# End-to-End Lakehouse Pipeline

* **Snowbear** Air's **mission-critical systems** manage a vast amount of flight data every minute.
* The company's enterprise application—affectionately known as the **Bear Traffic Controller**—pushes real-time flight information to cloud storage.
* In this lab, you will build a continuous pipeline that ingests, transforms, and refines this data into structured tables.
* By leveraging Snowflake external stages, Snowpipe, and dynamic Apache Iceberg™ tables, you will create a lakehouse architecture that:
  + A dedicated data engineer at Snowbear Air is tasked with monitoring on-time performance and improving customer satisfaction.
  + Given a huge data set from the Bear Traffic Controller, your pipeline will be essential for delivering insights that drive operational improvements.

**Starting Objects and States**

* You are provided with the following:
  + External Stage, External Volume, & Stored Procedure: An external stage, an external volume, and a stored procedure, stream\_flight\_csv\_data(...), that generates on-time flight performance data in CSV format.
  + Database Setup: A database named SQUIRREL\_de\_DB with three schemas:
    - RAW: Contains the untransformed, raw flight data (the “single source of truth”).
    - CONFORMED: Contains data after cleaning and standardization.
    - MODELED: Contains the final, structured tables that are ready for consumption by downstream workflows and dashboards.

**Setup**

|  |
| --- |
| USE ROLE de\_role;  ALTER SESSION SET QUERY\_TAG= '(SQUIRREL) Lab: Lakehouse End-to-End Pipeline'; |

**Next, run these commands to create and configure your compute resources and database context:**

|  |
| --- |
| CREATE WAREHOUSE IF NOT EXISTS SQUIRREL\_de\_wh;  ALTER WAREHOUSE SQUIRREL\_de\_wh SET  WAREHOUSE\_SIZE = 'XSMALL'  AUTO\_SUSPEND = 300  AUTO\_RESUME = TRUE;  USE WAREHOUSE SQUIRREL\_de\_wh;  CREATE DATABASE IF NOT EXISTS SQUIRREL\_de\_db;  USE SCHEMA SQUIRREL\_de\_db.raw; |

**Lab Sections (Optional Beyond Silver Layer)**

* After creating the "flights\_silver\_dt" dynamic table, the remaining sections of the lab are OPTIONAL.
* They show how to further transform and model data into specific flight categories—using dynamic Iceberg tables—in preparation for dashboards and reporting.
* This lab demonstrates a true lakehouse approach in which external Iceberg tables serve as the high-performance, pre-materialized layer for downstream analytics.
* Iceberg tables enable interoperability with other systems and offer near-real-time refresh capabilities.
* The subsequent sections include dynamic tables for: Each dynamic table refreshes its data automatically based on the defined target lag, ensuring that your business intelligence dashboards always query up-to-date data.

**Initially, this stage will be empty.**

A screenshot of a computer

AI-generated content may be incorrect.

**Generate flight data by calling the stored procedure:**

A screenshot of a computer

AI-generated content may be incorrect.

**The logic in this stored procedure that uploads to the stage can be shown here:**

select get\_ddl('procedure', 'raw.**stream\_csv\_flight\_data**(varchar, varchar, varchar, varchar)');

A white background with black and red dots

AI-generated content may be incorrect.

|  |
| --- |
| **Note:**  **Snowflake does not store or return the actual Python or JavaScript source code inside the procedure using get\_ddl()** if the procedure is a **Python or JavaScript stored procedure**. Instead, get\_ddl() returns only the *metadata* of the procedure (signature, language, runtime, etc.) but not the full source code inside the AS clause.  In other words:   * For SQL procedures, get\_ddl() returns the full SQL text. * For Python or JavaScript procedures, it returns only the metadata, **NOT** the code logic inside the handler.   This is a limitation of Snowflake’s metadata extraction for non-SQL stored procedures. |

|  |
| --- |
| **Probable Code from ChatGPT**  CREATE OR REPLACE PROCEDURE stream\_csv\_flight\_data(FOLDER VARCHAR, YEAR VARCHAR, MONTH VARCHAR, AIRPORT VARCHAR) RETURNS FLOAT LANGUAGE PYTHON RUNTIME\_VERSION = '3.9' PACKAGES = ('snowflake-snowpark-python', 'pandas') HANDLER = 'stream\_flight\_data' AS $$ import pandas as pd from io import StringIO    def stream\_flight\_data(session, FOLDER, YEAR, MONTH, AIRPORT):     # Compose the stage path     stage\_path = f"@raw.class\_stage/SQUIRREL/{FOLDER}/{YEAR}/{MONTH}/{AIRPORT}/flight\_data.csv"      # Example: load data from a source table (replace with your actual table name)     df = session.table("RAW.FLIGHT\_DATA") \                 .filter(                     (session.col("YEAR") == YEAR) &                      (session.col("MONTH") == MONTH) &                      (session.col("AIRPORT") == AIRPORT)                 ).to\_pandas()      if df.empty:         return 0.0  # No data to upload      # Convert DataFrame to CSV string     csv\_buffer = StringIO()     df.to\_csv(csv\_buffer, index=False)     csv\_content = csv\_buffer.getvalue()      # Upload CSV string to stage using Snowflake PUT command via Snowpark session     # Snowpark Python API does not support PUT directly, so we use Snowflake Python Connector     # Fetch connection info from session     conn = session.\_conn     cursor = conn.cursor()      try:         # Create a local temporary file for uploading         with open("/tmp/flight\_data.csv", "w") as f:             f.write(csv\_content)          # Put the file to stage path - overwrites existing file         put\_sql = f"PUT file:///tmp/flight\_data.csv {stage\_path} OVERWRITE = TRUE"         cursor.execute(put\_sql)          # Return number of rows uploaded         return float(len(df))      finally:         cursor.close()    $$;  **Explanation:**   * The procedure uses the Snowpark session to query the source data (RAW.FLIGHT\_DATA) filtered by year, month, and airport. * Converts the resulting data to a pandas DataFrame and then to CSV string. * Saves CSV content temporarily to /tmp/flight\_data.csv on the Snowflake compute node. * Uses PUT command to upload the file to the specified stage path. * Returns the number of rows uploaded.   **Important notes:**   * **Local file usage:** This example assumes that the Python environment running the procedure can write to the local /tmp directory, which Snowflake managed compute nodes generally allow. * **PUT command via Python connector:** Snowpark session doesn’t expose PUT directly, so this example uses the Snowflake Python connector cursor (session.\_conn.cursor()) to run the PUT command. * Adjust the source table name (RAW.FLIGHT\_DATA) to your actual flight data source. * Make sure the role executing this procedure has write access to the stage. |

Confirm that data has been generated on the external stage:

A screenshot of a computer

AI-generated content may be incorrect.

**Raw Data – Bronze Layer**

Next, we’ll load the raw flight data into an Apache Iceberg™ table. Then we’ll query the table’s contents and checks the row count.

|  |
| --- |
| CREATE OR REPLACE ICEBERG TABLE raw.flights\_bronze (  FLIGHT\_ID STRING,  TRAVEL\_DATE DATE,  STATUS STRING,  ORIGIN STRING,  DESTINATION STRING,  CANCELLATION\_REASON STRING  )  CATALOG='SNOWFLAKE'  EXTERNAL\_VOLUME='ext\_volume'  BASE\_LOCATION='SQUIRREL\_iceberg/flights/'  DATA\_RETENTION\_TIME\_IN\_DAYS=7  COMMENT='Iceberg table loads raw flight data.'; |

Verify the table was created with the following statement:

A screenshot of a computer

AI-generated content may be incorrect.

* This Iceberg table stores data in an external volume, ext\_volume (e.g. Amazon S3), and even though the data is external, Snowflake manages and queries it efficiently.
* With this Lakehouse design, the data will stay external to Snowflake to enable interoperability and storage savings.
* CATALOG=SNOWFLAKE: This specifies that the table is registered within the Snowflake catalog. In other words, Snowflake is managing the table metadata and its connection to external storage.
* EXTERNAL\_VOLUME=ext\_volume: This indicates the external storage volume where the table’s data files are stored. The external volume typically points to a cloud storage location (for example, an Amazon S3 bucket) that the Iceberg tables uses to persist data outside of Snowflake.
* \*\*BASE\_LOCATION=’SQUIRREL\_iceberg/flights/’:\*\* This defines the base path or directory within the external volume where the table’s files will reside. It organizes the data files for this particular table, making it easier to manage and locate the data.
* DATA\_RETENTION\_TIME\_IN\_DAYS=7: This setting specifies how long Snowflake retains historical data changes (for example, for time travel or change tracking) in this Iceberg table. In this case, changes are kept for 7 days.
* Now let’s enable the change tracking on the **flights\_bronze** table to ensure that the dynamic tables can detect and process only the changes made to the bronze table since the last refresh, rather than reprocessing the entire table each time.

ALTER ICEBERG TABLE raw.flights\_bronze SET CHANGE\_TRACKING = TRUE;

**Data Ingestion – File Format and Snowpipe**

This section creates a file format tells Snowflake how to parse CSV files in our external stage. Later, we’ll use this file format to load into the Apache Iceberg™ table via the pipe.

*Create a File Format*

|  |
| --- |
| CREATE OR REPLACE FILE FORMAT raw.my\_csv\_format  TYPE=CSV  FIELD\_DELIMITER=','  SKIP\_HEADER=0  FIELD\_OPTIONALLY\_ENCLOSED\_BY='"'  NULL\_IF=('')  EMPTY\_FIELD\_AS\_NULL=TRUE; |

A screenshot of a computer

AI-generated content may be incorrect.

