

Snowflake Data Exchanges are a great way to scale your business by inviting partners, customers, vendors, and others to participate.

Snowflake Data Clean Rooms

A data clean room on Snowflake is a design pattern for sharing data between two or more parties in a privacy protected way. A popular use case of data clean rooms is to link anonymized marketing and advertising data from multiple participants. This gives participants a way to adhere to privacy laws, such as the GDPR and the California Consumer Privacy Act (CCPA), because data clean rooms don't allow data points that can be tied back to a specific user to leave the environment.

Traditional data clean rooms require all data to be stored in one physical location. In contrast, a data clean room on Snowflake does not require participants to move their data from its existing location in order to list their data in the exchange. Instead, par-

ticipants use Secure Data Sharing where PII data included in the clean room is secured and encrypted.

Using Snowflake to create distributed data clean room functionality is unique in that Snowflake provides the ability for each data clean room participant to control its own data without having to trust a single party with all the data. In addition to limiting who can access the data, private data exchange participants can use secure functions and access controls to protect their data while still allowing join data analysis to occur.

A data clean room on Snowflake starts with setting up a private data exchange and uses current Snowflake features to allow the parties involved to join data in a single query that produces aggregate results, without allowing access to the underlying low-level data. Secure functions, as well as joins, can be used to establish individual links to people, devices, or cookies without needing to exchange or make visible any personally identifiable information.

In addition to the Snowflake secure data sharing technology, which automatically and securely shares tables between multiple Snowflake accounts, there are other technology features used for data clean rooms. Row access policies match customer data without exposing PII, and stored procedures validate query requests. Snowflake streams and tasks monitor for data clean room requests and respond automatically. Guidelines are established to determine what data is allowed in and what data, if any, can leave from a distributed data clean room on Snowflake.

Important Design, Security and Performance Considerations

Creating Snowflake shares are relatively quick and easy for providers to set up and make available for consumers. Before creating a new share, however, it is a good idea to think about the design, security, and performance implications for shares.

Share Design Considerations

As records get added to a provider's object that is shared with the consumer, those records are immediately available to the consumer. However, if an object such as a new table gets added to the database, the new table is not automatically shared with the consumer. The producer must take action to share that new object.

From a technical perspective, different shares can use the same schema, but then it will appear as if all the objects are part of a share even when they aren't. This makes it particularly difficult when adding new objects to existing shares that are part of the same schema.



TIP: Separate schemas for each listing is not required. However, from a design perspective, planning to create a separate schema for each listing could reduce the possibility of confusion and accidentally sharing an object by mistake.

An inbound share can only contain one database but it is possible for a producer to share data from multiple databases by using a secure view. A Snowflake provider can create a secure view which references schemas, tables, and other views from several databases as long as those databases belong to the same account.

Share Security Considerations

Sharing a secure object works the same way as sharing a table. Rather than sharing the table, you could easily devise a way to share securely. One option is to create an internally available schema where the base table resides and then create an externally available schema for the secure object. That way, you can share the external schema and secure object without exposing the base table.



The following Snowflake objects can be shared: tables, external tables, secure views, secure materialized views, and secure UDFs. However, to ensure sensitive data in a shared database is not exposed to users in consumer accounts, it is strongly recommended that you share secure views or secure UDFs rather than directly sharing tables.

By using dynamic data masking or row access policies, it is possible to share a portion of the data in a base table with different consumer accounts without having to create separate views for each consumer.



It is also possible to share only specific rows of data through the use of the `CURRENT_ACCOUNT()` function. It's important to note that this is only possible to use `CURRENT_ACCOUNT()` for Secure Data Sharing when using secure views and secure materialized views.

Shared Table Performance Considerations

When sharing data from massively large tables, it is recommended that you, as the provider, define clustering keys on the tables. This will prevent your consumers from being negatively impacted with degraded performance when they query the database share.

Difference Between Database Sharing and Database Cloning

Often times, the question is asked *What is the difference between cloning a database and using the Snowflake Secure Data Sharing feature?* The answer is that database cloning effectively takes a snapshot of the database and while the original database continues to change, the cloned database does not. It is important to note that instead of actually making a physical copy of the data, the metadata is used to display a view of the original database as it existed at that point in time. The original database and the cloned database both exist within the same account. In contrast, data sharing occurs across different accounts and the shared data is a live view of the data which changes whenever changes are made in the original database.

Data Shares and Time Travel Considerations

Snowflake time travel makes it possible to access historical data at a specified point in time. For security reasons, time travel is not enabled for data shares. Otherwise, the consumer might travel back to a time when they could access a view they shouldn't be allowed to access.

Sharing of Data Shares

A consumer cannot reshare a shared object to another account. A shared object also cannot be cloned by a consumer. However, a consumer could copy the data into a table to create a copy of the data that technically could be shared. This risk isn't just limited to data shares, though. Any data shared with others via traditional methods such as an API or FTP, for example, are at risk of being further shared. Really, anyone who owns a mobile phone and has access to your data can copy and share it. As such, any restrictions, such as not allowing shared data to be shared with anyone outside of the organization, is best taken care of using contractual agreements.

Summary

In this chapter, we've been reminded how Snowflake's architecture supports Secure Data Sharing and have considered specific use cases to highlight the importance of data sharing. We've taken a deep dive into the four Snowflake Secure Data Sharing approaches; direct Secure Data Sharing, the private Data Exchange, the public Snowflake Data Exchange, and Data Clean Rooms. Differentiating between the provider and consumer roles for Secure Data Sharing became more evident as we focused our attention on learning how to create outbound shares and understanding how inbound shares are used by Snowflake data consumers.

Having enough data, going both deep and wide, as well as having the right kind of data often means we need to incorporate external, or third-party, data. Snowflake's Secure Data Sharing capabilities allow us to do just that. Keeping our own data secure

and having access to the shared data we need is a necessary first step for being able to extract valuable data insights. One important way we can obtain better insights is by visualizing data. In the next chapter, *Visualizing Data for Better Insights*, we'll get some hands-on experience to understand the visualization capabilities of Snowsight and we'll learn about Snowflake partners who offer powerful data visualization tools native to Snowflake.

As a reminder, we used the ACCOUNTADMIN role throughout the chapter for simplicity, but best practice is to use the SYSADMIN role to create new Snowflake objects and to grant privileges as needed to custom roles.

Code Cleanup

All that is necessary for the code cleanup for this chapter is to drop the database we created. However, what happens when you attempt to do that?

```
USE ROLE ACCOUNTADMIN;  
DROP DATABASE DEMO10_DB;
```

You'll notice that you receive an error because you cannot drop a database when there are active shares associated with the database (Figure 6-29). So, we need to revoke share access to the database if we want to drop it.

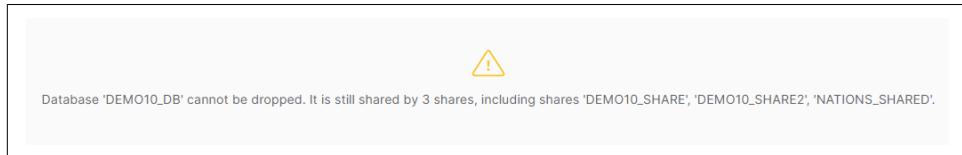


Figure 6-29. Error Message Received When Attempting to Drop a Database Which Has Active Shares

Let's revoke privileges and then see if we are able to drop the database.

```
REVOKE USAGE ON DATABASE DEMO10_DB FROM SHARE NATIONS_SHARED;  
REVOKE USAGE ON DATABASE DEMO10_DB FROM SHARE DEMO10_SHARE;  
REVOKE USAGE ON DATABASE DEMO10_DB FROM SHARE DEMO10_SHARE2;  
DROP DATABASE DEMO10_DB;
```

What are some things you just noticed? We only needed to revoke access at the database level because once that database access was removed, there was no longer access at the schema or table level. Also, we only need to drop the database which will then result in any schemas, tables, and views within the database to be dropped.

Knowledge Check

The following exercises are based on the data sharing topics in this chapter.

1. You need to use a secure view, rather than a non-secure view, to create an out-bound data share. What is one visual cue to easily know that a view is secure?
2. There is a standard database symbol assigned to all newly created databases. However, the SNOWFLAKE database symbol looks different. Why?
3. Can a data share include more than one database?
4. What is the difference between the Snowflake Data Marketplace and a Snowflake Data Exchange?
5. Which role(s) can be used to create and maintain data shares?
6. Which Snowflake account is assessed data storage charges, in relation to data shares?
7. Who pays the virtual warehouse compute charges incurred when querying a data share?
8. Explain the term *Snowgid*.
9. What Snowflake objects can be used to create data shares? Are there any best practices related to selecting the objects to create data shares?
10. What are the requirements that must be met in order to receive approval to list on the Snowflake Data Marketplace?

Solutions to these exercises are available in Appendix A.

Visualizing Data for Better Insights

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 11th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

Snowsight replaced the Snowflake SQL Worksheet as the new Snowflake web user interface. In large part, Snowsight has increased Snowflake’s usability by providing access to worksheet history and by providing the ability to switch between Snowflake accounts. Additionally, Snowsight’s SQL query formatting, UI keyboard shortcuts, and autocomplete allow Snowflake users to write queries more efficiently. Thus, Snowsight has become a valuable data visualization tool with the ability to collaborate with other Snowflake users.

In this chapter, we will work together through the following topics. The code in the chapter is also available on Github.

