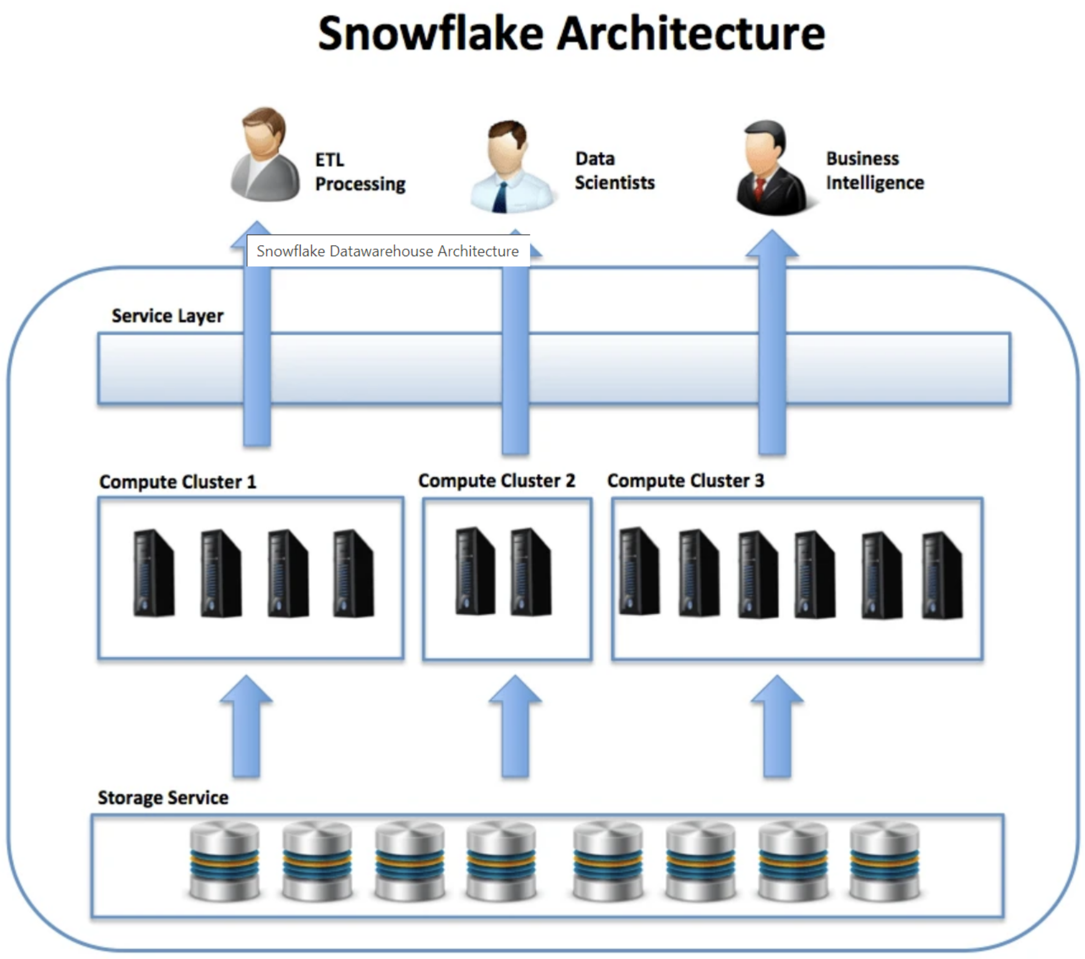
Caching, Optimization and Multi-Cluster Warehouses

In this experiment, we will learn to use caching, optimization and multi cluster warehouses in Snowflake

Snowflake Database Architecture

Before starting it's worth considering the underlying Snowflake architecture, and explaining when Snowflake caches data. The diagram below illustrates the overall architecture which consists of three layers:-



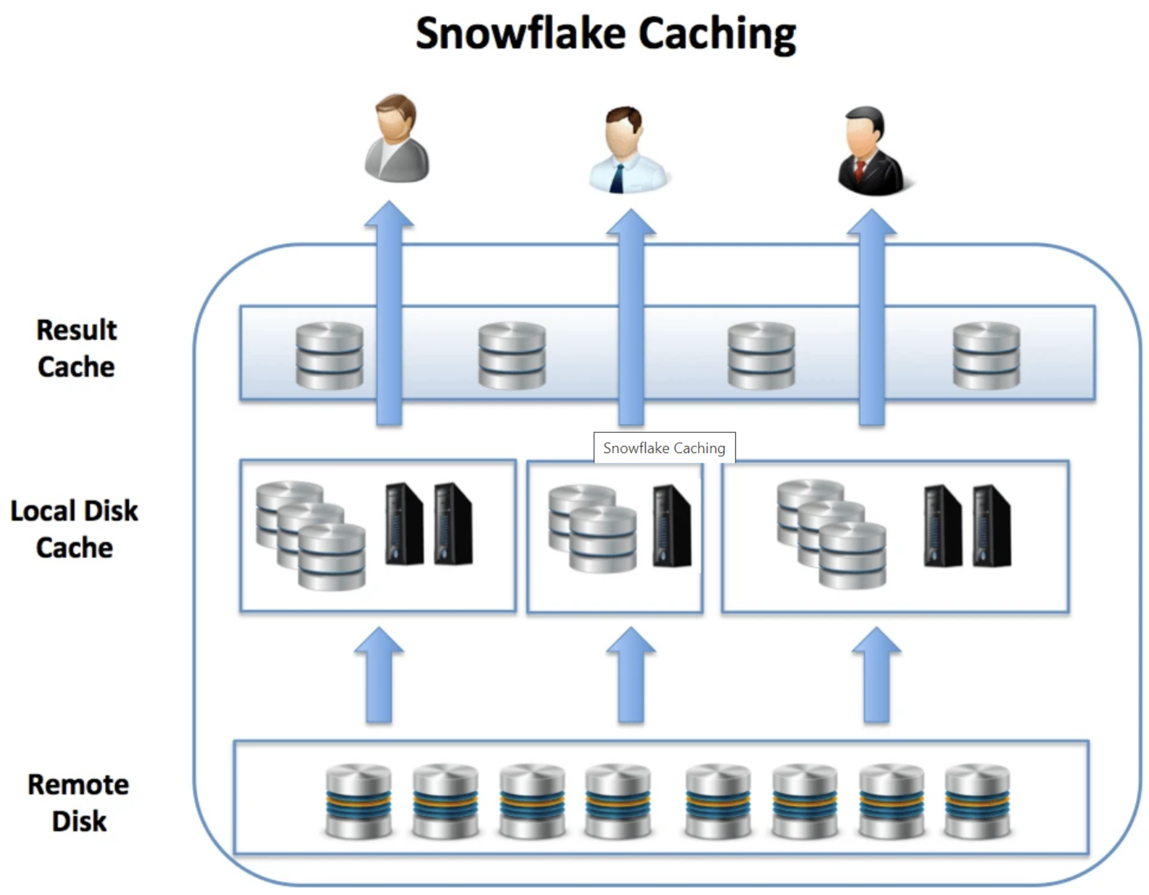
1. **Service Layer:** Which accepts SQL requests from users, coordinates queries, managing transactions and results. Logically, this can be assumed to hold the *result cache* – a cached copy of the results of every query executed.
2. **Compute Layer:** Which actually does the heavy lifting. This is where the actual SQL is executed across the nodes of a *Virtual Data Warehouse*. This layer holds a cache of data queried, and is often referred to as *Local Disk I/O* although in reality this is implemented using SSD storage. All data in the compute layer is temporary, and only held as long as the virtual warehouse is active.