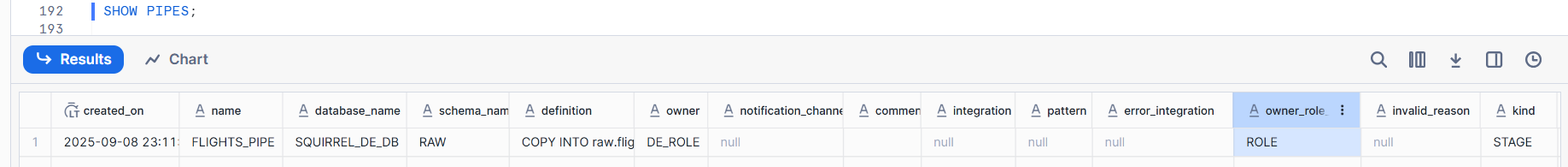
|  |
| --- |
| {"TYPE":"CSV","RECORD\_DELIMITER":"\n","FIELD\_DELIMITER":",","FILE\_EXTENSION":null,"SKIP\_HEADER":0,"PARSE\_HEADER":false,"DATE\_FORMAT":"AUTO","TIME\_FORMAT":"AUTO","TIMESTAMP\_FORMAT":"AUTO","BINARY\_FORMAT":"HEX","ESCAPE":"NONE","ESCAPE\_UNENCLOSED\_FIELD":"\\","TRIM\_SPACE":false,"FIELD\_OPTIONALLY\_ENCLOSED\_BY":"\"","NULL\_IF":[""],"COMPRESSION":"AUTO","ERROR\_ON\_COLUMN\_COUNT\_MISMATCH":true,"VALIDATE\_UTF8":true,"SKIP\_BLANK\_LINES":false,"REPLACE\_INVALID\_CHARACTERS":false,"EMPTY\_FIELD\_AS\_NULL":true,"SKIP\_BYTE\_ORDER\_MARK":true,"ENCODING":"UTF8","MULTI\_LINE":true} |
| {"TYPE":"CSV","RECORD\_DELIMITER":"\n","FIELD\_DELIMITER":",","FILE\_EXTENSION":null,"SKIP\_HEADER":1,"PARSE\_HEADER":false,"DATE\_FORMAT":"AUTO","TIME\_FORMAT":"AUTO","TIMESTAMP\_FORMAT":"AUTO","BINARY\_FORMAT":"HEX","ESCAPE":"NONE","ESCAPE\_UNENCLOSED\_FIELD":"\\","TRIM\_SPACE":false,"FIELD\_OPTIONALLY\_ENCLOSED\_BY":"\"","NULL\_IF":["\\N"],"COMPRESSION":"AUTO","ERROR\_ON\_COLUMN\_COUNT\_MISMATCH":false,"VALIDATE\_UTF8":true,"SKIP\_BLANK\_LINES":false,"REPLACE\_INVALID\_CHARACTERS":false,"EMPTY\_FIELD\_AS\_NULL":true,"SKIP\_BYTE\_ORDER\_MARK":true,"ENCODING":"UTF8","MULTI\_LINE":true} |
| {"TYPE":"PARQUET","TRIM\_SPACE":false,"NULL\_IF":[],"COMPRESSION":"AUTO","BINARY\_AS\_TEXT":true,"REPLACE\_INVALID\_CHARACTERS":false,"USE\_LOGICAL\_TYPE":false,"USE\_VECTORIZED\_SCANNER":false} |

**Create the Pipe to Load Data**

* In this section, this file format is used by a pipe that loads the parsed CSV data into the Iceberg table, flights\_bronze.
* When the data is loaded into an Iceberg table, Snowflake automatically converts and stores it in a highly optimized internal format (often leveraging Parquet), making it Iceberg-compatible.

A screen shot of a computer code

AI-generated content may be incorrect.



|  |
| --- |
| COPY INTO raw.flights\_bronze  FROM @raw.class\_stage/SQUIRREL/  FILE\_FORMAT = (FORMAT\_NAME = my\_csv\_format)  ON\_ERROR = 'CONTINUE' |

We can set up a serverless task to schedule the pipe ingestion process for production as shown here:

|  |
| --- |
| CREATE OR REPLACE TASK refresh\_flights\_pipe  SCHEDULE = 'USING CRON 5 \* \* \* \* America/Los\_Angeles'  AS  ALTER PIPE raw.flights\_pipe REFRESH; |

By default, the task is not activated on its schedule upon creation. To set this task on schedule, run the following command:

ALTER TASK refresh\_flights\_pipe RESUME;

* In this example, we are refreshing the pipe at five past the hour, every hour.
* A scheduled task can be quite costly since the warehouse is resumed regardless if the process is necessary.
* In production, an event driven approach is often preferred with AUTOINGEST = TRUE option. In this case, Snowpipe will use the cloud provider’s native event notifications (such as SNS or SQS for AWS, Azure Event Grid or GCPs Pub/Sub) to be notified when new data has landed in an external cloud storage resource.
* In each case, the external storage is “pushing” events to Snowflake, which then triggers the pipe to ingest the new files automatically.
* In order to validate that the new data is being loaded into our Iceberg table let’s perform a manual refresh for the sake of this lab's time restraints.

**Refresh the pipe to manually load any staged files into the raw.flights\_bronze table:**

ALTER PIPE raw.flights\_pipe REFRESH;

A white rectangular object with black lines

AI-generated content may be incorrect.

**Check the pipe status with the system function:**

Checking status without Refresh

**A screenshot of a computer

AI-generated content may be incorrect.**

Checking status after ALTER.. Refresh

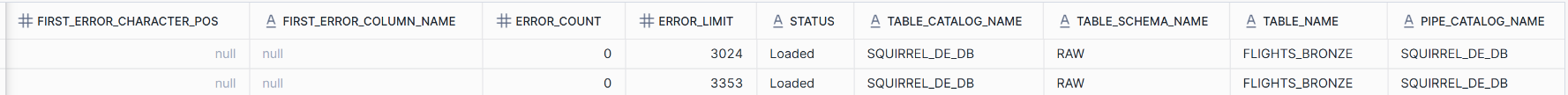
**A screenshot of a computer

AI-generated content may be incorrect.**

**Review the table history to see load details:**

If there are no results yet, please be patient. Depending on the number of files being loaded, this could take a minute or two.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

Finally, query the raw flights table:

A screenshot of a computer screen

AI-generated content may be incorrect.

**Transform Raw Data into Conformed Data Using Dynamic Tables**

In this section, we create a dynamic Apache Iceberg™ table named **flights\_silver\_dt** to clean and standardize the raw flight data from the bronze layer. We will place it in the CONFORMED schema to separate refined and raw data.

The transformation performs the following operations:

1. Standardizing Flight IDs: Flight IDs are trimmed to remove extraneous spaces and converted to uppercase, ensuring consistent formatting across the dataset.
2. Normalizing Flight Status: A CASE expression is used to standardize the flight status. This converts various textual representations (such as cancelled, delayed, early, and on time) into a consistent set of values: CANCELLED, DELAYED, EARLY, and ON\_TIME. Any unexpected value is set to UNKNOWN.
3. Cleaning Cancellation Reasons: The transformation replaces empty or whitespace-only CANCELLATION\_REASON values with NULL. This step is crucial to prevent errors in downstream processing, such as when using classification functions.
4. Enriching Data with Airport Information: The raw flight data is joined with the conformed.dim\_airports table to add corresponding city names for each airport. This enrichment makes the data morehuman-readable and useful for reporting.

* Dynamic tables continuously refresh on a predefined schedule (in this lab, every 5 minutes) to provide a near-real-time, materialized view of the data.
* This approach is particularly beneficial for data engineers who require pre-computed, consistent data for dashboards and business intelligence reports without the overhead of recalculating transformations at query time.

A screenshot of a computer

AI-generated content may be incorrect.

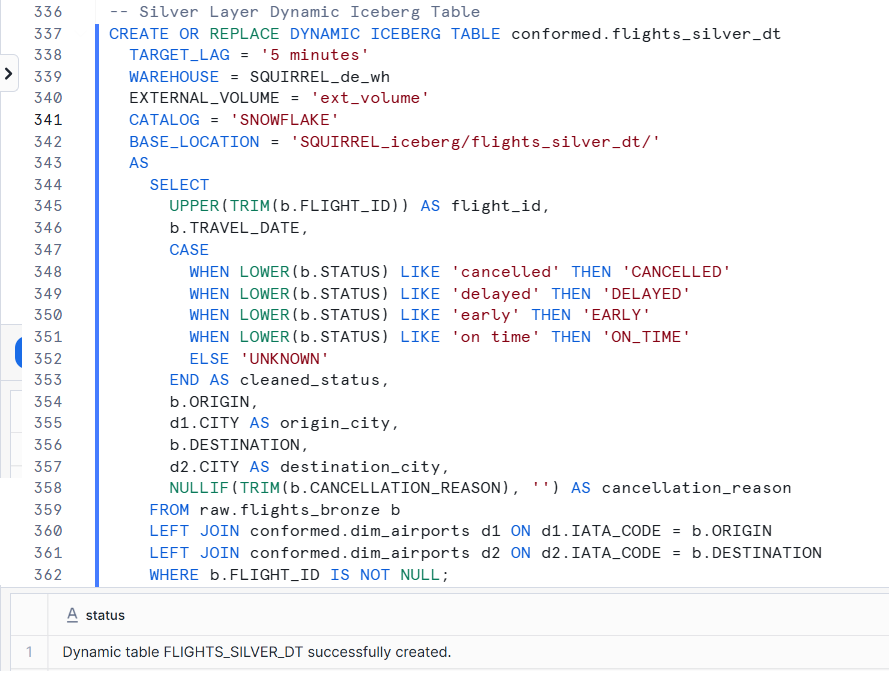
Now let’s add IATA code data into this dimension table.

|  |
| --- |
| INSERT INTO conformed.dim\_airports VALUES  ('SEA','Seattle','WA','USA'),  ('SFO','San Francisco','CA','USA'),  ('ORD','Chicago','IL','USA'),  ('JFK','New York','NY','USA'),  ('LAX','Los Angeles','CA','USA'),  ('ATL','Atlanta','GA','USA'),  ('DEN','Denver','CO','USA'),  ('BOS','Boston','MA','USA'),  ('IAH','Houston','TX','USA'),  ('SAN','San Diego','CA','USA'),  ('DTW','Detroit','MI','USA'),  ('DFW','Dallas/Fort Worth','TX','USA'),  ('SLC','Salt Lake City','UT','USA'),  ('DCA','Washington DC','','USA'),  ('OAK','Oakland','CA','USA'),  ('PDX','Portland','OR','USA'),  ('SLC','Salt Lake City','UT','USA'),  ('MSP','Minneapolis','MN','USA'),  ('SNA','Santa Ana','CA','USA'),  ('LAS','Las Vegas','NV','USA'),  ('MCO','Orlando','FL','USA'),  ('FLL','Fort Lauderdale','FL','USA'),  ('TPA','Tampa','FL','USA'),  ('MIA','Miami','FL','USA'),  ('SJU','San Juan','PR','USA'),  ('PBI','West Palm Beach','FL','USA'),  ('MSY','New Orleans','LA','USA'),  ('AUS','Austin','TX','USA'),  ('PHX','Phoenix','AZ','USA')  ; |

A screenshot of a computer

AI-generated content may be incorrect.