- How to Access Snowsight
- Navigating Snowsight and Data Sampling
- Improved Productivity
- Visualization

- Collaboration
- Test Your Knowledge

How to Access Snowsight

If you are not currently logged in as an ACCOUNTADMIN, be sure to change your role to ACCOUNTADMIN. You can access Snowsight in one of two ways. In the Classic Console Web Interface, you click on the “Preview App” button in the top right corner of the screen (Figure 7-1) or you can log directly into Snowsight: <https://app.snowflake.com/>.

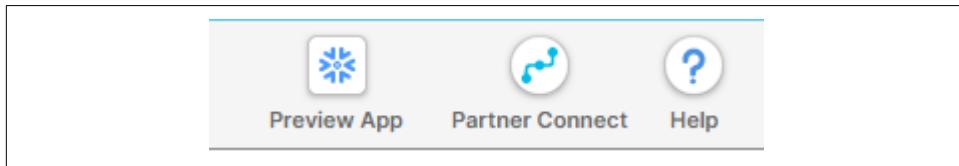


Figure 7-1. Classic Console Web Interface showing the “Preview App” button

Once inside Snowsight, “Worksheets” is the default tab (Figure 7-2). You can also click on some of the different tabs, including the “Data” tab and the “Compute” tab to see some of the available options. As we will see later, the “Databases” sub-tab will display the databases available to you within your access rights.

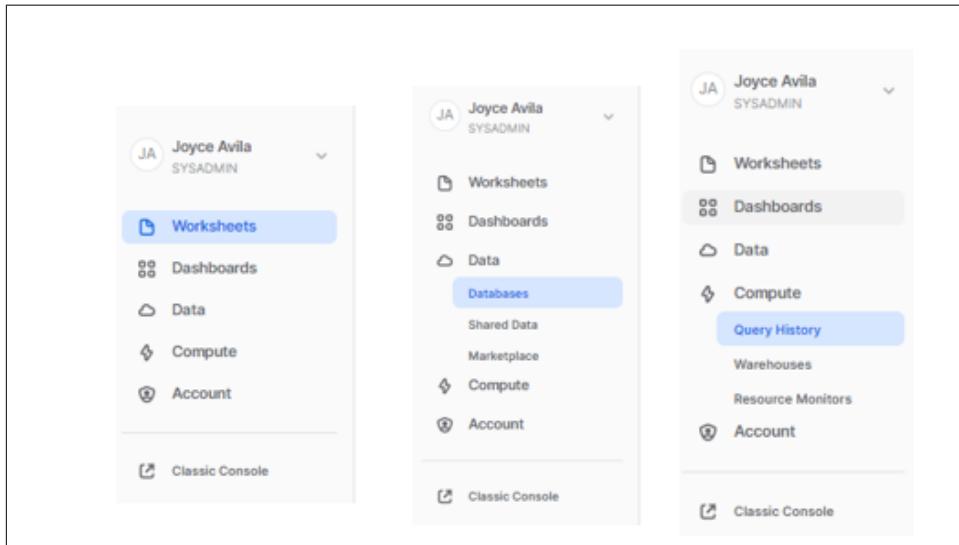


Figure 7-2. Snowsight UI tabs with Worksheet tab being the default

If you have been working in the Classic Web Interface, you'll be presented the option to import your Worksheets when you first come over into Snowsight ([Figure 7-3](#)).

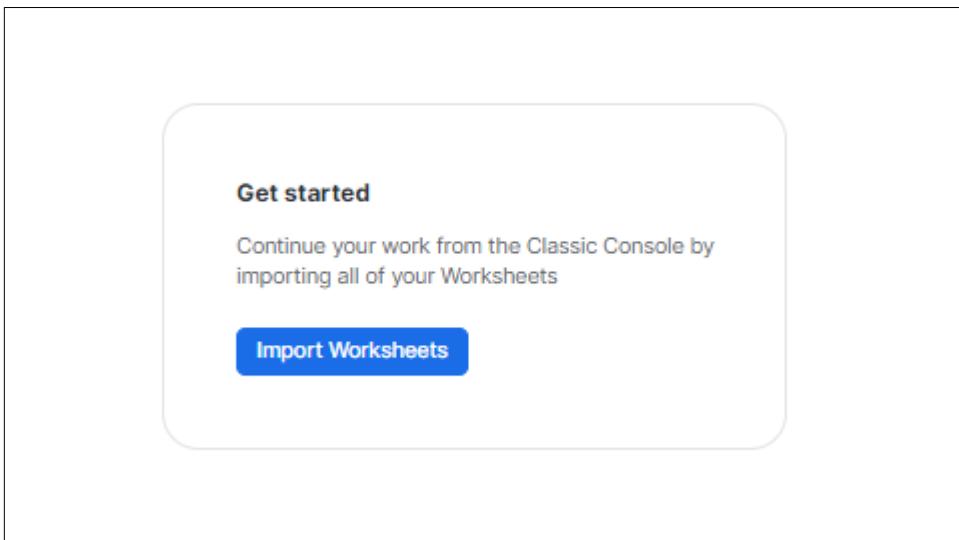


Figure 7-3. Option to import worksheets is presented to you the first time you use Snowsight

If you import a Worksheet from the Classic UI, a new timestamped folder will be created ([Figure 7-4](#)).

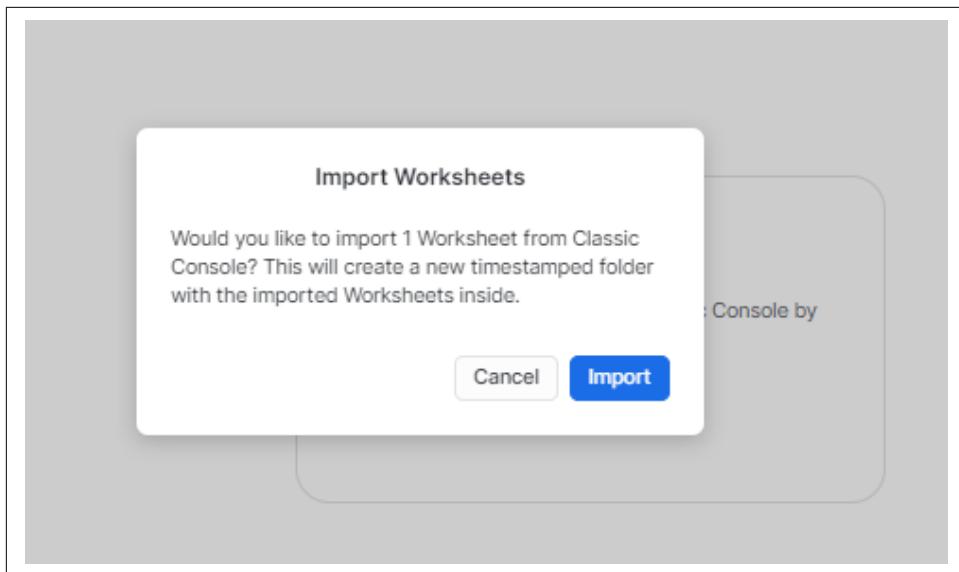


Figure 7-4. Importing worksheets from the Classic Console will create a new timestamped folder

If you click on the Worksheets tab and then “Folders”, you will see the new folder that was created (Figure 7-5).

A screenshot of the Snowsight interface. On the left is a sidebar with a user profile (Joyce Avila, SYSADMIN) and navigation links: Worksheets (highlighted in blue), Dashboards, Data, Compute, and Account. The main area is titled 'Worksheets' and shows tabs for Recent, Shared with me, My Worksheets, and Folders (which is selected). Below the tabs is a 'TITLE' dropdown and a list of folders. One folder is visible: 'Import 2021-06-19'.

Figure 7-5. New folder “Import 2021-06-19” was created when a worksheet was imported

As you navigate around Snowsight and do the exercises in this chapter, one of the things you will notice is that everything in Snowsight is auto saved.

Navigating Snowsight and Data Sampling

In this section, we'll discover how to navigate in Snowsight while we learn about data sampling in Snowflake. Sampling data is helpful for both productivity and visualization purposes. After loading large datasets, ETL developers or software engineers who are responsible for loading data can scan a sample of the data to ensure that nothing unusual stands out. Likewise, business analysts can view a sample of unfamiliar data as a first step in exploring the data and auditors can obtain a sample of the data on which to perform analysis. There are limits to the number of records that can be used for Snowsight charts and dashboards.

Sampling allows you to create a dashboard with a limited subset of the data that closely resembles the full data set. Sampling returns a subset of rows from a specified table. The number of rows returned from sampling a Snowflake table depends on whether you choose to return rows based on a specific number or return a certain percentage of table rows:

Number of rows

It is possible to sample a fixed number of rows specified in the query. If you want a data sample rather than all the data rows returned, be sure to specify the number of rows for sampling to be less than the total number of rows in the table. The number of rows can be any integer between 0 and one million. Note that the Bernoulli sampling method is used when you sample by number of rows and System/Block and Seed are not supported in this case. Bernoulli sampling is a sampling process where each row would have an equal chance of being selected.

Probability

You can sample a portion of the table with the exact number of rows returned dependent on the size of the table. It is possible to specify a seed, making the sampling deterministic, and then state the specified probability for a particular row being included in the sample. Either the Bernoulli or System sampling method can be used. System sampling is often faster than Bernoulli sampling, but the sample could be biased, especially for small tables. If not specified, Bernoulli is the default sampling method.

Keywords that can be used interchangeable include: “SAMPLE” and “TABLESAMPLE”, “BERNOULLI” and “ROW”, “SYSTEM” and “BLOCK”, and “REPEATABLE” and “SEED”.

To prepare for our first chapter exercise, let's create a new folder. On the top right corner, click on the ellipsis and then click on “New Folder” ([Figure 7-6](#)).