3. **Storage Layer:** Which provides long term storage of results. This is often referred to as *Remote Disk*, and is currently implemented on either Amazon S3 or Microsoft Blob storage.

Snowflake Cache Layers

The diagram below illustrates the levels at which data and results are cached for subsequent use. These are:-

1. **Result Cache:** Which holds the results of every query executed in the past 24 hours. These are available across virtual warehouses, so query results returned to one user is available to any other user on the system who executes the same query, provided the underlying data has not changed.
2. **Local Disk Cache:** Which is used to cache data used by SQL queries. Whenever data is needed for a given query it's retrieved from the *Remote Disk* storage, and cached in SSD and memory.
3. **Remote Disk:** Which holds the long term storage. This level is responsible for data resilience, which in the case of Amazon Web Services, means 99.999999999% durability. Even in the event of an entire data centre failure.



1. **Create new Citibike_Pipeline database**

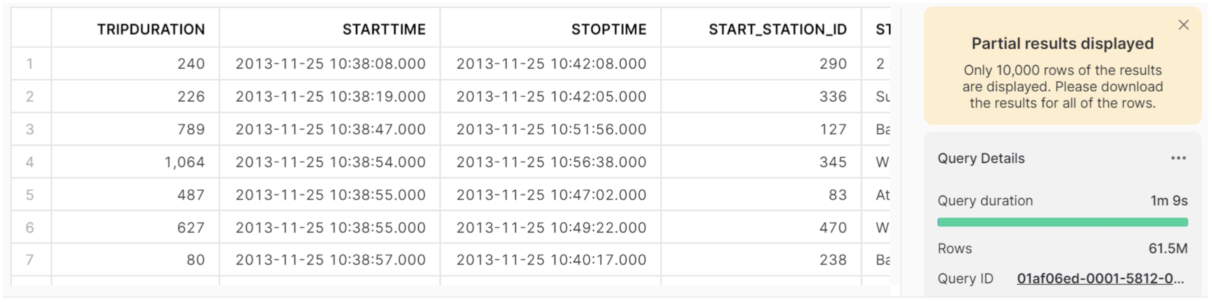
```
DROP DATABASE IF EXISTS CITIBIKE_PIPELINES;

-- Switch Context
USE ROLE ACCOUNTADMIN;

--Create the Warehouse
CREATE WAREHOUSE IF NOT EXISTS DATAPIPELINES_WH
   WITH WAREHOUSE_SIZE = 'XSMALL'
   AUTO_SUSPEND = 60
   AUTO_RESUME = TRUE;

--- Create the database and grant access to the new role create
CREATE DATABASE IF NOT EXISTS CITIBIKE_PIPELINES;

-- Switch Context
USE CITIBIKE_PIPELINES.PUBLIC;
USE WAREHOUSE DATAPIPELINES_WH;


-- Create the table for Trips
CREATE OR REPLACE TABLE TRIPS
(tripduration integer,
  starttime timestamp,
  stoptime timestamp,
  start_station_id integer,
  start_station_name string,
  start_station_latitude float,
  start_station_longitude float,
  end_station_id integer,
  end_station_name string,
  end_station_latitude float,
  end_station_longitude float,
  bikeid integer,
  membership_type string,
  usertype string,
  birth_year integer,
  gender integer);


-- Create the stage with the S3 bucket
CREATE or replace STAGE CITIBIKE_PIPELINES.PUBLIC.citibike_trips URL =
's3://snowflake-workshop-lab/citibike-trips-csv/';
```

```
list @citibike_trips;


-- Define the file format
create or replace FILE FORMAT CITIBIKE_PIPELINES.PUBLIC.CSV
    COMPRESSION = 'AUTO'
    FIELD_DELIMITER = ','
    RECORD_DELIMITER = '\n'
    SKIP_HEADER = 0
    FIELD_OPTIONALLY_ENCLOSED_BY = '\042'
    TRIM_SPACE = FALSE
    ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE
    ESCAPE = 'NONE'
    ESCAPE_UNENCLOSED_FIELD = '\134'
    DATE_FORMAT = 'AUTO'
    TIMESTAMP_FORMAT = 'AUTO'
    NULL_IF = ('');


alter warehouse DATAPIPELINES_WH set WAREHOUSE_SIZE = 'LARGE';
copy into trips from @citibike_trips file_format=CSV pattern = '.*.*[.]csv[.]gz';
alter warehouse DATAPIPELINES_WH set WAREHOUSE_SIZE = 'XSMALL';
```

2. Let’s test the caching features in Snowflake

```
--Run from Cold

SELECT *
FROM CITIBIKE_PIPELINES.PUBLIC.TRIPS;
```

We can look at the query results to observe how the query was processed.