* In this step, we create a dynamic Iceberg table in our conformed schema that serves as the silver layer for our flight data.
* This table continuously refreshes itself every 5 minutes, ensuring that any new or updated records from our raw flights bronze table are automatically captured.
* By leveraging the dynamic table capability, we simplify our data pipeline and provide near real-time data for downstream analytics. Notice that the table’s schema is automatically inferred from the SELECT clause.



The silver layer is the basis for further transformations. In a real-world scenario, this is where a DE at Snowbear Air refines the incoming data so that on-time performance can be tracked to improve customer satisfaction. In this transformation, we are doing the following refinement steps to the raw data:

\*\* Dynamic Table Attributes:\*\*

Immediately after creation, the dynamic table runs its defining query to populate the table. Validate by running the following commands:

A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| If you prefer to pause the lab and finish it later, then please use the following code to stop resource consumption. Note that if you resume at a later time, you will need to restart the lab from the beginning.  DROP TASK IF EXISTS raw.refresh\_flights\_pipe;  DROP PIPE IF EXISTS raw.flights\_pipe;  DROP DYNAMIC TABLE IF EXISTS conformed.flights\_silver\_dt;  ALTER WAREHOUSE SQUIRREL\_de\_wh SUSPEND;  ALTER WAREHOUSE SQUIRREL\_de\_load\_wh SUSPEND; |

**Data Modeling – Modeled (Gold) Layer**

In the gold layer, we create dynamic Apache Iceberg™ tables that are designed for business intelligence and dashboards. They are pre-materialized views with automatic refreshes that support fast queries. In this example, we create tables to capture:

*Classify the cancellations.*

* This table transforms cancelled flight data by applying the SNOWFLAKE.CORTEX.CLASSIFY\_TEXT function to the cancellation\_reason column.
* The query branches using a UNION ALL to handle rows with non-empty and empty cancellation\_reason separately.

|  |
| --- |
| -- Gold Layer Dynamic Table  CREATE OR REPLACE DYNAMIC ICEBERG TABLE modeled.flights\_gold\_dt  TARGET\_LAG='10 minutes'  WAREHOUSE=SQUIRREL\_de\_WH  EXTERNAL\_VOLUME='ext\_volume'  CATALOG='SNOWFLAKE'  BASE\_LOCATION='SQUIRREL\_iceberg/flights\_gold\_dt/'  AS  WITH classified AS (  SELECT  flight\_id,  travel\_date,  cleaned\_status AS flight\_status,  cancellation\_reason,  SNOWFLAKE.CORTEX.CLASSIFY\_TEXT(  NULLIF(TRIM(cancellation\_reason), ''),  ['weather','crew','mechanical','air traffic']  )['label']::STRING AS cancellation\_category,  origin,  origin\_city,  destination,  destination\_city  FROM conformed.flights\_silver\_dt  WHERE cleaned\_status IN ('ON\_TIME','DELAYED','CANCELLED')  AND NULLIF(TRIM(cancellation\_reason), '') IS NOT NULL  )  SELECT \* FROM classified  UNION ALL  SELECT  flight\_id,  travel\_date,  cleaned\_status AS flight\_status,  cancellation\_reason,  NULL AS cancellation\_category,  origin,  origin\_city,  destination,  destination\_city  FROM conformed.flights\_silver\_dt  WHERE cleaned\_status IN ('ON\_TIME','DELAYED','CANCELLED')  AND NULLIF(TRIM(cancellation\_reason), '') IS NULL; |

Note that since the returned object type of the CLASSIFY\_TEXT function is type VARIANT, it had to be cast as a STRING because VARIANT types are not allowed in Apache Iceberg™ tables.

**Verify the data is transformed on table creation.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Build the cancelled flight summary dynamic table.**

* In this part of the lab, you will create another dynamic Iceberg table that aggregates flight cancellation data from our gold layer table (**flights\_gold\_dt**).
* This table—**cancelled\_flight\_summary\_dt**—provides a summarized view of cancellations by cancellation category and origin city over a specified date range. The table is defined as dynamic, meaning it automatically refreshes on a schedule (every 10 minutes in this case) to capture the most recent changes from its source data.
* Key aspects of this design include: Now we can use this table to find out information like what were the main reasons flights were cancelled in 2019.

A screenshot of a computer

AI-generated content may be incorrect.

Now a report or dashboard can be used to visualize the data or feed into a machine learning model to predict trend analysis.

Since this information has been pre-computed and aggregated, the query can quickly return results.

A screenshot of a computer

AI-generated content may be incorrect.

**Dynamic Apache Iceberg™ table for delayed flights.**

Similarly, this table filters for flights that are marked as DELAYED. It creates another materialized table for the DELAYED flight status.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Dynamic Apache Iceberg™ table for early flights.**

This table captures flights with an EARLY status.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Generate more data.**

* Now that our data pipeline is in place, let’s simulate more data being added to the pipeline and refresh our pipe and dynamic tables manually to validate the data pipeline.
* Please allow for some time for the data to load and compute depending on the number of files being processed.

A screenshot of a computer

AI-generated content may be incorrect.

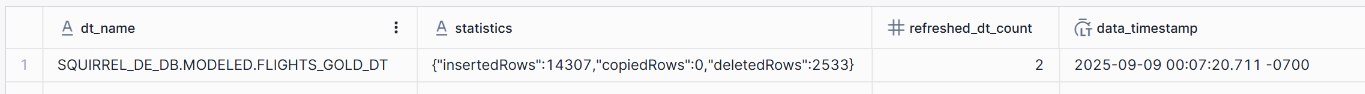
A screenshot of a computer

AI-generated content may be incorrect.

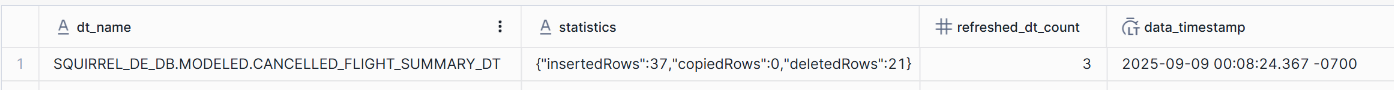
A screenshot of a computer

AI-generated content may be incorrect.

ALTER DYNAMIC TABLE modeled.flights\_gold\_dt REFRESH;



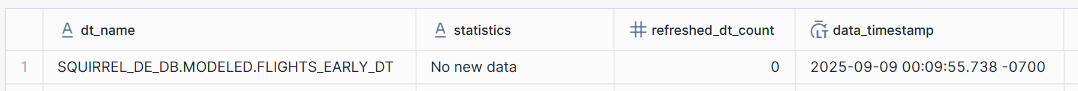
ALTER DYNAMIC TABLE modeled.cancelled\_flight\_summary\_dt REFRESH;



ALTER DYNAMIC TABLE modeled.flights\_delayed\_dt REFRESH;



ALTER DYNAMIC TABLE modeled.flights\_early\_dt REFRESH;



SELECT COUNT(\*) FROM conformed.flights\_silver\_dt; --46743

SELECT COUNT(\*) FROM modeled.flights\_gold\_dt;--14307

SELECT COUNT(\*) FROM modeled.cancelled\_flight\_summary\_dt;--37

SELECT COUNT(\*) FROM modeled.flights\_delayed\_dt;--13345

SELECT COUNT(\*) FROM modeled.flights\_early\_dt;--32436

**Resource Cleanup**

|  |
| --- |
| -- Drop the task to stop schedule warehouse consumption  DROP TASK IF EXISTS raw.refresh\_flights\_pipe;  -- Drop dynamic tables when the lab is complete to reduce storage costs:  DROP DYNAMIC TABLE IF EXISTS conformed.flights\_silver\_dt;  DROP DYNAMIC TABLE IF EXISTS modeled.flights\_gold\_dt;  DROP DYNAMIC TABLE IF EXISTS modeled.cancelled\_flight\_summary\_dt;  DROP DYNAMIC TABLE IF EXISTS modeled.flights\_delayed\_dt;  DROP DYNAMIC TABLE IF EXISTS modeled.flights\_early\_dt;  -- Optionally, drop the raw pipe to stop further ingestion:  DROP PIPE IF EXISTS raw.flights\_pipe;  -- Suspend warehouses to conserve compute costs:  ALTER WAREHOUSE SQUIRREL\_de\_wh SUSPEND; |

# Performance Analysis Toolkit and Tuning Metrics

Setup

|  |
| --- |
| USE ROLE de\_role;  ALTER SESSION SET QUERY\_TAG='(SQUIRREL) Lab: Performance Analysis Toolkit and Tuning Metrics';  USE WAREHOUSE SQUIRREL\_de\_QUERY\_WH;  USE database SQUIRREL\_de\_DB;  USE SCHEMA snowflake\_sample\_data.tpcds\_sf10tcl; |

Disable the use of cached query results (for testing purposes).

