# **Unistore, Hybrid Tables, External Connectivity, and Dashboarding**

## **1. Introduction to Unistore**

Unistore is Snowflake’s cutting-edge innovation that bridges transactional and analytical workloads within a single platform. Traditionally, transactional (OLTP) and analytical (OLAP) systems were siloed — businesses had to manage operational databases separately from analytical warehouses. With Unistore, Snowflake enables the management of real-time, row-level operations without sacrificing the scale and flexibility of analytical queries.

### **Industry Example – Netflix**

Netflix handles millions of concurrent users providing feedback on shows, updating watch history, and pausing/resuming streams — all of which are transactional in nature. Using Snowflake’s Unistore, Netflix can instantly write user interaction logs into hybrid tables, while immediately making them queryable by recommendation algorithms running analytical queries.

## **2. Key Concepts of Unistore**

* **Unified Workloads**: Ability to manage transactional and analytical operations side by side.
* **Hybrid Tables**: New table type purpose-built for low-latency, row-based operations.
* **Transactional Guarantees**: Full ACID compliance within Snowflake's cloud-native architecture.
* **Low-latency Updates**: High-performance, record-level modifications suitable for real-time applications.

### **Industry Example – Uber**

To power its driver-rider matching logic, Uber requires access to rapidly changing geolocation and availability data. Hybrid tables allow low-latency updates from mobile devices while simultaneously supporting live dashboards and surge pricing algorithms.

## **3. Hybrid Tables vs. Traditional Snowflake Tables**

* **Traditional Tables** are optimized for OLAP workloads: large-scale scans, aggregations, and distributed computing.
* **Hybrid Tables** support row-based operations and are designed for high-speed inserts, updates, and deletes.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Traditional Table** | **Hybrid Table** |
| Best For | Analytics | Operational data |
| Latency | Moderate | Low |
| Indexing | None (uses pruning) | Row-level, transactionally indexed |
| Workload Type | OLAP | OLTP + OLAP |

### **Industry Example – Google**

In Google's internal data ops, traditional tables serve analytics (e.g., trend analysis on billions of search queries), while hybrid tables help power real-time alerting systems on configuration changes and policy enforcement.

## **4. Storage, Performance, and Use Case Differences**

### **Storage:**

* Traditional tables use immutable micro-partitions.
* Hybrid tables allow row-level indexing for fast lookup and updates.

### **Performance:**

* Analytical tables are faster for full scans, aggregations.
* Hybrid tables are faster for frequent inserts/updates/deletes.

### **Use Cases:**

* **Hybrid Tables**:
  + Session management (e.g., real-time user sessions in Airbnb)
  + Cart systems (e.g., Amazon real-time shopping carts)
  + Live telemetry (e.g., Tesla vehicle logs processed in Snowflake)

## **5. How Unistore Supports Transactional Workloads**

* **ACID transactions** are supported with commit and rollback.
* **Row-level isolation** ensures concurrent access doesn’t interfere.
* **Fast writes and point lookups** are handled through hybrid table architecture.
* **Integration with Streams and Tasks** enables downstream analytics on the transactional changes.

### **Industry Example – Amazon**

In managing its logistics operations, Amazon tracks millions of package statuses in real-time. Hybrid tables act as a live event store. Streams read from these tables to trigger fulfillment analytics, update delivery predictions, and notify systems.

## **6. Integration with Existing Services (Streams, Tasks, Time Travel)**

* **Streams** can capture change data (CDC) from hybrid tables.
* **Tasks** can automate actions on top of transactional changes.
* **Time Travel** (to a limited extent) still works on hybrid tables for short-term recovery.

### **Industry Example – Meta**

Meta processes platform events (likes, shares, real-time feature flags) and stores them into hybrid tables. Tasks execute downstream transformations into summary tables, while Streams power the near-real-time ML retraining system.

## **7. Usage Best Practices & Real-Time Examples**

* Avoid overusing hybrid tables for bulk analytics — they're built for row-wise access.
* Pair hybrid tables with Streams for change-driven design.
* Use indexing-aware filters in queries.

### **Industry Example – Apple**

Apple handles real-time event ingestion from iOS crash logs into hybrid tables. Engineers query specific device IDs or timeframes during incident analysis. Cluster-aware query design ensures low latency, even at scale.

