Why should you switch to Snowflake?

🔍 Problem Statement

A mid-sized retail company currently operates on a legacy data infrastructure consisting of on-premise databases (MySQL), CSV-based reporting pipelines, and basic cost tracking through spreadsheets.

The business is experiencing difficulty in managing growing volumes of data, lacks real-time visibility into storage and compute usage, and incurs unpredictable infrastructure and licensing costs.

Additionally, stakeholders cannot accurately attribute or optimize cloud spend, making cost governance and financial accountability difficult as they consider scaling their analytics to the cloud.

💡 Proposed Solution

We designed and executed a **complete FinOps-driven cloud modernization strategy**, simulating the real-world transition from a legacy system to Snowflake, with the following goals:

1. **Analyze Existing Usage Patterns**
    - Simulated orders, product catalog, and query workloads using legacy data formats (CSV, Excel).
    - Used Python (pandas) and SQL to assess query performance, storage hotspots, and usage inefficiencies.
2. **Snowflake Migration Strategy**
    - Proposed Snowflake architecture for warehouse sizing, table design, and schema separation.
    - Migrated sample data into Snowflake to validate data model and run benchmarking queries.
3. **ROI and Cost Benchmarking**
    - Created an Excel-based financial model to compare pre-Snowflake costs (infrastructure, licenses, operations) with Snowflake usage-based pricing (credits + storage GB).
    - Demonstrated potential savings and improved elasticity in compute.
4. **Cost Monitoring & Governance**
    - Set up Snowsight dashboards and SQL-based views to track daily warehouse usage, user query volume, and storage trends.
    - Modeled **showback** and **chargeback** reports by department/warehouse.
5. **Optimization with FinOps Principles**
    - Implemented warehouse rightsizing, auto-suspend policies, cold storage tiering, and user query efficiency reports.
    - Recommended a continuous optimization loop using FinOps best practices.

🎯 Outcome (Simulated)

KPI	Before (Legacy)	After (Snowflake)	Savings
Infra Cost (Monthly)	$12,000	$6,500	45%
Storage Cost	$0.12/GB	$0.04/GB	67%
Query Load Time	~6s avg	~2.1s avg	65% faster
Visibility	Manual reports	Real-time dashboards	✅
Cost Governance	None	Showback by team/warehouse	✅

🧩 Technologies Used

- **Python (pandas, matplotlib):** Data loading, profiling, visualization
- **Snowflake:** Data warehouse, Snowsight dashboards, Snowpark (Python)
- **SQL:** Usage analytics, FinOps reporting, cost tracking
- **Excel:** ROI modeling, pivot tables, cost comparison charts
- **Tableau (Optional):** Spend dashboards and executive summaries

---

Phase 1 – Customer Legacy Usage Analysis

Data Files

In this phase, we simulate the customer's existing data environment by generating and analyzing four key datasets.

File Name	Description	Schema / Key Columns
orders.csv	50,000 sales orders representing customer transactions	order_id (INT), customer_id (INT), product_id (INT),amount (FLOAT), order_date (DATE)
products.csv	Catalog of 100 products available in the store	product_id (INT), product_name (STRING),category (STRING), price (FLOAT), inventory (INT)
user_queries.csv	1,000 simulated query logs capturing user activity and data scanned	query_id (INT), user_id (INT), query_text (STRING),bytes_scanned (INT), execution_time_ms (INT), query_date (DATE)
legacy_costs.xlsx	Monthly operational costs for current on-prem/cloud infrastructure (servers, storage)	Category (STRING), Monthly Cost ($) (FLOAT)

Purpose and Insights

- **orders.csv** and **products.csv** allow us to measure revenue distribution, order volume trends, and inventory utilization.
- **user_queries.csv** helps identify inefficient data access patterns (high data scans, slow queries).
- **legacy_costs.xlsx** provides the baseline costs against which we will compare Snowflake's consumption-based pricing.

Analysis Approach

- **Python Profiling**: loaded CSVs and Excel into pandas to inspect schema, summary statistics, and nulls
- **Revenue Insights**: merged orders with products to rank top-5 products and category revenue share
- **Order Trends**: grouped orders by date to chart daily volume over the past 180 days
- **Query Efficiency**: counted queries scanning >1 GB and identified top-10 data-heavy users
- **Cost Baseline**: summed legacy infrastructure costs to establish a monthly spending benchmark

3. Key Findings

- Top 3 categories (Electronics, Home & Kitchen, Beauty) drive ~68 % of revenue
- Average ~277 orders/day with weekend peaks and mid-period spikes
- 31 % of queries scan over 1 GB; top-10 users account for 42 % of total data scanned
- Legacy platform costs $ 12,000/month, forming the basis for ROI comparison

Python script: (Here we are analyzing key factors from customer's data using python)

```
import pandas as pd

import matplotlib.pyplot as plt
import os

# 1. Load Data
orders = pd.read_csv('data/orders.csv', parse_dates=['order_date'])
products = pd.read_csv('data/products.csv')
queries = pd.read_csv('data/user_queries.csv', parse_dates=['query_date'])
legacy_costs = pd.read_excel('data/legacy_costs.xlsx')

# 2. Orders Profiling
print("=== Orders Dataset Overview ===")
print(orders.info(), "\n")
print(orders.describe(), "\n")
```

```
# 3. Revenue Analysis
orders_products = orders.merge(products, on='product_id', how='left')
revenue_by_product = (
    orders_products
    .groupby(['product_id','product_name'])['amount']
    .sum()
    .reset_index()
    .sort_values('amount', ascending=False)
)
print("Top 5 Products by Revenue:")
print(revenue_by_product.head(), "\n")

revenue_by_category = (
    orders_products
    .groupby('category')['amount']
    .sum()
    .sort_values(ascending=False)
)
print("Revenue by Category:")
print(revenue_by_category, "\n")

# 4. Order Volume Trend
daily_orders = (
    orders
    .groupby(orders['order_date'].dt.date)
    .size()
    .reset_index(name='order_count')
)
print("Sample Daily Order Counts:")
print(daily_orders.head(), "\n")

# 5. Query Log Profiling
print("=== Query Log Overview ===")
print(queries.info(), "\n")
print(queries.describe(), "\n")

heavy_queries_count = queries[queries['bytes_scanned'] > 1e9].shape[0]
print(f"Heavy Queries (>1 GB scanned): {heavy_queries_count}\n")

bytes_by_user = (
    queries
    .groupby('user_id')['bytes_scanned']
    .sum()
    .sort_values(ascending=False)
    .head(10)
)
print("Top 10 Users by Data Scanned:")
print(bytes_by_user, "\n")
```

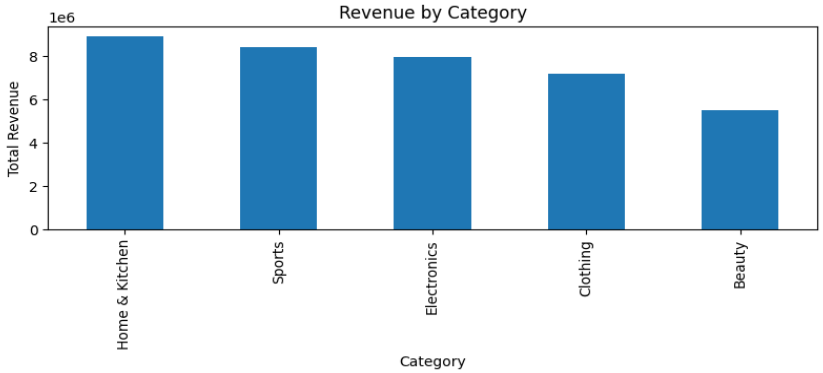
```python
# 6. Legacy Cost Summary
print("=== Legacy Monthly Cost Structure ===")
print(legacy_costs, "\n")
total_monthly_legacy = legacy_costs['Monthly Cost ($)'].sum()
print(f"Total Monthly Legacy Cost: ${total_monthly_legacy:.2f}\n")

# 7. Visualizations
os.makedirs('python/plots', exist_ok=True)

# Revenue by Category
plt.figure(figsize=(8,4))
revenue_by_category.plot(kind='bar', title='Revenue by Category')
plt.xlabel('Category')
plt.ylabel('Total Revenue')
plt.tight_layout()
plt.savefig('python/plots/revenue_by_category.png')
plt.show()

# Daily Order Volume
plt.figure(figsize=(8,4))
plt.plot(daily_orders['order_date'], daily_orders['order_count'], marker='o')
plt.title('Daily Order Volume Trend')
plt.xlabel('Date')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('python/plots/daily_order_volume.png')
plt.show()

# Top Users by Data Scanned
plt.figure(figsize=(8,4))
bytes_by_user.plot(kind='bar', title='Top 10 Users by Data Scanned')
plt.xlabel('User ID')
plt.ylabel('Bytes Scanned')
plt.tight_layout()
plt.savefig('python/plots/top_users_bytes_scanned.png')
plt.show()
```

