MANIPAL
ACADEMY of HIGHER EDUCATION
(Deemed to be University under Section 3 of the UGC Act, 1956)

MANIPAL SCHOOL OF INFORMATION SCIENCES
(A Constituent unit of MAHE, Manipal)

Snowflake Cost Reporting System

ResMed Technology India Private Limited

Manthana H K
231058012

Master of Engineering
M.E (Big Data Analytics)

Project Start Date: 01/08/2024

Industry Guide:
Kevin Logsdon
Senior Manager, Data Analytics Management
ResMed Technology India Private Limited
Bloomington, United States

Internal Guide:
Deepak Rao
Associate Professor
MSIS, MAHE
Manipal

Signed by:
kevin logsdon
E32C5EC1100D486...

14-May-2025 | 11:55 EDT

MANIPAL
ACADEMY of HIGHER EDUCATION
(Deemed to be University under Section 3 of the UGC Act, 1956)

MANIPAL SCHOOL OF INFORMATION SCIENCES

(A Constituent unit of MAHE, Manipal)

Manipal 14-05-2025

CERTIFICATE

This is to certify that the project titled "**Snowflake Cost Reporting System"** is a record of the Bonafede work done by **Manthana H K** (231058012) submitted in partial fulfillment of the requirements for the award of the degree of Master of Engineering – **M.E (Big Data Analytics)** of Manipal School of Information Sciences, Manipal, Karnataka (A Constituent Unit of Manipal Academy of Higher Education), during the academic year 2024- 25, and the same has not been submitted elsewhere for the award of any other degree. The project report does not contain any part or chapter plagiarized from other sources.

Deepak Rao
Associate Professor
MSIS
MAHE, Manipal

Dr. Keerthana Prasad
Director
MSIS
MAHE, Manipal

INTERNSHIP CERTIFICATE

Resmed

09 May 2025

To Whom It May Concern

This is to certify that Manthana H K, has been undergoing an internship with Resmed Technology India Pvt. Ltd.

The internship commenced on 22 July 2024 and is scheduled to continue until 30 May 2025. During this period, he has been working under the supervision of Kevin Logsdon, Senior Manager, Data Analytics Management.

We are pleased to have Manthana as part of our team during this time and appreciate his contributions.

For Resmed Technology India Pvt. Ltd.

DocuSigned by:
People Shared Services
1625BF76AE32445...

People Shared Services

ResMed Technology India Private Limited (Formerly MatrixCare India Private Limited) CIN: U72900TN2016FTC104223
AMBER Building, 6th Floor, Bagmane World Technology Center, Mahadevpura, Bengaluru, Pin code: 560048
Regd Off: DLF SEZ IT Park Block 6, Level 6, GKS Technology Park, 1/124, Shivaji Garden, Manapakkam, Chennai -600089

ACKNOWLEDGMENTS

I would like to sincerely thank **Dr. Keerthana Prasad**, Director of Manipal school of Information Sciences, Manipal, for giving me the chance to use my skills in the industry. Her support has been vital for my career development.

I am very grateful to my guide, **Mr. Deepak Rao**, Associate Professor, Manipal School of Information Sciences, Manipal, for his consistent support, helpful guidance, and valuable advice throughout my internship. His mentorship has been key to my learning and success.

I would like to extend my heartfelt gratitude to **ResMed Technology India Private Limited**, my workplace, for granting me the incredible opportunity to work on cutting-edge technologies. The resources, support, and enriching environment provided by the organization have been instrumental in my professional growth and learning.

I also want to extend my thanks to **Mr. Kevin Logsdon**, Senior Manager, Data Analytics Management of the SLTC Data and Analytics team in the ResMed Technology India Private Limited, for his guidance on my project and for ensuring that the training and onboarding process went smoothly.

I deeply appreciate the members of the SLTC Data and Analytics team in the ResMed Technology India Private Limited, for their warm welcome, encouragement, and the freedom they gave me to innovate and contribute. Their support has significantly enriched my professional experience.

ABSTRACT

This project focuses on designing and implementing a comprehensive cost estimation and analysis framework for Snowflake's Accounts and Reader Accounts using Snowflake's robust architecture. The project explores the key components of providing a detailed understanding of how cost optimization can be achieved. The data models leverage data warehousing concepts, efficient data loading strategies, and visualization techniques for query, warehouse, storage and ETL cost analysis. The importance of Reader Accounts is thoroughly discussed, emphasizing their role in securely sharing data with external stakeholders while managing costs effectively. A detailed data models are developed to calculate costs associated with queries and warehouses, storage and ETL. The data models comprise fact tables and dimension tables. These tables are interconnected to provide granular insights into query and warehouse and storage usage. For instance, the model captures key details such as user roles, query execution times, warehouse sizes, and credits consumed, enabling accurate cost estimation. Key aspects of data loading strategies full loads and incremental loads are explained in the context of this data model. Full loads are used for initial population and periodic synchronization, ensuring data integrity, while incremental loads are applied for efficient, ongoing updates to minimize resource usage and costs. This balance ensures the model remains optimized for performance and scalability. To analyze and visualize the cost metrics effectively, Power BI is integrated into the framework as the primary analytics and reporting tool. Once the transformed and modeled data is available in Snowflake, Power BI connects directly to Snowflake using native connectors, leveraging Direct Query modes based on the analytical needs and data volume. The semantic layer designed in Power BI mirrors the logical structure of the Snowflake data model, including relationships between fact and dimension tables, time-based hierarchies, and calculated metrics.

INDEX

Table of Contents

CERTIFICATE i

INTERNSHIP CERTIFICATE ii

ACKNOWLEDGMENTS iii

ABSTRACT iv

INTRODUCTION 1

Snowflake 1

Database Storage Layer 7

Query Processing Layer (Virtual Warehouses) 8

Cloud Services Layer 9

How these layers work 10

Key Features of Snowflake 11

Reader Accounts 14

Cost considerations for Reader Accounts 16

Airflow 18

Apache Airflow Introduction 18

Key Features of Apache Airflow: 18

Key Components of Apache Airflow 19

Airflow Workflow Execution 25

Airflow Use Cases 25

Core features of Airflow 26

Architecture of Airflow 27

How These Components Work Together 30

Data Flow Summary 31

Benefits of This Architecture 31

Amazon Managed Workflows for Apache Airflow (MWAA) 31

The Need for MWAA 32

Key Features of MWAA 32

MWAA Architecture 34

Typical Use Cases 34

Benefits of Using MWAA 35
DETAILS OF THE WORK DONE 37
Query Cost 37
Overview of the Model 38
How the Model Works 39
Warehouse Cost 39
How the Model Works 41
Security Model 42
Overview of the Model 42
How the Model Works 43
Storage Cost 45
Overview of the Data Model 45
How the Model Works 46
ETL cost 47
Overview of the Data Model 47
How the Model Works 49
Implementation of Data Models 49
Full Load 50
Incremental Load 51
Analysis using Power BI 52
Core Components of Power BI 53
Data Integration and Connectivity 54
Data Modeling 55
DAX (Data Analysis Expressions) 56
CONCLUSION 61
BIBLIOGRAPHY 64

Table of Figures

Figure 1: Snowflake Architecture……6

Figure 2: Credits used by various type warehouses in Snowflake……....9

Figure 3: Architecture of Reader Account...……14

Figure 4: Browser UI of Airflow……24

Figure 5: Information of a DAG execution……24

Figure 6: Workflow of Airflow components…..……27

Figure 7: Data model of Query Cost……..37

Figure 8: Data model of Warehouse Cost……..39

Figure 9: Data model of Security Model…...……42

Figure 10: Fact Tables for Storage Cost……45

Figure 11: Data model of ETL Cost……....47

Figure 12: Data Modeling in Power BI……..56

Figure 13: Dashboard for Query Model……60

Figure 14: Dashboard for Month-Over-Month Trend……...60

INTRODUCTION

Snowflake

In the digital era, data is not merely a byproduct of business operations—it is the foundation for innovation, strategy, and competitive advantage. As organizations increasingly rely on vast and diverse data sources for real-time decision-making, traditional data management solutions have struggled to keep pace. Issues related to scalability, performance, cost-efficiency, and operational complexity have underscored the need for a paradigm shift in how data is stored, processed, and analyzed. Enter Snowflake: a cloud-native data platform designed from the ground up to address the limitations of legacy systems and redefine what's possible in modern data architecture. [1]

A New Approach to Data Warehousing

Snowflake was founded in 2012 by Benoit Dageville, Thierry Cruanes, and Marcin Żukowski—veterans of Oracle and other major database technology companies. They envisioned a data platform that could harness the full potential of the cloud while eliminating the bottlenecks and complexities associated with traditional on-premises or Hadoop-based systems. Rather than adapting an existing database engine to the cloud, they built an entirely new architecture tailored specifically for cloud environments. This fundamental design choice has proven transformative. Snowflake is not merely a cloud-hosted data warehouse—it is a fully managed, cloud-native data platform that integrates data warehousing, data lakes, data engineering, data science, and data sharing into a unified solution. Unlike legacy solutions that require customers to provision hardware, manage infrastructure, and perform manual scaling or tuning, Snowflake abstracts away nearly all operational complexity.

It delivers the power of a full-featured SQL analytics database along with powerful enhancements such as automatic elasticity, high concurrency, cross-cloud capabilities, and built-in data sharing.

The Challenges with Traditional Architectures

To fully appreciate Snowflake's value proposition, it's important to understand the problems it was designed to solve. Traditional on-premises data warehouses typically suffer from the following issues:

- **Rigid Infrastructure**: Scaling resources up or down involves significant planning, procurement, and manual intervention. This results in either underutilization or bottlenecks.
- **Tightly Coupled Resources**: Compute and storage are often interlinked, making it difficult to scale one without affecting the other.
- **Operational Overhead**: Managing patches, backups, software upgrades, and tuning is time-consuming and error prone.
- **Limited Concurrency**: Supporting simultaneous users and workloads can degrade performance due to resource contention.
- **Lack of Agility**: Integrating diverse data sources (structured, semi-structured, unstructured) and supporting modern analytics (AI/ML) is often complex and costly.

Cloud-based adaptations of traditional warehouses (lift-and-shift) attempted to address some of these issues, but many inherited the architectural constraints of their legacy systems. Snowflake, by contrast, was conceived in the cloud and designed to take full advantage of cloud elasticity, resiliency, and distributed computing.

Snowflake's Cloud-Native Differentiators

At the heart of Snowflake's appeal is its unique architecture that separates compute, storage, and services. This design enables unmatched scalability, performance, and cost control.

- **Separation of Compute and Storage**: Unlike legacy systems, Snowflake allows storage and compute resources to scale independently. This ensures that data can be stored affordably and that compute resources (called Virtual Warehouses) can be scaled or spun up/down instantly based on workload demand.
- **Multi-Cluster Architecture**: Snowflake uses a Massively Parallel Processing (MPP) engine under the hood, and virtual warehouses can be set up as multi-cluster environments to handle high-concurrency workloads without bottlenecks.
- **Fully Managed Platform**: Snowflake automates maintenance tasks such as infrastructure provisioning, tuning, backups, and failover. As a result, users focus on data analysis and modeling rather than platform administration.
- **Cross-Cloud and Multi-Region Support**: Snowflake runs on all three major cloud providers—AWS, Azure, and Google Cloud Platform—and supports seamless data replication across regions and providers, giving customers the ability to adopt a truly multi-cloud strategy.
- **Secure Data Sharing**: One of Snowflake's most innovative features is Secure Data Sharing, which allows organizations to share live data across accounts, regions, or cloud platforms without duplicating or moving data. This enables real-time collaboration and data monetization.

- **Support for Semi-Structured Data**: Snowflake natively supports JSON, Avro, Parquet, ORC, and XML. Its unique VARIANT data type allows semi-structured data to be stored alongside structured data and queried with standard SQL, simplifying ETL pipelines.