	TRIPDURATION	STARTTIME	STOPTIME	START_STATION_ID	ST
1	240	2013-11-25 10:38:08.000	2013-11-25 10:42:08.000	290	2
2	226	2013-11-25 10:38:19.000	2013-11-25 10:42:05.000	336	Su
3	789	2013-11-25 10:38:47.000	2013-11-25 10:51:56.000	127	Ba
4	1,064	2013-11-25 10:38:54.000	2013-11-25 10:56:38.000	345	W
5	487	2013-11-25 10:38:55.000	2013-11-25 10:47:02.000	83	At
6	627	2013-11-25 10:38:55.000	2013-11-25 10:49:22.000	470	W
7	80	2013-11-25 10:38:57.000	2013-11-25 10:40:17.000	238	Ba

Partial results displayed

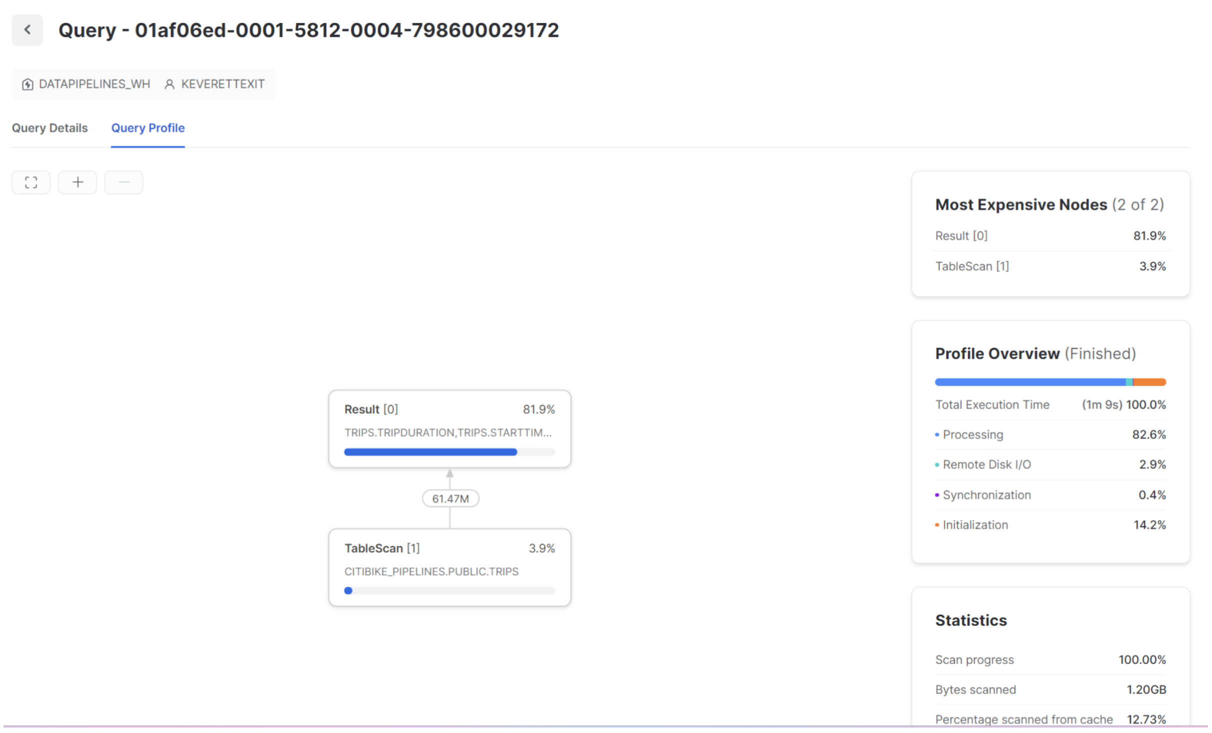
Only 10,000 rows of the results are displayed. Please download the results for all of the rows.

Query Details

Query duration 1m 9s

Rows 61.5M

Query ID 01af06ed-0001-5812-0...



Query - 01af06ed-0001-5812-0004-798600029172

DATAPIPELINES_WH KEVERETTEXIT

Query Details | Query Profile

Result [0] 81.9%
TRIPS.TRIPDURATION,TRIPS.STARTTIM...

61.47M

TableScan [1] 3.9%
CITIBIKE_PIPELINES.PUBLIC.TRIPS

Most Expensive Nodes (2 of 2)

Result [0]	81.9%
TableScan [1]	3.9%

Profile Overview (Finished)

Total Execution Time	(1m 9s) 100.0%
• Processing	82.6%
• Remote Disk I/O	2.9%
• Synchronization	0.4%
• Initialization	14.2%

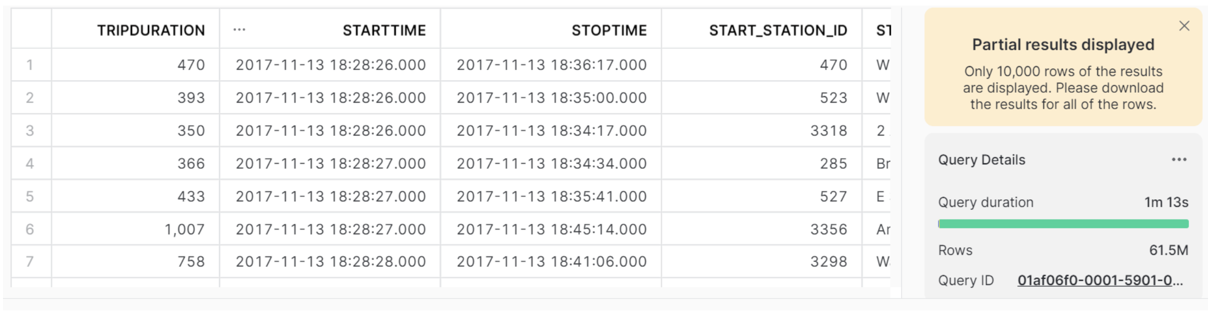
Statistics

Scan progress	100.00%
Bytes scanned	1.20GB
Percentage scanned from cache	12.73%

--Turn off results cache and only use local disk cache

Alter session set use_cached_result = false;