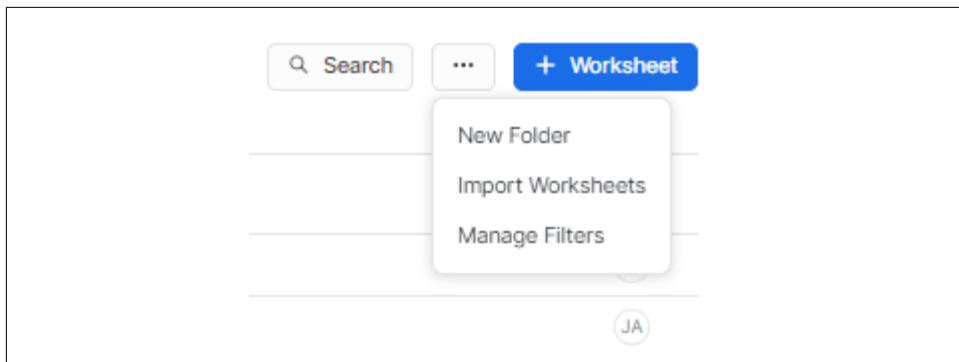


Figure 7-6. Clicking on the ellipsis offers choices for the user, including creating a new folder

Name the new folder “Sampling” and then click on (+ Worksheet) to create a new worksheet (Figure 7-7).

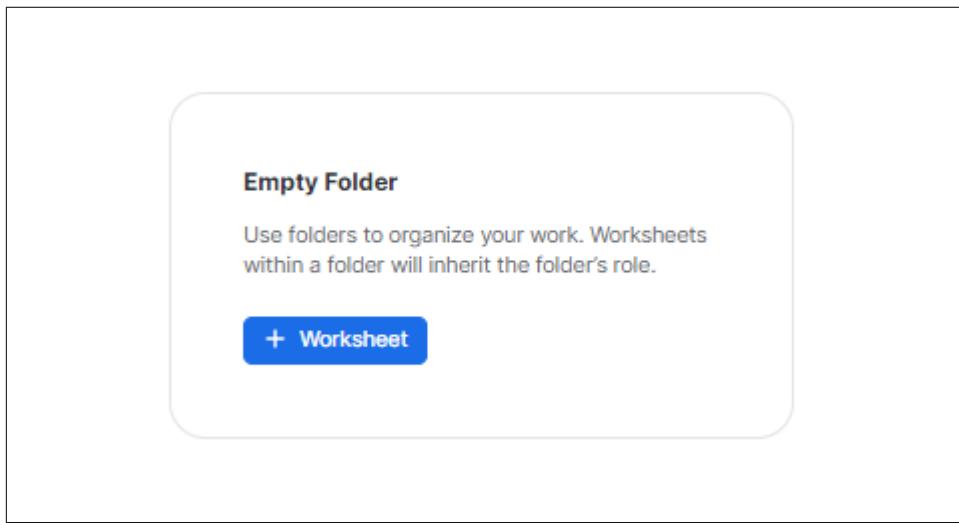


Figure 7-7. Creating a new worksheet within Snowsight

You should now see where you can begin entering in your query to obtain a sample (Figure 7-8).



Figure 7-8. Entering a Query in Snowsight

First, we have to select the database and schema (Figure 7-9). To the right of “No Database selected”, click on the drop-down arrow and select the database “SNOWFLAKE_SAMPLE_DATA”.

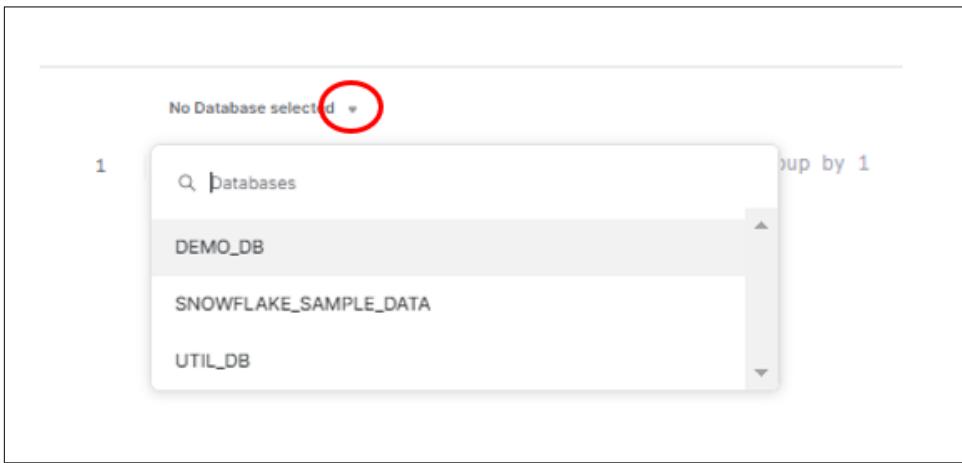


Figure 7-9. Selecting the database to use for the query

Next, select the TPCDS_SF100TCL schema. You can also open the schema and the tables on the left so you can see the tables you will have to work with (Figure 7-10).

The screenshot shows the Snowflake UI interface. At the top, there is a header bar with a home icon, the word "Sampling", the date "2021-06-20 1:52pm", and a dropdown arrow. Below the header is a pinned objects section with a note "No pinned objects". To the right, a code editor window displays a SQL query: "select :datebucket(created), count(1) from table group by 1". The main area is titled "Databases" and shows a tree view of schemas: DEMO_DB, SNOWFLAKE_SAMPLE_DATA (which is expanded to show INFORMATION_SCHEMA, TPCDS_SF1000, TPCDS_SF10000, and TPCDS_SF100TCL), and a Tables section which is also expanded to show all 31 tables listed below.

- Pinned (0)
- No pinned objects

```
SNOWFLAKE_SAMPLE_DATA.TPCDS_SF100TCL
1 | select :datebucket(created), count(1) from table group by 1
```

Databases

- > DEMO_DB
- > SNOWFLAKE_SAMPLE_DATA
 - > INFORMATION_SCHEMA
 - > TPCDS_SF1000
 - > TPCDS_SF10000
 - > TPCDS_SF100TCL
 - Tables
 - CALL_CENTER
 - CATALOG_PAGE
 - CATALOG RETURNS
 - CATALOG SALES
 - CUSTOMER
 - CUSTOMER ADDRESS
 - CUSTOMER DEMOGR...
 - DATE DIM
 - DBGEN VERSION
 - HOUSEHOLD DEMOOG...
 - INCOME BAND
 - INVENTORY
 - ITEM
 - PROMOTION
 - REASON
 - SHIP MODE
 - STORE
 - STORE RETURNS
 - STORE SALES
 - TIME DIM
 - WAREHOUSE
 - WEB PAGE
 - WEB RETURNS
 - WEB SALES
 - WEB SITE

Figure 7-10. List of tables within the SNOWFLAKE_SAMPLE_DATA schema

Click on the “STORE_SALES” table and you’ll see a preview that shows there are more than 288 billion rows in the table. Click on the magnifying glass on the right

side ([Figure 7-11](#)) and Snowflake will return the first 100 rows for you to view. Scan through the rows to get a sense of the data that exists.

STORE_SALES	288.0B Rows	
123 SS_SOLD_DATE_SK		
123 SS_SOLD_TIME_SK		
123 SS_ITEM_SK		
123 SS_CUSTOMER_SK		
123 SS_CDEMO_SK		
123 SS_HDEMO_SK		
123 SS_ADDR_SK		
123 SS_STORE_SK		

Figure 7-11. Using the Magnifying Glass will result in the first 100 rows being returned

Once you have entered your SQL statement to obtain 100 records by limiting the number of rows returned, position your cursor either above or within the query text in the text editor. Next, click the Run button which is a blue arrow within a circle located at the top right of the screen, or type [CMD] + [RETURN] for Mac or [CTRL] + [ENTER] for Windows.

```
Select * from store_sales limit 100;
```

When you scroll down through the results, you'll see that exactly 100 rows were returned. They are different from the 100 rows that you previewed. Run the query again and you'll notice that exactly the same 100 records are returned. Will the same thing happen when we use sampling? Let's see. Query the same table to have 100 rows returned but, this time, use "sample" instead of "limit".

```
Select * from store_sales sample (100 rows);
```

Sample query results are unique in that the results are not cached. As a result, you'll receive a different subset of 100 records if you run the same query again. Try running the query again and see what happens.

One of the things you'll notice about Snowsight that is different from the Classic UI is that there are four buttons below the query and just to the right of the object listings. Currently there are three of the buttons highlighted in blue ([Figure 7-12](#)).

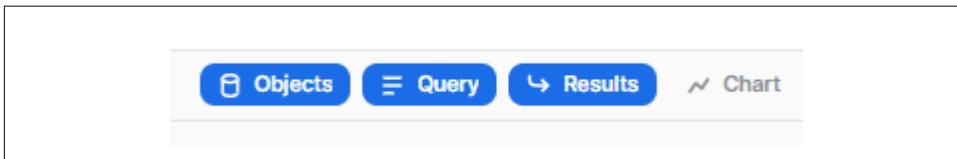


Figure 7-12. Four buttons in Snowsight just below the query and to the right of the object listings

Deselect each of the three buttons one at a time to see what happens. Notice what happens to your workspace area as you deselect and then reselect each one. The “Objects” button determines whether you see the objects to the left of your screen. You can deselect the “Query” button which gives you a full screen view of the results or you can deselect the “Results” button to get a full screen view of the worksheet queries. If you select “Chart” button, the “Results” button is deselected. Changing up the Snowsight viewing area allows you to customize the Snowflake UI and there are also many other improvements Snowflake has made in their new web UI.