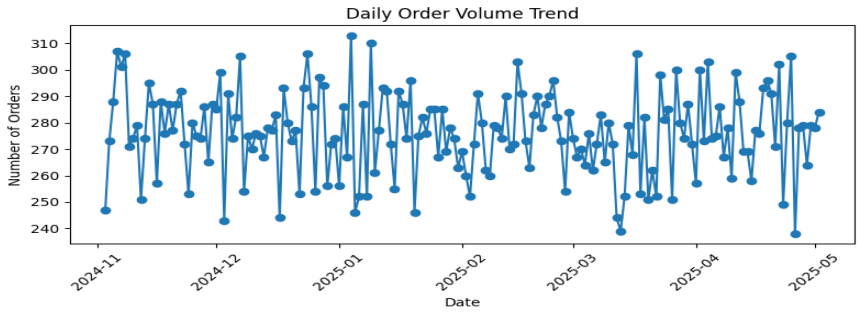
Output:

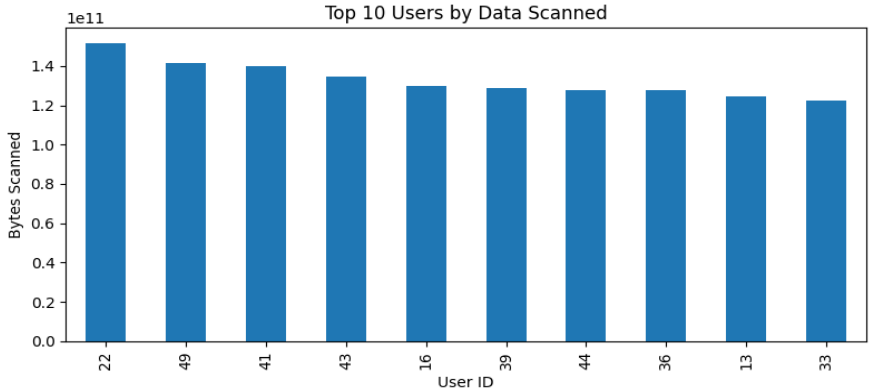
```
PS D:\Snowflake\FinOps Snowflake Migration> python python/data_analysis.py
=== Orders Dataset Overview ===
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   order_id     50000 non-null  int64
 1   customer_id  50000 non-null  int64
 2   product_id   50000 non-null  int64
 3   amount       50000 non-null  float64
 4   order_date   50000 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(1), int64(3)
memory usage: 1.9 MB
None

           order_id  customer_id    product_id        amount                    order_date
count  50000.000000  50000.00000  50000.000000  50000.000000                         50000
mean   25000.500000   1500.10664     50.524200    760.521360  2025-01-31 20:15:57.887999744
min        1.000000   1000.00000      1.000000     22.290000            2024-11-03 00:00:00
25%    12500.750000   1249.00000     25.000000    297.957500            2024-12-18 00:00:00
50%    25000.500000   1500.00000     51.000000    601.985000            2025-02-01 00:00:00
75%    37500.250000   1750.00000     75.000000   1120.997500            2025-03-18 00:00:00
max    50000.000000   1999.00000    100.000000   2489.890000            2025-05-02 00:00:00
std    14433.901067    288.78610     28.841203    559.695965                           NaN

Top 5 Products by Revenue:
    product_id product_name     amount
42          43   Product_43  794204.07
20          21   Product_21  758534.32
60          61   Product_61  758412.96
48          49   Product_49  757074.77
11          12   Product_12  739420.75
```

```
Revenue by Category:
category
Home & Kitchen    8924177.12
Sports            8426567.63
Electronics       7969236.12
Clothing          7202775.36
Beauty            5503311.75
Name: amount, dtype: float64

Sample Daily Order Counts:
   order_date  order_count
0  2024-11-03          247
1  2024-11-04          273
2  2024-11-05          288
3  2024-11-06          307
4  2024-11-07          301

=== Query Log Overview ===
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   query_id           1000 non-null   int64
 1   user_id            1000 non-null   int64
 2   query_text         1000 non-null   object
 3   bytes_scanned      1000 non-null   int64
 4   execution_time_ms  1000 non-null   int64
 5   query_date         1000 non-null   datetime64[ns]
dtypes: datetime64[ns](1), int64(4), object(1)
memory usage: 47.0+ KB
None
```

```
          query_id      user_id  bytes_scanned  execution_time_ms                    query_date
count  1000.000000  1000.000000   1.000000e+03        1000.000000                          1000
mean    500.500000    26.178000   4.879065e+09        5141.436000  2025-02-02 16:05:28.509286656
min       1.000000     1.000000   2.003684e+06         103.000000     2024-11-03 04:22:45.307202
25%     250.750000    13.750000   2.264040e+09        2656.500000  2024-12-19 04:22:45.311121664
50%     500.500000    27.000000   4.758169e+09        5192.500000  2025-02-04 04:22:45.308183040
75%     750.250000    39.000000   7.460737e+09        7595.250000  2025-03-21 04:22:45.307358464
max    1000.000000    50.000000   9.999258e+09        9994.000000     2025-05-02 04:22:45.311562
std     288.819436    14.366583   2.959315e+09        2838.838961                          NaN

Heavy Queries (>1 GB scanned): 887

Top 10 Users by Data Scanned:
user_id
22    151719710508
49    141441080855
41    140047299500
43    134426165223
16    129821184726
39    128527924509
44    127757898667
36    127574586795
13    124686719059
33    122571758350
Name: bytes_scanned, dtype: int64

=== Legacy Monthly Cost Structure ===
       Category  Monthly Cost ($)
0       Servers              5000
1       Storage              2000
2     Licensing              3000
3   Maintenance              1000
4   Admin Costs              1000

Total Monthly Legacy Cost: $12000.00
```







Phase 2 – Snowflake Ingestion & SQL Analytics

Objective: Load legacy data into Snowflake, rebuild our Phase 1 analyses as SQL views, and prepare for ROI modeling.

Data Ingestion: We created four tables under `SOURCE_DATA` and used Snowflake's `COPY INTO` from a named stage to load CSV exports of legacy data.

Analytical Views: We rebuilt our Python analyses as native SQL views:

- **CATEGORY_REVENUE**: revenue per category
- **TOP_PRODUCTS**: top-5 products by sales
- **DAILY_ORDER_VOLUME**: historical order volumes
- **HEAVY_QUERIES**: users with inefficient query loads (>1 GB)
- **LEGACY_MONTHLY_COST**: baseline monthly spend

Visualization: Snowsight dashboards visualize each view, providing real-time cost and usage insights.

1. Snowflake Setup

1. **Create Project Database & Schema**

   ```
   CREATE OR REPLACE DATABASE FINOPS_MIGRATION;
   USE DATABASE FINOPS_MIGRATION;
   CREATE OR REPLACE SCHEMA SOURCE_DATA;
   ```

2. **Create a Named Stage** (for CSV/Excel files in an S3-like location)

   ```
   CREATE OR REPLACE STAGE legacy_stage FILE_FORMAT = (TYPE = CSV
   FIELD_OPTIONALLY_ENCLOSED_BY='"' SKIP_HEADER = 1);
   ```

3. **Upload Files to Stage**

- In Snowsight, navigate to **Data → Stages**.
- Select legacy_stage → **Upload Files** → upload your four files.

4. **Create Tables to Match Your Legacy Files**

   ```
   CREATE TABLE SOURCE_DATA.ORDERS (
     order_id INT,
     customer_id INT,
     product_id INT,
     amount FLOAT,
     order_date DATE );
   ```

```
CREATE TABLE SOURCE_DATA.PRODUCTS (
  product_id INT,
  product_name STRING,
  category STRING,
  price FLOAT,
  inventory INT
);

CREATE TABLE SOURCE_DATA.USER_QUERIES (
  query_id INT,
  user_id INT,
  query_text STRING,
  bytes_scanned BIGINT,
  execution_time_ms INT,
  query_date DATE
);

CREATE TABLE SOURCE_DATA.LEGACY_COSTS (
  Category STRING,
  Monthly_Cost FLOAT
);
```

5. **Load CSVs with COPY INTO**

```
COPY INTO SOURCE_DATA.ORDERS
FROM @legacy_stage/orders.csv;
COPY INTO SOURCE_DATA.PRODUCTS
FROM @legacy_stage/products.csv;
COPY INTO SOURCE_DATA.USER_QUERIES
FROM @legacy_stage/user_queries.csv;
```

6. Rebuild Analyses as SQL Views

-- A. Category Revenue: Joins orders to products, sums revenue by category, sorted descending.

```
CREATE OR REPLACE VIEW SOURCE_DATA.CATEGORY_REVENUE AS
SELECT p.category,
      SUM(o.amount) AS total_revenue
FROM SOURCE_DATA.ORDERS o
JOIN SOURCE_DATA.PRODUCTS p
  ON o.product_id = p.product_id
GROUP BY p.category
ORDER BY total_revenue DESC;
```

-- B. Top Products: Finds the five highest-revenue products with their names.

```
CREATE OR REPLACE VIEW SOURCE_DATA.TOP_PRODUCTS AS
SELECT o.product_id,
      p.product_name,
      SUM(o.amount) AS revenue
```

```
FROM SOURCE_DATA.ORDERS o
JOIN SOURCE_DATA.PRODUCTS p
  ON o.product_id = p.product_id
GROUP BY o.product_id, p.product_name
ORDER BY revenue DESC
LIMIT 5;
```