SELECT *
FROM CITIBIKE_PIPELINES.PUBLIC.TRIPS;



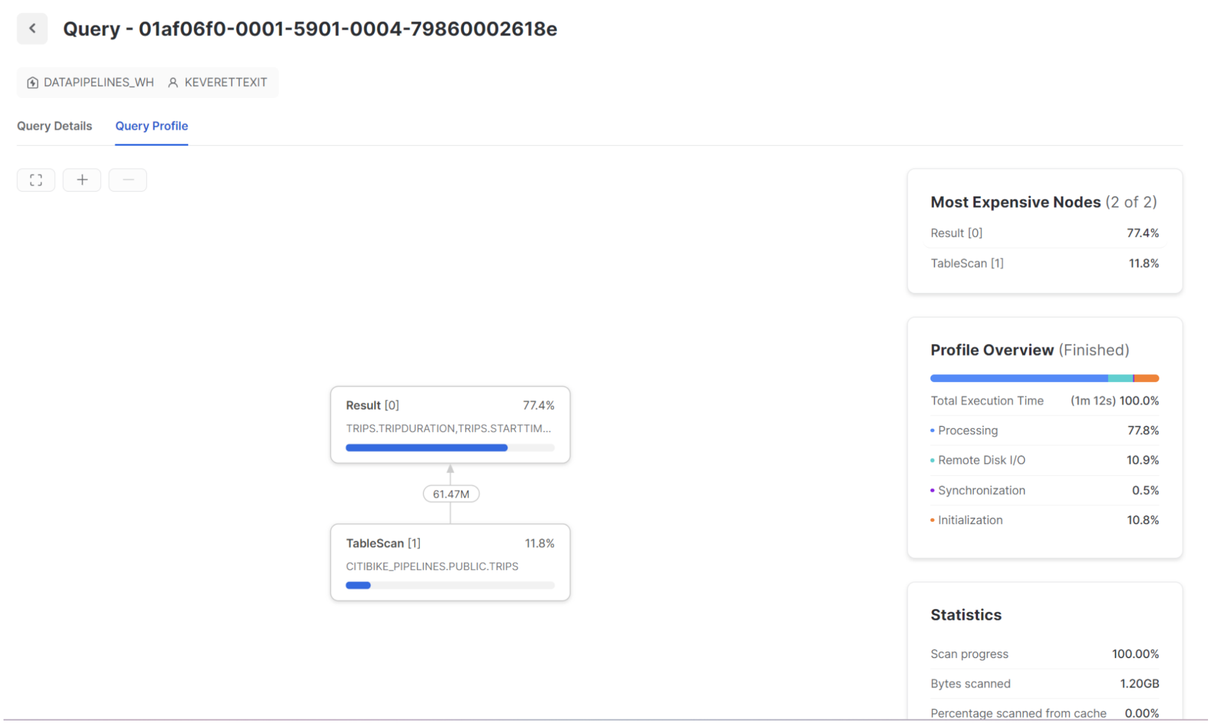
	TRIPDURATION	STARTTIME	STOPTIME	START_STATION_ID	ST
1	470	2017-11-13 18:28:26.000	2017-11-13 18:36:17.000	470	W
2	393	2017-11-13 18:28:26.000	2017-11-13 18:35:00.000	523	W
3	350	2017-11-13 18:28:26.000	2017-11-13 18:34:17.000	3318	2
4	366	2017-11-13 18:28:27.000	2017-11-13 18:34:34.000	285	Br
5	433	2017-11-13 18:28:27.000	2017-11-13 18:35:41.000	527	E
6	1,007	2017-11-13 18:28:27.000	2017-11-13 18:45:14.000	3356	Ar
7	758	2017-11-13 18:28:28.000	2017-11-13 18:41:06.000	3298	W

Partial results displayed

Only 10,000 rows of the results are displayed. Please download the results for all of the rows.

Query Details

Query duration	1m 13s
Rows	61.5M
Query ID	01af06f0-0001-5901-0...



Query - 01af06f0-0001-5901-0004-79860002618e

DATAPIPELINES_WH KEVERETTEXIT

Query Details Query Profile

Result [0] 77.4%
TRIPS.TRIPDURATION,TRIPS.STARTTIM...

61.47M

TableScan [1] 11.8%
CITIBIKE_PIPELINES.PUBLIC.TRIPS

Most Expensive Nodes (2 of 2)

Result [0]	77.4%
TableScan [1]	11.8%

Profile Overview (Finished)

Total Execution Time	(1m 12s) 100.0%
• Processing	77.8%
• Remote Disk I/O	10.9%
• Synchronization	0.5%
• Initialization	10.8%

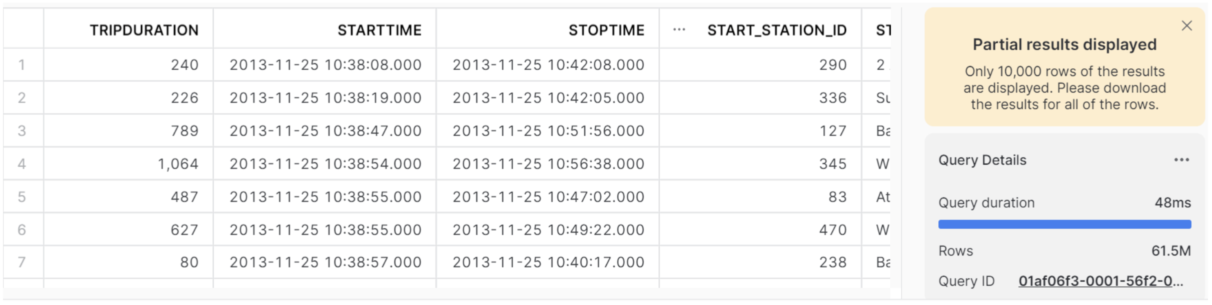
Statistics

Scan progress	100.00%
Bytes scanned	1.20GB
Percentage scanned from cache	0.00%

--Turn on results cache

Alter session set use_cached_result = true;

SELECT *
FROM CITIBIKE_PIPELINES.PUBLIC.TRIPS;