Key Use Cases

Thanks to its modern architecture and feature-rich design, Snowflake is well-suited for a wide range of use cases:

- **Enterprise Data Warehousing**: Store and analyze large volumes of structured business data with SQL.
- **Data Lakes**: Use Snowflake as a query able data lake, ingesting raw data from a variety of formats.
- **Data Engineering**: Build and orchestrate robust data pipelines using Snowflake's support for streams, tasks, and stored procedures.
- **Data Science and ML**: Enable data scientists to explore, transform, and analyze massive datasets for model training and inference.
- **Real-Time Analytics**: Drive operational insights with near real-time data processing and high-concurrency querying.
- **Data Sharing and Monetization**: Create data marketplaces or share data with external partners securely and instantly.

A Growing Ecosystem

Snowflake's capabilities are further enhanced by its ecosystem of integrations and partners. It integrates seamlessly with modern data stack components, including:

- **ETL/ELT Tools**: Fivetran, dbt, Matillion, Talend
- **BI and Visualization**: Tableau, Power BI, Looker, Qlik
- **Data Science and Notebooks**: Jupyter, Dataiku, DataRobot, SageMaker
- **Orchestration**: Airflow, Prefect, Dagster
- **Security and Governance**: Collibra, Alation, Immuta, Okta

In addition, Snowflake's Marketplace allows users to access third-party datasets and services directly from their Snowflake interface, expanding analytics capabilities without complex integrations.

Market Impact and Adoption

Snowflake has rapidly become a leading force in the data platform market. Its IPO in 2020 was the largest in software history, and the platform is used by thousands of enterprises across industries—from healthcare and finance to retail and technology. Organizations choose Snowflake not only for its technical capabilities but also for the agility, cost-efficiency, and simplicity it brings to managing and analyzing data at scale. Here's how Snowflake has found footholds across different verticals:

- **Healthcare and Life Sciences**: Snowflake is used for aggregating clinical, research, and patient data, facilitating population health analytics, real-time reporting, and regulatory compliance. Its ability to handle semi-structured data allows for effective integration of genomic, imaging, and EHR data.

- **Financial Services**: Banks, insurance firms, and fintech companies leverage Snowflake for fraud detection, customer segmentation, risk modeling, and regulatory reporting. Its data sharing capabilities support collaborative ecosystems between partners, regulators, and service providers.
- **Retail and Consumer Goods**: Retailers use Snowflake to unify point-of-sale, e-commerce, inventory, and customer data to drive personalization, supply chain optimization, and predictive analytics. With support for real-time data ingestion, Snowflake enables agile decision-making in dynamic markets.
- **Media and Entertainment**: Streaming services and media platforms process vast amounts of user interaction and content data with Snowflake, enabling real-time recommendations, audience analytics, and ad optimization.
- **Technology and SaaS**: Tech companies often use Snowflake as their primary analytics backbone, supporting internal business intelligence, telemetry, and application-level analytics. SaaS providers use Snowflake's data sharing features to create data products and embedded analytics for their customers.

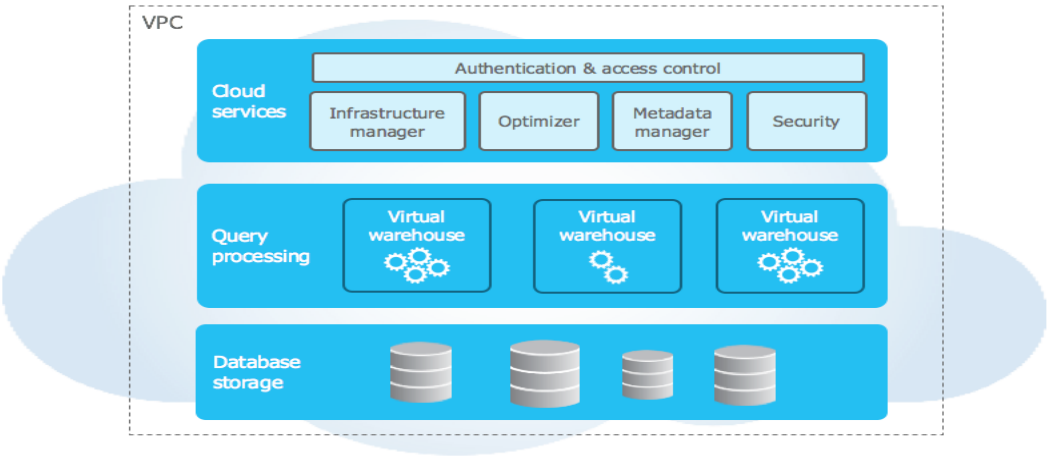

Figure 1: Snowflake Architecture

Snowflake is a cloud-native data platform that separates compute, storage, and cloud services. This separation is the foundation of its flexibility, scalability, and performance. Snowflake's architecture has three main layers: Database Storage, Query Processing, and Cloud Services.

Database Storage Layer

This foundational layer is responsible for storing all structured and semi-structured data. Key characteristics include:

- **Columnar Storage Format**: Data is stored in a columnar format, optimizing analytical query performance by allowing efficient data retrieval and compression.
- **Micro-Partitioning**: Data is automatically divided into micro-partitions, typically ranging from 50 MB to 500 MB in size. Each micro-partition contains metadata about the data it holds, facilitating efficient query pruning and performance optimization.
- **Automatic Clustering**: Snowflake automatically manages the clustering of data within micro-partitions, eliminating the need for manual partitioning and ensuring optimal query performance.
- **Support for Semi-Structured Data**: Snowflake natively supports semi-structured data formats such as JSON, Avro, Parquet, and XML, allowing users to store and query diverse data types without complex transformations.[2]

Query Processing Layer (Virtual Warehouses)

This layer handles all query execution and computation tasks. Key features include:

- **Virtual Warehouses**: Snowflake uses virtual warehouses, which are clusters of compute resources that can be independently scaled up or down based on workload demands. Each virtual warehouse operates independently, ensuring that workloads do not interfere with one another.
- **Massively Parallel Processing (MPP)**: Virtual warehouses utilize MPP to execute queries efficiently by distributing the workload across multiple compute nodes.
- **Auto-Scaling and Auto-Suspension**: Virtual warehouses can automatically scale to accommodate varying workloads and suspend when not in use, optimizing resource utilization and cost. [3]
- **Workload Isolation:** Each virtual warehouse operates independently from others, meaning multiple teams or processes can run queries simultaneously without affecting each other's performance. For example, ETL processes, reporting dashboards, and ad hoc analytics can run on separate warehouses, ensuring consistent performance and avoiding resource contention.
- **Multi-Cluster Warehouses for Concurrency Scaling:** Snowflake offers multi-cluster virtual warehouses, which automatically start additional compute clusters when concurrent user demand spikes. This feature is ideal for handling large numbers of simultaneous users or queries (e.g., in BI dashboards or shared environments) without degrading performance.

Warehouse Size	Credits / Hour	Credits / Second
X-Small	1	0.0003
Small	2	0.0006
Medium	4	0.0011
Large	8	0.0022
X-Large	16	0.0044
2X-Large	32	0.0089
3X-Large	64	0.0178
4X-Large	128	0.0356
5X-Large	256	0.0711
6X-Large	512	0.1422

Figure 2: Credits used by various type warehouses in Snowflake

Cloud Services Layer

This layer orchestrates and manages all operations within Snowflake. Key responsibilities include:

- **Metadata Management**: Maintains metadata related to data storage, structure, and access, enabling efficient query planning and execution.
- **Security and Access Control**: Handles authentication, authorization, and role-based access control to ensure data security and compliance.
- **Query Compilation and Optimization**: Parses SQL queries, generates optimized execution plans, and coordinates with the query processing layer for execution.
- **Infrastructure Management**: Manages the provisioning and scaling of virtual warehouses, ensuring seamless operation and performance. [4]

How these layers work

This decoupled architecture is a cornerstone of Snowflake's cloud-native design and is what allows the platform to deliver elastic scalability, workload isolation, performance optimization, and automated management. Each layer has a unique function but collaborates with the others to deliver a seamless user experience.

1. Data Storage and Retrieval:

Data is ingested into the Database Storage Layer, where it is compressed and stored securely. When a query is executed, it retrieves the relevant data from storage without interfering with other workloads.

2. Query Execution:

Queries are processed by Virtual Warehouses, which dynamically scale to handle workload demands. Compute resources are billed based on usage, ensuring cost efficiency.

3. Management and Optimization:

The Cloud Services Layer manages all system activities, including security, scaling, and query optimization, ensuring seamless operations.

Key Features of Snowflake

1. Elasticity and Scalability

Virtual Warehouses: Snowflake's architecture is built on the concept of virtual warehouses, which are clusters of compute resources that can be independently scaled. This separation of compute and storage allows for flexible resource management, enabling users to allocate compute power based on specific workload requirements.

Instant Scaling: Virtual warehouses can be resized instantly, allowing organizations to scale up for intensive workloads or scale down during periods of low activity. This elasticity ensures optimal performance without the need for over-provisioning resources.

Auto-Suspend and Auto-Resume: To optimize resource utilization and cost, Snowflake offers auto-suspend and auto-resume features. Warehouses can be configured to suspend automatically after a period of inactivity and resume when new queries are submitted, ensuring that compute resources are used efficiently.

Multi-Cluster Warehouses: For workloads with high concurrency requirements, Snowflake provides multi-cluster warehouses that can automatically start and stop clusters based on query demand. This feature ensures consistent performance during peak usage times by distributing the load across multiple clusters.

Scaling Policies: Snowflake allows users to define scaling policies for multi-cluster warehouses, balancing performance and cost. Policies can prioritize either minimizing query latency or conserving credits, depending on organizational needs.

2. High-Performance Query Execution

Massively Parallel Processing (MPP): Snowflake employs a Massively Parallel Processing architecture, enabling it to execute queries by dividing tasks across multiple compute nodes. This parallelism significantly enhances query performance, especially for large datasets.

Result Caching: To expedite query responses, Snowflake caches the results of previous queries. If an identical query is executed, Snowflake can return the cached result instantly, reducing compute usage and improving response times.

Query Optimization: Snowflake's optimizer analyzes queries to determine the most efficient execution plan. It considers factors like data distribution, available indexes, and statistics to optimize performance.

Query Acceleration Service (QAS): For resource-intensive queries, Snowflake offers the Query Acceleration Service, which dynamically allocates additional compute resources to accelerate query execution. This service is particularly beneficial for complex analytical workloads.

Adaptive Network Optimization: Snowflake continuously enhances its network infrastructure to reduce data transfer times and improve query performance. These optimizations ensure efficient data movement within and across regions.

Automatic Query Concurrency Management: Snowflake automatically manages concurrent query execution by intelligently queuing and distributing workloads across compute clusters. In environments with high user demand—such as business intelligence dashboards or concurrent data pipelines

3. Cross-Cloud and Multi-Region Support

Cloud-Agnostic Platform: Snowflake operates seamlessly across major cloud providers, including AWS, Azure, and Google Cloud Platform. This cloud-agnostic approach allows organizations to choose their preferred cloud provider or operate in a multi-cloud environment.

Global Region Availability: With support for numerous regions worldwide, Snowflake enables organizations to store and process data close to their users, reducing latency and complying with data residency requirements.

Cross-Region Data Replication: Snowflake's cross-region replication feature allows for the duplication of databases across different regions or cloud providers. This capability supports disaster recovery, data sharing, and business continuity strategies.

Secure Data Sharing: Snowflake facilitates secure data sharing across accounts, regions, and cloud platforms without the need to move or copy data. This feature enables real-time collaboration and data exchange between different organizational units or partners.

Client Redirect for Business Continuity: In the event of a regional outage, Snowflake's Client Redirect feature ensures uninterrupted access by redirecting clients to a secondary region. This automatic failover mechanism maintains application availability and user access.

Global Metadata and Catalog Synchronization: Snowflake maintains a unified metadata layer across all regions and clouds, ensuring that object definitions, access permissions, and data catalogs are consistently synchronized. This provides users with a seamless, single view of their data assets regardless of where the data physically resides simplifying governance, discovery, and administration in multi-region deployments.