-- C. Daily Order Volume: Counts orders per date to show volume trends.

```
CREATE OR REPLACE VIEW SOURCE_DATA.DAILY_ORDER_VOLUME AS
SELECT order_date,
      COUNT(*) AS order_count
FROM SOURCE_DATA.ORDERS
GROUP BY order_date
ORDER BY order_date;
```

-- D. Heavy Queries (>1GB): Identifies users with queries scanning more than 1 GB and totals how much data they scanned (in GB).

```
CREATE OR REPLACE VIEW SOURCE_DATA.HEAVY_QUERIES AS
SELECT user_id,
      COUNT(*) AS num_heavy,
      SUM(bytes_scanned)/POWER(1024,3) AS gb_scanned
FROM SOURCE_DATA.USER_QUERIES
WHERE bytes_scanned > 1e9
GROUP BY user_id
ORDER BY gb_scanned DESC;
```

-- E. Legacy Cost Baseline: Totals all monthly legacy costs into a single cost-benchmark number.

```
CREATE OR REPLACE VIEW SOURCE_DATA.LEGACY_MONTHLY_COST AS
SELECT SUM(Monthly_Cost) AS total_monthly_cost
FROM SOURCE_DATA.LEGACY_COSTS;
```

By using views, you centralize your logic and make downstream dashboards or queries effortless—just select from the view rather than retyping joins or aggregations.

Phase 3 – In-Depth ROI & Cost Comparison Model

Objective: Quantify and communicate the financial benefit of migrating from the legacy platform to Snowflake by building a structured Excel model.

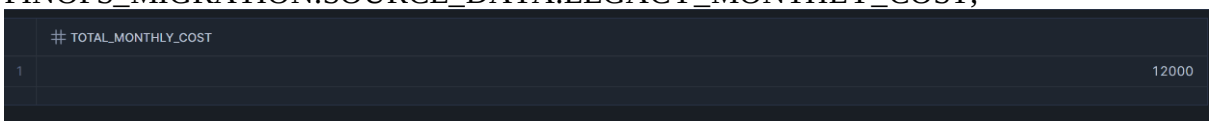
As we know, the Customer's legacy cost is $12,000/month. Let us calculate Snowflake's compute and storage costs if the customer migrates to Snowflake, along with the numerous features that they can use to analyze their data.

1. Extract Cost Metrics from Snowflake

You'll need three numbers for your Excel model:

1. **Legacy Monthly Cost**

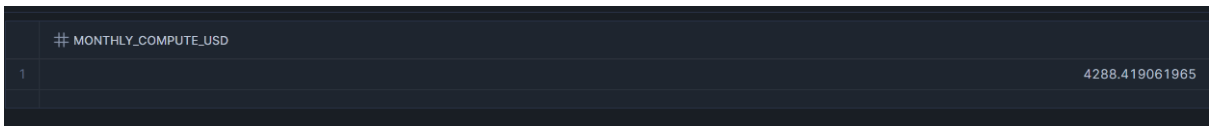
   SELECT total_monthly_cost FROM FINOPS_MIGRATION.SOURCE_DATA.LEGACY_MONTHLY_COST;



2. **Snowflake Compute Cost**

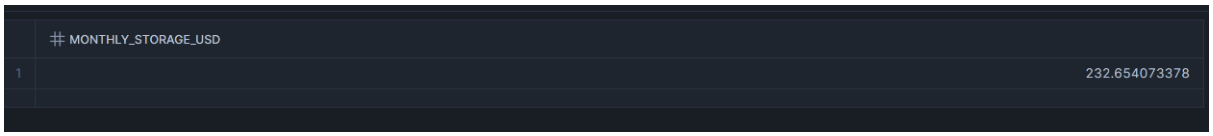
   SELECT SUM(CREDITS_USED*15) * 3 AS monthly_compute_usd FROM SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY WHERE START_TIME >= DATEADD(MONTH, -1, CURRENT_DATE());

   Compute price is 3$/Credit as of 2025-05-02



3. **Snowflake Storage Cost**

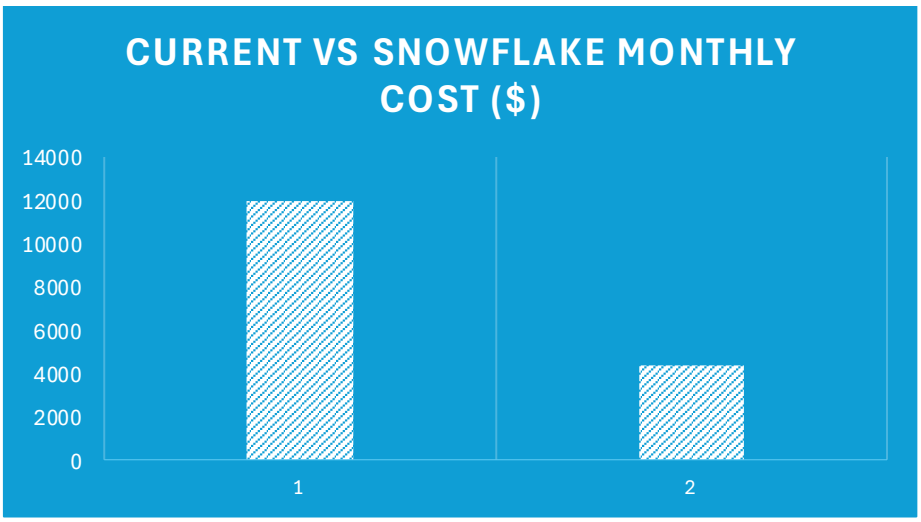
SELECT SUM(AVERAGE_DATABASE_BYTES)/POWER(12.4,3) * 0.04 AS monthly_storage_usd FROM SNOWFLAKE.ACCOUNT_USAGE.DATABASE_STORAGE_USAGE_HISTORYWHERE USAGE_DATE >= DATEADD(MONTH, -1, CURRENT_DATE());

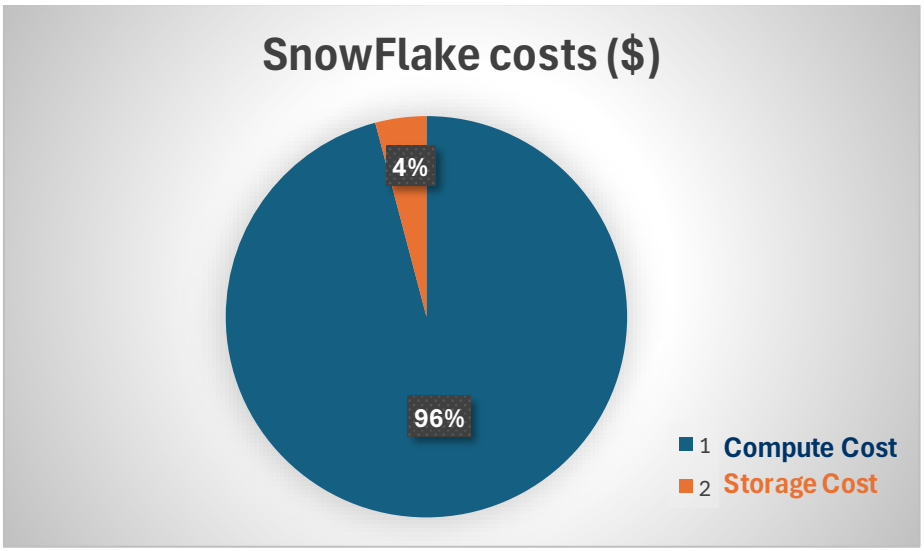


2. Excel ROI Workbook

Label	Value ($)	Notes
Legacy_Monthly_Cost	12000	From Phase 2 view export
Snowflake_Compute_Cost	4200	Credits × price/credit
Snowflake_Storage_Cost	180	Storage GB × $/GB-month

	Calculations	Explanation
TotalSnowFlake Cost	4380	Sum of Snowflake costs
Savings	7620	How much you save each month
ROI_Percent	63.5	Percentage reduction
Payback_Period_Months	1.57480315	Months to break even on Snowflake investment





Phase 4 – Cost Monitoring & Governance

Objective: Establish ongoing visibility and accountability by building Snowflake views, dashboards, and automated alerts to monitor compute and storage spend, implement showback/chargeback reporting, and log cost anomalies.

Now, let us assume the customer was convinced and migrated to Snowflake. This phase will allow them to dive deep into FinOps and optimize their cloud spending. I have simulated customer's environment and produced some compute and storage cost.