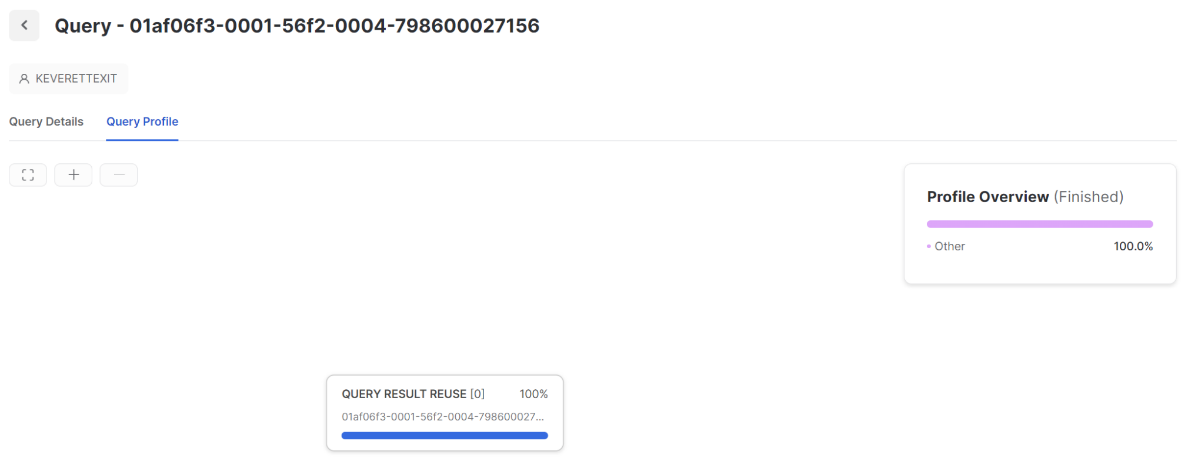
	TRIPDURATION	STARTTIME	STOPTIME	...	START_STATION_ID	ST
1	240	2013-11-25 10:38:08.000	2013-11-25 10:42:08.000		290	2
2	226	2013-11-25 10:38:19.000	2013-11-25 10:42:05.000		336	Su
3	789	2013-11-25 10:38:47.000	2013-11-25 10:51:56.000		127	Ba
4	1,064	2013-11-25 10:38:54.000	2013-11-25 10:56:38.000		345	W
5	487	2013-11-25 10:38:55.000	2013-11-25 10:47:02.000		83	At
6	627	2013-11-25 10:38:55.000	2013-11-25 10:49:22.000		470	W
7	80	2013-11-25 10:38:57.000	2013-11-25 10:40:17.000		238	Ba

Partial results displayed

Only 10,000 rows of the results are displayed. Please download the results for all of the rows.

Query Details

Query duration	48ms
Rows	61.5M
Query ID	01af06f3-0001-56f2-0...


Query - 01af06f3-0001-56f2-0004-798600027156

KEVERETTEXIT

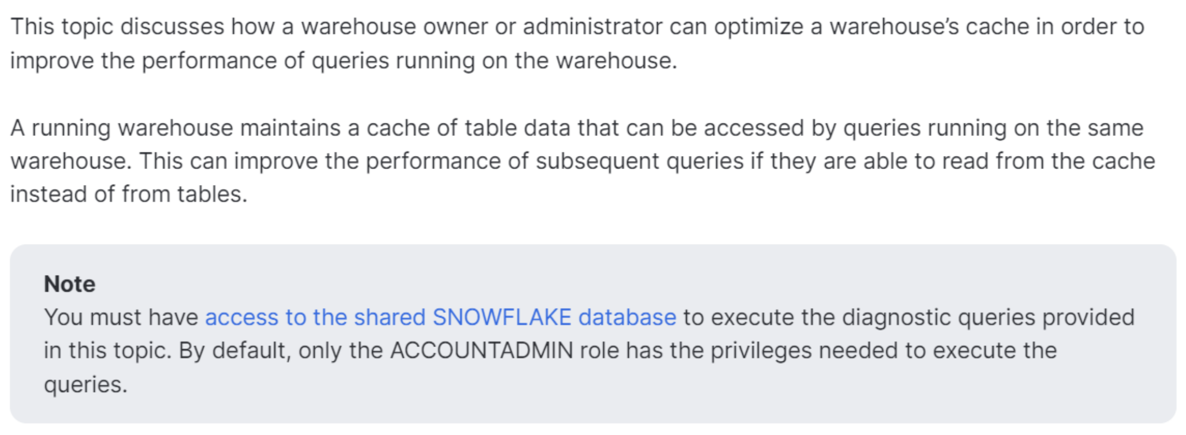
Profile Overview (Finished)

● Other 100.0%

QUERY RESULT REUSE [0] 100%
01af06f3-0001-56f2-0004-798600027...

```
-- Analyze Data Scanned from Cache

SELECT warehouse_name
  ,COUNT(*) AS query_count
  ,SUM(bytes_scanned) AS bytes_scanned
  ,SUM(bytes_scanned*percentage_scanned_from_cache) AS
bytes_scanned_from_cache
  ,SUM(bytes_scanned*percentage_scanned_from_cache) / SUM(bytes_scanned)
AS percent_scanned_from_cache
FROM snowflake.account_usage.query_history
WHERE start_time >= dateadd(month,-1,current_timestamp())
  AND bytes_scanned > 0
GROUP BY 1
ORDER BY 5;
```

Optimizing performance



This topic discusses how a warehouse owner or administrator can optimize a warehouse's cache in order to improve the performance of queries running on the warehouse.

A running warehouse maintains a cache of table data that can be accessed by queries running on the same warehouse. This can improve the performance of subsequent queries if they are able to read from the cache instead of from tables.

Note
You must have access to the shared SNOWFLAKE database to execute the diagnostic queries provided in this topic. By default, only the ACCOUNTADMIN role has the privileges needed to execute the queries.