Reader Accounts

In Snowflake, a Reader Account is a type of account that enables external organizations to access shared data without the need for them to purchase a full Snowflake subscription. Previously referred to as read-only accounts, Reader Accounts are designed to facilitate data sharing between Snowflake customers and third-party users such as customers, partners, vendors, or other stakeholders.

These accounts are part of Snowflake's powerful Data Sharing capabilities, which allow data providers to grant secure, granular, and controlled access to their data without needing to move or duplicate the data. Reader Accounts allow external parties to run queries and gain insights from the shared data directly on Snowflake, without incurring infrastructure or maintenance costs associated with managing their own Snowflake environment.

By leveraging Reader Accounts, businesses can expand the reach of their data to external users, enabling easier collaboration and fostering data-driven decision-making among partners or customers, while maintaining control over their data. [5]

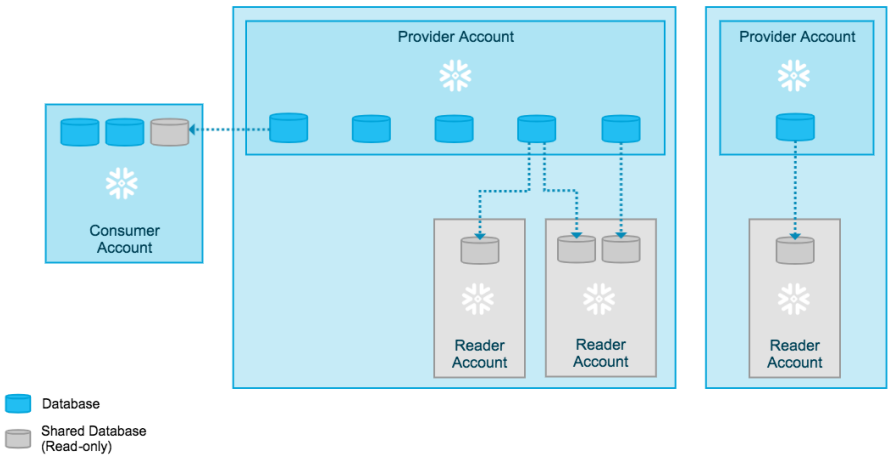

Figure 3: Architecture of Reader Account

Snowflake's Reader Accounts are structured as follows:

- **Data Sharing**: A Snowflake customer (the provider) shares data with an external organization (the consumer) via a Reader Account. This sharing is done using Snowflake's Secure Data Sharing feature, which allows the provider to create a share containing the specific datasets to be made accessible.
- **Access Control**: The external user, through their Reader Account, can query the shared data but cannot modify or manage it. They can execute SELECT queries to view data but cannot alter or delete it.
- **No Additional Snowflake Account Required for Consumers**: The key distinction with Reader Accounts is that the external party does not need their own Snowflake account. Instead, the provider provisions the Reader Account, and the consumer accesses shared data directly.
- **Consumption-Based**: The Reader Account model is cost-effective because the external users do not incur any compute or storage charges for using the data. These costs are borne by the data provider, which makes the consumption model significantly more affordable for external stakeholders.

Setting up a Reader Account involves several steps, including the creation of a share, the provisioning of the account, and the assignment of the necessary permissions.

Steps to Set Up a Reader Account:

The first step is for the Snowflake data provider to create a Share object that contains the data they wish to share with external users. The data provider can control which specific databases, schemas, and tables are included in the share.

Once the share is created, the provider can provision a Reader Account for the external consumer. This is done through the Snowflake Account Administration interface or through SQL commands in Snowflake. The external user is given login credentials and access to the shared data.

The data provider can assign the necessary permissions to the Reader Account. The permissions can be customized to allow the external party to run specific queries, while preventing any data modifications.

After the Reader Account is set up and granted access to the share, the external user can log in to Snowflake using the Reader Account credentials. They can now access the shared data using the standard Snowflake query interface and begin running analytical queries.

Cost considerations for Reader Accounts

1. Compute Costs

In Snowflake, Reader Accounts are designed to allow users to access shared data without having to provision their own compute resources. However, any queries executed by users in a Reader Account utilize the compute resources of the data provider, not the reader's account. This means that whenever a user in a Reader Account runs a query to retrieve or interact with the shared data, the provider's virtual warehouses are used to process the query.

2. Storage Costs

When data is shared with Reader Accounts, the shared data is stored in the provider's Snowflake account. This means that any shared data contributes to the total storage usage and associated costs for the provider. Storage costs are influenced by the amount of data being shared and stored, including both structured data (tables, views) and any associated metadata. Even though Reader Accounts don't incur storage costs for the data they access, the provider is responsible for storing and managing that data in their own Snowflake environment, which will be factored into their overall storage bill.

3. Monitoring Costs

While Reader Accounts are designed for read-only access to shared data, the provider still needs to monitor and manage the data access activity. This can incur additional costs, especially if detailed logging, usage tracking, or resource management services are enabled to ensure proper governance and security. Snowflake provides monitoring tools to track the activity of Reader Accounts.

Airflow

Apache Airflow Introduction

Apache Airflow is a powerful open-source platform primarily used for orchestrating workflows, managing tasks, and automating complex data pipelines. With Airflow, users can define workflows as Directed Acyclic Graphs (DAGs) that allow for flexible, scalable, and dynamic execution of tasks. Airflow is designed to simplify the process of building and managing complex workflows in a distributed environment. Its architecture and extensibility make it a go-to choice for various industries and use cases, particularly in data engineering and machine learning. [6] Airflow was initially developed by Airbnb to handle their growing set of workflows, with the intent to streamline task execution and enhance the scalability of data processing pipelines. Over time, it became an Apache project due to its popularity and broad adoption in the open-source community.

Key Features of Apache Airflow:

- **Scalability**: Can scale from running on a single machine to a fully distributed environment capable of handling large and complex workflows.

- **Extensibility**: With support for custom plugins, operators, and hooks, Airflow can easily integrate with virtually any system or API.

- **Dynamic Workflow Management**: Workflows can be dynamically defined using Python scripts, giving users full flexibility in defining task dependencies and logic.

- **Scheduling**: Airflow has a powerful scheduler that manages when tasks should be executed, allowing for fine-grained control over job scheduling.

- **Web Interface**: Provides an intuitive web interface to monitor, troubleshoot, and manage workflows and tasks.

Key Components of Apache Airflow

Apache Airflow consists of several key components, each serving a specific role in the orchestration and execution of workflows. The following explains each component in detail:

1. Directed Acyclic Graph (DAG)

The core concept of Apache Airflow is the Directed Acyclic Graph (DAG), which represents a collection of tasks that should be executed in a particular order. A DAG in Airflow is essentially a Python script where the workflow is defined. Each task within a DAG can represent any unit of work, such as querying a database, calling an API, running a Python function, or transferring data between systems.

- DAG Structure: DAGs are defined as Python classes or scripts, where the tasks and their dependencies are set up using operators.
- Task Dependencies: The tasks within a DAG are connected via directed edges, defining their order of execution. This dependency structure ensures that tasks are executed in a proper sequence based on their prerequisites.
- Dynamic DAGs: Airflow supports dynamic DAG generation, which allows users to programmatically create tasks and dependencies based on variables, configurations, or external inputs.

- Scheduling: DAGs can be scheduled to run at specific intervals (e.g., daily, hourly, or even on specific dates). This scheduling functionality is crucial for automating workflows.

2. Tasks

In Airflow, a task represents a single unit of work within a DAG. Tasks are implemented as instances of operators that define the work to be performed. For example, a task could run a SQL query, execute a shell command, or invoke a Python function.

- Task Instantiation: Each task in a DAG is instantiated using an operator. The task is then executed when its turn comes in the execution sequence, based on the dependencies set within the DAG.
- Atomic Units: Tasks are atomic, meaning that they cannot be broken down further once defined. They execute a specific action and then report their status (success, failure, etc.).
- Task State: Each task has a state that represents whether it is running, completed, skipped, or failed. Airflow's UI helps visualize these states in real-time.

3. Operators

Operators are the building blocks for tasks in Airflow. Each operator defines a specific type of action that is performed by a task. There are several types of operators, each serving a unique purpose:

- PythonOperator: Executes a Python function.
- BashOperator: Executes a bash command.

- EmailOperator: Sends an email notification.
- SQLOperator: Executes an SQL query.
- DummyOperator: Used as a placeholder for tasks that don’t perform any actions but help in structuring the DAG.
- Custom Operators: Users can define custom operators to meet specific needs, such as interacting with proprietary systems, APIs, or services.

Operators are instantiated with various parameters and can include arguments, task dependencies, and other configurations that define how the task behaves during execution.

4. Scheduler

The Scheduler is responsible for determining when and how tasks in a DAG are executed. It ensures that tasks are triggered based on their schedule or upon task dependency completion.

- Task Scheduling: Airflow allows users to specify when a DAG should run using cron-like syntax or specific intervals (e.g., once a day at midnight, every Monday, etc.).
- Task Queueing: The Scheduler places tasks into the execution queue based on their schedule and dependencies. It checks the task's dependencies and ensures that tasks are only executed once their upstream tasks are completed.
- Backfilling: If there are missed executions (e.g., a task did not run at its scheduled time), the Scheduler can backfill missing tasks based on the defined schedule.

5. Executor

The Executor is responsible for determining how and where tasks should run. It is the component that manages the execution environment for tasks and can be configured to run tasks in different setups based on the scale and infrastructure of the environment.

There are several types of executors in Airflow:

- LocalExecutor: Runs tasks on the same machine where Airflow is installed. This is suitable for small-scale environments or testing purposes.
- CeleryExecutor: Distributes task execution across a cluster of workers using the Celery distributed task queue. This is useful for large-scale deployments where tasks can be run concurrently.
- KubernetesExecutor: Runs each task in its own Kubernetes pod. This is suitable for cloud-native environments where tasks need to run in isolated containers.
- DaskExecutor: A more recent addition that allows for parallel task execution using the Dask distributed computing framework.

The choice of Executor depends on the requirements for concurrency, scaling, and infrastructure.

6. Metadata Database

The Metadata Database is a core component of Airflow. It stores all the metadata related to the DAGs, tasks, and their execution statuses. This includes information such as:

- Task states (whether a task is running, completed, failed, etc.).

- Task logs.
- DAG execution history.
- Scheduler and task logs.

The metadata database is typically a relational database such as MySQL, PostgreSQL, or SQLite (for small environments). It is essential for task tracking, troubleshooting, and monitoring.

7. Web Interface

Airflow provides a Web Interface that serves as the front-end for managing and monitoring workflows. The web interface offers an intuitive graphical representation of DAGs, tasks, and their states. Key features of the Airflow web interface include:

- DAG Visualization: A visual representation of the DAG structure, including task dependencies and their execution statuses.
- Task Logs: Real-time logging of task execution, which helps identify errors and issues in task processing.
- Triggering and Pausing DAGs: Users can manually trigger a DAG or pause its execution from the web interface.
- Monitoring: Airflow's UI provides insights into the performance and health of workflows. Users can view task duration, failures, retries, and logs.
- Admin Panel: Provides administrative controls for managing users, DAGs, and configurations.