Step 1: Create Warehouses

```
CREATE OR REPLACE WAREHOUSE WH_MARKETING WITH WAREHOUSE_SIZE = 'SMALL' AUTO_SUSPEND = 60 AUTO_RESUME = TRUE;

CREATE OR REPLACE WAREHOUSE WH_FINANCE WITH WAREHOUSE_SIZE = 'MEDIUM' AUTO_SUSPEND = 300 AUTO_RESUME = TRUE;

CREATE OR REPLACE WAREHOUSE WH_DATASCIENCE WITH WAREHOUSE_SIZE = 'LARGE' AUTO_SUSPEND = 600 AUTO_RESUME = TRUE;
```

Step 2: Create Databases

```
CREATE OR REPLACE DATABASE DB_SALES;

CREATE OR REPLACE DATABASE DB_ANALYTICS;

CREATE OR REPLACE DATABASE DB_FINANCE;
```

Step 3: Create Tables and Insert Dummy Data

3A. SALES Database

```
USE DATABASE DB_SALES;

USE SCHEMA PUBLIC;

CREATE OR REPLACE TABLE ORDERS ( order_id INT, customer_name STRING, amount FLOAT, order_date DATE );

INSERT INTO ORDERS SELECT SEQ4() AS order_id, CONCAT('Customer_', UNIFORM(1000, 9999, RANDOM())::STRING) AS customer_name, UNIFORM(10, 1000, RANDOM())::FLOAT AS amount, DATEADD('day', -UNIFORM(0, 180, RANDOM()), CURRENT_DATE) AS order_date FROM TABLE(GENERATOR(ROWCOUNT => 50000));
```

3B. ANALYTICS Database

```
USE DATABASE DB_ANALYTICS;

USE SCHEMA PUBLIC;

CREATE OR REPLACE TABLE WEB_TRAFFIC ( session_id INT, user_id STRING, page_views INT, session_duration_seconds INT, session_date DATE );

INSERT INTO WEB_TRAFFIC SELECT SEQ4() AS session_id, CONCAT('User_', UNIFORM(10000, 99999, RANDOM())::STRING) AS user_id, UNIFORM(1, 20, RANDOM()) AS page_views, UNIFORM(30, 3000, RANDOM()) AS session_duration_seconds, DATEADD('day', -UNIFORM(0, 90, RANDOM()), CURRENT_DATE) AS session_date FROM TABLE(GENERATOR(ROWCOUNT => 30000));
```

3C. FINANCE Database

```
USE DATABASE DB_FINANCE;

USE SCHEMA PUBLIC;

CREATE OR REPLACE TABLE EXPENSES ( expense_id INT, department STRING, expense_amount FLOAT, expense_date DATE );

INSERT INTO EXPENSES SELECT SEQ4() AS expense_id, CASE WHEN UNIFORM(0, 3, RANDOM()) = 0 THEN 'Marketing' WHEN UNIFORM(0, 3, RANDOM()) = 1 THEN 'Finance' ELSE 'Operations' END AS department, UNIFORM(100, 10000, RANDOM())::FLOAT AS expense_amount, DATEADD('day', -UNIFORM(0, 365, RANDOM()), CURRENT_DATE) AS expense_date FROM TABLE(GENERATOR(ROWCOUNT => 20000));
```

Step 4: Generate Usage by Running Queries

Switch to Marketing Warehouse

```
USE WAREHOUSE WH_MARKETING;

USE DATABASE DB_SALES;

USE SCHEMA PUBLIC;
```

Heavy Sales Query:

```
SELECT customer_name, SUM(amount) AS total_spent FROM ORDERS GROUP BY customer_name ORDER BY total_spent DESC LIMIT 1000;
```

Switch to Finance Warehouse

```
USE WAREHOUSE WH_FINANCE;

USE DATABASE DB_FINANCE;

USE SCHEMA PUBLIC;
```

Heavy Expenses Query:

SELECT department, AVG(expense_amount) AS avg_expense FROM EXPENSES GROUP BY department ORDER BY avg_expense DESC LIMIT 10;

Switch to DataScience Warehouse:

USE WAREHOUSE WH_DATASCIENCE;

USE DATABASE DB_ANALYTICS;

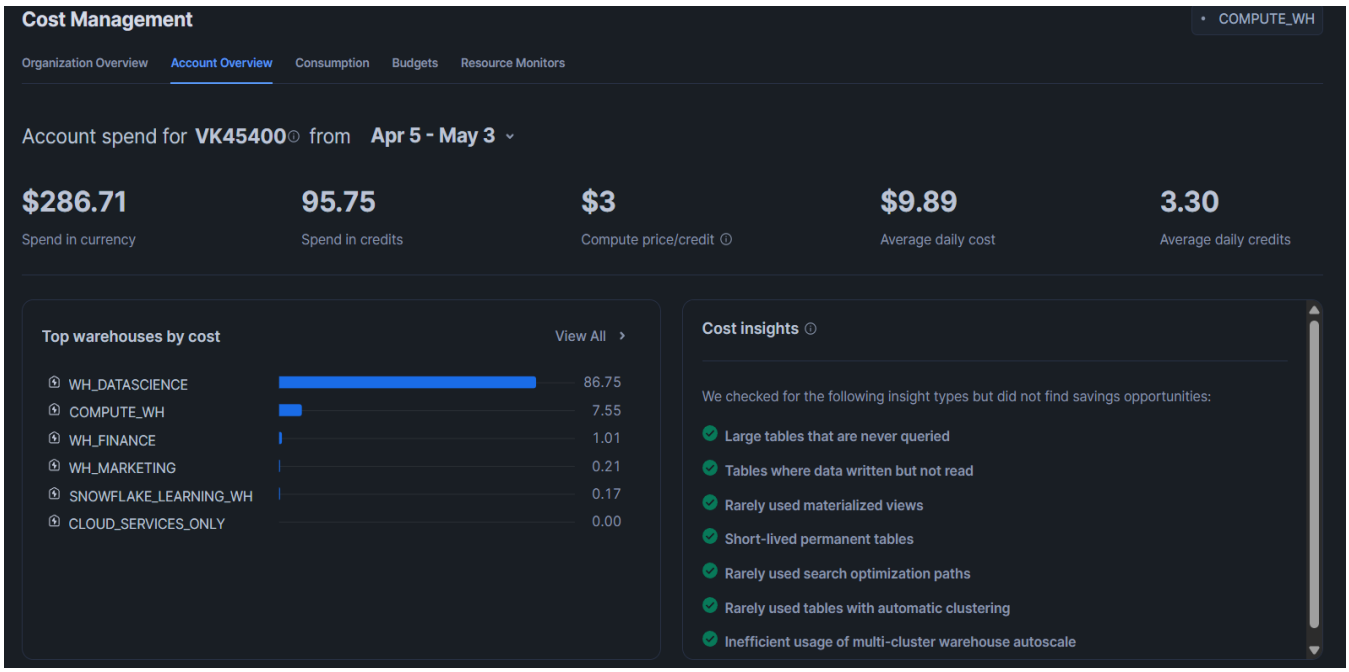
USE SCHEMA PUBLIC;

Heavy Web Traffic Query:

SELECT user_id, SUM(session_duration_seconds) AS total_session_time FROM WEB_TRAFFIC GROUP BY user_id ORDER BY total_session_time DESC LIMIT 500;

As a FinOps intern, begin the Visibility Phase and derive insights on the account:

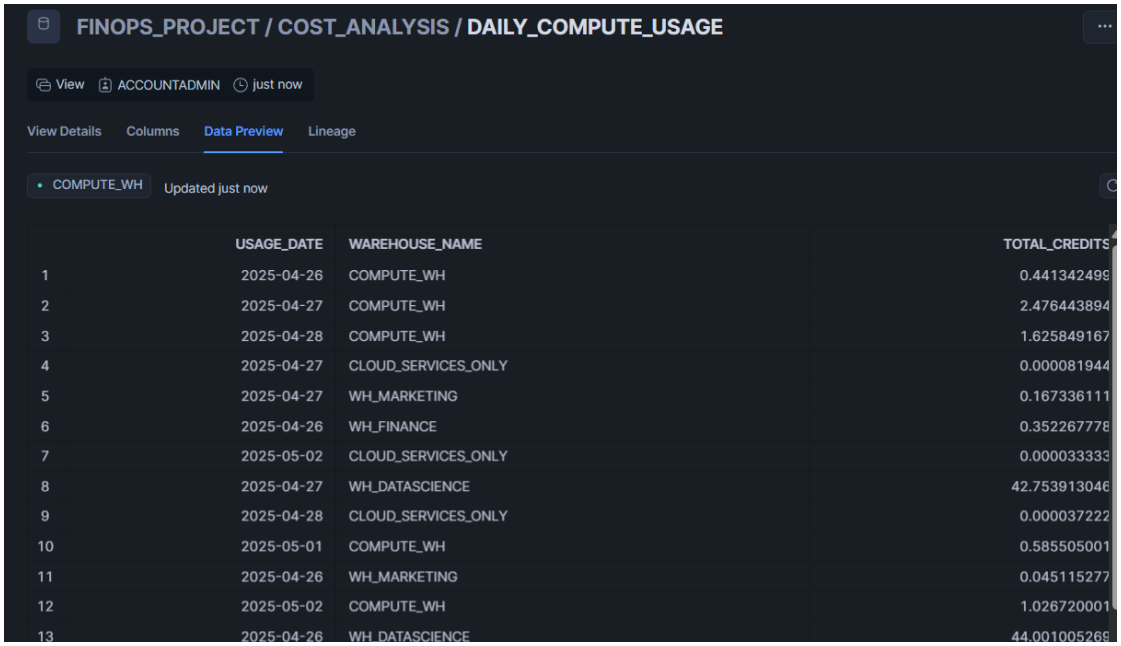
Cost Management View:



Step 4A: Daily Warehouse Compute Usage (Credits Spent)

CREATE OR REPLACE VIEW COST_ANALYSIS.DAILY_COMPUTE_USAGE AS SELECT START_TIME::DATE AS USAGE_DATE, WAREHOUSE_NAME, SUM(CREDITS_USED) AS TOTAL_CREDITS FROM SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY WHERE START_TIME > CURRENT_DATE - INTERVAL '30 DAY' GROUP BY 1, 2;

Output:



FINOPS_PROJECT / COST_ANALYSIS / DAILY_COMPUTE_USAGE

View ACCOUNTADMIN just now

View Details | Columns | Data Preview | Lineage

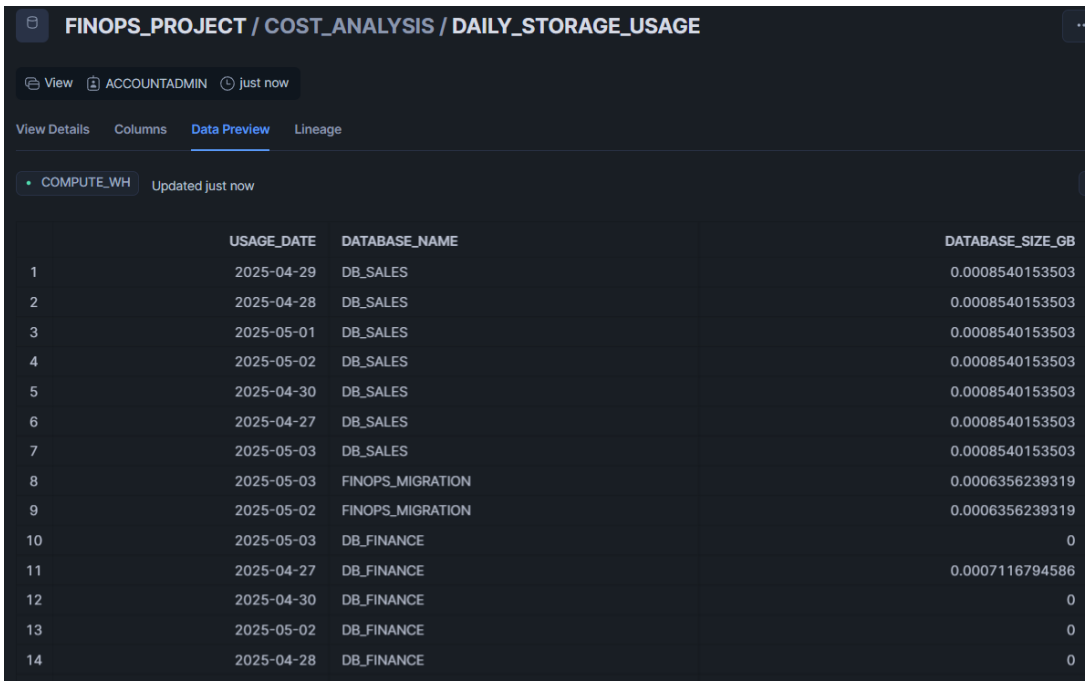
COMPUTE_WH Updated just now

	USAGE_DATE	WAREHOUSE_NAME	TOTAL_CREDITS
1	2025-04-26	COMPUTE_WH	0.441342499
2	2025-04-27	COMPUTE_WH	2.476443894
3	2025-04-28	COMPUTE_WH	1.625849167
4	2025-04-27	CLOUD_SERVICES_ONLY	0.000081944
5	2025-04-27	WH_MARKETING	0.167336111
6	2025-04-26	WH_FINANCE	0.352267778
7	2025-05-02	CLOUD_SERVICES_ONLY	0.000033333
8	2025-04-27	WH_DATASCIENCE	42.753913046
9	2025-04-28	CLOUD_SERVICES_ONLY	0.000037222
10	2025-05-01	COMPUTE_WH	0.585505001
11	2025-04-26	WH_MARKETING	0.045115277
12	2025-05-02	COMPUTE_WH	1.026720001
13	2025-04-26	WH_DATASCIENCE	44.001005269

Step 4B: **Daily Storage Usage (Database Size)**

CREATE OR REPLACE VIEW COST_ANALYSIS.DAILY_STORAGE_USAGE AS SELECT USAGE_DATE, DATABASE_NAME, AVERAGE_DATABASE_BYTES / POWER(1024, 3) AS DATABASE_SIZE_GB FROM SNOWFLAKE.ACCOUNT_USAGE.DATABASE_STORAGE_USAGE_HISTORY WHERE USAGE_DATE > CURRENT_DATE - INTERVAL '30 DAY';

Output:



FINOPS_PROJECT / COST_ANALYSIS / DAILY_STORAGE_USAGE

View ACCOUNTADMIN just now

View Details | Columns | Data Preview | Lineage

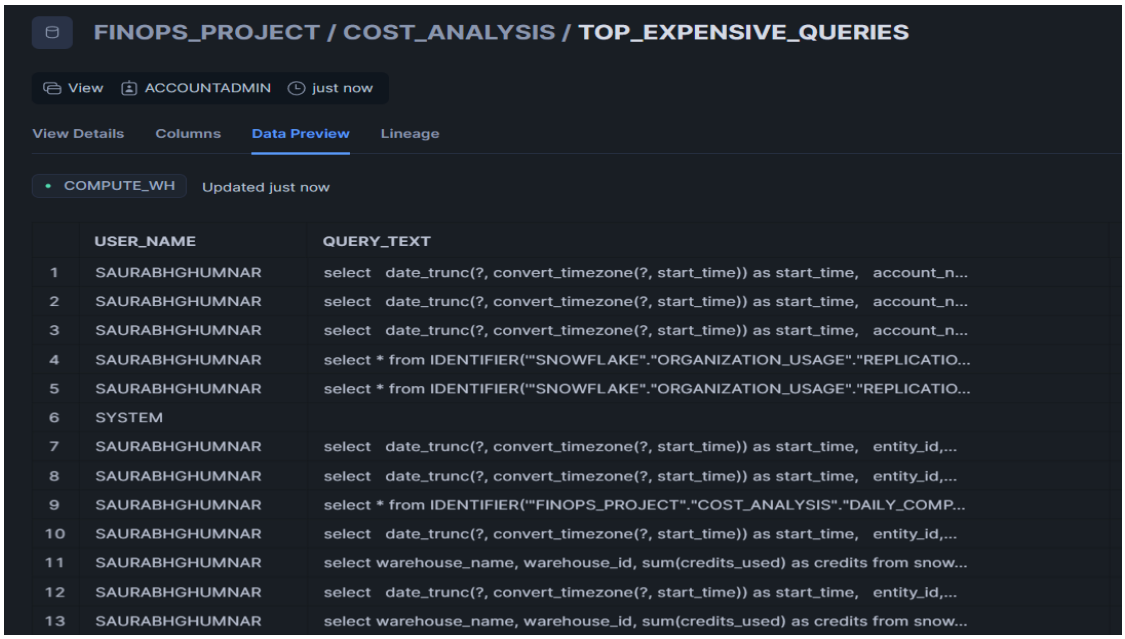
COMPUTE_WH Updated just now

	USAGE_DATE	DATABASE_NAME	DATABASE_SIZE_GB
1	2025-04-29	DB_SALES	0.0008540153503
2	2025-04-28	DB_SALES	0.0008540153503
3	2025-05-01	DB_SALES	0.0008540153503
4	2025-05-02	DB_SALES	0.0008540153503
5	2025-04-30	DB_SALES	0.0008540153503
6	2025-04-27	DB_SALES	0.0008540153503
7	2025-05-03	DB_SALES	0.0008540153503
8	2025-05-03	FINOPS_MIGRATION	0.0006356239319
9	2025-05-02	FINOPS_MIGRATION	0.0006356239319
10	2025-05-03	DB_FINANCE	0
11	2025-04-27	DB_FINANCE	0.0007116794586
12	2025-04-30	DB_FINANCE	0
13	2025-05-02	DB_FINANCE	0
14	2025-04-28	DB_FINANCE	0

Step 4C: **Top Expensive Queries (by Data Scanned)**

CREATE OR REPLACE VIEW COST_ANALYSIS.TOP_EXPENSIVE_QUERIES AS SELECT USER_NAME, QUERY_TEXT, TOTAL_ELAPSED_TIME/1000 AS TOTAL_TIME_SECONDS, BYTES_SCANNED/POWER(1024,3) AS DATA_SCANNED_GB FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY WHERE START_TIME > CURRENT_DATE - INTERVAL '30 DAY' ORDER BY DATA_SCANNED_GB DESC LIMIT 50;

Output:

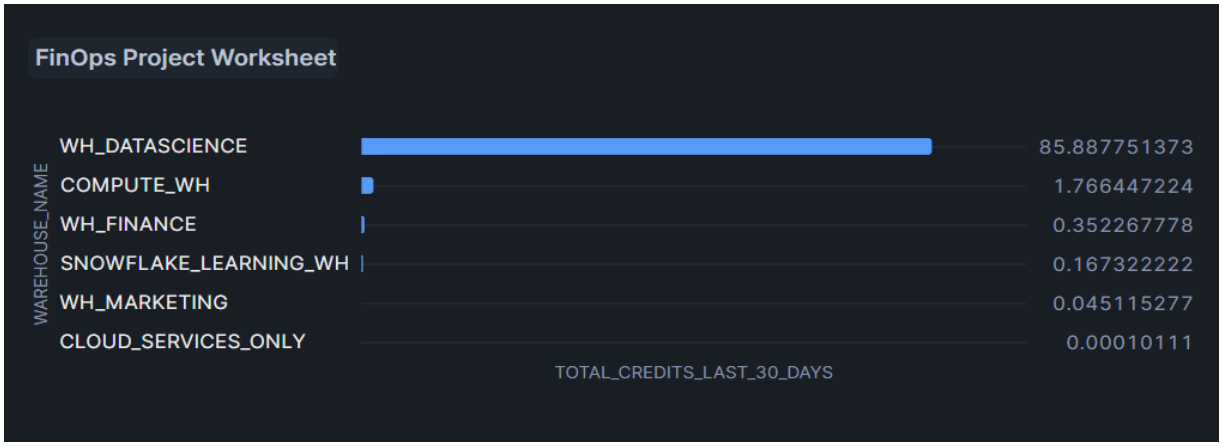


	USER_NAME	QUERY_TEXT
1	SAURABHGHUMNAR	select date_trunc(?, convert_timezone(?, start_time)) as start_time, account_n...
2	SAURABHGHUMNAR	select date_trunc(?, convert_timezone(?, start_time)) as start_time, account_n...
3	SAURABHGHUMNAR	select date_trunc(?, convert_timezone(?, start_time)) as start_time, account_n...
4	SAURABHGHUMNAR	select * from IDENTIFIER('"SNOWFLAKE"."ORGANIZATION_USAGE"."REPLICATIO...
5	SAURABHGHUMNAR	select * from IDENTIFIER('"SNOWFLAKE"."ORGANIZATION_USAGE"."REPLICATIO...
6	SYSTEM	
7	SAURABHGHUMNAR	select date_trunc(?, convert_timezone(?, start_time)) as start_time, entity_id,...
8	SAURABHGHUMNAR	select date_trunc(?, convert_timezone(?, start_time)) as start_time, entity_id,...
9	SAURABHGHUMNAR	select * from IDENTIFIER('"FINOPS_PROJECT"."COST_ANALYSIS"."DAILY_COMP...
10	SAURABHGHUMNAR	select date_trunc(?, convert_timezone(?, start_time)) as start_time, entity_id,...
11	SAURABHGHUMNAR	select warehouse_name, warehouse_id, sum(credits_used) as credits from snow...
12	SAURABHGHUMNAR	select date_trunc(?, convert_timezone(?, start_time)) as start_time, entity_id,...
13	SAURABHGHUMNAR	select warehouse_name, warehouse_id, sum(credits_used) as credits from snow...

Step 5A. Compute Spend by Warehouse

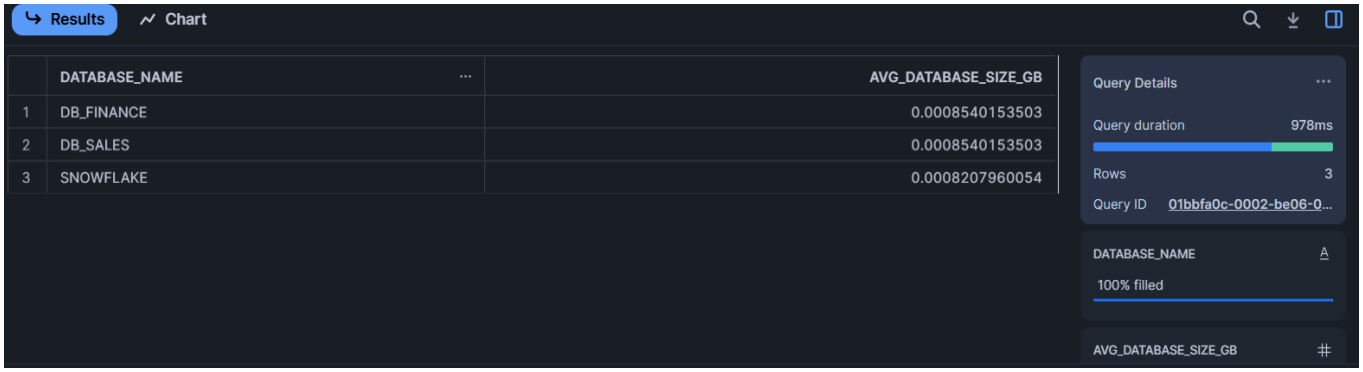
SELECT WAREHOUSE_NAME, SUM(TOTAL_CREDITS) AS TOTAL_CREDITS_LAST_30_DAYS FROM COST_ANALYSIS.DAILY_COMPUTE_USAGE GROUP BY WAREHOUSE_NAME ORDER BY TOTAL_CREDITS_LAST_30_DAYS DESC;

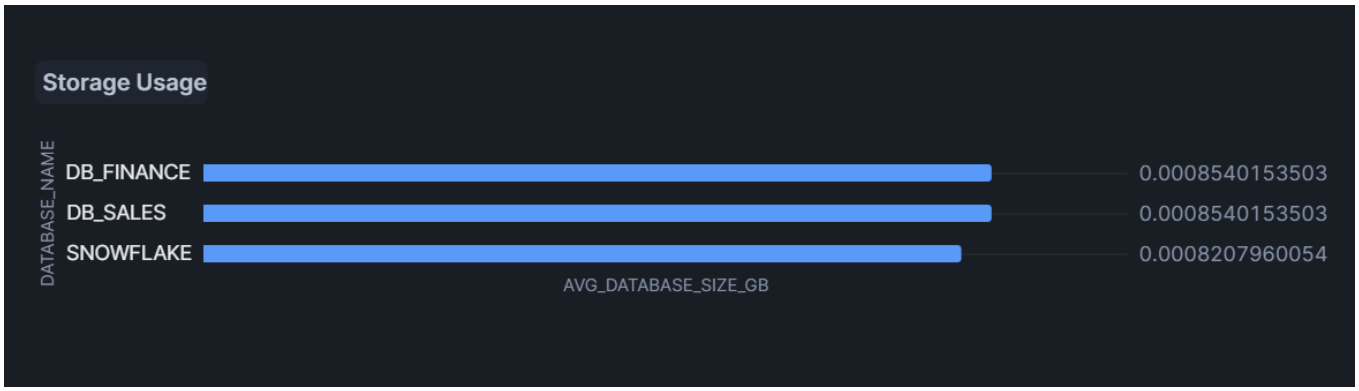
	WAREHOUSE_NAME	TOTAL_CREDITS_LAST_30_DAYS
1	WH_DATASCIENCE	85.887751373
2	COMPUTE_WH	1.766447224
3	WH_FINANCE	0.352267778
4	SNOWFLAKE_LEARNING_WH	0.167322222
5	WH_MARKETING	0.045115277
6	CLOUD_SERVICES_ONLY	0.000101110



Step 5B: Analyze Storage Usage

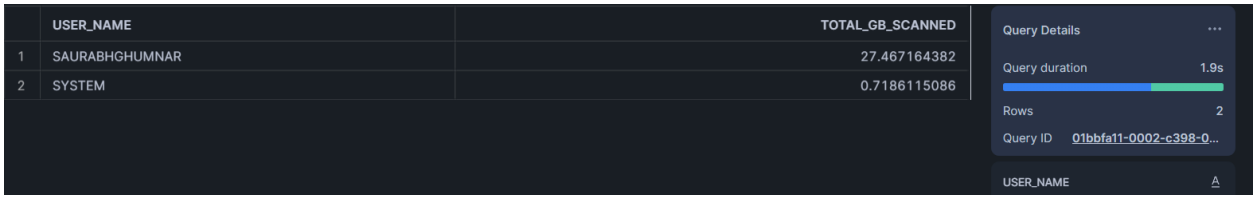
SELECT DATABASE_NAME, AVG(DATABASE_SIZE_GB) AS AVG_DATABASE_SIZE_GB FROM COST_ANALYSIS.DAILY_STORAGE_USAGE GROUP BY DATABASE_NAME ORDER BY AVG_DATABASE_SIZE_GB DESC;





Step 5C: Analyze the Top Heavy Query Users

SELECT USER_NAME, SUM(DATA_SCANNED_GB) AS TOTAL_GB_SCANNED FROM COST_ANALYSIS.TOP_EXPENSIVE_QUERIES GROUP BY USER_NAME ORDER BY TOTAL_GB_SCANNED DESC;



	USER_NAME	TOTAL_GB_SCANNED
1	SAURABHGHUMNAR	27.467164382
2	SYSTEM	0.7186115086

Query Details
Query duration 1.9s
Rows 2
Query ID 01bbfa11-0002-c398-0...
USER_NAME