In the next section, we'll take a look at new Snowsight features that improve user productivity, then we'll look at Snowsight charts and dashboards, and finally we'll look at some new collaboration features.

Improved Productivity

First off, you'll appreciate the many improvements that come with Snowsight, if you've been working in the Snowflake Classic UI. Snowsight provides automatic contextual statistics, script version history, worksheet folder structures, shortcuts and much more.

Using Contextual Suggestions from Smart Autocomplete

You may have noticed when we were writing our sampling query that the Snowflake smart autocomplete gave you contextual suggestions. Indeed, Snowflake showed you your choices before you could complete the name of the table ([Figure 7-13](#)). And if you were to begin typing in a function name, Snowflake will give you information about the parameters needed as well as link to the Snowflake docs.



Figure 7-13. Snowsight contextual suggestions from smart autocomplete

Formatting SQL

Snowsight gives you the ability to neatly format your SQL code. Use the drop-down menu to select “Format query” to have your SQL query reformatted (Figure 7-14).

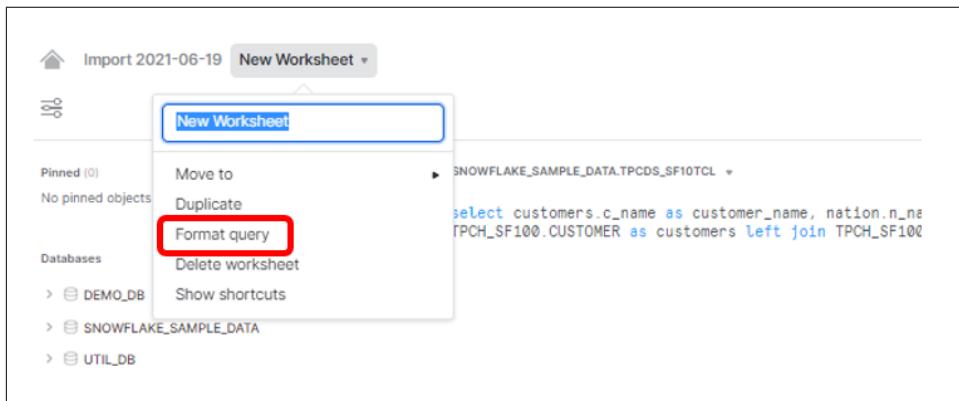


Figure 7-14. Snowsight option to Format Query

It is no longer necessary to spend time formatting your messy code before sharing with others or collaborating on your dashboards. The “Format query” option takes care of that for you. Whereas the original query was two long lines of code; the reformatted query is perfectly formatted (Example 11-1).

Example 7-1. Formatted Query

```
select
  customers.c_name as customer_name,
  nation.n_name as nation_name,
  customers.c_acctbal as account_balance,
  customers.c_mktsegment as market_segment
from
```

```

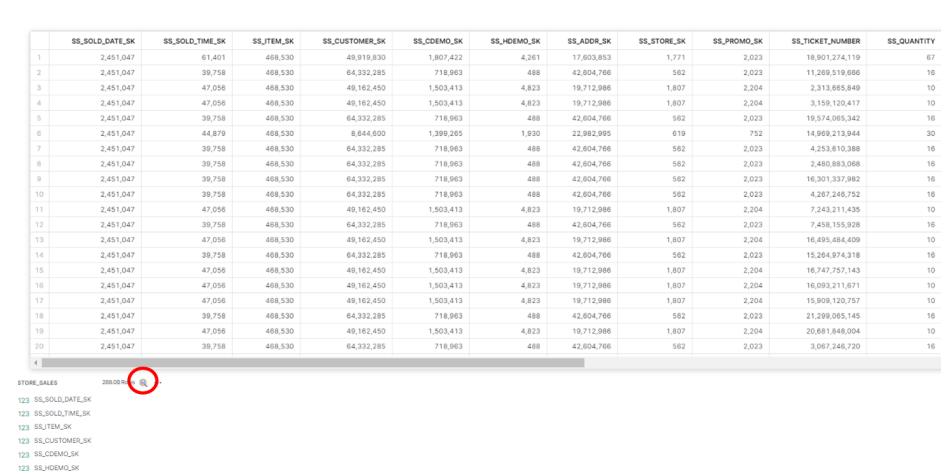
TPCH_SF100.CUSTOMER as customers
left join TPCH_SF100.NATION as nation on customers.c_nationkey = nation.n_nationkey
where
    nation.n_nationkey = 24;

```

You use the “Format query” option to clean up any messy formatting before you share your SQL code or collaborate on visualizations together.

Previewing Data Quickly

In the previous section, we took advantage of being able to preview the first 100 records of a table by clicking on the magnifying glass (Figure 7-15).



	SS_SOLD_DATE_SK	SS_SOLD_TIME_SK	SS_ITEM_SK	SS_CUSTOMER_SK	SS_CDEMO_SK	SS_HDEMO_SK	SS_ADDR_SK	SS_STORE_SK	SS_PROMO_SK	SS_TICKET_NUMBER	SS_QUANTITY
1	2,451,047	61,401	468,530	49,918,830	1,807,422	4,261	17,803,853	1,771	2,023	18,901,274,119	67
2	2,451,047	39,758	468,530	64,331,285	718,963	488	42,804,766	582	2,023	11,269,519,668	16
3	2,451,047	47,056	468,530	49,162,450	1,503,413	4,823	19,712,988	1,807	2,204	2,313,685,849	10
4	2,451,047	47,056	468,530	49,162,450	1,503,413	4,823	19,712,988	1,807	2,204	3,159,120,417	10
5	2,451,047	39,758	468,530	64,331,285	718,963	488	42,804,766	582	2,023	19,574,065,342	16
6	2,451,047	44,879	468,530	8,644,600	1,399,265	1,930	22,982,995	619	752	14,969,213,944	30
7	2,451,047	39,758	468,530	64,331,285	718,963	488	42,804,766	582	2,023	4,253,810,388	16
8	2,451,047	39,758	468,530	64,331,285	718,963	488	42,804,766	582	2,023	2,480,883,068	16
9	2,451,047	39,758	468,530	64,331,285	718,963	488	42,804,766	582	2,023	16,301,337,982	16
10	2,451,047	39,758	468,530	64,331,285	718,963	488	42,804,766	582	2,023	4,267,246,752	16
11	2,451,047	47,056	468,530	49,162,450	1,503,413	4,823	19,712,988	1,807	2,204	7,243,211,435	10
12	2,451,047	39,758	468,530	64,331,285	718,963	488	42,804,766	582	2,023	7,456,155,928	16
13	2,451,047	47,056	468,530	49,162,450	1,503,413	4,823	19,712,988	1,807	2,204	16,495,484,409	10
14	2,451,047	39,758	468,530	64,331,285	718,963	488	42,804,766	582	2,023	15,264,974,318	16
15	2,451,047	47,056	468,530	49,162,450	1,503,413	4,823	19,712,988	1,807	2,204	16,747,757,143	10
16	2,451,047	47,056	468,530	49,162,450	1,503,413	4,823	19,712,988	1,807	2,204	16,093,211,871	10
17	2,451,047	47,056	468,530	49,162,450	1,503,413	4,823	19,712,988	1,807	2,204	15,908,120,757	10
18	2,451,047	39,758	468,530	64,331,285	718,963	488	42,804,766	582	2,023	21,299,065,145	16
19	2,451,047	47,056	468,530	49,162,450	1,503,413	4,823	19,712,988	1,807	2,204	20,881,848,004	10
20	2,451,047	39,758	468,530	64,331,285	718,963	488	42,804,766	582	2,023	3,067,246,720	16

Figure 7-15. First 100 records of a table in Snowsight, viewable by clicking on the magnifying glass



An active warehouse is required to preview data.

Using Shortcuts

You’ll find that there are a number of shortcut commands in Snowsight commands. Click on the dropdown arrow beside the name of the worksheet and the click on “Show Shortcuts” to access those commands (Figure 7-16).

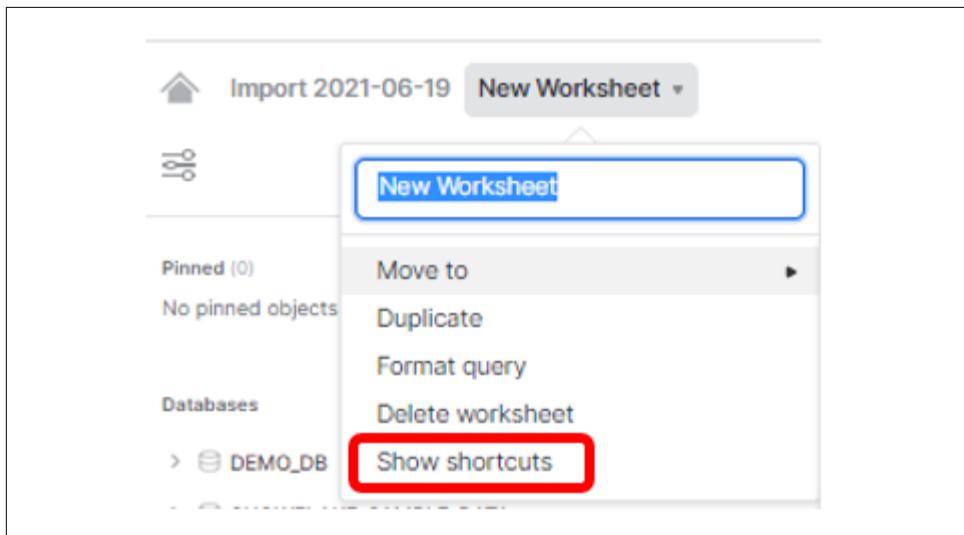


Figure 7-16. Option to Show Shortcuts is available in Snowsight

After you click enter, you'll be shown the list of shortcuts (Figure 7-17).