1. Analyze Warehouse Usage Scanned from Cache

```
SELECT warehouse_name
  ,COUNT(*) AS query_count
  ,SUM(bytes_scanned) AS bytes_scanned
  ,SUM(bytes_scanned*percentage_scanned_from_cache) AS bytes_scanned_from_cache
  ,SUM(bytes_scanned*percentage_scanned_from_cache) / SUM(bytes_scanned) AS
percent_scanned_from_cache
FROM snowflake.account_usage.query_history
WHERE start_time >= dateadd(hour,-1,current_timestamp())
  AND bytes_scanned > 0
GROUP BY 1
ORDER BY 5;
```

About the Cache and Auto-suspension

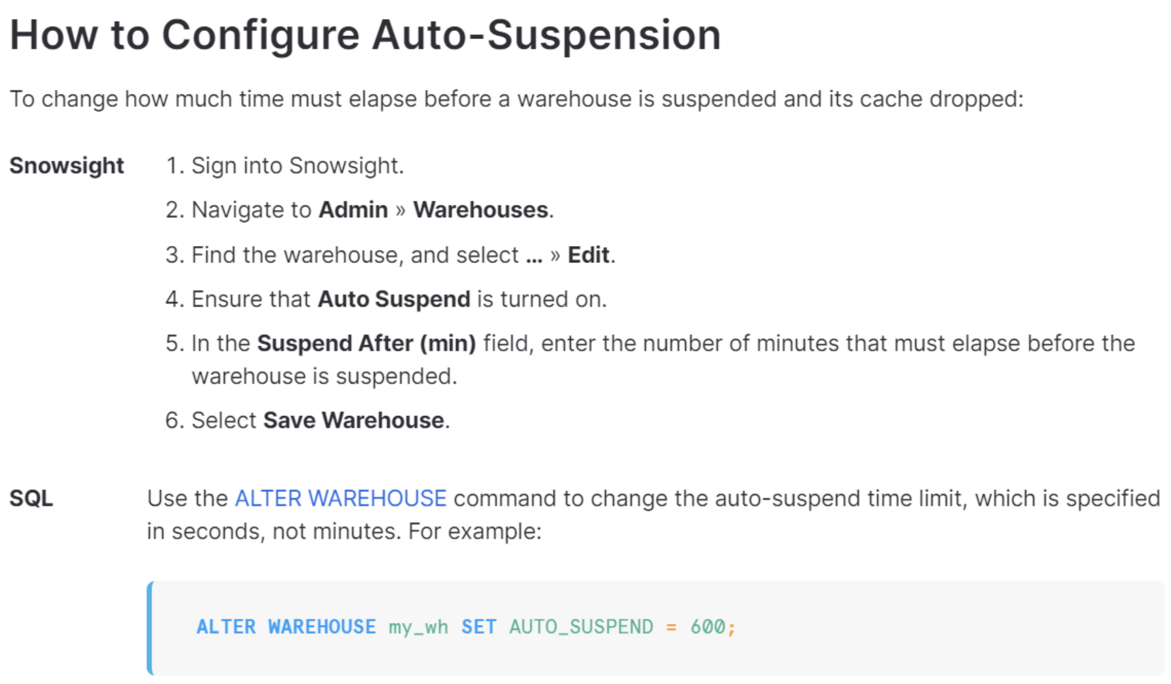
The auto-suspension setting of the warehouse can have a direct impact on query performance because the cache is dropped when the warehouse is suspended. If a warehouse is running frequent and similar queries, it might not make sense to suspend the warehouse in between queries because the cache might be dropped before the next query is executed.

You can use the following general guidelines when setting that auto-suspension time limit:

- For tasks, Snowflake recommends immediate suspension.
- For DevOps, DataOps and Data Science use cases, Snowflake recommends setting auto-suspension to approximately 5 minutes, as the cache is not as important for ad-hoc and unique queries.
- For query warehouses, for example BI and SELECT use cases, Snowflake recommends setting auto-suspension to at least 10 minutes to maintain the cache for users.

Cost Considerations

Keep in mind that a running warehouse consumes credits even if it is not processing queries. Be sure that your auto-suspension setting matches your workload. For example, if a warehouse executes a query every 30 minutes, it does not make sense to set the auto-suspension setting to 10 minutes. The warehouse will consume credits while sitting idle without gaining the benefits of a cache because it will be dropped before the next query executes.



How to Configure Auto-Suspension

To change how much time must elapse before a warehouse is suspended and its cache dropped:

Snowsight

1. Sign into Snowsight.
2. Navigate to **Admin** » **Warehouses**.
3. Find the warehouse, and select **...** » **Edit**.
4. Ensure that **Auto Suspend** is turned on.
5. In the **Suspend After (min)** field, enter the number of minutes that must elapse before the warehouse is suspended.
6. Select **Save Warehouse**.