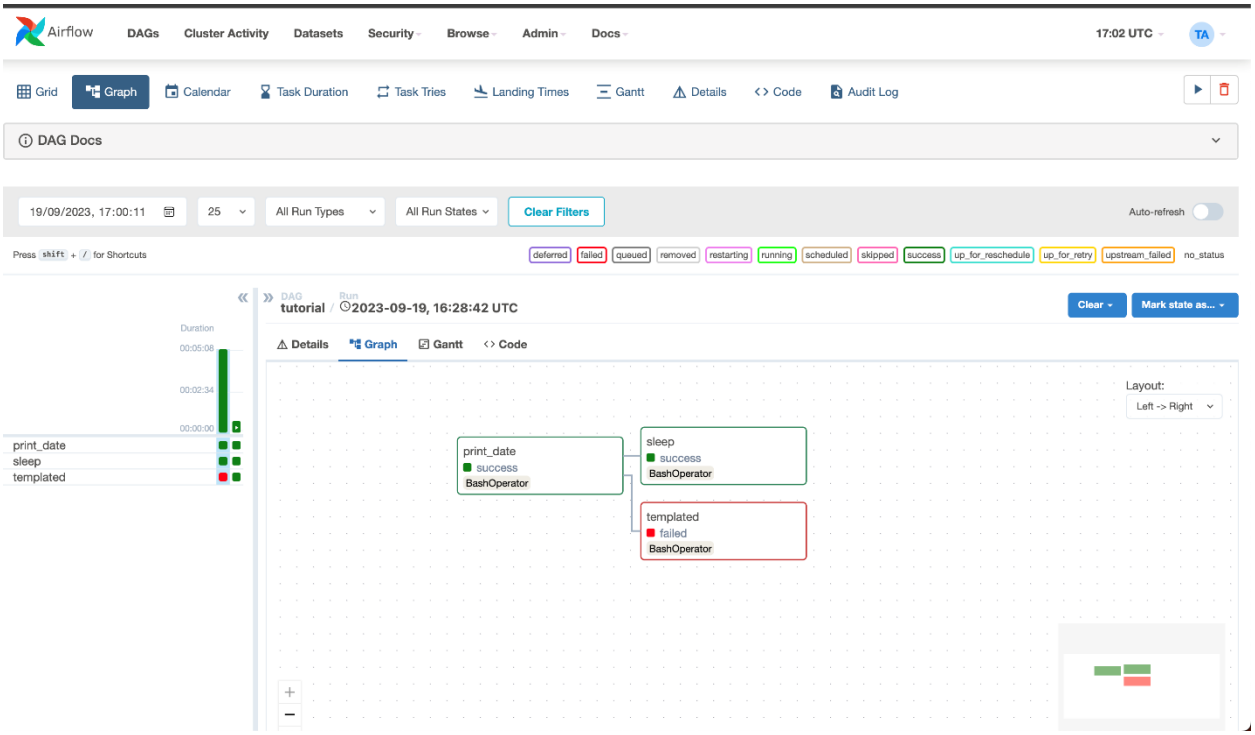

Figure 4: Browser UI of Airflow

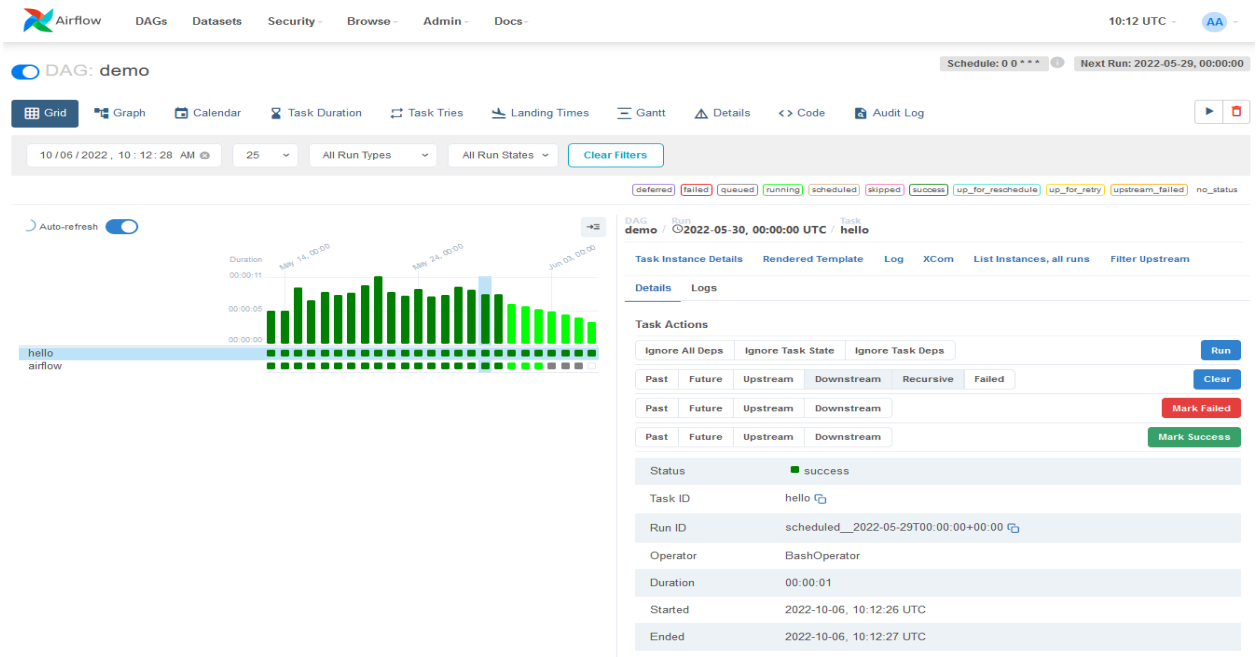

Figure 5: Information of a DAG execution

Airflow Workflow Execution

Airflow works by executing workflows (DAGs) in the following sequence:

Define the DAG: The DAG defines the tasks and their dependencies, including schedules and trigger mechanisms.

Task Scheduling: The Scheduler picks up tasks based on the DAG's defined schedule.

Task Execution: The Executor runs the tasks on the designated worker (using the appropriate executor like Celery or Kubernetes).

Task Monitoring: Task progress is monitored through the Web Interface, and the task's state is updated in the Metadata Database.

Task Completion: Upon task completion, its state is logged, and subsequent tasks are triggered based on the DAG's dependencies.

Airflow Use Cases

Apache Airflow is widely used for various use cases across industries. Some common examples include:

- ETL Pipelines: Airflow is commonly used for building and managing ETL (Extract, Transform, Load) pipelines, where it automates data extraction, transformation, and loading tasks from various sources to data lakes or data warehouses.
- Machine Learning Pipelines: Data scientists use Airflow to automate the stages of machine learning workflows, from data preparation and model training to model deployment and monitoring.
- Data Migration: Airflow is used to orchestrate the movement of data between different systems, databases, and storage platforms.

- Batch Processing: Tasks that need to be run in batches at specific intervals can be managed using Airflow, ensuring timely execution and monitoring.

Core features of Airflow

1. Dynamic Workflow Creation:

Workflows are defined programmatically using Python, allowing for dynamic workflows that adjust based on parameters and external conditions.

2. Task Dependencies:

Dependencies between tasks can be defined explicitly, ensuring proper execution order.

3. Rich Scheduling Capabilities:

Workflows can be scheduled at fixed intervals (e.g., daily, hourly) or triggered on demand.

4. Extensibility:

Custom operators, sensors, and hooks can be created to support various use cases.

5. Scalability:

Scales across multiple machines using executors like Celery and Kubernetes.

6. Monitoring and Logging:

Provides detailed logging and an intuitive UI for monitoring workflow execution.

Architecture of Airflow

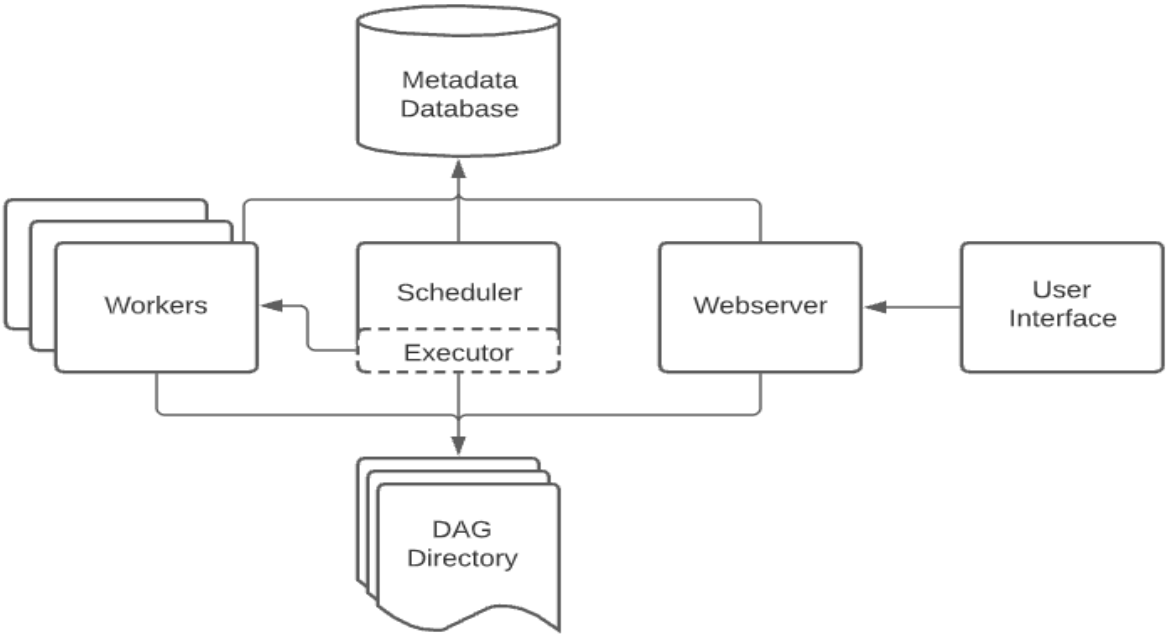

Figure 6: Workflow of Airflow components

Apache Airflow is an open-source platform designed to programmatically author, schedule, and monitor complex data workflows. Built with scalability and flexibility in mind, Airflow allows users to define workflows as code using Python, making them dynamic and easy to maintain. At its core, Airflow uses Directed Acyclic Graphs (DAGs) to represent task dependencies and execution order. With its modular architecture—including components like the Scheduler, Webserver, Metadata Database, and Workers—Airflow supports distributed execution, enabling seamless orchestration of tasks across a variety of systems and environments. Its intuitive web-based UI provides robust monitoring and management capabilities, making it a preferred choice for data engineering teams handling batch processing, ETL pipelines, and other data-centric workflows.

1. DAG Directory

What it is: This is a folder in the Airflow environment where all the DAGs (Directed Acyclic Graphs) are stored as Python scripts.

Purpose: DAG files define workflows — including task dependencies, schedules, operators, and logic.

Functionality:

Airflow continuously scans this directory.

The Scheduler reads these DAG definitions to plan and queue tasks.

The Webserver loads them to display in the UI.

2. Scheduler

What it is: The brain of Airflow that handles scheduling of tasks.

Purpose: Determines when each task needs to run, based on the DAG definitions and schedule intervals.

Responsibilities:

Monitors the DAG directory.

Parses DAG files to build task execution plans.

Queues tasks in the Executor for execution.

Updates metadata in the Metadata Database.

3. Executor

What it is: The interface between the Scheduler and the Workers.

Purpose: Sends tasks to the appropriate Worker(s) to run them.

Types:

SequentialExecutor: For local testing.

LocalExecutor: Executes tasks in parallel on the same machine.

CeleryExecutor/KubernetesExecutor: For distributed execution across machines or containers.

4. Workers

What they are: Background processes that perform the actual execution of tasks.

Purpose: These run the logic defined in each Airflow task (e.g., calling a Python function, executing SQL, pulling data).

How it works:

Workers receive tasks from the Executor.

They execute the tasks and report back the status (success, failure, etc.).

They can be scaled horizontally depending on the Executor type.

5. Metadata Database

What it is: A relational database (e.g., PostgreSQL, MySQL) used internally by Airflow.

Purpose: Stores metadata about DAGs, task instances, execution logs, user configurations, and more.

Functionality:

Tracks DAG runs, task statuses, retries, start/end times.

Accessed by the Scheduler, Webserver, and Workers for state consistency.

6. Webserver

What it is: A Flask-based web application that provides a GUI for Airflow.

Purpose: Hosts the User Interface (UI) so users can:

View DAGs and their structure.

Trigger DAGs manually.

Monitor task runs and view logs.

Pause/unpause DAGs.

Manage users (if RBAC is enabled).

7. User Interface

What it is: The frontend through which users interact with Airflow.

Purpose:

Enables users to manage and monitor workflows visually.

Offers logs, DAG run statuses, Gantt charts, and more.