## **8. External Connectivity (ODBC/JDBC, DSN Setup)**

* **ODBC** and **JDBC** connectors allow Snowflake to interface with BI/reporting tools, backend apps, and notebooks.
* **DSN Setup** enables easier access through client tools (Excel, Power BI, etc.).

### **Industry Example – Microsoft**

Microsoft integrates Power BI dashboards directly with Snowflake datasets using DSNs and ODBC drivers. Live connections empower product and marketing teams to run daily funnel reports on hybrid table data.

## **9. Query Optimization for Dashboards**

* Use materialized views and summary tables
* Minimize joins and subqueries
* Parameterized queries and caching for repeatable performance
* Avoid SELECT \* in dashboard queries

### **Industry Example – Airbnb**

Airbnb optimizes its internal dashboards by using pre-aggregated tables on booking data, minimizing dashboard loading time for executives. Hybrid tables store session-level logs, while dashboards only reference necessary columns.

## **10. Monitoring Query Performance for BI Workloads**

* Use Snowflake’s Query Profile tool to trace long-running queries
* Leverage warehouse and query history for cost and latency tracking
* Tune warehouse sizing and suspend policies

### **Industry Example – Spotify**

Spotify uses query performance insights to maintain stable dashboard experiences during peak usage (e.g., Wrapped campaign). Warehouse scaling policies ensure Spotify for Artists dashboards remain responsive in high-traffic periods.

## **Hands-On Tasks (Listed Separately)**

1. Create and compare a hybrid table vs traditional table
2. Simulate a transactional workload with Unistore and Streams
3. Set up an ODBC connection and visualize data in Power BI
4. Analyze and optimize a dashboard query
5. Monitor dashboard performance using Query Profile and history views

### **Guided Lab : Create and Compare a Hybrid Table vs Traditional Table**

#### **🧠 Objective:** Understand the structural and performance differences between a traditional table and a hybrid table in Snowflake by creating both, inserting data, and running targeted queries.

### **Step-by-Step Lab Guide**

#### **🔧 Prerequisites:**

* Snowflake account (any edition with Unistore/Hybrid Table support — currently in Enterprise or higher)
* Role with SYSADMIN or custom role with create privileges
* Use the Snowflake Web UI or SnowSQL CLI

### **🪜 Step 1: Create a Database and Schema**

CREATE OR REPLACE DATABASE lab\_unistore\_demo;

USE DATABASE lab\_unistore\_demo;

CREATE OR REPLACE SCHEMA hybrid\_vs\_traditional;

USE SCHEMA hybrid\_vs\_traditional;

### **🧱 Step 2: Create a Traditional Table**

Traditional tables in Snowflake are optimized for analytical (OLAP) queries.

CREATE OR REPLACE TABLE user\_activity\_traditional (

user\_id STRING,

event\_type STRING,

device STRING,

event\_timestamp TIMESTAMP\_LTZ

);

### **🌐 Step 3: Create a Hybrid Table**

Hybrid tables are row-store, indexed, and support fast transactional reads/writes.

CREATE OR REPLACE HYBRID TABLE user\_activity\_hybrid (

user\_id STRING,

event\_type STRING,

device STRING,

event\_timestamp TIMESTAMP\_LTZ

);

🔍 **Note**: If you get a syntax error here, check whether your Snowflake edition supports Unistore. You may need to request access or use a supported trial.

### **📥 Step 4: Insert Sample Data into Both Tables**

-- Insert into traditional

INSERT INTO user\_activity\_traditional VALUES

('U1', 'click', 'mobile', CURRENT\_TIMESTAMP),

('U2', 'scroll', 'desktop', CURRENT\_TIMESTAMP),

('U3', 'search', 'tablet', CURRENT\_TIMESTAMP);

-- Insert into hybrid

INSERT INTO user\_activity\_hybrid VALUES

('U1', 'click', 'mobile', CURRENT\_TIMESTAMP),

('U2', 'scroll', 'desktop', CURRENT\_TIMESTAMP),

('U3', 'search', 'tablet', CURRENT\_TIMESTAMP);

### **🔍 Step 5: Run Analytical Queries**

Run aggregate queries on both tables to compare performance.

-- On traditional table

SELECT event\_type, COUNT(\*) AS event\_count

FROM user\_activity\_traditional

GROUP BY event\_type;

-- On hybrid table

SELECT event\_type, COUNT(\*) AS event\_count

FROM user\_activity\_hybrid

GROUP BY event\_type;

Observe response time and performance differences in the **Query Profile tab**.