ALTER SESSION SET USE\_CACHED\_RESULT = false;

**Explore Spilling to Local and Remote Storage**

Resize your warehouse to SMALL.

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH RESUME IF SUSPENDED;

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH SET WAREHOUSE\_SIZE = XSMALL;

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH SET WAREHOUSE\_SIZE = SMALL WAIT\_FOR\_COMPLETION = TRUE;

* The WAIT\_FOR\_COMPLETION parameter can be used when resizing a warehouse to block the return of the ALTER WAREHOUSE command until the resize has finished provisioning all its compute resources.
* Blocking the command’s return when resizing to a larger warehouse serves to notify you that your compute resources have been fully provisioned and the warehouse is now ready to execute queries using all the new resources.
* This is useful when conducting performance testing to enforce resource uniformity across warehouses of the same size.

Run a query with a window function on the small warehouse. The query lists detailed Catalog sales data and a running sum of sales price within the order. On a small warehouse, this will take a few minutes to complete.

A screenshot of a computer

AI-generated content may be incorrect.

**View the query profile and select the operator WindowFunction[1]**

Note how long the query took and how much query time was spent on the window function. Also, note that your query is spilling to local storage and possibly to remote storage.

**Resize your warehouse to MEDIUM.**

--Suspend the warehouse to flush the data cache

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH SUSPEND;

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH RESUME;

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH SET WAREHOUSE\_SIZE = MEDIUM WAIT\_FOR\_COMPLETION = TRUE;

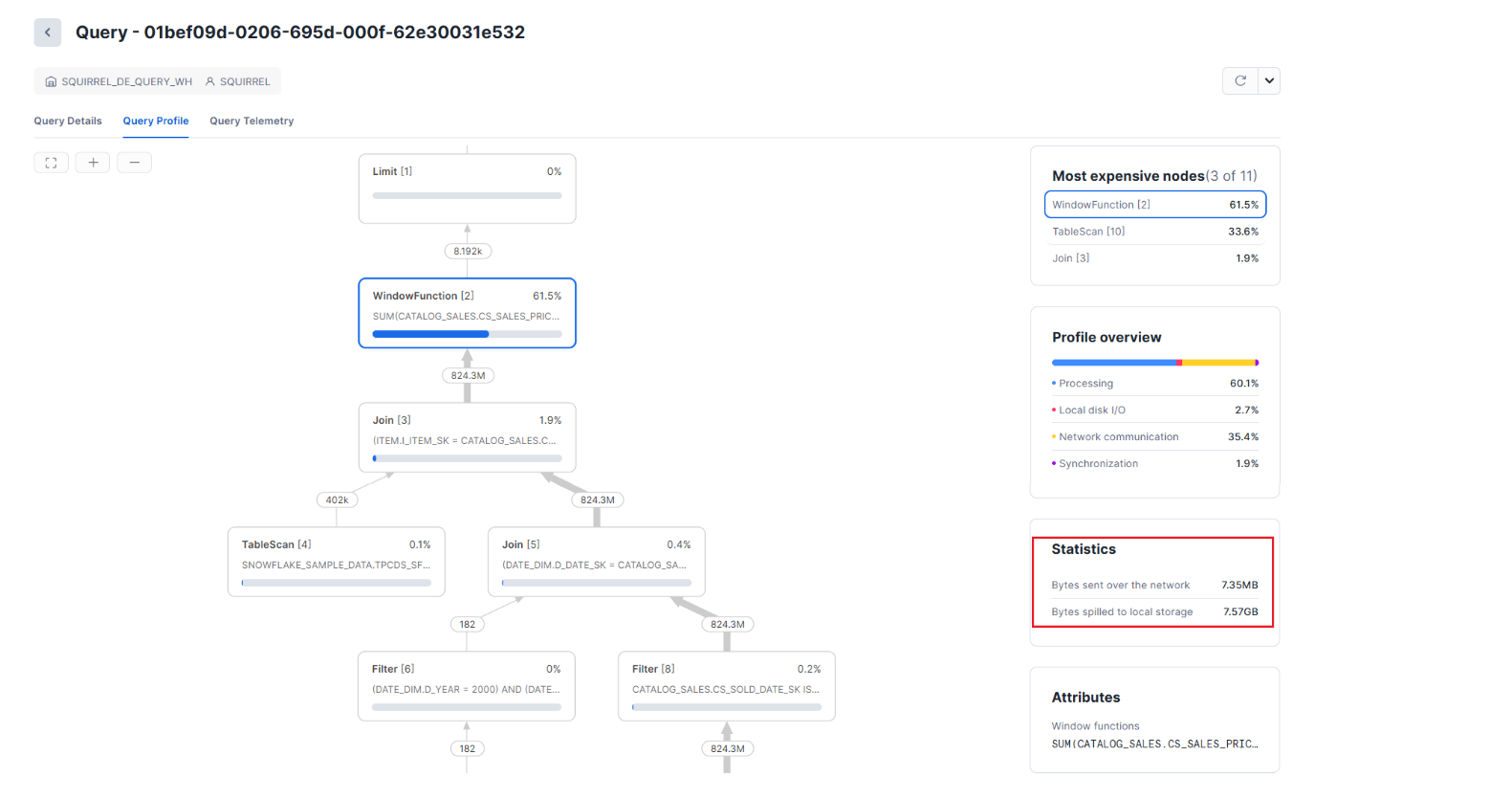
**Re-run the query using a medium warehouse.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**View the Query Profile.**

What was the query time? Click on the diagram’s **WindowFunction[2]** node and see if the query is still spilling to local or remote storage.



**Resize your virtual warehouse to LARGE.**

**--** Again, suspend the warehouse to flush the data cache

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH SUSPEND;

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH RESUME;

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH SET WAREHOUSE\_SIZE = LARGE WAIT\_FOR\_COMPLETION = TRUE;

**Re-run the query using a large warehouse.**

A screenshot of a computer

AI-generated content may be incorrect.

**View the query profile.**

What was the execution time? Is **WindowFunction[2]** still spilling? Snowflake’s architecture allows for linear scaling of compute resources, simultaneously enabling better performance and lower cost.

Example response times:

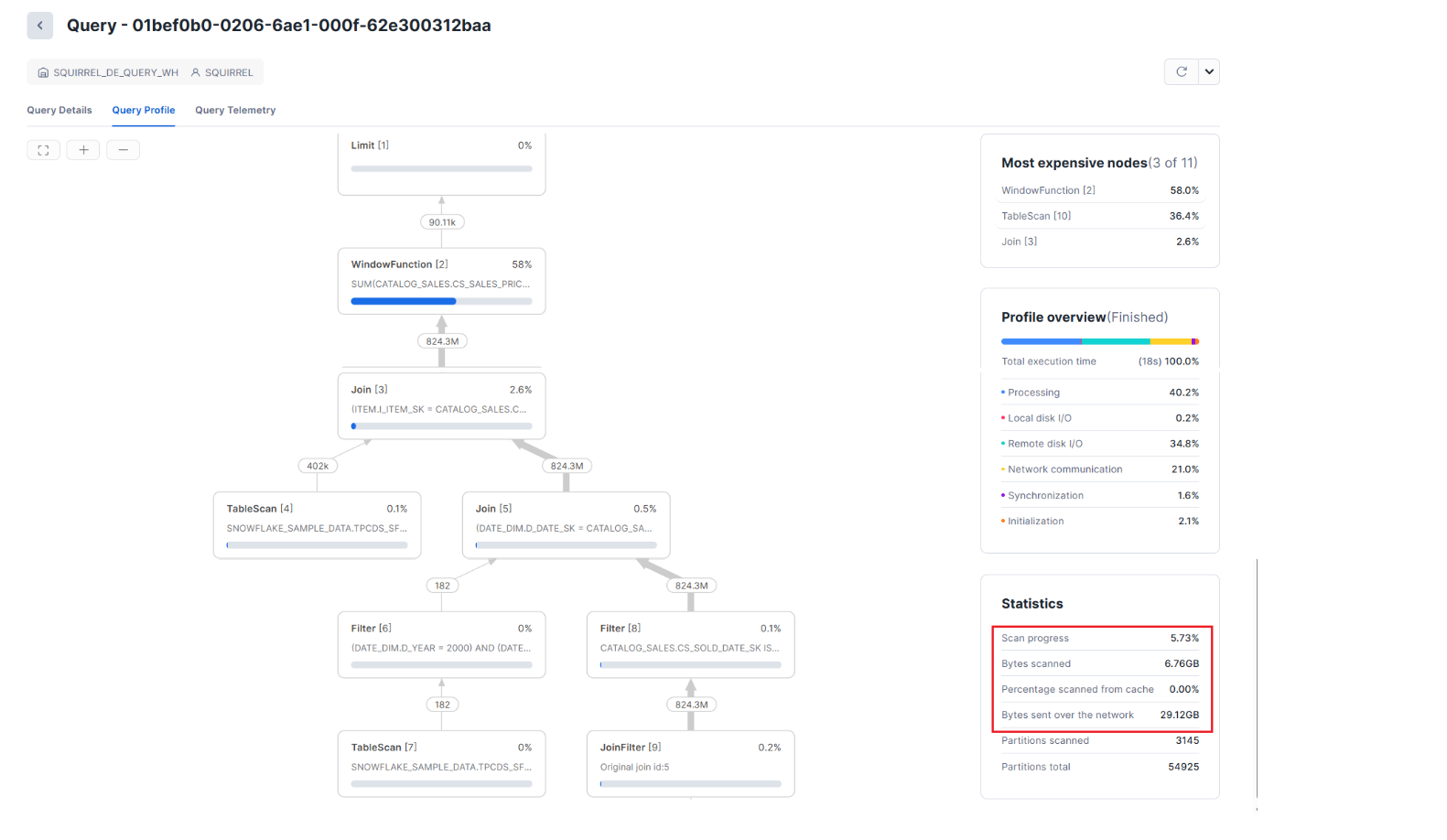
Cluster Size Response Time Credit Cost

-------------- --------------- -------------

Small 2 min 14s 0.074

Medium 1 min 7s 0.074

Large 33s 0.073



**Evaluate WHERE Clauses Based on Clustering Efficiency**

In this scenario, we look into large table scan symptoms caused by the fact that the query’s filter column is not the clustering dimension of the base table.