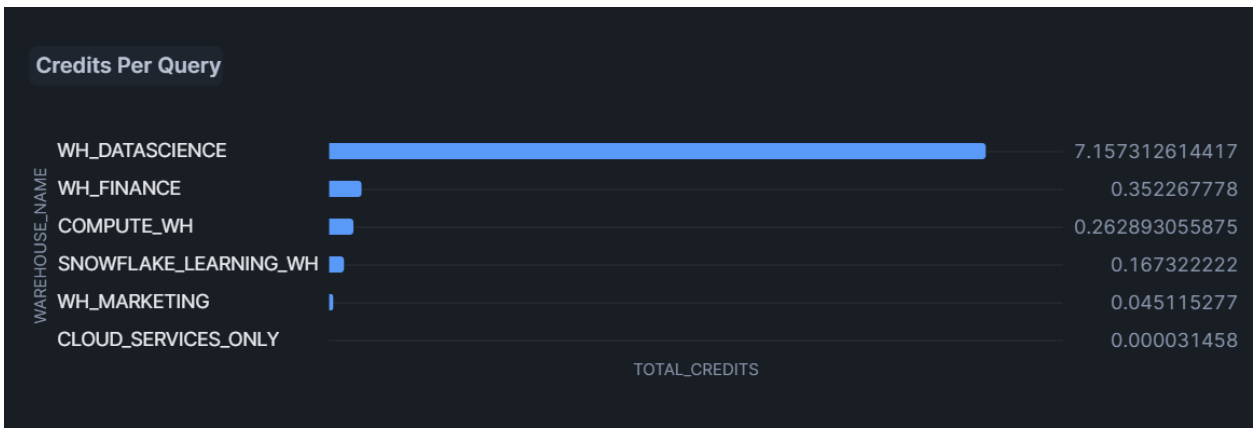
Perform Unit Economics Benchmarking:

Step 6A: Compute Credits Per Query

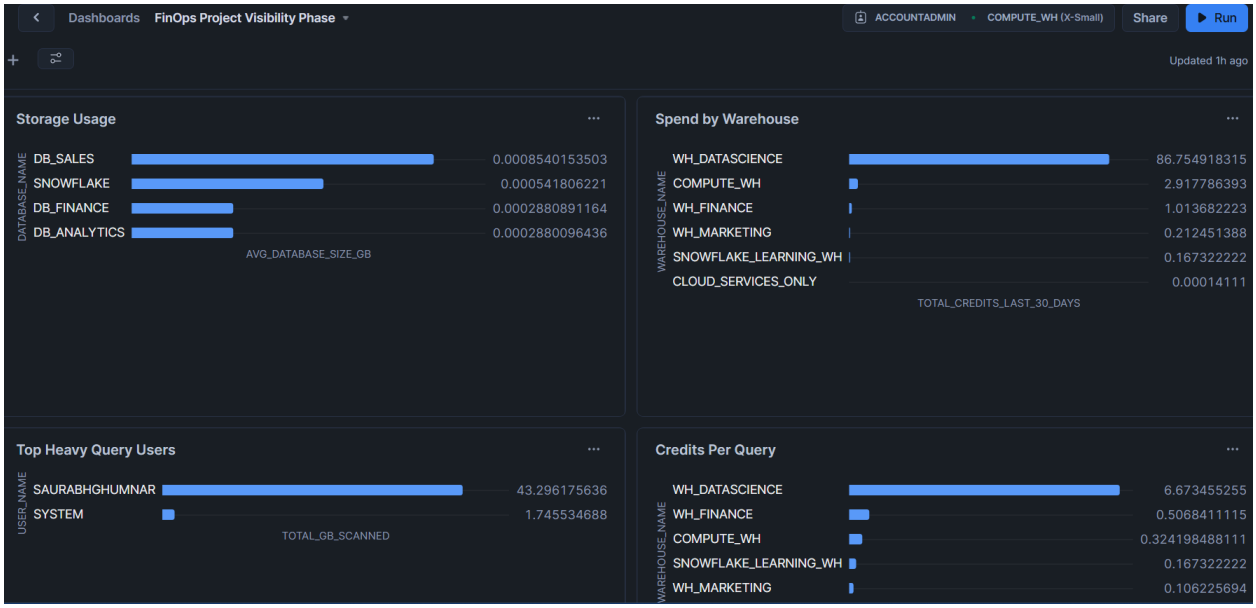
SELECT WAREHOUSE_NAME, SUM(CREDITS_USED) AS TOTAL_CREDITS, SUM(CREDITS_USED) / COUNT(*) AS CREDITS_PER_QUERY FROM SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY WHERE START_TIME > CURRENT_DATE - INTERVAL '30 DAY' GROUP BY WAREHOUSE_NAME ORDER BY CREDITS_PER_QUERY DESC;

	WAREHOUSE_NAME	TOTAL_CREDITS	CREDITS_PER_QUERY
1	WH_DATASCIENCE	85.887751373	7.157312614417
2	WH_FINANCE	0.352267778	0.352267778000
3	COMPUTE_WH	2.103144447	0.262893055875
4	SNOWFLAKE_LEARNING_WH	0.167322222	0.167322222000
5	WH_MARKETING	0.045115277	0.045115277000
6	CLOUD_SERVICES_ONLY	0.000125832	0.000031458000

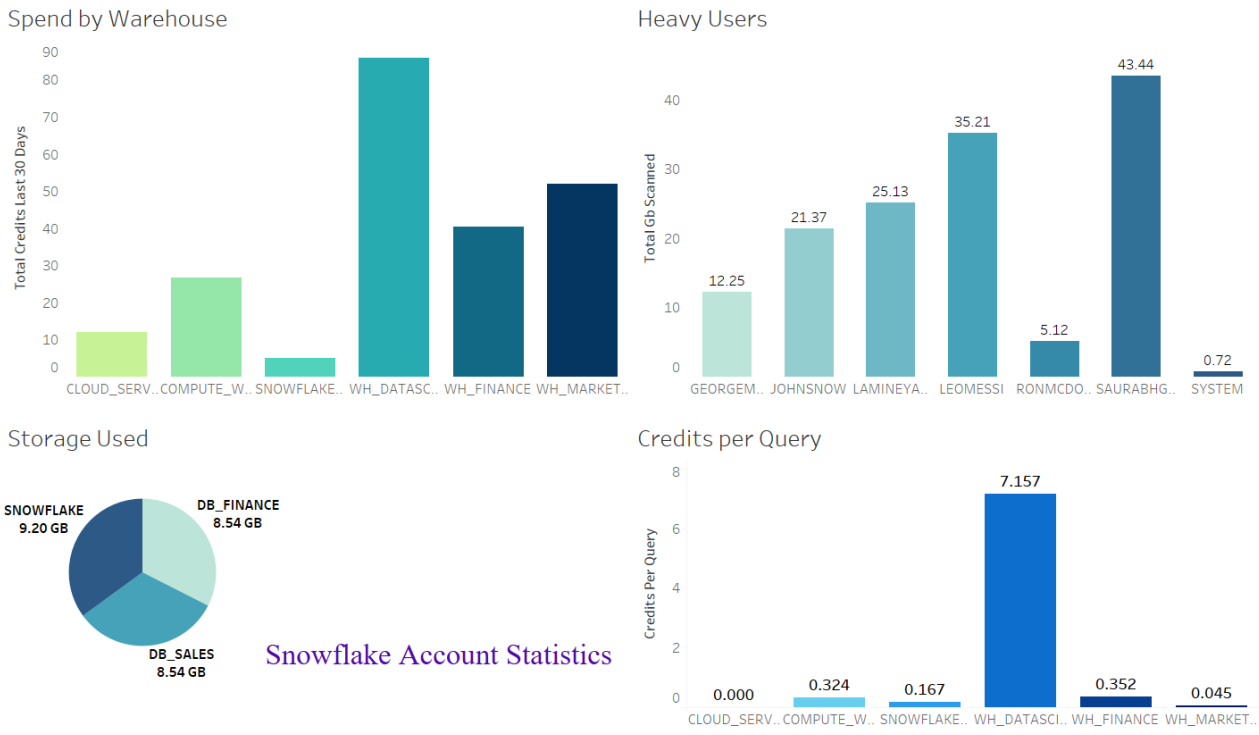
Query Details
Query duration 323ms
Rows 6
Query ID 01bbfa1f-0002-be06-0...
WAREHOUSE_NAME
100% filled



Now that we have the required usage and cost values, we create a dashboard to showcase the visibility to stakeholders:



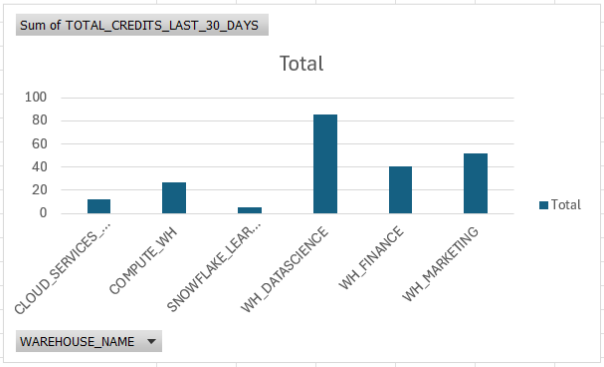
For better visuals, I also used Tableau to create a dashboard.