### **⚡ Step 6: Run Point Lookup and Update Queries**

These simulate transactional workloads.

-- Point lookup (faster in hybrid)

SELECT \* FROM user\_activity\_hybrid WHERE user\_id = 'U2';

-- Update a record

UPDATE user\_activity\_hybrid SET device = 'smart\_tv'

WHERE user\_id = 'U2';

Repeat the same on the traditional table:

SELECT \* FROM user\_activity\_traditional WHERE user\_id = 'U2';

UPDATE user\_activity\_traditional SET device = 'smart\_tv'

WHERE user\_id = 'U2';

Compare execution time and latency in both.

### **📊 Step 7: Analyze Query Profiles**

* Go to the **History** tab in the Snowflake UI.
* Open each query’s **Query Profile**.
* Examine:
  + Scan type (Full vs Indexed)
  + Duration
  + Data scanned vs rows returned
  + Caching impact

💡 Hybrid tables use row indexing. For point lookups or small updates, they should outperform traditional tables.

### **🧾 Step 8: Summary Table Comparison**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Traditional Table** | **Hybrid Table** |
| Query Latency (Point) | Higher | Lower |
| Query Latency (Aggregate) | Lower | Slightly Higher |
| Write Latency | Moderate | Very Low |
| Optimized For | OLAP | OLTP/Hybrid |

### **✅ Conclusion**

You’ve now seen firsthand:

* How to create both table types
* Performance differences in analytical vs transactional scenarios
* How real-time updates are handled

This mirrors real-world usage at companies like **Amazon (cart updates), Uber (real-time GPS pings), and Netflix (session events)** where high-speed writes and immediate reads are critical.

### **Guided Lab: Simulate a Transactional Workload with Unistore and Streams**

#### **🧠 Objective:** Use a hybrid table in Snowflake to simulate a real-world transactional workload (e.g., ride-booking or cart updates), then track changes using a Snowflake Stream.

### **Step-by-Step Lab Guide**

#### **🔧 Prerequisites:**

* Snowflake Enterprise Edition or higher with Unistore support
* A role with SYSADMIN privileges or equivalent
* Familiarity with basic DML (INSERT/UPDATE/DELETE) operations

### **🪜 Step 1: Create a Database and Schema**

CREATE OR REPLACE DATABASE lab\_unistore\_streaming;

USE DATABASE lab\_unistore\_streaming;

CREATE OR REPLACE SCHEMA tx\_stream\_demo;

USE SCHEMA tx\_stream\_demo;

### **🧱 Step 2: Create a Hybrid Table to Simulate OLTP**

We'll use a ride booking simulation (like Uber).

CREATE OR REPLACE HYBRID TABLE ride\_transactions (

ride\_id STRING,

user\_id STRING,

driver\_id STRING,

pickup\_location STRING,

drop\_location STRING,

status STRING,

updated\_at TIMESTAMP\_LTZ

);

### **🌊 Step 3: Create a Stream on the Hybrid Table**

This will track all changes (INSERT, UPDATE, DELETE).

CREATE OR REPLACE STREAM ride\_txn\_stream ON TABLE ride\_transactions;

🧠 **Why?** Streams allow us to capture real-time changes and process them incrementally, a critical feature for CDC pipelines and micro-batch ETL.

### **🚕 Step 4: Simulate Inserts (New Rides Booked)**

INSERT INTO ride\_transactions VALUES

('R1001', 'U500', 'D100', 'MG Road', 'Airport', 'requested', CURRENT\_TIMESTAMP),

('R1002', 'U501', 'D101', 'Koramangala', 'Whitefield', 'requested', CURRENT\_TIMESTAMP);

### **🕒 Step 5: Simulate Updates (Driver Assigned or Ride Started)**

UPDATE ride\_transactions

SET status = 'driver\_assigned', updated\_at = CURRENT\_TIMESTAMP

WHERE ride\_id = 'R1001';

### **🗑️ Step 6: Simulate Cancellations (Deleted Rides)**

DELETE FROM ride\_transactions WHERE ride\_id = 'R1002';

### **🔎 Step 7: Query the Stream to See Captured Changes**

SELECT \* FROM ride\_txn\_stream;

Expected columns include:

* METADATA$ACTION (INSERT, UPDATE, DELETE)
* METADATA$ISUPDATE (TRUE/FALSE)
* All columns of the base table

🔬 **Observation Tip**: Each action (INSERT, UPDATE, DELETE) will be recorded, allowing downstream systems or tasks to process only new changes.