How These Components Work Together

1. **DAG Creation:**
    - A user writes DAG files and places them in the DAG directory.
    - These files are picked up by the Scheduler and the Webserver.
2. **Scheduling:**
    - The Scheduler reads the DAGs and determines which tasks should run and when.
    - Tasks are then sent to the Executor.
3. **Execution:**
    - The Executor assigns tasks to Workers (depending on executor type).
    - Workers perform the task logic and update the Metadata Database with the task status.
4. **Monitoring:**
    - The Webserver queries the Metadata Database to display DAG statuses and logs in the UI.

Data Flow Summary

- **DAG Directory → Scheduler**: Scheduler reads DAG definitions.
- **Scheduler → Executor**: Scheduler queues tasks.
- **Executor → Workers**: Tasks are assigned to Workers.
- **Workers → Metadata DB**: Task execution results (success/failure) are stored.
- **Webserver → Metadata DB**: UI pulls metadata for display.
- **User Interface → Webserver**: Users interact with the UI to control and monitor workflows.

Benefits of This Architecture

- **Modular and Pluggable:** Components can be replaced or extended independently.
- **Scalable:** Supports small jobs on a laptop to massive, distributed workflows in the cloud.
- **Reliable:** Metadata tracking ensures recoverability and consistency.
- **Observable:** Rich UI and logging for debugging and monitoring.

Amazon Managed Workflows for Apache Airflow (MWAA)

Amazon Managed Workflows for Apache Airflow (MWAA) is a fully managed orchestration service provided by Amazon Web Services (AWS) that allows users to run Apache Airflow workflows in a scalable, secure, and highly available environment. Apache Airflow is a widely used open-source platform for programmatically authoring, scheduling, and monitoring workflows. By integrating Airflow into AWS's managed ecosystem, MWAA significantly simplifies the deployment and management of complex data workflows, allowing teams to focus more on workflow logic rather than infrastructure concerns.

The Need for MWAA

While Airflow is powerful, deploying and managing it on your own comes with challenges:

- Provisioning servers and maintaining infrastructure
- Configuring dependencies and plugins
- Handling auto-scaling, availability, and security
- Performing upgrades and patching

For organizations using AWS, managing these concerns manually can become cumbersome. MWAA solves these issues by offering a fully managed Airflow **service**, allowing users to deploy Airflow workflows without managing the underlying infrastructure.

Key Features of MWAA

1. Fully Managed Service

AWS handles the heavy lifting of setting up and maintaining the infrastructure. This includes provisioning Airflow environments, patching, scaling, and integrating with AWS security and monitoring services.

2. Seamless Integration with AWS Services

MWAA is tightly integrated with a wide array of AWS services:

- **Amazon S3**: Stores DAG files, logs, and plugins
- **Amazon CloudWatch**: Collects logs and metrics
- **AWS Identity and Access Management (IAM)**: Secures task permissions
- **Amazon RDS**: Manages the Airflow metadata database

- **Amazon VPC**: Ensures private network connectivity
  This integration makes it easy to build workflows that interact with services like Redshift, Glue, EMR, Lambda, SageMaker, and more.

3. Built-In Security and Compliance

Security is handled through IAM, AWS Key Management Service (KMS), and VPC integration. MWAA supports encryption at rest and in transit. It is also compliant with many industry standards including HIPAA, SOC, and ISO.

4. Scalability and Availability

MWAA automatically scales the Airflow worker fleet up and down based on the volume of tasks in the queue, ensuring optimal resource usage. It also distributes workloads across multiple availability zones to enhance fault tolerance.

5. Version Control and Environment Customization

You can choose from supported versions of Apache Airflow (e.g., 2.2.2, 2.4.3), and customize environments with Python requirements, custom plugins, and environment variables stored in S3.

6. Monitoring and Logging

Airflow logs are stored in CloudWatch and accessible from the Airflow UI. This central logging helps in monitoring and troubleshooting issues quickly.

MWAA Architecture

At a high level, MWAA includes the following components:

1. **Amazon S3 Bucket** – Holds your DAG code, plugins, and Python requirements.
2. **Environment** – A managed environment where Airflow runs, including scheduler, web server, and workers.
3. **Amazon RDS for PostgreSQL** – Acts as the metadata database for Airflow.
4. **Amazon CloudWatch Logs** – Stores logs for task execution and Airflow system components.
5. **Amazon VPC** – Provides network isolation, routing, and connectivity for your environment.
6. **IAM Roles and Policies** – Secure task execution by granting fine-grained permissions to access AWS services.

You interact with MWAA through the Airflow web UI or CLI, submitting DAGs and monitoring execution in real-time.

Typical Use Cases

1. ETL and Data Pipelines

Organizations often use MWAA to orchestrate ETL pipelines that move and transform data between S3, Redshift, RDS, and external sources. Tasks might include data cleaning, enrichment, and loading.

2. Machine Learning Workflows

You can schedule training jobs in SageMaker, preprocess data, evaluate models, and automate deployment using Airflow DAGs running in MWAA.

3. Data Lake and Data Warehouse Management

Schedule and monitor Glue jobs or Redshift COPY/UNLOAD commands as part of a broader data engineering pipeline.

4. Batch Processing

Trigger EMR clusters for batch jobs or invoke Lambda functions to process small-scale transformations or file events.

5. Business Intelligence Refreshes

Update Power BI or QuickSight dashboards, perform data quality checks, and publish results using automated workflows.

Benefits of Using MWAA

- **Reduced Operational Overhead**: No need to manage Airflow infrastructure.
- **Faster Time-to-Value**: Quickly deploy workflows in production.
- **Integrated Security**: Leverages IAM and VPC for enterprise-grade protection.
- **Cost-Effective**: Pay only for what you use based on environment uptime and compute scale.
-

Limitations and Considerations

- **Cold Start Time**: The environment may take several minutes to spin up after being paused.
- **Version Support**: Only specific Airflow versions are supported at any time.
- **Limited Customization**: Compared to self-managed Airflow, MWAA restricts deep-level configuration changes.

DETAILS OF THE WORK DONE

For the calculation of query and warehouse cost with respect to reader accounts, I have built a data model. For each of the data model, the data source is SNOWFLAKE.READER_ACCOUNT_USAGE schema. Inside the schema we have QUERY_HISTORY, stores data related to queries ran by READER_ACCOUNTS, WAREHOUSE_METERING_HISTORY, stores data related to warehouse usage by READER_ACCOUNTS, STORAGE_HISTORY views. Each of these views has data retention period of 1 year. [7]

Query Cost

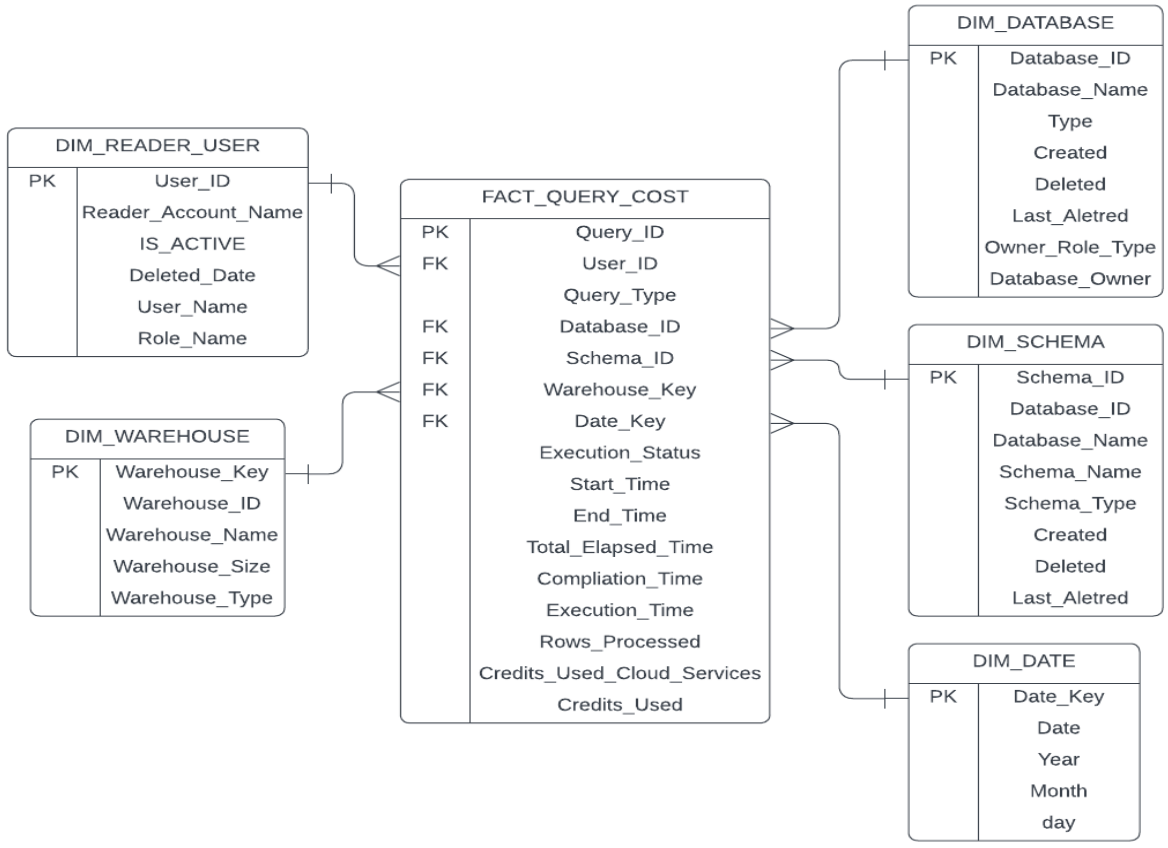

Figure 7: Data model of Query Cost

Overview of the Model

This model represents a dimensional schema designed to track and analyze query costs in Snowflake. It provides a detailed representation of the various entities and their relationships within the system, allowing for effective cost reporting and insights into query execution patterns. The schema includes fact and dimension tables that capture the core details of query costs, metadata, and resource usage.

1. Fact Table:

FACT_QUERY_COST: This is the central table of the schema that captures query-level details and associated costs. Each row in this table represents a specific query execution and includes attributes like execution time, rows processed, and credits consumed.

2. Dimension Tables:

DIM_READER_USER: Provides information about the users executing queries, their roles, and their association with reader accounts.

DIM_WAREHOUSE: Captures details about the virtual warehouses involved in query execution, including size, type, and usage.

DIM_DATABASE: Stores metadata about the databases accessed during queries.

DIM_SCHEMA: Holds schema-level information tied to the databases.

DIM_DATE: A time dimension table to provide temporal analysis capabilities.

How the Model Works

1. Relationships:

The FACT_QUERY_COST table is at the center of the schema and connects to all dimension tables through foreign keys. This star schema design ensures efficient querying and reporting.

2. Data Flow:

Raw query execution data is ingested into FACT_QUERY_COST. Dimension tables are populated with metadata about users, warehouses, databases, schemas, and dates. Queries join the fact table with dimensions for comprehensive analysis.

Warehouse Cost

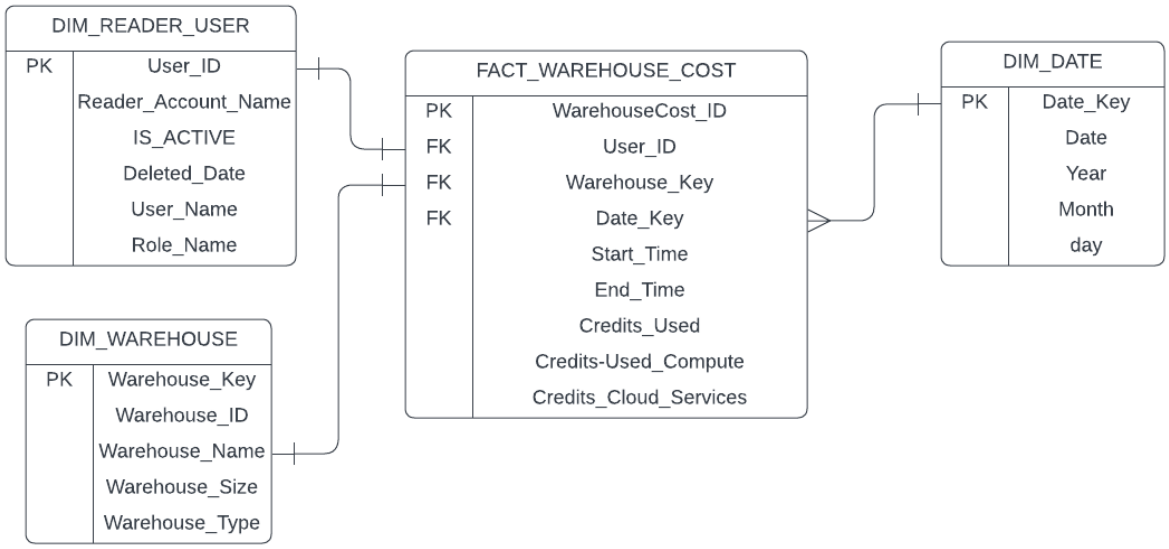

Figure 8: Data model of Warehouse Cost

Overview of the Model

The data model presented is designed for warehouse cost analysis in Snowflake. It integrates multiple dimensions—user, warehouse, and time—to track and analyze warehouse usage and associated costs. This model is highly scalable, enabling comprehensive insights into resource consumption and cost distribution across users, warehouses, and time periods.