Set your context and warehouse size.

USE SCHEMA training\_db.tpch\_sf1000;

ALTER SESSION SET USE\_CACHED\_RESULT = FALSE;

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH SUSPEND;

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH SET WAREHOUSE\_SIZE = MEDIUM;

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH RESUME;

**Run a query with a filter on a column that is not well-clustered.**

A screenshot of a computer program

AI-generated content may be incorrect.

**View the query profile and note the micro-partition pruning. Note that all micro-partitions were scanned.**

Evaluate the clustering efficiency on the L\_EXTENDEDPRICE column.

SELECT SYSTEM$CLUSTERING\_INFORMATION( 'lineitem' , '(l\_extendedprice)' );



Note that the table is poorly clustered on the L\_EXTENDEDPRICE dimension. The clustering depth shows almost 100% overlap in the micro-partitions.

**Run a query that filters on a well-clustered column.**

A screenshot of a computer

AI-generated content may be incorrect.

Open the query profile and view micro-partition pruning.

****

Evaluate the clustering efficiency of the filter column.

**A screenshot of a computer

AI-generated content may be incorrect.**

|  |
| --- |
| {  "cluster\_by\_keys" : "LINEAR(l\_shipdate)",  "total\_partition\_count" : 9183,  "total\_constant\_partition\_count" : 6750,  "average\_overlaps" : 0.516,  "average\_depth" : 1.2649,  "partition\_depth\_histogram" : {  "00000" : 0,  "00001" : 6750,  "00002" : 2433,  "00003" : 0,  "00004" : 0,  "00005" : 0,  "00006" : 0,  "00007" : 0,  "00008" : 0,  "00009" : 0,  "00010" : 0,  "00011" : 0,  "00012" : 0,  "00013" : 0,  "00014" : 0,  "00015" : 0,  "00016" : 0  },  "clustering\_errors" : [ ]  } |

**Summary:** The amount of data scanned by a query is directly related to how well a query’s WHERE clause column correlates to the clustering dimension of the table being accessed.

**Rogue Query Symptom from JOIN Explosion**

This scenario will explore runaway query symptoms common in real workloads. One common type of rogue query includes join pitfalls like an explosion of output and unintentional cross-joins.

**Set your context.**

|  |
| --- |
| USE SCHEMA snowflake\_sample\_data.tpcds\_sf100tcl;  ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH SET WAREHOUSE\_SIZE = MEDIUM;  USE WAREHOUSE SQUIRREL\_de\_QUERY\_WH; |

**Run the following query: This may take just over 4 minutes to complete.**

A screenshot of a computer

AI-generated content may be incorrect.

**View the profile for this query.**

Note that the JOIN produces many output records relative to the sizes of the two input data sets.

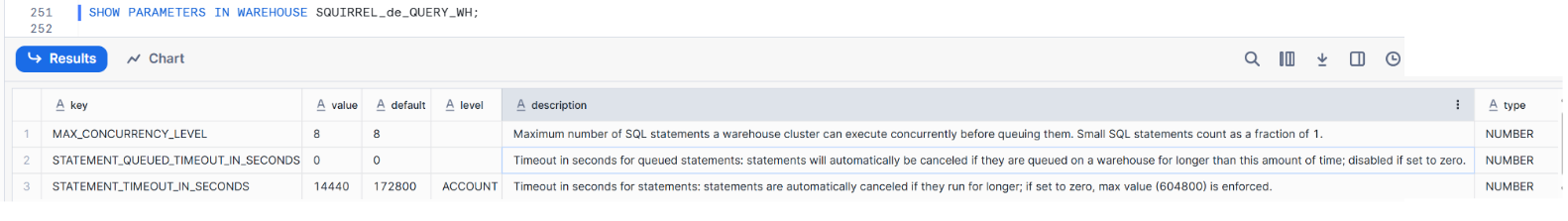
To avoid performance issues with a JOIN that could explode the outputs, you should JOIN on unique keys whenever possible. If not possible, consider other options, such as adding more filters to the WHERE clause or scaling up the warehouse.

**Tune Timeout Parameters**

This exercise will explore tuning the timeout parameter available to a virtual warehouse to manage long-running workloads.

**Review the existing values of the timeout parameters:**

SHOW PARAMETERS IN WAREHOUSE SQUIRREL\_de\_QUERY\_WH;



You will see that the maximum timeout for a queued statement is 0 seconds (meaning it will never time out).

The maximum timeout of a statement is just over 4 hours.

Update the statement\_timeout\_in\_seconds parameter to 30 seconds.

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH SET STATEMENT\_TIMEOUT\_IN\_SECONDS = 30;

Re-run the query from above.

SELECT S.SS\_SOLD\_DATE\_SK

, R.SR\_RETURNED\_DATE\_SK

, S.SS\_STORE\_SK

, S.SS\_ITEM\_SK

, S.SS\_SALES\_PRICE

, S.SS\_SALES\_PRICE

, R.SR\_RETURN\_AMT

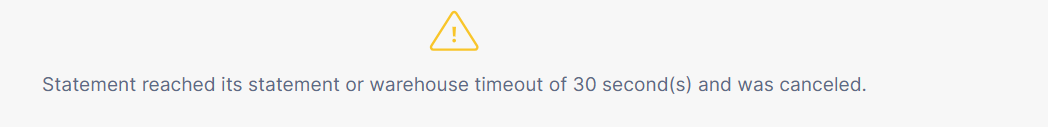
FROM STORE\_SALES S

INNER JOIN STORE\_RETURNS R

on R.SR\_ITEM\_SK=S.SS\_ITEM\_SK

WHERE S.SS\_ITEM\_SK =4164;

The statement will **timeout before** it completes and generate an error message.



Set the timeout back to its default using UNSET.

ALTER WAREHOUSE SQUIRREL\_de\_QUERY\_WH UNSET STATEMENT\_TIMEOUT\_IN\_SECONDS;

Confirm that the timeout parameter has been restored to 14,440 seconds (just over 4 hours)

SHOW PARAMETERS IN WAREHOUSE SQUIRREL\_de\_QUERY\_WH;

A screenshot of a computer

AI-generated content may be incorrect.

**Use Query Tags**

This exercise will explore setting the QUERY\_TAG parameter to track queries executed within a session. This additional metadata can be queried later for tracking purposes. We can search the INFORMATION\_SCHEMA.QUERY\_HISTORY view or by filtering on the Query Tag in the History view of the Snowflake Web UI.

See if an existing query tag is set.

SHOW PARAMETERS LIKE '%QUERY\_TAG%' FOR SESSION;

A screenshot of a computer

AI-generated content may be incorrect.

Update the QUERY\_TAG session parameter:

ALTER SESSION SET QUERY\_TAG = 'Lab - Performance Analysis Toolkit and Tuning Metrics - SQUIRREL\_join\_test';

SHOW PARAMETERS LIKE '%QUERY\_TAG%' FOR SESSION;

A screenshot of a computer

AI-generated content may be incorrect.

**Run the join explosion query from the previous exercise with a LIMIT clause.**

SELECT s.ss\_sold\_date\_sk

, r.sr\_returned\_date\_sk

, s.ss\_store\_sk

, r.sr\_returned\_date\_sk

, s.ss\_item\_sk

, s.ss\_sales\_price

, r.sr\_return\_amt

FROM store\_sales s

INNER JOIN store\_returns r

ON r.sr\_item\_sk=s.ss\_item\_sk

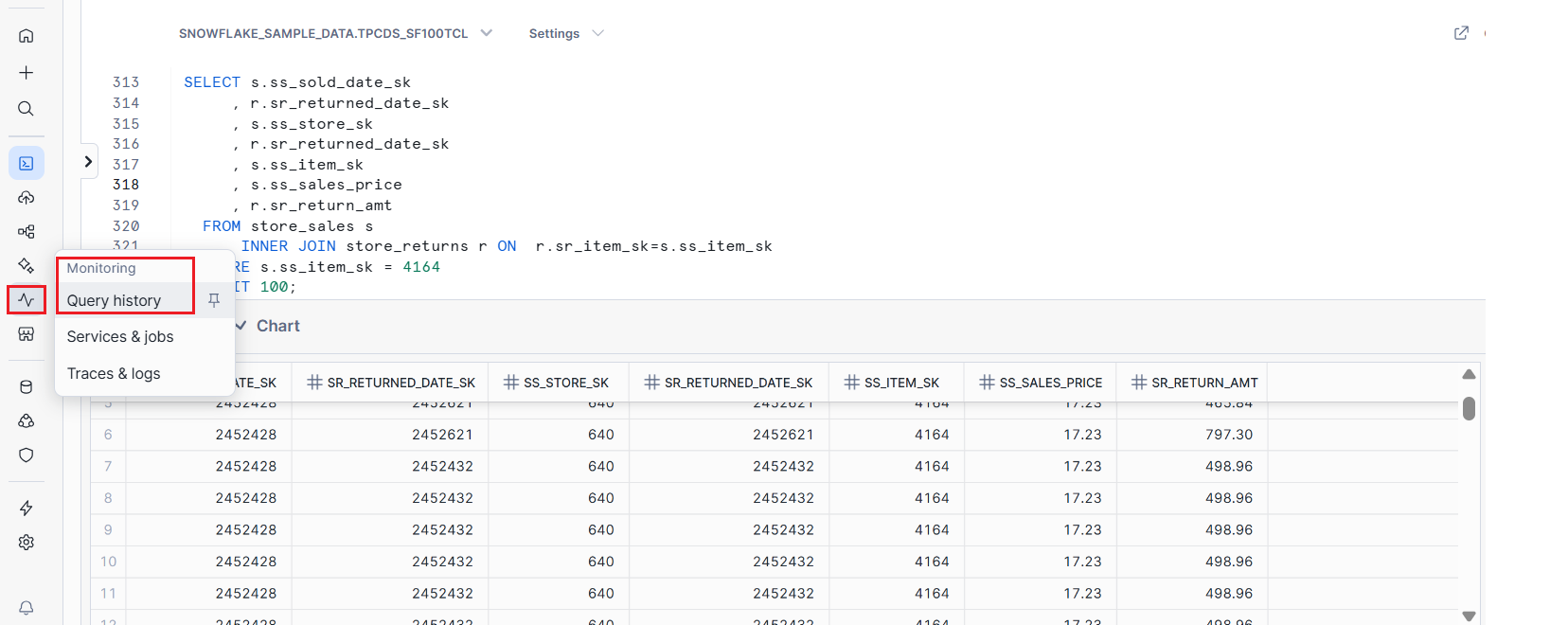
WHERE s.ss\_item\_sk = 4164

LIMIT 100;