SQL

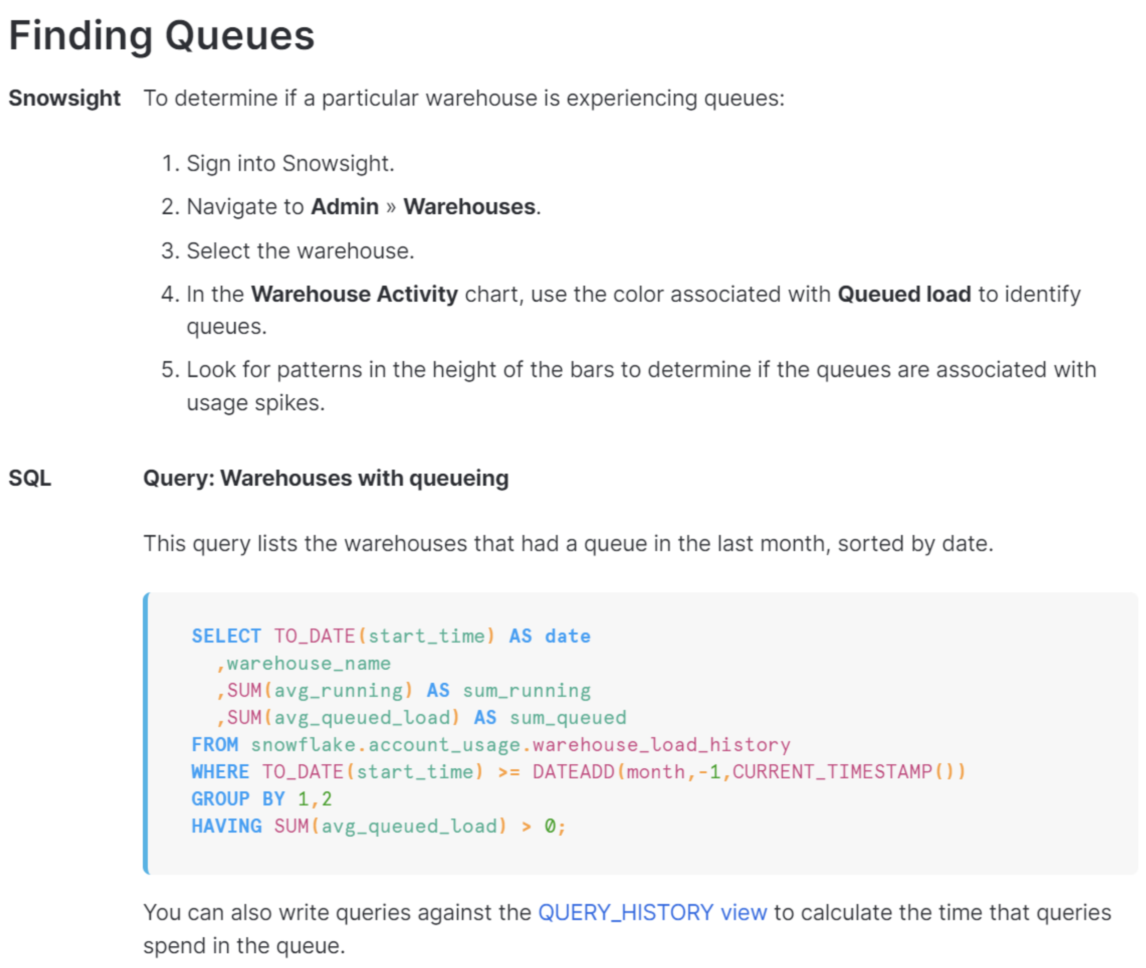
Use the ALTER WAREHOUSE command to change the auto-suspend time limit, which is specified in seconds, not minutes. For example:

```
ALTER WAREHOUSE my_wh SET AUTO_SUSPEND = 600;
```

Reducing Queues

This topic discusses how a warehouse owner or administrator can reduce queuing in order to improve the performance of queries running on a warehouse.

If too many queries are sent to a warehouse at the same time, the warehouse’s compute resources become exhausted and subsequent queries are queued until resources become available. The time between submitting a query and getting its results is longer when the query must wait in a queue before starting.



Finding Queues

Snowsight To determine if a particular warehouse is experiencing queues:

1. Sign into Snowsight.
2. Navigate to **Admin** » **Warehouses**.
3. Select the warehouse.
4. In the **Warehouse Activity** chart, use the color associated with **Queued load** to identify queues.
5. Look for patterns in the height of the bars to determine if the queues are associated with usage spikes.

SQL **Query: Warehouses with queueing**

This query lists the warehouses that had a queue in the last month, sorted by date.

```
SELECT TO_DATE(start_time) AS date
  ,warehouse_name
  ,SUM(avg_running) AS sum_running
  ,SUM(avg_queued_load) AS sum_queued
FROM snowflake.account_usage.warehouse_load_history
WHERE TO_DATE(start_time) >= DATEADD(month,-1,CURRENT_TIMESTAMP())
GROUP BY 1,2
HAVING SUM(avg_queued_load) > 0;
```

You can also write queries against the QUERY_HISTORY view to calculate the time that queries spend in the queue.

Resolving Memory Spillage

This topic discusses how a warehouse owner or administrator can resolve memory spillage in order to improve the performance of a query.

Performance degrades drastically when a warehouse runs out of memory while executing a query because memory bytes must “spill” onto local disk storage. If the query requires even more memory, it spills onto remote cloud-provider storage, which results in even worse performance.

```
-- Query Memory Spillage

SELECT query_id, SUBSTR(query_text, 1, 50) partial_query_text, user_name,
warehouse_name,
  bytes_spilled_to_local_storage, bytes_spilled_to_remote_storage
FROM  snowflake.account_usage.query_history
WHERE (bytes_spilled_to_local_storage > 0
  OR  bytes_spilled_to_remote_storage > 0 )
  AND start_time::date > dateadd('days', -45, current_date)
```

```
ORDER BY bytes_spilled_to_remote_storage, bytes_spilled_to_local_storage DESC
LIMIT 10;
```

Options for Resolving Memory Spillage

When memory spillage is the issue, you can convert your existing warehouse to a Snowpark-optimized warehouse, which provides 16x more memory per node and 10x the local cache compared to a standard warehouse. Though a larger warehouse also has more memory available, a query might not require its expanded compute resources.

If you want to try resolving the spillage of a query without adjusting the warehouse that runs it, use the Query Profile to identify which operation nodes are causing the spillage.

```
-- Create Snowpark optimized warehouses

ALTER WAREHOUSE DATAPIPELINES_WH SUSPEND;
ALTER WAREHOUSE DATAPIPELINES_WH SET
WAREHOUSE_TYPE='snowpark-optimized';
```

What is a Multi-cluster Warehouse?

By default, a virtual warehouse consists of a single cluster of compute resources available to the warehouse for executing queries. As queries are submitted to a warehouse, the warehouse allocates resources to each query and begins executing the queries. If sufficient resources are not available to execute all the queries submitted to the warehouse, Snowflake queues the additional queries until the necessary resources become available.

With multi-cluster warehouses, Snowflake supports allocating, either statically or dynamically, additional clusters to make a larger pool of compute resources available. A multi-cluster warehouse is defined by specifying the following properties:

- Maximum number of clusters, greater than 1 (up to 10).
- Minimum number of clusters, equal to or less than the maximum (up to 10).

Additionally, multi-cluster warehouses support all the same properties and actions as single-cluster warehouses, including:

- Specifying a warehouse size.
- Resizing a warehouse at any time.
- Auto-suspending a running warehouse due to inactivity; note that this does not apply to individual clusters, but rather the entire multi-cluster warehouse.
- Auto-resuming a suspended warehouse when new queries are submitted.