1. Fact Table:

The central FACT_WAREHOUSE_COST table records credits consumed during warehouse operations, while the surrounding dimension tables provide detailed metadata to contextualize the cost data.

Credits_Used: Total credits consumed during the operation.

Credits_Used_Compute: Compute-related credits.

Credits_Cloud_Services: Cloud service-related credits.

Attributes including timestamps (Start_Time, End_Time) to track the duration of operations.

2. Dimension Tables:

DIM_READER_USER: Tracks user details like usernames, roles, and account activity status.

DIM_WAREHOUSE: Captures metadata about warehouses, including size, type, and unique identifiers.

DIM_DATE: Provides a temporal context for warehouse usage by breaking down dates into year, month, and day components.

How the Model Works

The model consists of a fact table and multiple dimension tables that interact through primary and foreign key relationships to provide a complete view of warehouse costs. The data model is designed to support a range of use cases aimed at optimizing Snowflake usage and managing costs effectively. It facilitates cost attribution by allowing organizations to assign warehouse costs to individual users or roles, enabling internal billing or chargeback models for cost recovery. Resource optimization is supported by identifying underutilized or oversized warehouses, allowing adjustments to warehouse configurations to minimize unnecessary expenses. Usage analysis helps track warehouse usage patterns across users and time, identifying peak periods to plan resources more effectively. The model also aids in budgeting and forecasting, using historical data to predict future costs and create budget plans aligned with usage trends.

1. Relationships:

FACT_WAREHOUSE_COST ↔ DIM_READER_USER: Links warehouse usage to the users responsible for those operations.

FACT_WAREHOUSE_COST ↔ DIM_WAREHOUSE: Associates cost data with specific warehouse configurations.

FACT_WAREHOUSE_COST ↔ DIM_DATE: Aligns warehouse operations with a temporal dimension for time-based analysis.

Security Model

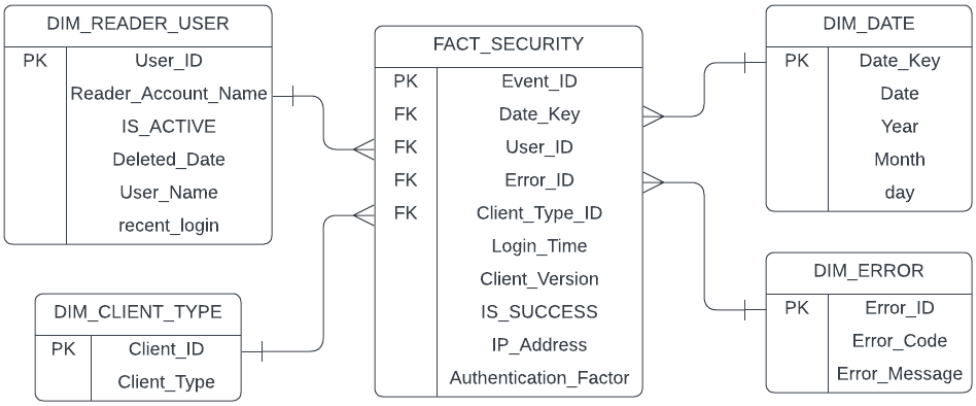

Figure 9: Data model of Security Model

Overview of the Model

This data model is designed for security event analysis and monitoring in Snowflake. This model is specifically built to link security events with user attributes, client types, error information, and time dimensions to provide a holistic view of security activities.

1. Fact Table:

The central fact table, FACT_SECURITY, is a vital element of an organization's data architecture, capturing a wide range of security event data. It tracks detailed information related to security activities, such as user login events, errors encountered during authentication, and user-specific metrics for authentication. The table records login attempts, including timestamps, usernames, IP addresses, devices used, and the success or failure of the attempt. This table is the central component of the model, holding the core transactional or event-level data related to security. Each row in this table represents a specific event or security-related transaction.

2. Dimension Tables:

DIM_READER_USER: Provides user-related metadata such as user ID, account name, status (IS_ACTIVE), deleted date, username, and the most recent login timestamp.

DIM_CLIENT_TYPE: Captures details about the client type used during the events, such as desktop, mobile, or browser.

DIM_DATE: Stores temporal attributes to enable time-based analyses, like year, month, day, and specific dates.

DIM_ERROR: Catalogs error codes and messages that occurred during security events for error tracking and analysis.

How the Model Works

The model revolves around the central fact table FACT_SECURITY, which records events linked to users, errors, client types, and time. Here's how the model works in practice:

1. Relationships:

FACT_SECURITY to DIM_READER_USER (User_ID → User_ID)

This relationship ties security events to specific users. It enables analysis of User behavior patterns, monitoring of inactive or deleted accounts, identifying users frequently associated with failed login attempts or errors. This relationship helps organizations track user activities and pinpoint security vulnerabilities tied to specific accounts.

FACT_SECURITY to DIM_DATE (Date_Key → Date_Key)

Links each event to a specific date, enabling temporal analysis. Facilitates tracking of security trends over time, such as peak times for login attempts or recurring issues on specific dates.

FACT_SECURITY to DIM_ERROR (Error_ID → Error_ID)

Associates' security events with specific errors. This relationship helps in identifying and addressing recurring error patterns, such as failed logins due to incorrect credentials or system issues.

FACT_SECURITY to DIM_CLIENT_TYPE (Client_Type_ID → Client_ID)

Connects events to the type of client initiating them.Enables analysis of client-specific security trends, such as vulnerabilities in a specific client version or unauthorized access attempts through certain clients.

2. User Mapping:

The User_ID in FACT_SECURITY connects to DIM_READER_USER to provide user-specific attributes like account status, deleted status, and recent login details. This mapping allows for user-centric analysis, such as identifying inactive or frequently failing users.

3. Error Tracking:

The ERROR_ID in FACT_SECURITY links to DIM_ERROR, enabling error diagnostics. This dimension helps in aggregating and categorizing error types and providing the corresponding error messages.

4. Time Dimension:

The DATE_KEY in FACT_SECURITY joins with DIM_DATE to incorporate time-based analytics. This allows for tracking trends in security events over time.

5. Client Type Analysis:

The CLIENT_TYPE_ID in FACT_SECURITY connects to DIM_CLIENT_TYPE, providing insights into the devices or clients used during login attempts.

6. Authentication Factor Insights:

Attributes like Authentication_Factor and Is_Success in FACT_SECURITY allow for detailed insights into the effectiveness of authentication methods.

Storage Cost

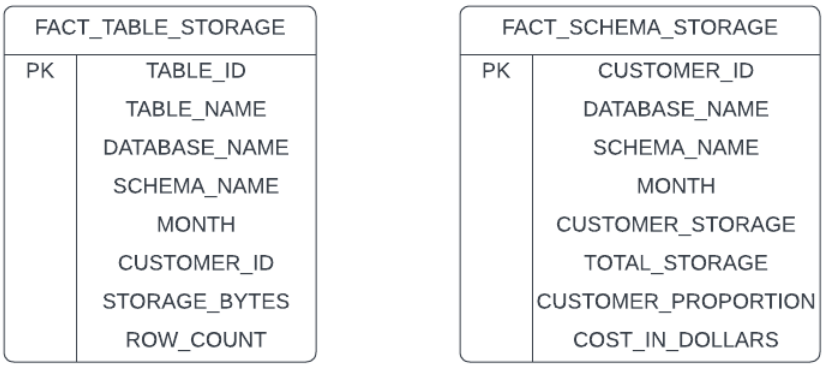

Figure 10: FACT Tables for Storage Cost

Overview of the Data Model

The data model is designed to analyze and allocate storage costs proportionally for customers in a schema.

The process involves calculating the storage usage for each customer at the table level and aggregating it at the schema level. This allows for detailed insights into the storage costs for individual customers.

FACT_TABLE_STORAGE:

This table provides granular details of storage usage for each customer at the table level.

1. Customer Row Proportion = Customer Row Count / Table Row Count

2. Customer Bytes = Table Storage Bytes * Customer Row Proportion

3. Customer Storage Proportion = Customer Bytes / Table Storage Bytes

FACT_SCHEMA_STORAGE:

This table provides aggregated storage and cost information for each customer at the schema level.

Calculations:

1. Cost in Dollar = Customer Storage Proportion * 33 $

How the Model Works

Step 1: Granular Customer Analysis at Table Level

The CUSTOMER_TABLE_STORAGE table is populated by:

1. Calculating the row proportion of each customer in a table.

2. Deriving the bytes consumed by the customer's rows.

3. Computing the customer's share of the table's total storage.

Step 2: Aggregated Customer Analysis at Schema Level

The CUSTOMER_SCHEMA_STORAGE table is populated by:

1. Summing up the storage bytes for each customer across all tables in the schema.

2. Comparing the customer's storage with the total schema storage to compute the proportion.

3. Calculating the cost incurred by the customer based on their storage proportion.

ETL cost

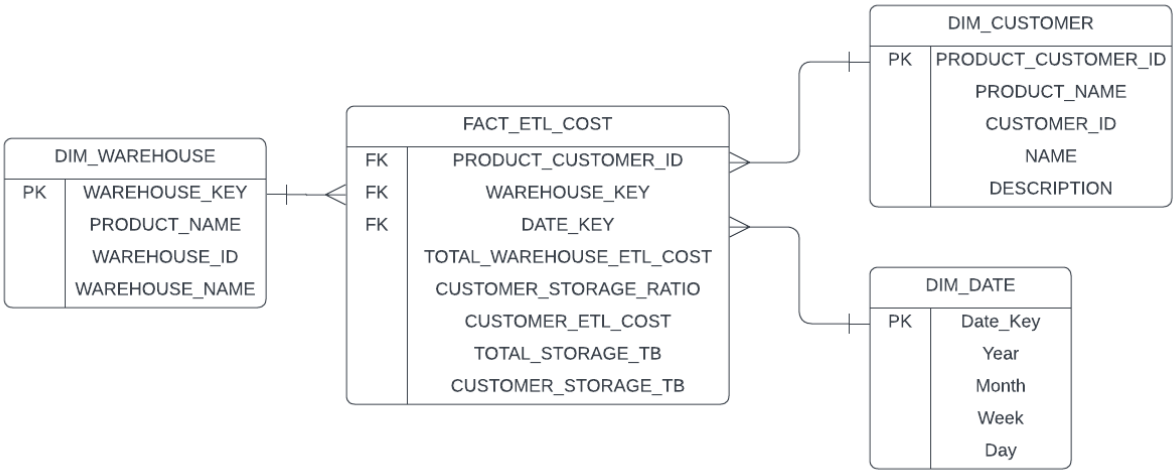

Figure 11: Data model of ETL Cost

Overview of the Data Model

For each of the warehouse, we have divided the weekly cost of that warehouse for each of the customer by multiplying with the customer's storage proportion. The data model consists of the following key tables and their relationships:

DIM_WAREHOUSE:

Contains metadata about warehouses, including:

- WAREHOUSE_KEY
- PRODUCT_KEY
- WAREHOUSE_ID
- WAREHOUSE_NAME

FACT_ETL_COST:

Central fact table with measures like:

- TOTAL_WAREHOUSE_COST,
- CUSTOMER_ETL_COST,
- STORAGE_RATIO

DIM_CUSTOMER:

Provides customer-related details, including:

- PRODUCT_CUSTOMER_ID
- PRODUCT_NAME
- CUSTOMER_ID
- NAME

DIM_DATE: Date dimension table with attributes like DATE_KEY, Month, Year.