### **♻️ Step 8: Consume the Stream (Optional for Practice)**

If you'd like to process the stream and mark changes as consumed:

INSERT INTO some\_processing\_table

SELECT \* FROM ride\_txn\_stream;

This clears the stream backlog (you can’t re-query the same changes again unless Time Travel is used).

### **🧪 Step 9: Run More Transaction Scenarios**

Try different combinations:

-- Another insert

INSERT INTO ride\_transactions VALUES

('R1003', 'U502', 'D102', 'Indiranagar', 'Yelahanka', 'requested', CURRENT\_TIMESTAMP);

-- Ride completed

UPDATE ride\_transactions

SET status = 'completed', updated\_at = CURRENT\_TIMESTAMP

WHERE ride\_id = 'R1003';

-- View changes

SELECT \* FROM ride\_txn\_stream;

### **✅ Conclusion**

This simulation reflects how companies like **Uber or Lyft** manage real-time transactional data:

* Each ride triggers changes
* A Stream captures them in real time
* Downstream systems (billing, notifications, analytics) consume these changes

You’ve now:

* Created a transactional hybrid table
* Captured changes with a Stream
* Queried changes like an enterprise-grade CDC system

### **Guided Lab: Set up an ODBC Connection and Visualize Data in Power BI**

#### **🧠 Objective:**

Establish a live ODBC connection from Power BI to Snowflake, import data from a hybrid or traditional table, and create an interactive dashboard to visualize real-time transactional or analytical data.

### **Step-by-Step Lab Guide**

#### **🔧 Prerequisites:**

* Snowflake account with access to an active warehouse
* Power BI Desktop installed (Windows only)
* ODBC Driver for Snowflake installed (version 2.25.4 or later)
* Snowflake credentials (username/password)

🧰 **Downloads:**

* Snowflake ODBC Driver (Windows)
* [Power BI Desktop](https://powerbi.microsoft.com/desktop)

### **🪜 Step 1: Install the ODBC Driver**

1. Go to the Snowflake ODBC driver download page.
2. Choose the appropriate driver for your OS and install it.
3. After installation, open **ODBC Data Source Administrator (64-bit)** from Windows search.
4. Go to **System DSN** > Click **Add**.
5. Select **SnowflakeDSIIDriver** > Click **Finish**.

### **🔧 Step 2: Configure a DSN (Data Source Name)**

Fill in the following details in the ODBC setup window:

* **Data Source Name**: SNOWFLAKE\_LAB
* **User**: your Snowflake username
* **Server**: <your\_account>.snowflakecomputing.com (e.g., xyz12345.ap-south-1.snowflakecomputing.com)
* **Warehouse**: your virtual warehouse (e.g., COMPUTE\_WH)
* **Database**: lab\_unistore\_demo (or your target database)
* **Schema**: tx\_stream\_demo (or your schema)
* **Role**: SYSADMIN

Click **Test** to ensure the connection is successful.

### **📊 Step 3: Launch Power BI Desktop and Connect to Snowflake**

1. Open Power BI Desktop.
2. Click **Home** > **Get Data** > **More**.
3. In the **Get Data** window, search for **Snowflake** and select it.
4. Click **Connect**.

### **🔑 Step 4: Enter Snowflake Connection Details**

* **Server**: <your\_account>.snowflakecomputing.com
* **Warehouse**: e.g., COMPUTE\_WH
* **Database**: lab\_unistore\_demo
* **Schema**: tx\_stream\_demo

Click **OK**.

🧠 You can use either a DSN-less connection or point to the DSN you configured earlier.

### **📂 Step 5: Load Tables into Power BI**

1. Power BI will prompt a Navigator window.
2. Select the table (e.g., ride\_transactions or user\_activity\_hybrid).
3. Click **Load** to bring the data into Power BI.

### **📈 Step 6: Create Visualizations**

Now that your data is loaded:

1. Go to the **Report** view.
2. Drag fields like ride\_id, status, pickup\_location, updated\_at into a **table visual**.
3. Add a **bar chart** to show ride count per status.
4. Add **filters** (e.g., by status or time).
5. Add a **card visual** to show total number of rides.

🧪 Experiment with different visuals like stacked bar, donut charts, and line graphs to tell the story of the data.