A screenshot of a computer

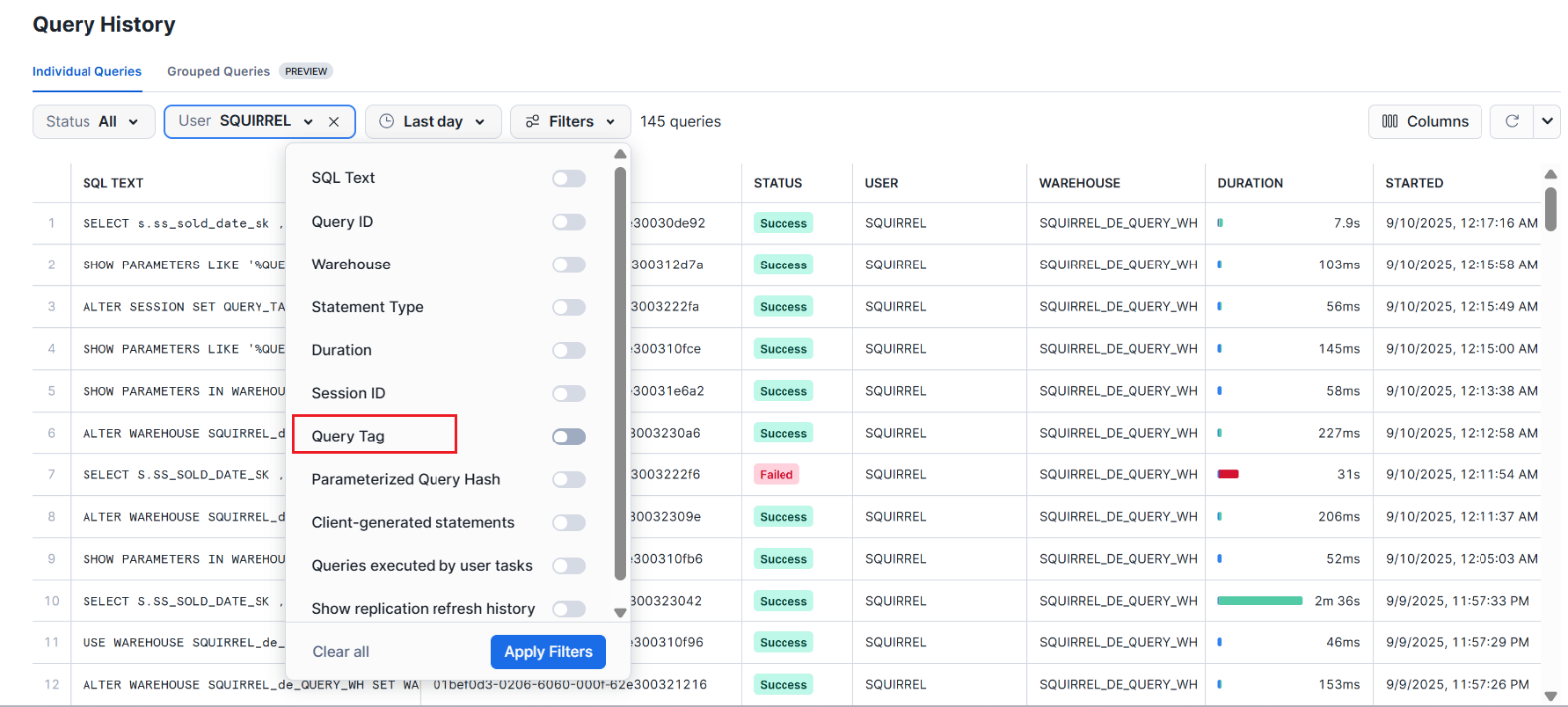
AI-generated content may be incorrect.

Use the **Query History** view in the Snowsight Web UI with the Query Tag filter to find tagged queries. Click the < navigation button, top left, then select the Query History page under the Monitoring area.



Ensure your username is selected in the User box at the top of the screen before clicking on Filters.

Select the **Query Tag** option and enter **JOIN** in the text box that appears before clicking on the **Apply Filters** button to return the results. It will look something like this:



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Alternatively, look for the query tag using the INFORMATION\_SCHEMA.QUERY\_HISTORY table function. Execute the following query in the worksheet for this lab:

SELECT QUERY\_ID, QUERY\_TAG

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

WHERE QUERY\_TAG ILIKE '%join%';

A screenshot of a computer

AI-generated content may be incorrect.

Finally, reset the QUERY\_TAG session parameter to its default value (empty string).

ALTER SESSION UNSET QUERY\_TAG;

SHOW PARAMETERS LIKE '%QUERY\_TAG%' FOR SESSION;

-- or

SHOW PARAMETERS LIKE '%QUERY\_TAG%' IN SESSION;

**A screenshot of a computer

AI-generated content may be incorrect.**

# Performance

**SCENARIO:**

Snow Bear Air has a pipeline that processes weather data so the operations team can decide which flights to cancel or delay. They've noticed that the weather data processing is taking too long and requested that someone investigate this issue. You've been asked to analyse the pipeline's compute and performance profile and make the necessary changes to improve its performance and efficiency.

Note that to ensure weather data was available in a timely fashion, the Service Level Agreement (SLA) for the query’s runtime was set to 8-10 seconds.

* Use the Query Profiler to review query performance
* Use knowledge of SQL to modify SELECT statements
* Apply knowledge on how partition pruning impacts query performance
* Apply knowledge on how warehouse sizing impacts query performance

**STARTING OBJECTS AND STATES**

* A query exists for moving weather data from the RAW schema to the CONFORMED schema.
* A task exists that moves weather data from the RAW to the CONFORMED schema.
* The query will have run before, so information about the query's performance should have been logged.

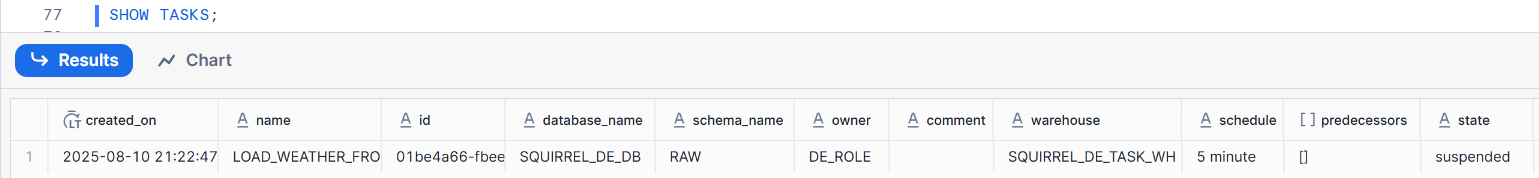
|  |
| --- |
| USE ROLE de\_role;  ALTER SESSION SET QUERY\_TAG='(SQUIRREL) Lab: Performance';  USE WAREHOUSE SQUIRREL\_de\_task\_wh;  ALTER WAREHOUSE SQUIRREL\_de\_task\_wh SET WAREHOUSE\_SIZE = 'SMALL';  USE DATABASE SQUIRREL\_de\_db;  USE SCHEMA RAW; |

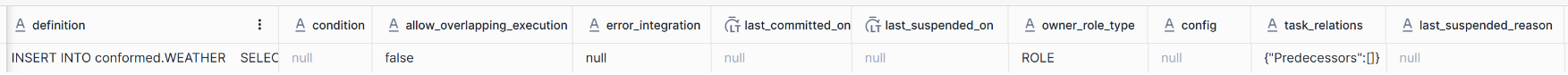
**Investigating the Problem**

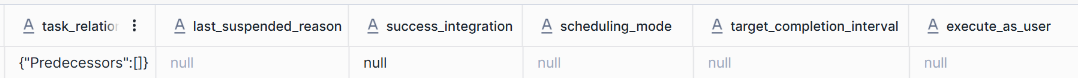
* Understanding the high-level overview of the components that move weather data from the RAW to CONFORMED schemas is essential.
* The raw weather data was previously loaded into our RAW.ISD\_TOTAL table from a 3rd party external public S3 bucket.
* A task LOAD\_WEATHER\_FROM\_RAW\_TO\_CONFORMED is scheduled to run to calculate the average of seven weather station measurements and INSERT the calculated results into CONFORMED.WEATHER table.
* In this lab, you will work with Snowflake TASK objects to execute SQL, DML, DDL, or Stored Procedures using trigger events (other Tasks completing) or scheduling (CRON or otherwise).