How the Model Works

From the WAREHOUSE_METERING_HISTORY in ACCOUNT_USAGE schema, inside the SNOWFLAKE database we have calculated the weekly cost for each of the warehouse that are assigned by the company for the ETL process.

1. Fact and Dimension Relationships:

The FACT_ETL_COST is linked to DIM_WAREHOUSE , DIM_CUSTOMER, and DIM_DATE via foreign keys WAREHOUSE_KEY, PRODUCT_CUSTOMER_ID, and DATE_KEY respectively. These relationships enable filtering and aggregations based on dimensions like time, warehouse, and customer.

2. Measure Calculations:

Key measure such as TOTAL_WAREHOUSECOST, CUSTOMER_ETL_COST allow cost analysis at various granularity levels.

The DIM_DATE table allows time-based aggregations (e.g., monthly, weekly, or yearly trends).

The DIM_WAREHOUSE and DIM_CUSTOMER tables support drill-down, and filtering based on specific warehouses or customers.

Implementation of Data Models

When managing data within a Snowflake data warehouse, particularly for a comprehensive data model like the one provided, the choice between full load and incremental load plays a crucial role in ensuring efficient, accurate, and timely data updates. Both approaches have their unique characteristics and are applied depending on the requirements of the system and business processes.

Below is a detailed explanation of these loading strategies in the context of an entire data model.[8]

Full Load

1. Overview

Definition: A full load operation involves completely replacing the target data with the entire dataset from the source system. All existing records in the target are purged before new data is loaded.

Purpose: Ensures that the data in the target system is an exact replica of the source system at the time of loading.

2. Characteristics

Simplicity: Implementation is straightforward as it involves dropping and recreating data in the target.

Resource-Intensive: Since all records are processed during each load, full loads consume more compute and storage resources, especially with large datasets.

Use of Resources: Requires more time and processing power compared to incremental loading, particularly for larger data models.

Data Integrity: Guarantees consistency between the source and target systems, as the entire dataset is refreshed.

Data Freshness**:** A full load ensures that the data in the target system is always up to date with the source system at the time of loading, providing the most current and accurate representation of the data.

Incremental Load

1. Overview

Definition: Incremental loading involves loading only the new or modified records from the source into the target. The rest of the dataset remains untouched.

Purpose: Minimize resource usage by avoiding redundant processing of unchanged data.

2. Characteristics

Efficiency: Processes only a subset of data, making it faster and less resource-intensive than a full load.

Selective Updates: Uses timestamps, change flags, or unique identifiers to identify and process only the updated records.

Reduced Storage Impact: Retains existing records in the target, adding only the new or changed records.

Complex Implementation: Requires additional logic to identify and handle changes, which might include managing deletions, updates, and inserts.

Data Consistency Challenges**:** Incremental loads may introduce challenges in ensuring data consistency, especially if the logic used to identify changes (such as timestamps or change flags) is not accurate or properly synchronized across systems, leading to potential discrepancies between the source and target systems.

Lower Processing Time**:** Since only new or modified data is loaded, incremental loads generally require less processing time compared to full loads.

Analysis using Power BI

At its core, Power BI is a collection of software services, apps, and connectors that work together to transform unrelated sources of data into coherent, visually immersive, and interactive insights. Whether data is stored in a simple Excel file or complex cloud-based databases, Power BI helps users bring it together, clean it, model it, and visualize it through a user-friendly interface. Power BI is part of the Microsoft Power Platform, which includes Power Apps, Power Automate, and Power Virtual Agents. This integration allows organizations not only to analyze data but also to act using apps and automated workflows.

The process begins with transforming the tables in Snowflake's data model into views. Views are virtual tables that provide a simplified interface for querying the underlying data without modifying the actual stored data. This approach ensures flexibility and abstraction, making it easier to manage complex queries and combine data from multiple tables. By using views, specific columns or calculations can be tailored to the analysis requirements without altering the data at the source. This setup creates a layer of abstraction that enhances both performance and usability in the downstream analytics. [9]

After creating the views in Snowflake, they are loaded into Power BI. Power BI natively supports Snowflake as a data source, enabling direct connectivity. The views are imported or queried live, depending on the analytical requirements. Import mode allows for faster analysis by bringing the data into Power BI's in-memory storage, while Direct Query mode facilitates real-time analysis by querying Snowflake directly. The choice between these modes depends on the size of the data and the performance considerations for the reports being developed. [10]

Core Components of Power BI

Power BI is made up of several elements, each playing a specific role in the data analytics lifecycle:[9]

- **Power BI Desktop**

  This is the primary development tool used by analysts. It provides a drag-and-drop interface to create reports, perform data transformations, define data models, and visualize insights. Power BI Desktop integrates seamlessly with various data sources and supports advanced features like DAX (Data Analysis Expressions) and Power Query (M language).

- **Power BI Service (Cloud)**

  The Power BI Service is a cloud-based SaaS platform used for sharing, collaboration, and enterprise-scale analytics. After creating reports in Power BI Desktop, users can publish them to the service, where dashboards and datasets are accessible across the organization.

- **Power BI Mobile**

  Power BI offers mobile applications for iOS and Android, enabling users to access and interact with dashboards and reports on the go. Mobile responsiveness and real-time alerts keep business leaders informed anytime, anywhere.

- **Power BI Gateway**

  To connect on-premises data sources with the Power BI Service, Microsoft provides the On-premises Data Gateway. This component bridges cloud and on-premises data infrastructures and supports scheduled refreshes and live queries.

- **Power BI Report Server**

  For organizations not ready for the cloud, Power BI Report Server provides an on-premises solution. It allows users to host, manage, and distribute reports behind the firewall, with the same rich features found in the cloud-based version.

- **Power BI Embedded**

  Power BI Embedded allows developers to integrate Power BI dashboards and reports into their own applications using REST APIs and SDKs. This is particularly useful for ISVs and enterprises building white-labeled solutions.

Data Integration and Connectivity

Power BI supports a wide range of data sources—from local files and on-premises databases to cloud-based services and APIs. Over 150 native connectors exist, including:

- Excel, CSV, XML, JSON
- SQL Server, Oracle, PostgreSQL
- Azure SQL Database, Snowflake, Google BigQuery
- Salesforce, Dynamics 365, SharePoint

- REST APIs and custom data connectors

Data can be ingested in two primary modes:

- **Import Mode**

  Data is extracted and loaded into Power BI's in-memory engine (VertiPaq), allowing for lightning-fast performance. This is the most commonly used mode for small to medium-sized datasets.

- **DirectQuery Mode**

  Queries are executed in real-time against the source system. This is ideal for large datasets or when up-to-date information is critical. However, it may impact performance depending on the backend database.[10]

- A third, hybrid approach—Composite Models—combines Import and DirectQuery modes for flexibility in large-scale deployments.

Data Modeling

A key strength of Power BI lies in its ability to define semantic models for analysis. Data modeling involves organizing data into tables, defining relationships, and adding business logic through calculated columns and measures using DAX. Power BI works best with dimensional modeling approaches like the star or snowflake schema. Fact tables are related to dimension tables via relationships, enabling slicing, filtering, and drill-down capabilities in visuals.[11]

DAX (Data Analysis Expressions)

DAX is a powerful formula language used to create custom calculations in Power BI. It supports time intelligence (e.g., YTD, QoQ), complex aggregations, row context evaluations, and calculated tables/measures.[12]

Hierarchies and Aggregations

Power BI allows users to define hierarchies (e.g., Year > Quarter > Month) for intuitive drilldowns in visuals. Aggregations can be defined to pre-compute common queries and improve performance.

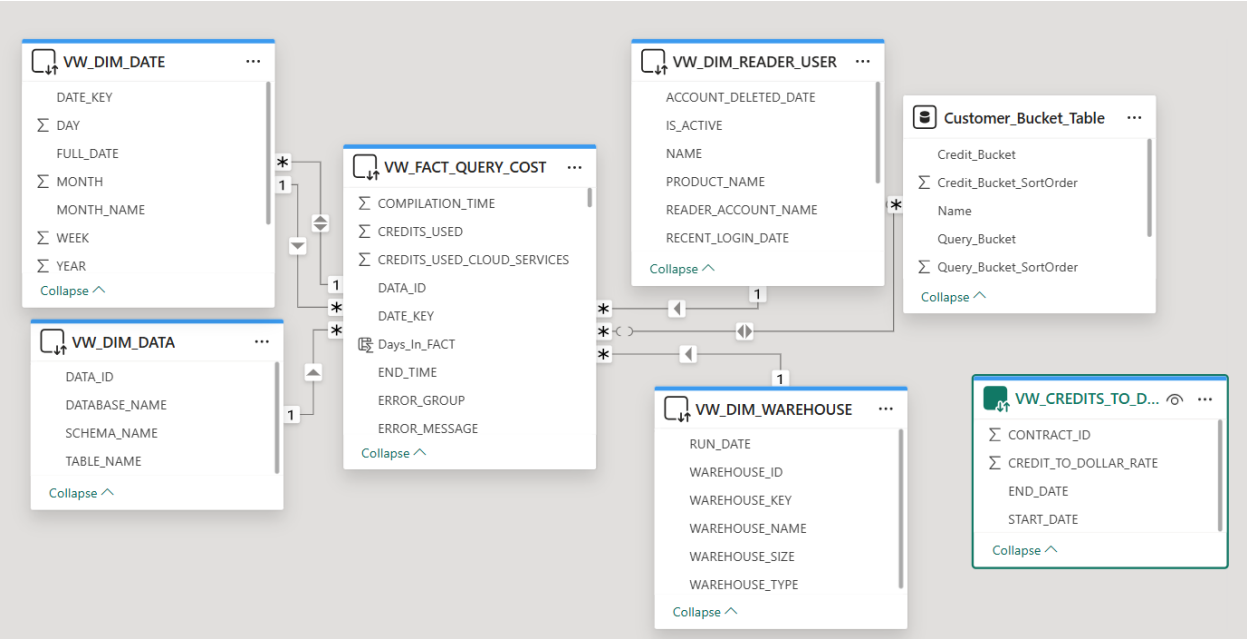

Figure 12: Data Modeling in Power BI

Data Transformation with Power Query

Power BI uses Power Query for data shaping, cleansing, and transformation. It follows the "Extract-Transform-Load" (ETL) pattern and uses the M language behind the scenes. Common Power Query features include:

- Column removal, renaming, and reordering
- Data type transformations
- Merging and appending queries
- Pivoting/unpivoting
- Grouping and summarization
- Error handling and custom functions

Power Query's GUI is intuitive, and the steps are recorded in a sequence that can be refreshed or modified.

- **Visualizations and Dashboards**
  Power BI offers a rich library of built-in visualizations—bar charts, pie charts, scatter plots, maps, gauges, KPIs, and more. Additional visuals can be downloaded from the Power BI Visuals Marketplace or custom-built using the Power BI Developer tools.
- **Interactive Reports**
  Users can click on visuals to apply filters dynamically across the report. This interactive behavior is one of Power BI's standout features, promoting exploration and discovery.

- **Dashboards vs Reports**

  A report is a multi-page view of related visuals, while a dashboard is a single-page view composed of visuals from one or more reports. Dashboards are ideal for high-level monitoring.

- **Sharing, Collaboration, and Governance**

  Workspaces and Apps

  Power BI organizes content into workspaces, where teams collaborate on datasets, reports, and dashboards. Finished content can be published as an App—a curated collection for wider consumption.

- **Row-Level Security (RLS)**

  RLS ensures that users see only the data they're authorized to view. Security rules can be applied within the Power BI data model, aligning with organizational access control policies.