### **📉 Step 7: Schedule Refresh (Optional)**

To ensure data stays current:

* You can use **DirectQuery** mode instead of Import.
* Or, configure **scheduled refresh** via Power BI Service (requires a gateway).

### 

### **✅ Conclusion**

You’ve now:

* Installed and configured Snowflake’s ODBC connector
* Connected Power BI to Snowflake
* Imported hybrid/analytical tables
* Created a dynamic dashboard reflecting real-time transactional data

This is exactly how companies like **Airbnb, Spotify, and Microsoft** build live dashboards for internal users to monitor sessions, orders, and system status.

💼 **Extension Idea**: Try visualizing stream results in Power BI to simulate near-real-time activity dashboards!

### **Lab Manual 4: Analyze and Optimize a Dashboard Query**

#### **🧠 Objective:** Understand how to analyze slow-running or high-cost dashboard queries in Snowflake and apply best practices for performance tuning, using real-time telemetry or transactional datasets.

### **Step-by-Step Lab Guide**

#### **🔧 Prerequisites:**

* Snowflake account with access to QUERY\_HISTORY and WAREHOUSE usage
* Power BI (or any BI tool) already connected to Snowflake
* Sample dashboard or report already built (can use from Lab 3)

🎯 **Goal**: Identify a query that's slow, optimize it using Snowflake tools and best practices, and validate performance improvement.

### **🧪 Step 1: Identify the Query from Dashboard History**

In Power BI, refresh a dashboard connected to a Snowflake dataset (e.g., ride\_transactions).

Now, in Snowflake Web UI:

SELECT \*

FROM TABLE(INFORMATION\_SCHEMA.QUERY\_HISTORY())

WHERE EXECUTION\_STATUS = 'SUCCESS'

AND USER\_NAME = CURRENT\_USER()

ORDER BY START\_TIME DESC

LIMIT 5;

🔍 Locate the dashboard query — often has large SELECT statements with aggregations or joins.

Copy the QUERY\_ID for deeper inspection.

### **🔍 Step 2: Use QUERY\_PROFILE for Deep Dive**

SELECT SYSTEM$QUERY\_PROFILE('your\_query\_id');

Then open the visual **Query Profile** panel in the Snowflake UI and observe:

* Total execution time
* Most expensive nodes (Scan, Join, Aggregation)
* Bytes scanned vs rows returned
* Whether result or metadata cache was used

### **⚙️ Step 3: Check for Common Issues**

Ask the following diagnostic questions:

* Are there unnecessary SELECT \* usages?
* Are filters applied **after** aggregations instead of before?
* Is there an expensive **JOIN** with no filter?
* Is there missing **pruning** (e.g., no partition or column filter)?
* Are complex **CTEs** repeated instead of reused?

### **💡 Step 4: Apply Optimizations**

Refactor the query with improvements. Here are common changes:

#### **❌ Original (Inefficient):**

SELECT \*

FROM ride\_transactions

WHERE status != 'completed';

#### **✅ Optimized:**

SELECT ride\_id, status, pickup\_location

FROM ride\_transactions

WHERE status != 'completed';

Other techniques:

* Use **materialized views** for pre-aggregation
* Replace **repeated subqueries/CTEs** with temp tables
* Add **filters early** in the query
* Use **SELECT specific columns** only
* Avoid functions on filtered columns (e.g., DATE(updated\_at) in WHERE clause)

### **📊 Step 5: Re-run and Compare Performance**

Use the optimized query manually in Snowflake first:

-- Original vs optimized timing

SELECT ... FROM ride\_transactions WHERE ...;

Then update the Power BI query to use the optimized form.

Use QUERY\_HISTORY again:

SELECT query\_text, execution\_time, rows\_scanned

FROM TABLE(INFORMATION\_SCHEMA.QUERY\_HISTORY())

WHERE user\_name = CURRENT\_USER()

ORDER BY start\_time DESC;

Compare the performance:

* **Execution time**
* **Bytes scanned**
* **Warehouse credits consumed**

### **🏁 Step 6: Save Optimization Learnings**

* Document slow queries and their causes
* Apply naming conventions to avoid SELECT \*
* Teach dashboard users about query cost awareness

🧠 Companies like **Netflix** and **Spotify** use internal dashboards daily for engineering and product metrics. Their BI teams regularly tune queries and establish governance policies to keep dashboards responsive and cost-efficient.