2. Create multi-clusters

```
CREATE WAREHOUSE IF NOT EXISTS MULTICLUSTER_WH
   WITH WAREHOUSE_SIZE = 'XSMALL'
   AUTO_SUSPEND = 60
   AUTO_RESUME = TRUE
   MAX_CLUSTER_COUNT = 4
   ;

SHOW WAREHOUSES;
```

Maximized vs. Auto-scale

You can choose to run a multi-cluster warehouse in either of the following modes:

Maximized
This mode is enabled by specifying the ***same*** value for both maximum and minimum number of clusters (note that the specified value must be larger than 1). In this mode, when the warehouse is started, Snowflake starts all the clusters so that maximum resources are available while the warehouse is running.

This mode is effective for statically controlling the available compute resources, particularly if you have large numbers of concurrent user sessions and/or queries and the numbers do not fluctuate significantly.

Auto-scale
This mode is enabled by specifying ***different*** values for maximum and minimum number of clusters. In this mode, Snowflake starts and stops clusters as needed to dynamically manage the load on the warehouse:

- As the number of concurrent user sessions and/or queries for the warehouse increases, and queries start to queue due to insufficient resources, Snowflake automatically starts additional clusters, up to the maximum number defined for the warehouse.
- Similarly, as the load on the warehouse decreases, Snowflake automatically shuts down clusters to reduce the number of running clusters and, correspondingly, the number of credits used by the warehouse.

To help control the usage of credits in Auto-scale mode, Snowflake provides a property, SCALING_POLICY, that determines the scaling policy to use when automatically starting or shutting down additional clusters. For more information, see Setting the Scaling Policy for a Multi-cluster Warehouse (in this topic).

```
-- You can add a scaling policy to the multicluster to shift from maximinized to auto-scale

ALTER WAREHOUSE MULTICLUSTER_WH SCALING_POLICY = 'STANDARD'  -- Standard policy

ALTER WAREHOUSE MULTICLUSTER_WH SCALING_POLICY = 'ECONOMY'  -- Economy policy
```

Working with Resource Monitors

To help control costs and avoid unexpected credit usage caused by running warehouses, Snowflake provides *resource monitors*. A virtual warehouse consumes Snowflake credits while it runs.

A resource monitor can be used to monitor credit usage by virtual warehouses and the cloud services needed to support those warehouses. If desired, the warehouse can be suspended when it reaches a credit limit.

The number of credits consumed depends on the size of the warehouse and how long it runs.

Limits can be set for a specified interval or date range. When these limits are reached and/or are approaching, the resource monitor can trigger various actions, such as sending alert notifications and/or suspending user-managed warehouses.

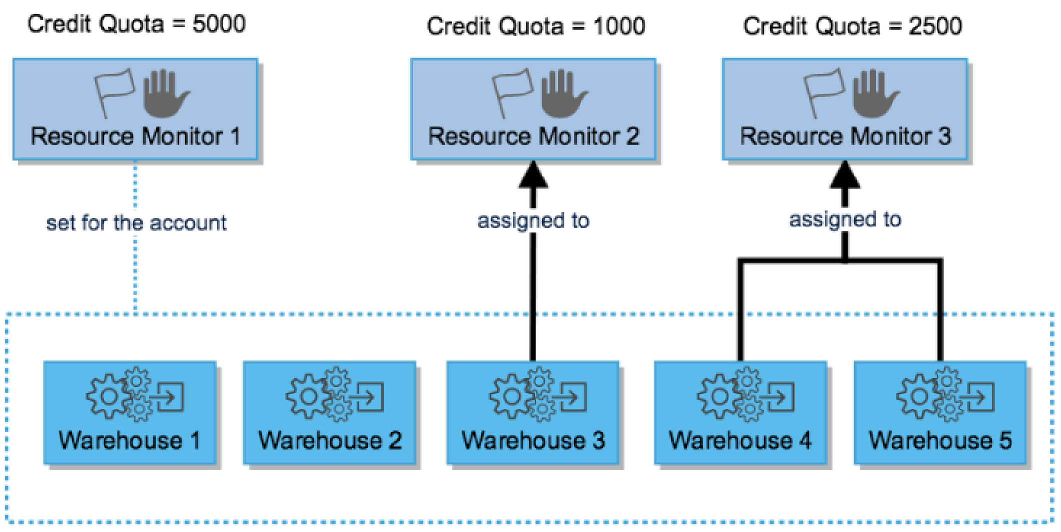
Resource monitors can only be created by account administrators (i.e. users with the ACCOUNTADMIN role); however, account administrators can choose to enable users with other roles to view and modify resource monitors using SQL.

Assignment of Resource Monitors

A single monitor can be set at the account level to control credit usage for all warehouses in your account.

In addition, a monitor can be assigned to one or more warehouses, thereby controlling the credit usage for each assigned warehouse. Note, however, that a warehouse can be assigned to only a single resource monitor below the account level.

The following diagram illustrates a scenario in which one resource monitor is set at the account level and individual warehouses are assigned to two other resource monitors:



Based on this diagram:

- The credit quota for the entire account is 5000 for the interval (month, week, etc.), as controlled by Resource Monitor 1; if this quota is reached within the interval, the actions defined for the resource monitor (**Suspend**, **Suspend Immediate**, etc.) are enforced for all five warehouses.
- Warehouse 3 can consume a maximum of 1000 credits within the interval.
- Warehouse 4 and 5 can consume a maximum combined total of 2500 credits within the interval.

Note that the actual credits consumed by Warehouses 3, 4, and 5 may be less than their quotas if the quota for the account is reached first.

3. Assign Resource Monitor on Warehouse

```
create or replace resource monitor limiter with credit_quota=5000
  notify_users = (jdoe, "jane smith", "john doe")
  triggers on 75 percent do notify
           on 100 percent do suspend
           on 110 percent do suspend_immediate;

ALTER WAREHOUSE MULTICLUSTER_WH SET RESOURCE_MONITOR = limiter;

USE WAREHOUSE COMPUTE_WH;
```