**Start the task that moves weather data.**

* Note that this task was created for you by the Education Services team. Run the SHOW TASK command:
* The show command output indicates that the State of the pre-defined TASK object is suspended. To start the task, use RESUME in an ALTER statement, and run SHOW TASKS again to confirm:



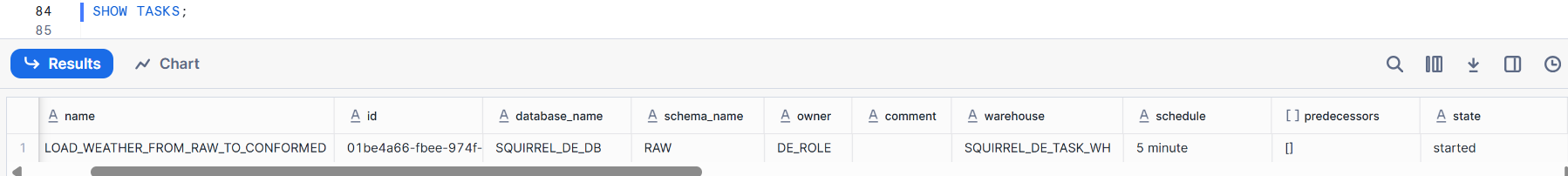




Definition

|  |
| --- |
| INSERT INTO conformed.WEATHER  SELECT  CURRENT\_TIMESTAMP AS WEATHER\_TIMESTAMP  , AVG(observations.value:air.temp)::decimal(18,2) AS AVG\_AIR\_TEMP  , AVG(observations.value:air."dew-point")::decimal(18,2) AS AVG\_DEW\_POINT  , AVG(observations.value:atmospheric.pressure)::decimal(18,2) AS AVG\_PRESSURE  , AVG(observations.value:sky.ceiling)::decimal(18,2) AS AVG\_CEILING  , AVG(observations.value:visibility.distance)::decimal(18,2) AS AVG\_DISTANCE  , AVG(observations.value:wind."direction-angle")::decimal(18,2) AS AVG\_DIRECTION\_ANGLE  , AVG(observations.value:wind."speed-rate")::decimal(18,2) AS AVG\_SPEED\_RATE  FROM  raw.ISD\_TOTAL weather, LATERAL FLATTEN(input => v:data.observations) observations  WHERE  weather.T::varchar BETWEEN '2010-02-03 08:00:00.000' AND '2011-02-03 08:00:00.000' |

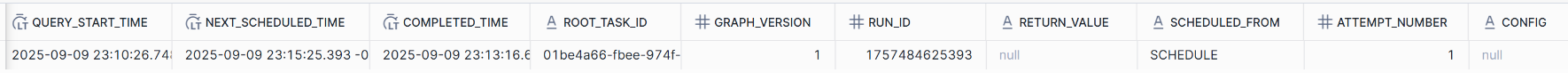
ALTER TASK raw.load\_weather\_from\_raw\_to\_conformed RESUME;

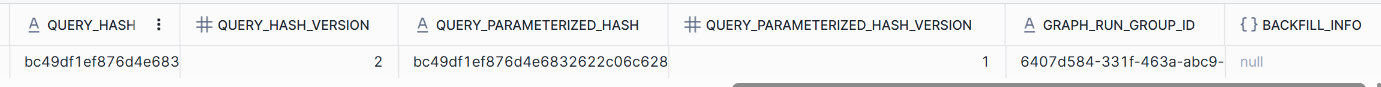


**Determine if the Query in the task meets the SLA.**

* Given the query that moves the weather data is executed from a TASK, we can use the built-in task\_history table function to examine the history of the task and determine if the SLA is within 8-10 seconds.
* Use the task\_history table function to retrieve the ten most recent executions of the LOAD\_WEATHER\_FROM\_RAW\_TO\_CONFORMED task (scheduled within the last hour) that has been completed or is still running:



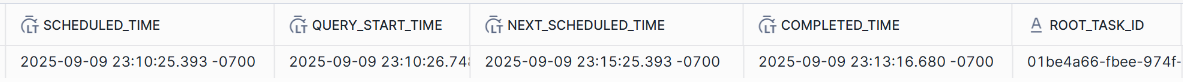




QUERY\_TEXT

|  |
| --- |
| INSERT INTO conformed.WEATHER  SELECT  CURRENT\_TIMESTAMP AS WEATHER\_TIMESTAMP  , AVG(observations.value:air.temp)::decimal(18,2) AS AVG\_AIR\_TEMP  , AVG(observations.value:air."dew-point")::decimal(18,2) AS AVG\_DEW\_POINT  , AVG(observations.value:atmospheric.pressure)::decimal(18,2) AS AVG\_PRESSURE  , AVG(observations.value:sky.ceiling)::decimal(18,2) AS AVG\_CEILING  , AVG(observations.value:visibility.distance)::decimal(18,2) AS AVG\_DISTANCE  , AVG(observations.value:wind."direction-angle")::decimal(18,2) AS AVG\_DIRECTION\_ANGLE  , AVG(observations.value:wind."speed-rate")::decimal(18,2) AS AVG\_SPEED\_RATE  FROM  raw.ISD\_TOTAL weather, LATERAL FLATTEN(input => v:data.observations) observations  WHERE  weather.T::varchar BETWEEN '2010-02-03 08:00:00.000' AND '2011-02-03 08:00:00.000' |

Examine the output, specifically the QUERY\_START\_TIME and COMPLETED\_TIME columns. Did the query compete within 8-10 seconds?



**Suspend the task.**

ALTER TASK raw.load\_weather\_from\_raw\_to\_conformed SUSPEND;

**Adjusting the Query**

The query defined in the task LOAD\_WEATHER\_FROM\_RAW\_TO\_CONFORMED does not complete within the required SLA. In this exercise, you will determine the root cause of the performance problem using the query profiler and try to improve the query performance by changing the SQL to take advantage of partition pruning.

**Disable the use of cached query results (for testing purposes).**

ALTER SESSION SET USE\_CACHED\_RESULT = FALSE;

**Locate the query in the query history.**

Use the query\_history table function to find the query in question. We haven’t covered this monitoring table function query\_history() yet, but we will cover specifics later in the course. For now, know it’s giving you telemetry on the queries run in Snowflake, which is how we’re using it here.

Run the following SQL and from the output , click on the links in the QUERY\_TEXT column to view the SQL until you find “INSERT INTO conformed.WEATHER SELECT CURRENT\_TIMESTAMP AS WEATHER\_TIMESTAMP ..” query.

A screenshot of a computer

AI-generated content may be incorrect.

Next run the SELECT portion extracted form INSERT INTO SELECT statement.

|  |
| --- |
| SELECT CURRENT\_TIMESTAMP AS WEATHER\_TIMESTAMP  , AVG(observations.value:air.temp)::decimal(18,2) AS AVG\_AIR\_TEMP  , AVG(observations.value:air."dew-point")::decimal(18,2) AS AVG\_DEW\_POINT  , AVG(observations.value:atmospheric.pressure)::decimal(18,2) AS AVG\_PRESSURE  , AVG(observations.value:sky.ceiling)::decimal(18,2) AS AVG\_CEILING  , AVG(observations.value:visibility.distance)::decimal(18,2) AS AVG\_DISTANCE  , AVG(observations.value:wind."direction-angle")::decimal(18,2) AS AVG\_DIRECTION\_ANGLE  , AVG(observations.value:wind."speed-rate")::decimal(18,2) AS AVG\_SPEED\_RATE  FROM  raw.ISD\_TOTAL weather, LATERAL FLATTEN(input => v:data.observations) observations  WHERE  weather.T::varchar BETWEEN '2010-02-03 08:00:00.000' AND '2011-02-03 08:00:00.000'; |

**Verify the query run time length**

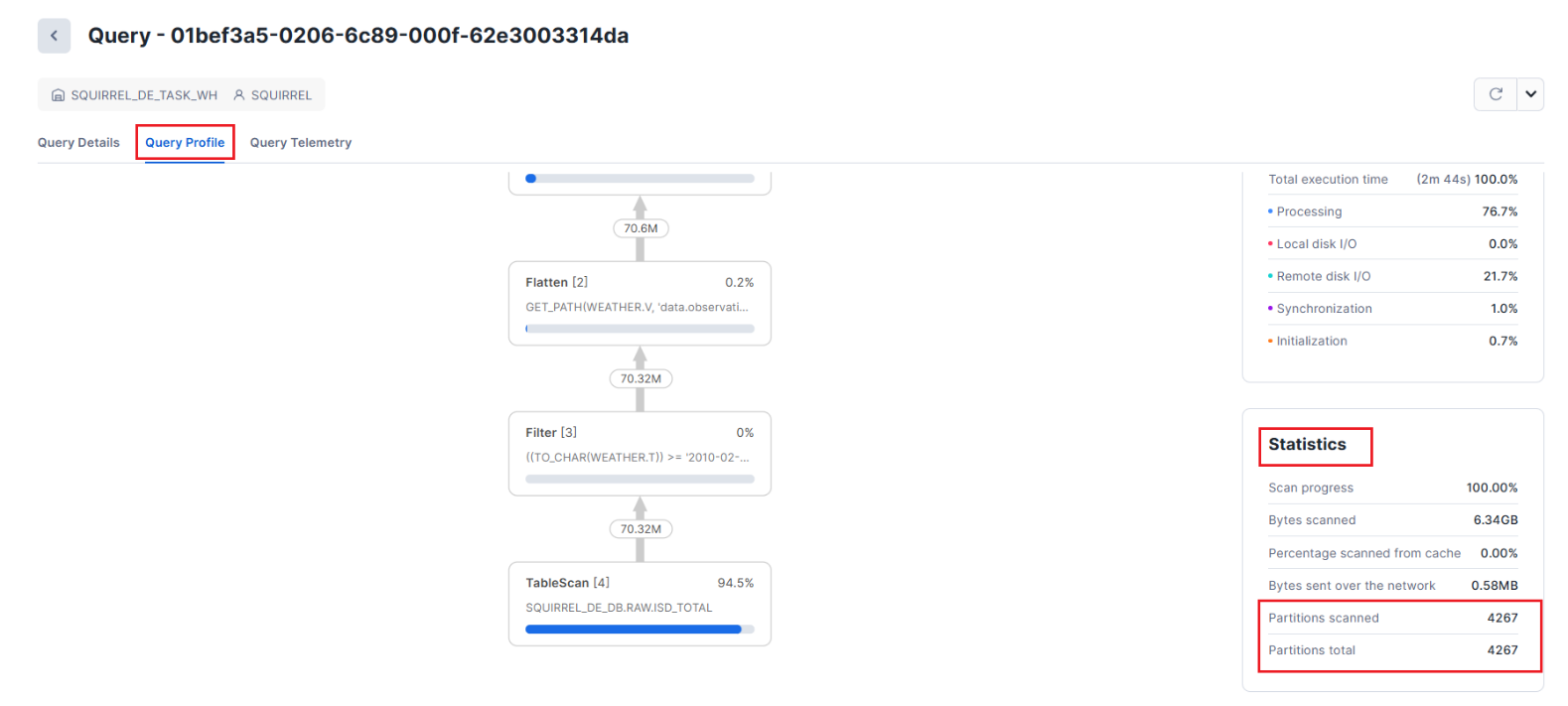
Once the query has run, check the runtime length in the Results pane. It should be approximately over a few minutes, considerably **longer** than the **SLA of 8-10 seconds**.