Shortcuts	
General	
ctrl + shift + ?	Show keyboard shortcuts
ctrl + alt + n	New query
ctrl + shift + f	Search schema or results
esc	Clear selection
Navigation	
ctrl + alt + ↑	Make bottom pane larger
ctrl + alt + ↓	Make bottom pane smaller
ctrl + alt + →	Go right one pane tab
ctrl + alt + ←	Go left one pane tab
Query editing	
ctrl + Enter	Run query
ctrl + Enter	Run query
ctrl + shift + o	Format query
ctrl +]	Indent line
ctrl + [De-indent line
ctrl + /	Toggle comment
Done	

Figure 7-17. List of Snowflake shortcuts available

Using Automatic Statistics and Interactive Results

Snowsight uses metadata proactively to provide expedited fast interactive results no matter how many rows are returned in query results. As you load and query the data, Snowflake puts all the rich metadata that is stored behind the scenes into a GUI interface so that you can start to understand some of the trends in your data and/or catch any errors early on. For example, you can tell right away whether there are a lot of null values in a field. This is especially advantageous to ETL engineers who are then able to detect loading errors or gaps in the data being received.

In addition to the Snowsight automatic contextual statistics which tell you how many rows are filled, you'll also see histograms for all date, time, and numeric columns, and frequency distributions for categorical columns. Additionally, Snowflake provides email domain distributions for email columns and key distributions for JSON objects.

Let's take a look at some of Snowsight's automatic statistics and interactive results. Run a select * statement on the CATALOG RETURNS table and limit the results to 500,000. You'll see the statistics to the right of the results (Figure 7-18).



Figure 7-18. Query Details and Statistics

The statistics results are interactive. If you click on the statistics for column CR_CATALOG_PAGE_SK, you'll see that 98% of the records have a page SK value (Figure 7-19). You can also see the rank of values.

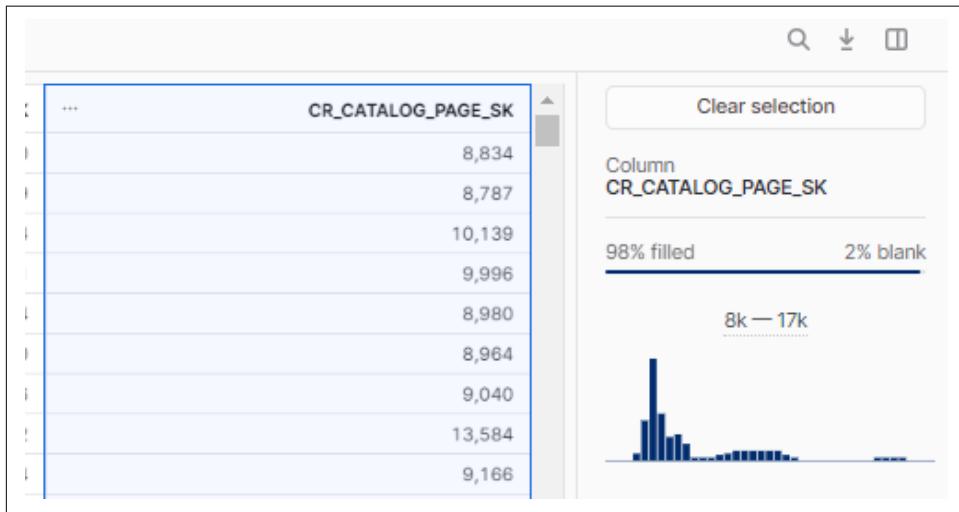


Figure 7-19. Query Results Statistics for an individual column

Rather than having to write individual queries or change over to a data science or visualization tool to find out how complete the data is and what are the distinct values or date ranges for each column, the user can now use Snowsight's automatic statistics and interactive results.

Accessing Version History

The current worksheet is automatically saved each time a SQL statement is run in a Snowsight worksheet. If a statement or script is re-executed multiple times, the version history will be available. The various versions can be accessed by accessing the drop-down menu on the right side of the screen where it is shown when the last time the worksheet was updated (Figure 7-20).

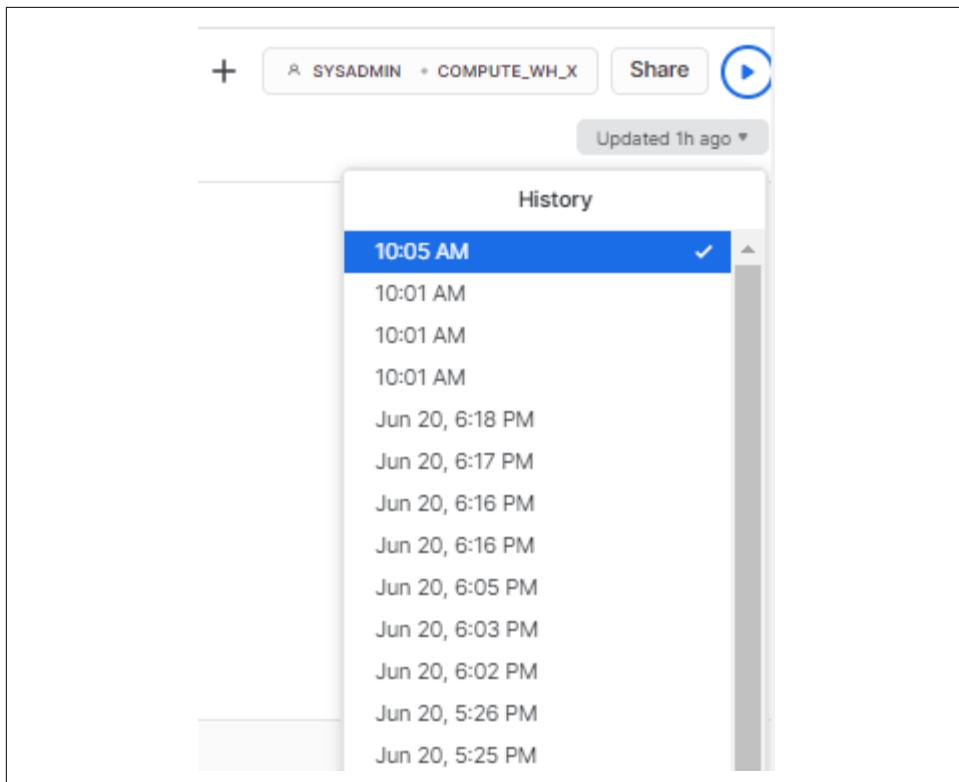


Figure 7-20. Query History showing recent updates

Visualization

It is incredibly helpful to have the ability to preview 100 rows of the data and especially to be able to use automatic statistics and interactive results to quickly learn much about the data. However, the metadata isn't enough. We'll want to use the new Snowsight visualization tools to get a more complete story about the data.

Data visualization makes it much easier to identify trends, patterns, and outliers and to discover insights within complex data. Information displayed in a visual context, such as a graph or a map, is a more natural way for the human mind to comprehend large amounts of data. Business Intelligence tools such as Tableau and Power BI are used for corporate reporting and for building complex dashboards where many users are consumers of a particular dashboard.

Sometimes, individuals or small groups of users in an organization just need to undertake ad-hoc data analysis, perform data validation while loading data, or quickly create simple charts and dashboards that can be shared and explored together

as a team. For those use cases, Snowflake's Snowsight visualization tool is a great choice.

Creating a Dashboard and Tiles

Tiles are created from within a dashboard. Let's begin by creating a dashboard where we can create our tiles. You will want to click on the folder. In this case, the folder name is "Sampling" (Figure 7-21).

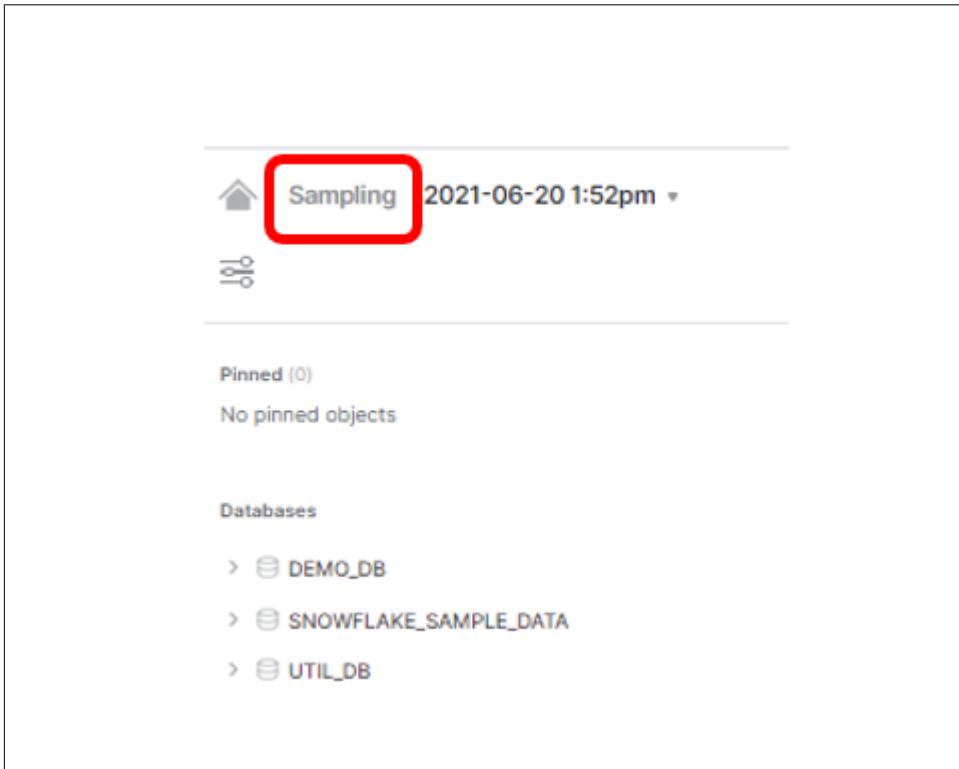


Figure 7-21. Currently viewing a folder named "Sampling"

Then click on Dashboards to the left and "+ Dashboard" to the right to create a new dashboard (Figure 7-22).