### **✅ Conclusion**

You’ve now:

* Identified heavy dashboard queries
* Analyzed performance using query history and profiling
* Refactored queries using Snowflake optimization principles
* Improved dashboard responsiveness

This reflects exactly how companies like **Amazon or Microsoft** keep their internal tools lean and responsive, even when serving thousands of employees.

### **Lab Manual 5: Monitor Dashboard Performance using Query Profile and History Views**

#### **🧠 Objective:**

Learn how to continuously monitor and troubleshoot dashboard performance using Snowflake’s built-in QUERY\_HISTORY, WAREHOUSE\_LOAD\_HISTORY, and Query Profile tools.

📊 This lab mimics how companies like Microsoft and Meta build BI governance by tracking cost-heavy queries and proactively resolving bottlenecks.

### **Step-by-Step Lab Guide**

#### **🔧 Prerequisites:**

* A Snowflake account with MONITOR privilege
* Power BI dashboard already built (e.g., Lab 3 or Lab 4)
* A virtual warehouse in use (e.g., COMPUTE\_WH)

### **🧪 Step 1: Refresh Your Dashboard to Trigger Queries**

In Power BI:

1. Open your existing Snowflake-connected dashboard.
2. Click **Refresh** to trigger a fresh set of queries.

Let this simulate a user accessing the dashboard in production.

### **🔍 Step 2: View Query History at User Level**

SELECT query\_text, execution\_status, start\_time, total\_elapsed\_time, warehouse\_size, rows\_produced

FROM TABLE(INFORMATION\_SCHEMA.QUERY\_HISTORY())

WHERE USER\_NAME = CURRENT\_USER()

AND START\_TIME > DATEADD(HOUR, -1, CURRENT\_TIMESTAMP)

ORDER BY START\_TIME DESC;

* Look for queries from Power BI (often large SELECT statements)
* Sort by total\_elapsed\_time to identify slowest queries

### **📉 Step 3: Visualize Warehouse Load Over Time**

SELECT start\_time, avg\_running, avg\_queued\_load, avg\_queued\_provisioning

FROM SNOWFLAKE.ACCOUNT\_USAGE.WAREHOUSE\_LOAD\_HISTORY

WHERE warehouse\_name = 'COMPUTE\_WH'

AND start\_time > DATEADD(HOUR, -2, CURRENT\_TIMESTAMP)

ORDER BY start\_time;

🔍 **Goal**: See if any query caused queuing, overload, or concurrency issues.

You may visualize this data in Excel or Power BI to understand load patterns.

### **📦 Step 4: Correlate Queries with Warehouse Performance**

1. Pick a slow query from QUERY\_HISTORY
2. Get its QUERY\_ID
3. Open its profile:

SELECT SYSTEM$QUERY\_PROFILE('your\_query\_id');

In the Snowflake UI, look for:

* Query Duration
* Join operations (nested loop vs hash)
* Scan type (Full vs Pruned)
* Whether result cache or metadata cache was hit

### **🛠 Step 5: Detect Repeated/Costly Queries**

SELECT query\_text, COUNT(\*) AS frequency, AVG(total\_elapsed\_time) AS avg\_runtime

FROM TABLE(INFORMATION\_SCHEMA.QUERY\_HISTORY())

WHERE USER\_NAME = CURRENT\_USER()

AND start\_time > DATEADD(DAY, -7, CURRENT\_TIMESTAMP)

GROUP BY query\_text

ORDER BY avg\_runtime DESC

LIMIT 10;

🎯 **Goal**: Spot which queries are most expensive or most repeated.

### **📌 Step 6: Set up Scheduled Monitoring Alerts (Optional)**

You can use:

* **Tasks** to run monitoring SQL every hour
* **Email/Slack integration** (via Snowflake alerts or external orchestration tools)

Examples of metrics to monitor:

* Queries longer than 60 seconds
* Warehouse credits consumed > 1
* Queuing time > 10 seconds

### **✅ Conclusion**

You’ve now:

* Monitored query performance from dashboards
* Diagnosed warehouse pressure via load history
* Correlated queries with expensive warehouse usage
* Identified patterns of costly or inefficient queries

🧠 This mimic real-world observability setups in companies like **Spotify** and **Airbnb**, where dashboard SLAs and cost governance are maintained rigorously.

HAPPY LEARNING

BEST REGARDS

SARANSH JAIN