- **Data Refresh and Scheduling**

  Users can schedule data refreshes in the Power BI Service. Depending on the data source, Power BI supports incremental refresh and near real-time datasets (push data model).

- **Enterprise Readiness and Scalability**

  Power BI is designed for both small businesses and large enterprises. Enterprise capabilities include:

- **Large dataset support**: Premium workspaces support models >100 GB.
- **Deployment Pipelines**: Version control and CI/CD for Power BI artifacts.
- **Service Principal and REST APIs**: Automate deployment and governance.
- **Compliance and Certifications**: GDPR, HIPAA, ISO certifications for enterprise data handling.

Power BI Premium offers dedicated capacity, multi-geo deployment, paginated reports, and XMLA endpoint support for third-party tools.

Once the data is in Power BI, the next step involves designing a data model tailored for analysis. In Power BI, the data model reflects the relationships and hierarchies among the tables (or views in this case), enabling advanced analysis.

The relationships defined in Snowflake's data model are carefully replicated in Power BI. For instance, fact tables such as FACT_QUERY_COST and FACT_WAREHOUSE_COST are connected to dimension tables like DIM_READER_USER, DIM_DATE, and DIM_WAREHOUSE through primary and foreign keys. These relationships are crucial because they allow Power BI to aggregate, filter, and analyze the data efficiently based on user queries. [11] The primary purpose of this data model in Power BI is to facilitate interactive analysis. Users can explore query and warehouse costs, track trends, and identify patterns through intuitive dashboards and reports. For example, the model allows analysis of query costs by user roles, reader accounts, or warehouse sizes. Similarly, it can provide insights into the temporal trends of costs, such as daily, monthly, or yearly expenses. Power BI's robust DAX (Data Analysis Expressions) language further enhances this analysis by enabling the creation of custom metrics and calculations that are not available in the original Snowflake views. [12]

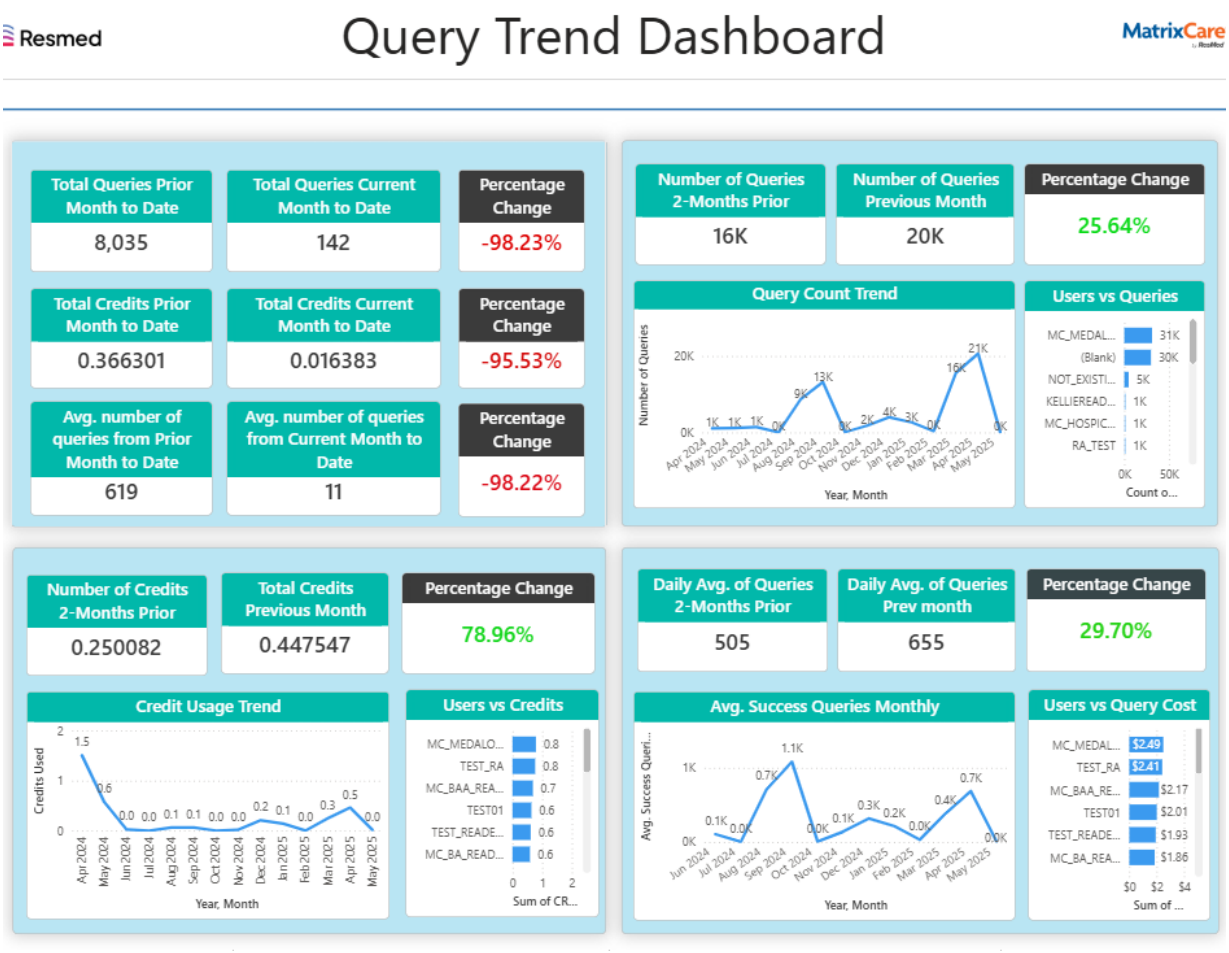

Figure 13: Dashboard for Query Cost Model

Month-Over-Month Analysis

Year, Month	Number of Queries	Percentage Change in Queries	Sum of Credits Used	Percentage Change in Credits	Sum of Execution Time (in MS)	Percentage Change in Execution Time
May 2025	142	-99.31 %	0.0164	-96.44 %	58,967	-96.44 %
Apr 2025	20,609	31.85 %	0.4602	84.00 %	1,656,565	83.98 %
Mar 2025	15,631	3,928.61 %	0.2501	5,886.96 %	900,417	5,886.42 %
Feb 2025	388	-85.31 %	0.0042	-97.07 %	15,041	-97.07 %
Jan 2025	2,642	-33.85 %	0.1425	-31.27 %	512,921	-31.30 %
Dec 2024	3,994	120.06 %	0.2074	975.94 %	746,590	975.67 %
Nov 2024	1,815	1,535.14 %	0.0193	1,169.83 %	69,407	1,168.87 %
Oct 2024	111	-99.16 %	0.0015	-97.59 %	5,470	-97.57 %
Sep 2024	13,200	51.34 %	0.0630	-1.17 %	224,787	-1.65 %
Aug 2024	8,722	13,318.46 %	0.0638	7,573.77 %	228,561	7,539.07 %
Jul 2024	65	-95.39 %	0.0008	-96.35 %	2,992	-96.35 %
Jun 2024	1,410	22.40 %	0.0228	-96.02 %	81,868	-59.04 %
May 2024	1,152	5.88 %	0.5720	-62.01 %	199,892	-45.32 %
Apr 2024	1,088	45.84 %	1.5058	20.60 %	365,574	27.35 %

Figure 14: Dashboard for Month-Over-Month Trend

CONCLUSION

In the evolving landscape of data management, the convergence of cloud-native technologies and intelligent analytics platforms is redefining how organizations handle, process, and derive value from data. The integration of Snowflake, Power BI, and Apache Airflow reflects this transformation, offering a scalable, secure, and cost-effective solution for modern enterprises that seek agility, precision, and deep insight. This project brought together these technologies to build an end-to-end data cost estimation and analysis framework—one that not only simplifies technical complexity but also drives informed business decisions.

At the core of this architecture is Snowflake, a purpose-built cloud data platform that fundamentally departs from the constraints of traditional databases. It introduces a decoupled architecture where compute and storage are isolated, enabling independent scaling and resource allocation based on the nature and size of workloads. This elasticity is a cornerstone for performance and cost control, allowing systems to respond dynamically to demands without incurring unnecessary overhead. As data volumes grow, so does the importance of this scalability. Snowflake's secure, multi-tenant environment ensures that various users and processes can coexist efficiently, without competition for resources or compromise on data governance. The value of this architecture becomes especially pronounced in multi-region and multi-cloud contexts, where global data access, replication, and business continuity are critical.

A unique and transformative feature of Snowflake is its Reader Account functionality. This innovation allows organizations to share live data with external entities without requiring those parties to be Snowflake customers. By eliminating data duplication and infrastructure concerns, Reader Accounts enable seamless,

secure collaboration while the provider retains control over costs and data access. In real-world scenarios, this model supports data-as-a-service offerings, regulatory data disclosures, and strategic data partnerships. Reader Accounts exemplify Snowflake's mission to break down the traditional barriers of data silos, replacing them with flexible, policy-driven access. Their impact on data monetization, customer engagement, and operational transparency is both immediate and scalable.

However, understanding usage and managing cost in such a dynamic environment requires more than raw logs and billing reports. Through data modeling techniques involving fact and dimension tables, it becomes possible to track granular activity across users, warehouses, and queries. These models introduce a semantic layer that allows for real-time and retrospective analysis of cost-driving factors such as compute utilization, storage consumption, ETL job frequency, and query complexity. The result is not just awareness, but control—empowering teams to optimize resources, plan budgets, and implement cost governance policies. This form of operational intelligence ensures that innovation is not only technologically feasible but also financially sustainable.

The ecosystem's analytical layer is enabled by Power BI, a powerful visualization and business intelligence tool that democratizes access to data insights. By directly connecting to Snowflake, Power BI enables stakeholders to explore, visualize, and interpret data through interactive dashboards and custom metrics. Whether through imported datasets or real-time DirectQuery, users are given the flexibility to choose performance over latency based on business requirements. Power BI's data modeling and expression language (DAX) supports complex logic, trend analysis, and predictive indicators—all of which are crucial in cost-sensitive environments where insight into user behavior, warehouse efficiency, or query patterns can lead to actionable change.

The future of data is cloud-native, intelligent, and automated. Organizations that embrace these principles not only survive disruption but lead it. This project serves as a template for such leadership—showing how to unify best-of-breed tools into a system that is greater than the sum of its parts. By integrating Snowflake's elasticity, Power BI's insight, and MWAA's orchestration, enterprises are equipped to navigate complexity with confidence and to convert data from a challenge into a competitive advantage.

In conclusion, the framework presented is not just a technical accomplishment but a strategic enabler. It demonstrates that with thoughtful architecture, proactive governance, and user-focused analytics, data can be transformed into one of an organization's most powerful assets. Whether optimizing internal processes, managing cost structures, or enabling external collaboration, the architecture holds the potential to redefine the way businesses operate in the digital age. As technologies evolve and data volumes grow, this foundation ensures that scalability, efficiency, and insight remain at the forefront of every decision.

BIBLIOGRAPHY

1. https://docs.snowflake.com/en/user-guide/intro-key-concepts

2. https://docs.snowflake.com/en/user-guide/intro-key-concepts#database-storage

3. https://docs.snowflake.com/en/user-guide/intro-key-concepts#query-processing

4. https://docs.snowflake.com/en/user-guide/intro-key-concepts#query-processing

5. https://docs.snowflake.com/en/user-guide/data-sharing-reader-create

6. https://airflow.apache.org/docs/apache-airflow/2.5.3/core-concepts/overview.html

7. https://docs.snowflake.com/en/sql-reference/account-usage#label-reader-account-usage-views

8. https://hevodata.com/learn/incremental-data-load-vs-full-load/

9. https://learn.microsoft.com/en-us/power-bi/connect-data/service-connect-snowflake

10. https://learn.microsoft.com/en-us/power-bi/connect-data/desktop-use-directquery

11. https://www.microsoft.com/en-in/power-platform/products/power-bi/topics/data-modeling/what-is-data-modeling

12. https://learn.microsoft.com/en-us/dax/