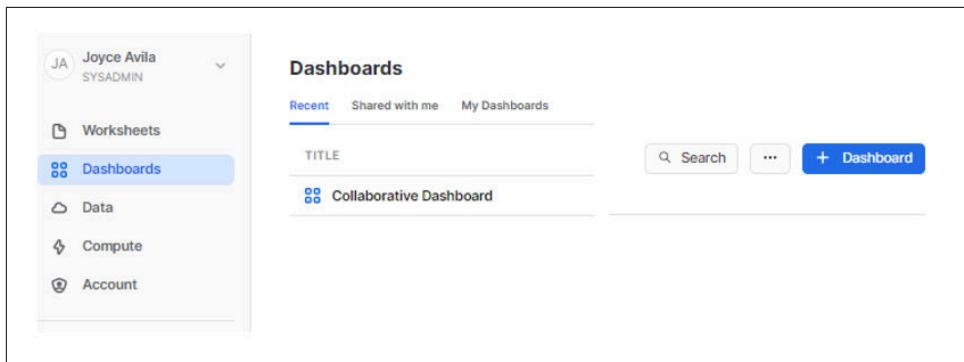


Figure 7-22. Creating a new Dashboard in Snowsight

Give the dashboard a name and click on the “Create Dashboard” button and then “+ New Tile” button (Figure 7-23).

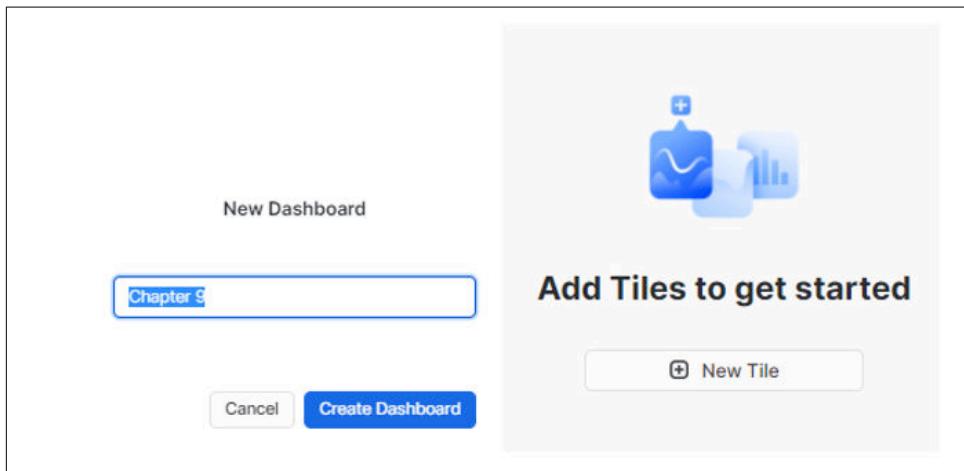


Figure 7-23. Creating a new dashboard titled “Chapter 11” and then adding a new tile

Just as before, select the database and schema (Figure 7-24).

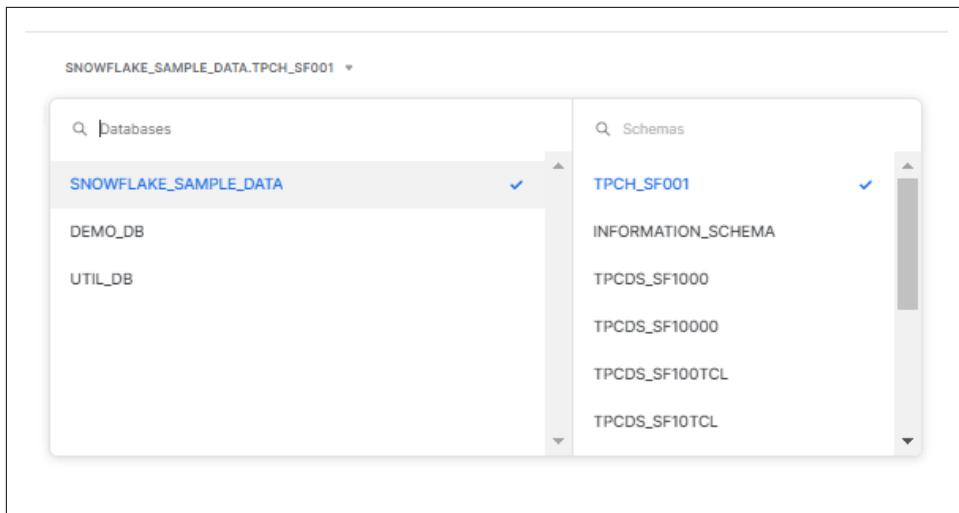


Figure 7-24. Selecting the database and schema for the new Chapter 9 dashboard

Click on the Query button and you'll see all the database tables to the right. Click on any of the tables and they'll be pinned to your workspace (Figure 7-25). If there is any table you no longer want to pin, just click on the pushpin and the table will be removed from the workspace. You'll also notice that the pushpin next to the table name, located on the right side of the screen, is no longer there.

A screenshot of the Snowflake interface showing three pinned tables. At the top, there are tabs for 'Objects' (selected) and 'Query'. Below are three tables: TPCH_SF001.NATION, TPCH_SF001.REGION, and TPCH_SF001.ORDERS. Each table has a 'Rows' count (25 rows, 5 rows, and 15k+ rows respectively), a 'Preview' button, and a pushpin icon. The pushpin icon for TPCH_SF001.ORDERS is yellow, indicating it is pinned. The table names are also pinned to the right of the screen.

Figure 7-25. Pinned Tables

Working with Chart Visualizations

We'll start with a basic query.

```
Select * from TPCH_SF1.LINEITEM limit 100000
```

Click on the Chart button at the bottom to see the chart. It will default to a line chart (Figure 7-26).

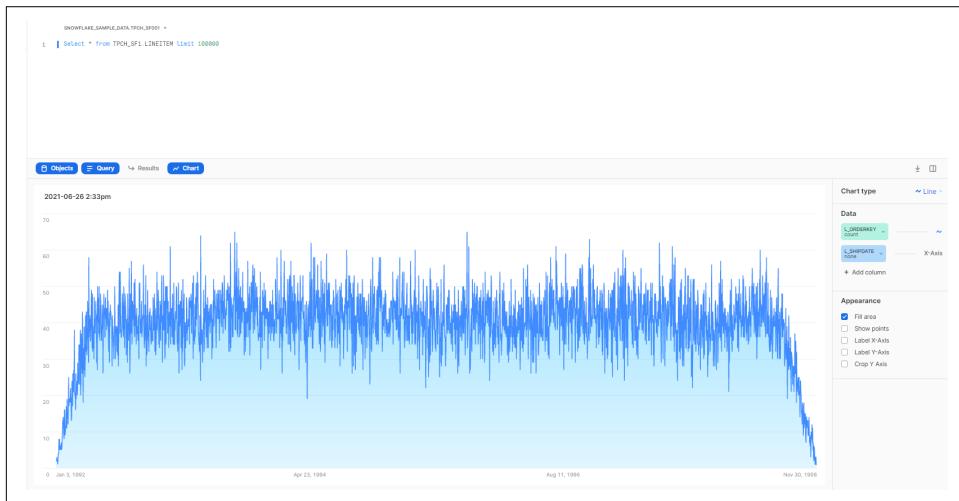


Figure 7-26. Line chart default results

On the right, there is a chart drop-down menu (Figure 7-27). Select Bar chart and see what happens.

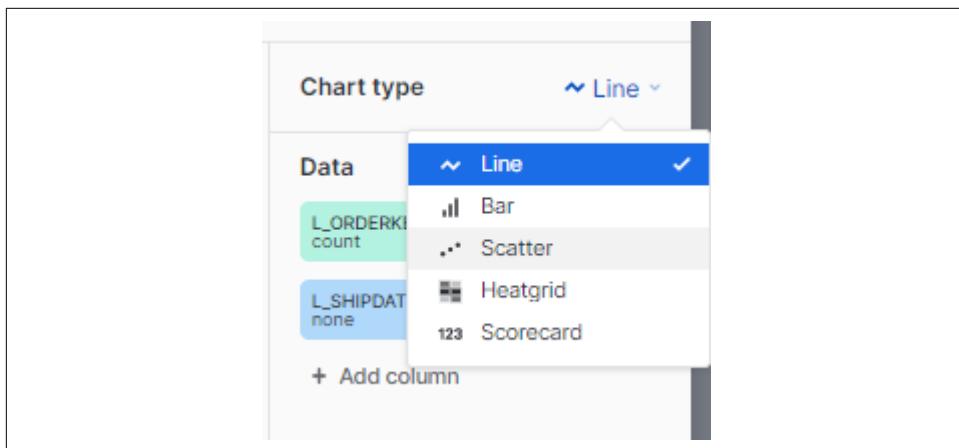


Figure 7-27. Different Chart Types that can be selected

At first, you won't see any chart. If you run the query again, though, you'll see the bar chart.