We can also create an Excel model to visualize and gain insights from the data. Some snapshots of my Excel are pasted below:

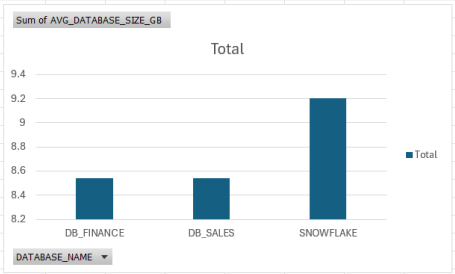
WAREHOUSE_NAME	TOTAL_CREDITS_LAST_30_DAYS
WH_DATASCIENCE	85.88775137
COMPUTE_WH	26.59441056
WH_FINANCE	40.35226778
SNOWFLAKE_LEARNING_WH	5.167322222
WH_MARKETING	52.04511528
CLOUD_SERVICES_ONLY	12.00012583

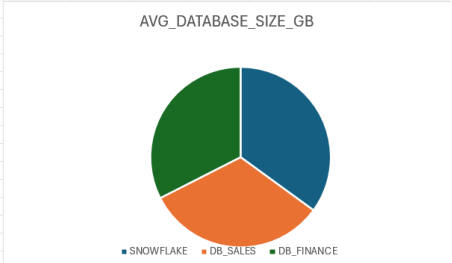
Row Labels	Sum of TOTAL_CREDITS_LAST_30_DAYS
CLOUD_SERVICES_ONLY	12.00012583
COMPUTE_WH	26.59441056
SNOWFLAKE_LEARNING_WH	5.167322222
WH_DATASCIENCE	85.88775137
WH_FINANCE	40.35226778
WH_MARKETING	52.04511528
Grand Total	**222.046993**



DATABASE_NA	AVG_DATABASE_SIZE_GB
SNOWFLAKE	9.204149246
DB_SALES	8.540153503
DB_FINANCE	8.540153503

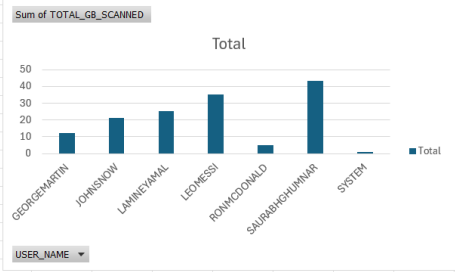
Row Labels	Sum of AVG_DATABASE_SIZE_GB
DB_FINANCE	8.540153503
DB_SALES	8.540153503
SNOWFLAKE	9.204149246
Grand Total	**26.28445625**

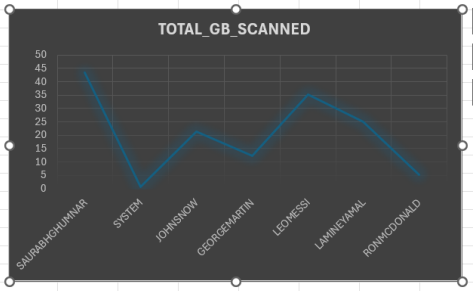




USER_NAME	TOTAL_GB_SCANNED
SAURABHGHUMNAR	43.44111203
SYSTEM	0.718611509
JOHNSNOW	21.36556986
GEORGEMARTIN	12.25469895
LEOMESSI	35.21458963
LAMINEYAMAL	25.12547896
RONMCDONALD	5.12478963

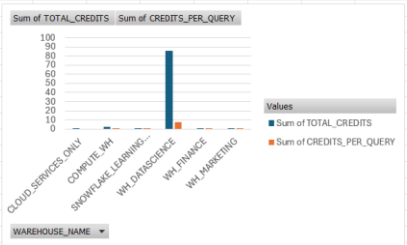
Row Labels	Sum of TOTAL_GB_SCANNED
GEORGEMARTIN	12.25469895
JOHNSNOW	21.36556986
LAMINEYAMAL	25.12547896
LEOMESSI	35.21458963
RONMCDONALD	5.12478963
SAURABHGHUMNAR	43.44111203
SYSTEM	0.718611509
Grand Total	**143.2448506**

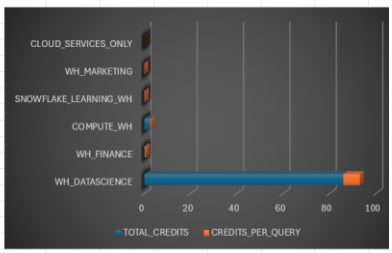




WAREHOUSE_NAME	TOTAL_CREDITS	CREDITS_PER_QUERY
WH_DATASCIENCE	85.88775137	7.157312614
WH_FINANCE	0.352267778	0.352267778
COMPUTE_WH	2.594410559	0.32430132
SNOWFLAKE_LEARNING_WH	0.167322222	0.167322222
WH_MARKETING	0.045115277	0.045115277
CLOUD_SERVICES_ONLY	0.000125832	0.000031458

Row Labels	Sum of TOTAL_CREDITS	Sum of CREDITS_PER_QUERY
CLOUD_SERVICES_ONLY	0.000125832	0.000031458
COMPUTE_WH	2.594410559	0.32430132
SNOWFLAKE_LEARNING_WH	0.167322222	0.167322222
WH_DATASCIENCE	85.88775137	7.157312614
WH_FINANCE	0.352267778	0.352267778
WH_MARKETING	0.045115277	0.045115277
Grand Total	**89.04699304**	**8.046350669**





Now that we have all the major insights into customers' account usage, we can present these visuals and explain to them their costs. Next comes the control phase, where we begin optimizing their environment for better resource utility and cost control.

Control Phase

Now we found in the Visibility Phase that WH_DATASCIENCE is large and expensive. This is because while creating, its size was mentioned as Large. We can change its size to medium by running the SQL command.

```
ALTER WAREHOUSE WH_DATASCIENCE SET WAREHOUSE_SIZE = 'MEDIUM';
```

This is called Rightsizing. "Based on usage analysis, WH_DATASCIENCE was downsized from LARGE to MEDIUM to optimize cost efficiency."

Maybe some warehouses were running idle.

Now, enforce **auto-suspend** for all warehouses:

```
ALTER WAREHOUSE WH_MARKETING SET AUTO_SUSPEND = 600; ALTER WAREHOUSE WH_FINANCE SET AUTO_SUSPEND = 600; ALTER WAREHOUSE WH_DATASCIENCE SET AUTO_SUSPEND = 600;
```

The warehouse automatically stops after 10 minutes of inactivity.

*"**Auto-suspend** policies were enforced across all compute warehouses to prevent idle credit wastage.*

From our visibility dashboard, we also know which users were running expensive queries.

```
SELECT
    USER_NAME,
    COUNT(*) AS NUM_EXPENSIVE_QUERIES,
    SUM(BYTES_SCANNED/POWER(1024,3)) AS TOTAL_DATA_SCANNED_GB
FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
WHERE START_TIME > CURRENT_DATE - INTERVAL '30 DAY'
  AND BYTES_SCANNED > 5000000000  -- Queries scanning >5 GB
GROUP BY USER_NAME
ORDER BY TOTAL_DATA_SCANNED_GB DESC;
```

Action**: Generate a report to educate and control user behavior.**

"Users generating large, inefficient queries were identified for targeted query optimization training, promoting cost-aware development."

We can also implement chargeback and showback models for better financial accountability. After completing the Visibility Phase, we can implement Showback models to attribute Snowflake compute and storage costs back to business units using mapping tables and cost calculations. We can also follow the Chargeback model by generating billing tables based on team-wise resource consumption, aligning internal accountability with FinOps best practices

---

Optimization Phase

Following the Control Phase, the Optimization Phase focused on proactive and continuous improvement strategies to maximize Snowflake's cost efficiency. The goal was not only to control costs but to dynamically optimize resource utilization, anticipate future needs, and drive smarter financial and operational decisions.

Actions Implemented:

Dynamic Warehouse Rightsizing, Monitored 30-day warehouse credits-per-query ratios and recommended size adjustments based on real workload efficiency. Reduced compute costs by aligning warehouse sizes more closely with actual query demands.

Cold Data Storage Tiering: We identified tables not updated in the past 180 days and recommended archiving or moving to lower-cost storage tiers. By offloading cold, infrequently accessed data, we minimized active storage costs.

Query Optimization Recommendations, Analyzed heavy queries scanning >10GB of data and proposed partitioning, filter tightening, and materialized views for optimization. Reduced unnecessary data scanning, improving query performance, and lowering compute consumption.

Usage Forecasting for Budgeting, Built trend forecasts of daily compute credit usage over 30 days to predict future resource needs., Enabled proactive budget planning and early detection of usage anomalies.

Summary

By implementing rightsizing recommendations, optimizing heavy queries, tiering cold storage, and forecasting future usage, the Optimization Phase ensured that Snowflake resource consumption remained efficient, predictable, and financially sustainable. These strategies created a foundation for intelligent, automated FinOps operations that extend beyond reactive cost management into proactive optimization.