A screenshot of a computer

AI-generated content may be incorrect.

**Analyse the Query Profile.**

Examine the "**Statistics**" section. The "Partitions scanned" and "Partitions total" should be the same, indicating no partition pruning was possible, meaning the query has been written so that all partitions will be scanned whenever this query is executed.



**Find the root cause.**

* This query returns a simple set of averages from the flattening of JSON data in table ISD\_TOTAL. The SELECT and FROM portions of the statement are not likely to be part of the root cause.
* Take a close look at the WHERE clause:

WHERE weather.T::**varchar** BETWEEN '2010-02-03 08:00:00.000' AND '2011-02-03 08:00:00.000'

* The WHERE clause casts column T from table ISD\_TOTAL (alias weather) to VARCHAR to compare it to the VARCHAR operands in the BETWEEN clause.
* Snowflake is a column store database that leverages partition pruning as one of its strategies for optimizing data retrieval.
* By forcing Snowflake to convert all values in column T to varchar, every single value for that column must be accessed to evaluate the WHERE clause conditions.
* Fix the root cause - Change the WHERE clause to cast ONLY the VARCHAR operands as data type TIMESTAMP\_NTZ and rerun the query:

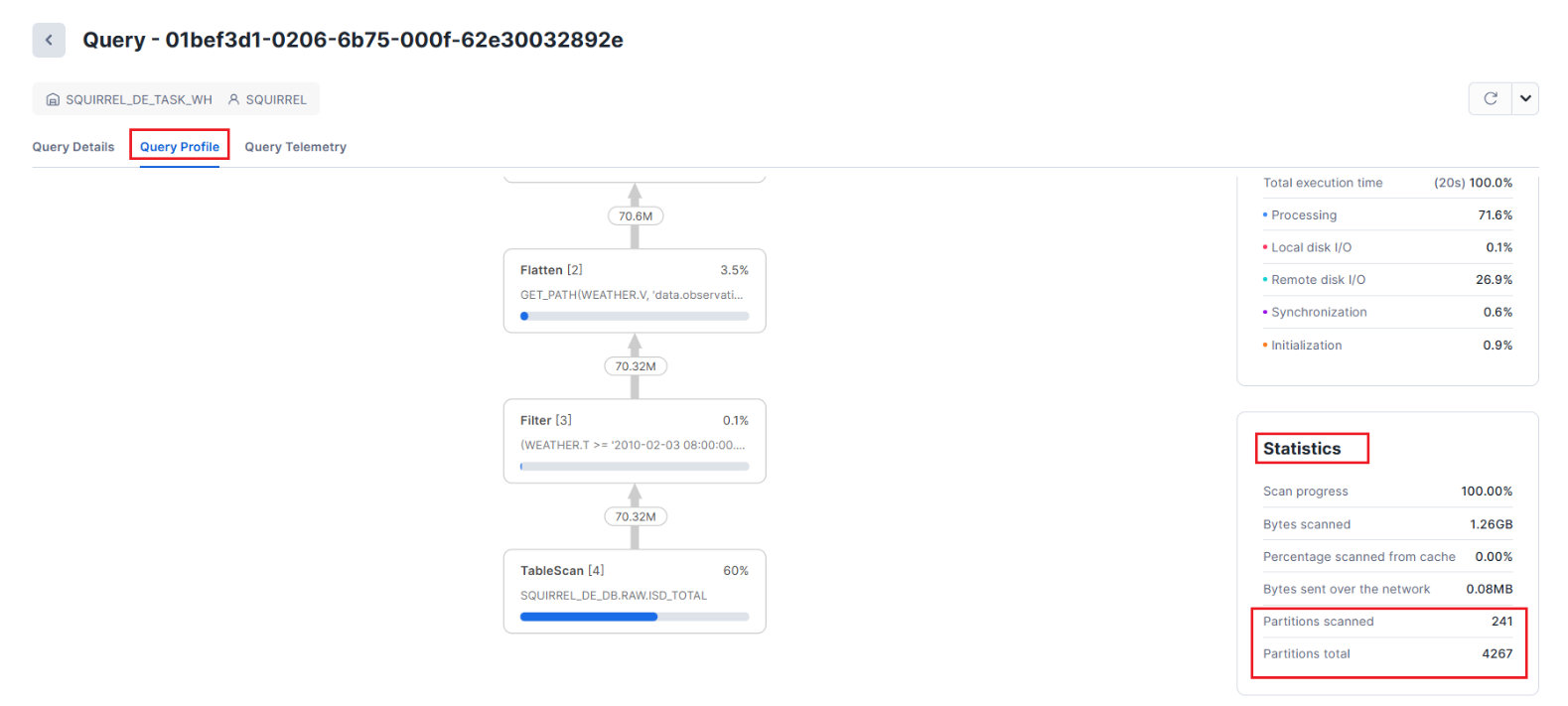
**Test your solution**

* Check the runtime length in the Results pane. The time should be around 20 seconds.
* Although you’ve significantly reduced the query execution time, you still aren’t meeting your SLA of 8-10 seconds.

A screenshot of a computer

AI-generated content may be incorrect.

Now query profile suggest that, the **Partitions scanned** value should be much smaller than the **Partitions total** value.



Check the warehouse size. Note that it is a small (SMALL) warehouse. Increasing the warehouse size adds more compute resources, improving query performance for larger and/or more complex queries. This might be the answer to the problem.

**Testing a Change in Warehouse Size**

--Suspend the warehouse to flush the data cache

ALTER WAREHOUSE SQUIRREL\_de\_task\_wh SUSPEND;

ALTER WAREHOUSE SQUIRREL\_de\_task\_wh RESUME;

--Ensure WH resources are fully provisioned before running workloads

ALTER WAREHOUSE SQUIRREL\_de\_task\_wh SET warehouse\_size = 'large' wait\_for\_completion = true;

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a chat

AI-generated content may be incorrect.

**Fix the query in the TASK object.**

Using the code below, update the body of the task. Note that you will also need to run ALTER TASK…RESUME to put the task into a running state from suspended:

A screen shot of a computer code

AI-generated content may be incorrect.

ALTER TASK raw.load\_weather\_from\_raw\_to\_conformed RESUME;

A screenshot of a computer

AI-generated content may be incorrect.

**Clean-up**

Once you have completed this activity, SUSPEND your task by running the following SQL and remove the query tag.

ALTER TASK raw.load\_weather\_from\_raw\_to\_conformed SUSPEND;

ALTER SESSION UNSET QUERY\_TAG;

# Advanced Alerts & Metadata-Driven Alerting Framework

* This lab focuses on building a flexible alerts framework that leverages metadata to drive alert conditions and actions.
* It includes condition and action handler stored procedures, as well as monitoring of task errors.
* Data Engineers can use this framework to implement sophisticated, configurable monitoring that adapts to evolving requirements.
* Snowflake Alerts allow you to be notified or take action when specific data conditions are met. In this lab, you will build a metadata-driven alerts framework that reads configuration parameters from a table and triggers email notifications when conditions are met.
* This approach enables Data Engineers to create flexible monitoring routines for operational data quality and task performance.
* Ensure you have a valid email address. Alerts in this lab will use email notifications to inform you when conditions are triggered.

**Setup**

|  |
| --- |
| USE ROLE de\_role;  ALTER SESSION SET QUERY\_TAG = '(SQUIRREL) lab - SCENARIO: Advanced Alerting Framework for Data Quality and Operations';  CREATE WAREHOUSE IF NOT EXISTS SQUIRREL\_de\_wh AUTO\_SUSPEND = 60 INITIALLY\_SUSPENDED = true;  ALTER WAREHOUSE SQUIRREL\_de\_wh SET WAREHOUSE\_SIZE = XSMALL;  USE WAREHOUSE SQUIRREL\_de\_wh;  CREATE SCHEMA IF NOT EXISTS SQUIRREL\_de\_db.util;  USE SCHEMA SQUIRREL\_de\_db.util; |

**Create an Email-Type Notification Integration**

For this lab, you’ll set up a dedicated email integration that will be used solely for alert notifications.

If you need more detailed instructions on setting up email integrations, please refer to the "**Management and Observability**" lab.

**Set Up your user profile in Snowsight.**

* You can skip the following steps if you already set up your profile information.
* Pick up the lab again in the Verify the integration has been created section.
* To create an email notification, we must add an email address to the profile.
* Since directly adding an email address to the profile in Snowsight requires elevated privileges, the education team has created a stored procedure to perform this operation with elevated privileges.
* First, add your email address to the following statement so you can receive the email notification.
* Also add your first and last name to complete the update to your profile.
* Note that the email address must be entered in lowercase; otherwise, the statement will fail.
* Replace the following with your personal information.

SET email\_address\_provided = 'REPLACE WITH YOUR EMAIL';

SET f\_name = 'REPLACE WITH YOUR FIRST NAME';

SET l\_name = 'REPLACE WITH YOUR LAST NAME';

**Call the update profile stored procedure to update your profile.**