Aggregating and Bucketing Data

Having the chart results at the daily level and in descending order doesn't provide the visual picture that we think would be helpful. Let's see if we can make some improvements.

Click on the SHIPDATE button to the right (Figure 7-28). Change the Bucketing from None to Month or Quarter, or whatever you think would be best. Try selecting Month and changing the direction to Ascending so that the time goes from left to right. Additionally, take time to label the X and Y axis.

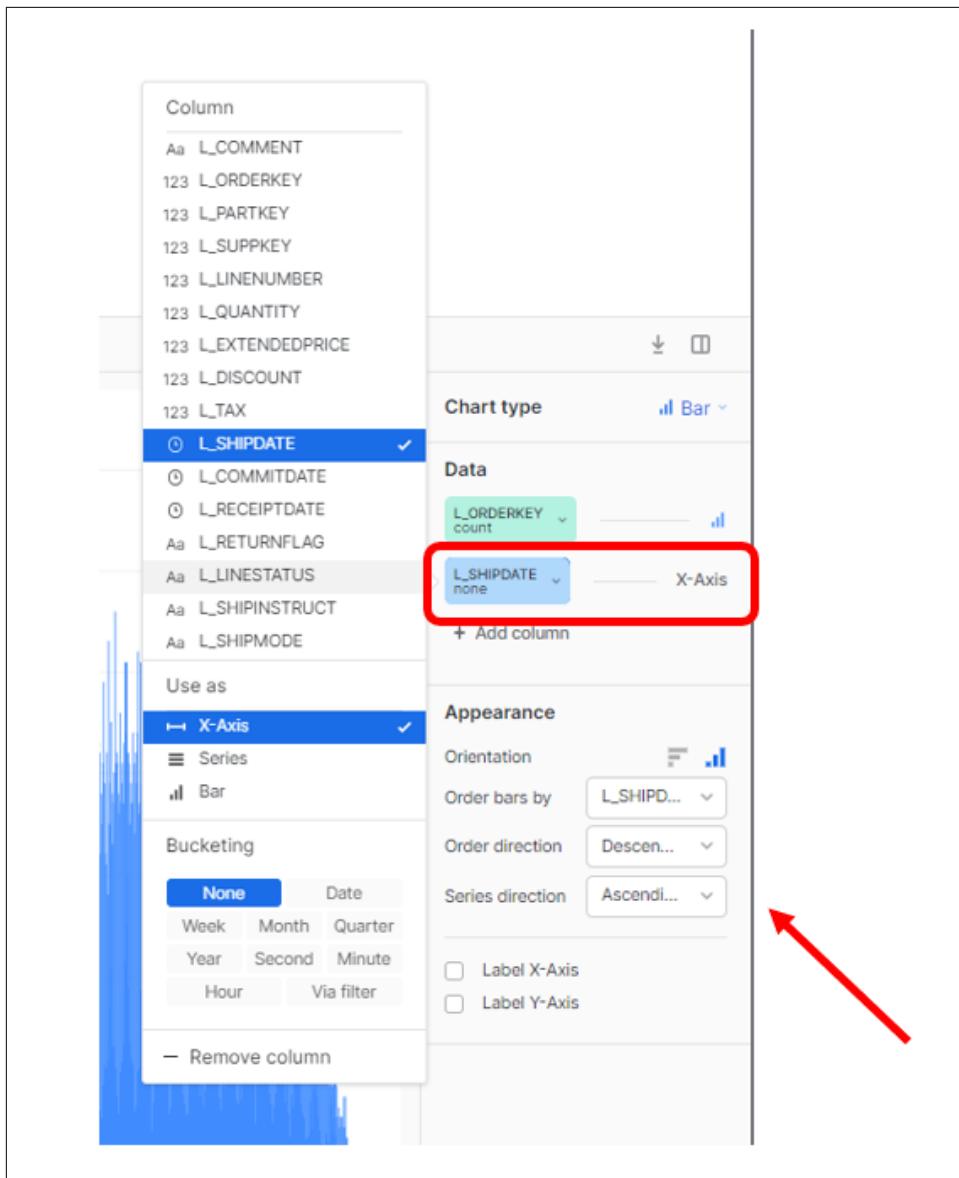


Figure 7-28. Bucketing the SHIPDATE column

The changes I made give me a more appealing visualization that I can gain some insights from (Figure 7-29).

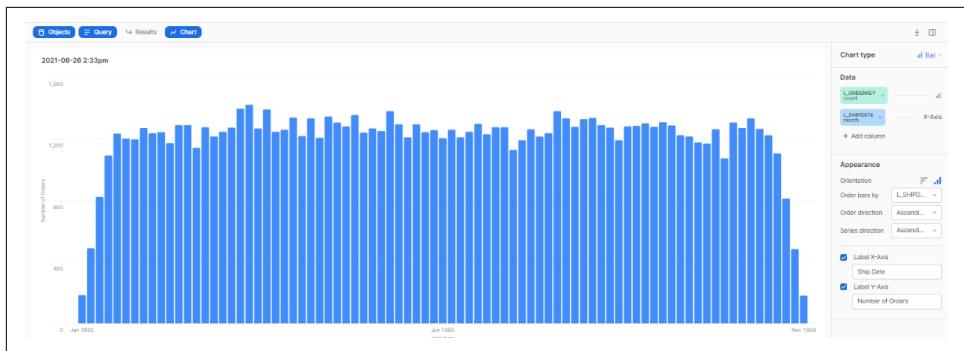


Figure 7-29. Visualization of data after bucketing SHIPDATE

Note that if you hover your cursor over the top of any of the bars, you'll see the month and year as well as the number of orders for that particular month.

In the top left corner, you'll see "Return to Chapter 11", with Chapter 11 being the name of your dashboard ([Figure 7-30](#)).

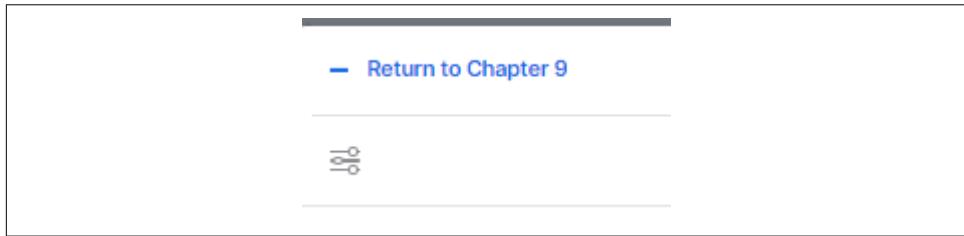


Figure 7-30. Ability to return to the dashboard is provided in the top left corner of the screen

To create a new worksheet, click on the + symbol in the top left corner then "New Tile from Worksheet" at the bottom ([Figure 7-31](#)).

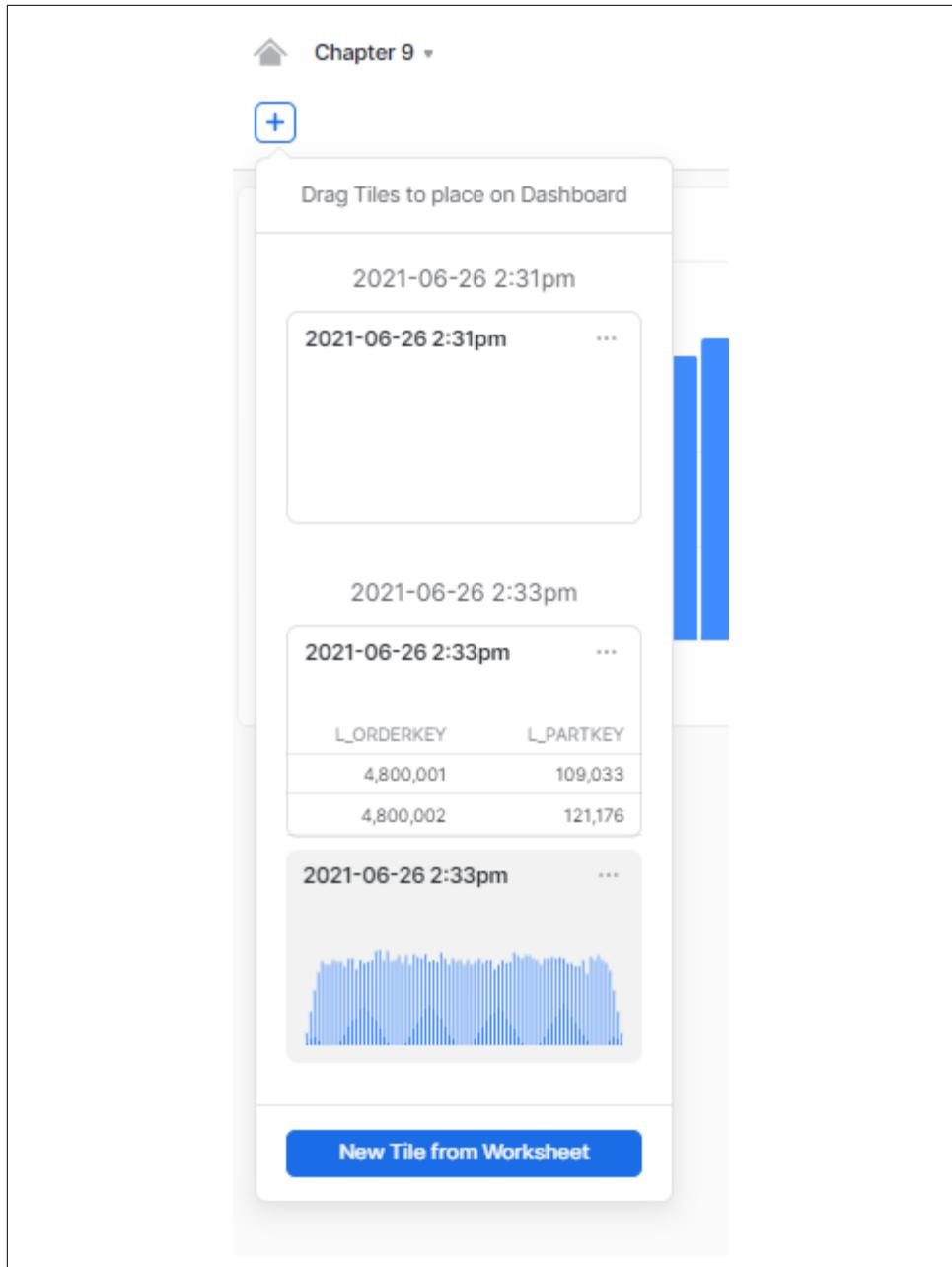


Figure 7-31. Creating a New Tile in a New Worksheet

Editing and Deleting Tiles

Even when placed on the dashboard, individual tiles can be edited. In the top right corner of the chart (Figure 7-32) click on the ellipsis and select from the options. Of course, you can edit the chart by editing the underlying query and one handy feature is that you can duplicate a tile. Duplicating a tile duplicates the underlying worksheet and queries.



Figure 7-32. Tile Options accessible after clicking on the ellipsis



Deleting a tile from a dashboard also permanently deletes the underlying worksheet. You'll not be able to retrieve the tile or worksheet after you perform a delete action.

Deleting a tile is not recommended, unless you are certain that you will not need the worksheet upon which the chart is built. One alternative to deleting a tile is to simply “Unplace Tile” so that it still exists but is not viewable on the dashboard.

Collaboration

The goal of sharing work product and collaborating is so that the group benefits from the work of individuals without each person or business unit having to recreate the wheel. It also provides the opportunity for more innovation and improvements since everyone can contribute to the knowledge base and move it forward faster.

Sharing Your Query Results

To share the results of any query, simply click on the down arrow to download a csv file of the query results. The results can then be shared with others (Figure 7-33). You'll notice that the files are saved with the date and timestamp.

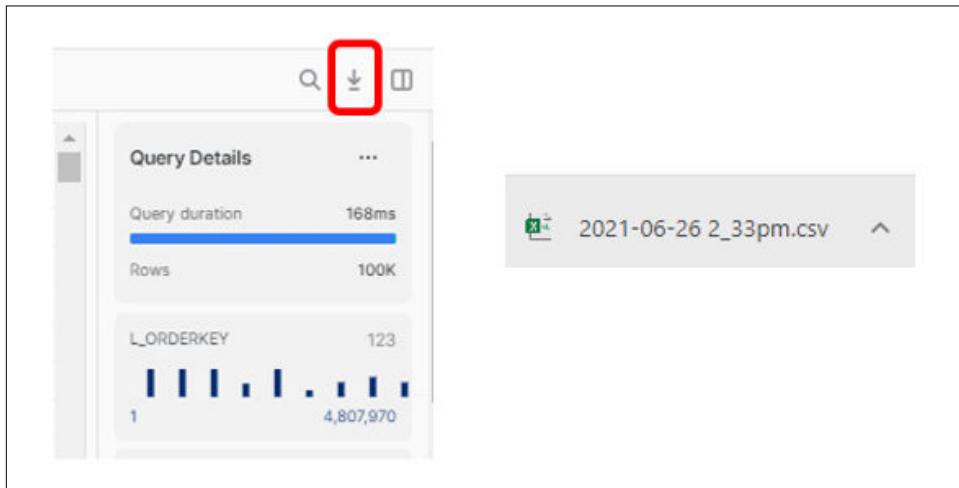


Figure 7-33. Use the Down Arrow to begin the process of sharing query results with others

Using a Private Link to Collaborate on Dashboards

One of the most important Snowsight features is the ability for individuals to collaborate on a dashboard. It's easy to do. In the top left corner, there is a "Share" button (Figure 7-34).

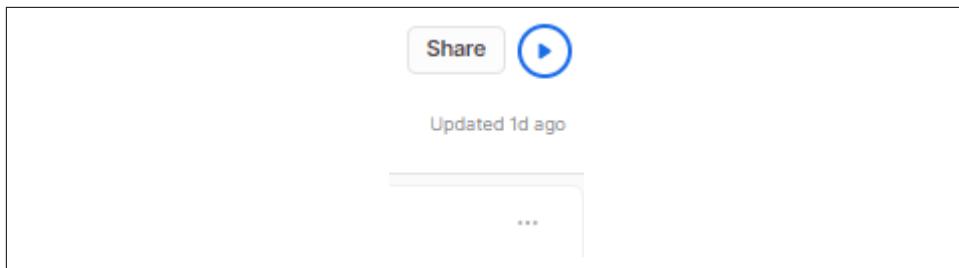


Figure 7-34. Share button allows for sharing with others for dashboard collaboration

After you click on the Share button, there is an option to invite users or to let everyone with a link have access to the dashboard. You can authorize a few different privileges for people with the dashboard link, including the ability to only view the results or to run the dashboard which gives them the ability to refresh for the latest results (Figure 7-35).

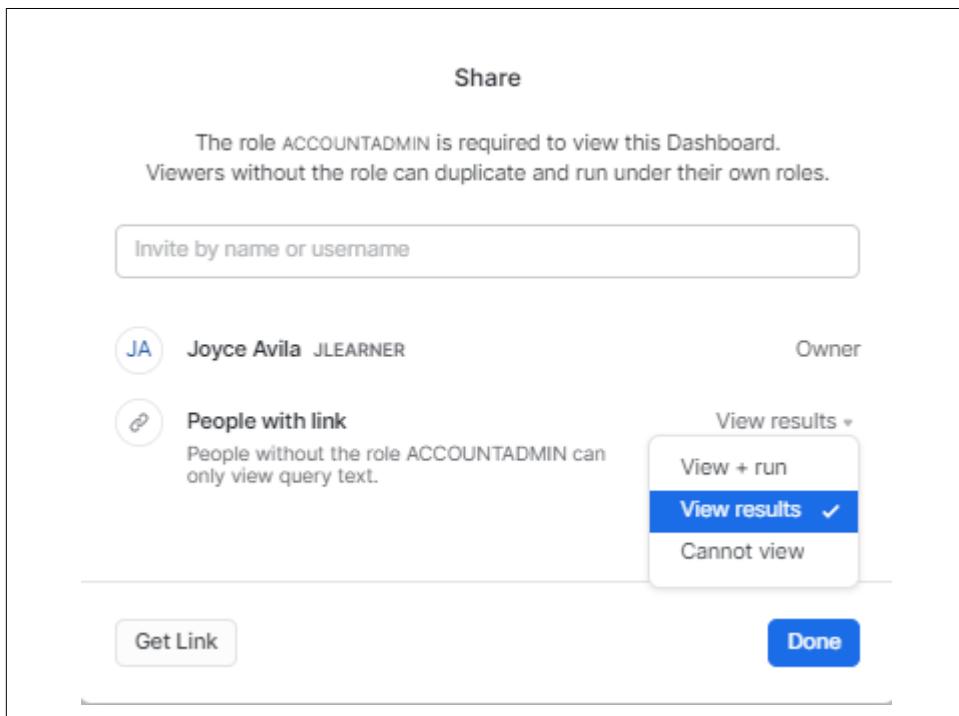


Figure 7-35. Assigning privileges for people with the link to the dashboard

A user can filter the results and chart the data to create a dashboard and share it with a colleague within a few minutes without ever having to leave Snowflake and with only using SQL.

Exercises to Test Your Knowledge

The following exercises are based on the coverage in this chapter.

1. What are the different ways you can instruct Snowflake to return a subset of rows via sampling technique?
2. To use the preview option, do you write a SQL query? How many rows are returned when you use the preview option?
3. When you re-run a sampling query, are the same rows returned the second time as were returned the first time?
4. Is an active warehouse required to run a sampling query? Is an active warehouse required to preview the data?
5. What does “Format SQL” do? Specifically, does “Format SQL” correct your spelling for commands or table names?
6. What happens when you delete a tile from a dashboard?
7. What are some of the use cases for using Snowsight visualizations?
8. What is an alternative to deleting a tile from the dashboard?
9. How many charts can you create from one worksheet?
10. Does Snowflake allow for visualization from internal data sources, external data sources, or both?

Solutions to these exercises are available in Appendix A.

About the Author

Joyce Kay Avila has more than 25 years as a business and technology leader. She is a Texas Certified Public Accountant and currently holds one Amazon Web Services certification and one Snowflake certification. She also holds twelve Salesforce certifications, including the Tableau CRM and Einstein Discovery certification along with several Salesforce Architect-level certifications.

Joyce works as the Snowflake Practice Manager at SpringML, a Snowflake partner who helps organizations unlock the power of data-driven insights using Snowflake's unique cloud data platform. Previously, Joyce earned bachelor's degrees in Computer Science and in Accounting Business Administration, a Master's degree of Business Administration, and completed her PhD coursework in Accounting Information Systems.

In support of the Salesforce and Snowflake communities, Joyce produces a running series of how-to videos on her YouTube channel. In 2022, she was selected as one of 48 Snowflake Data Superheroes worldwide.