Understanding Your Snowflake Utilization, Part 1: Warehouse Profiling

This series will take a deeper dive into the Information Schema (Snowflake's data dictionary) and show you some practical ways to use this data to better understand your usage of Snowflake.

In this lab, we will discuss how they can implement profiling for your Snowflake account on your own.

The answer is to utilize the Information Schema. Aside from providing a set of detailed views into Snowflake's metadata store, the Information Schema goes a step further and provides several powerful table functions that can be called directly in SQL. These functions can be used to return historical information about executed queries, data stored in databases and stages, and virtual warehouse (i.e. compute) usage.

In addition to these functions, I also recommend leveraging the recently implemented TABLE_STORAGE_METRICS view (also in the Information Schema) to dive even deeper into your analysis.

In this lab, I will show you how to leverage these easy-to-use function to gather detailed information about the usage of your virtual warehouses. So let's get started.

Warehouse Profiling

To profile your current warehouse usage, use the [WAREHOUSE_LOAD_HISTORY] and [WAREHOUSE_METERING_HISTORY] functions. A good way to think about the relationship between these two functions is that the first one shows how much work was done over a period of time (**load**) and the second one shows the cost for doing the work (**metering**).

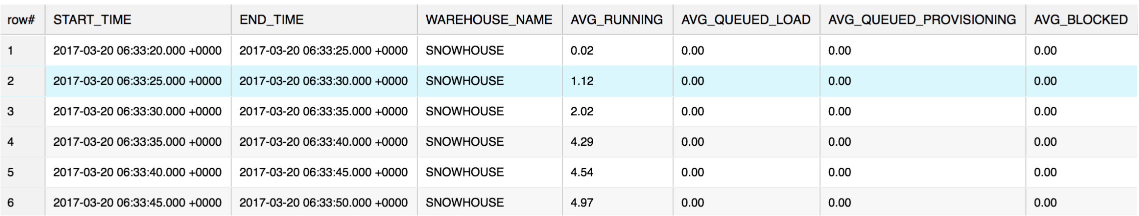
The syntax for calling these functions is simple, and can be executed in the **Worksheet** in the Snowflake web interface. For example:

```
use warehouse mywarehouse;

select * from
table(information_schema.warehouse_load_history(date_range_start=>dateadd('hour',-1,curr


select * from
table(information_schema.warehouse_metering_history(dateadd('hour',-1,current_date()),cu
```

The above queries show warehouse load and credits used for the past hour for all your warehouses. Be sure to check out the *Usage Notes* section (in the documentation) for each function to understand all the requirements and rules. For example, the WAREHOUSE_LOAD_HISTORY function returns results in different intervals based on the timeframe you specify:

- 5-second intervals when the timeframe is less than 7 hours.
- 5-minute intervals when the timeframe is greater than 7 hours.

Here's an example of the output from the WAREHOUSE_LOAD_HISTORY query against SNOWHOUSE, a warehouse that we use internally:

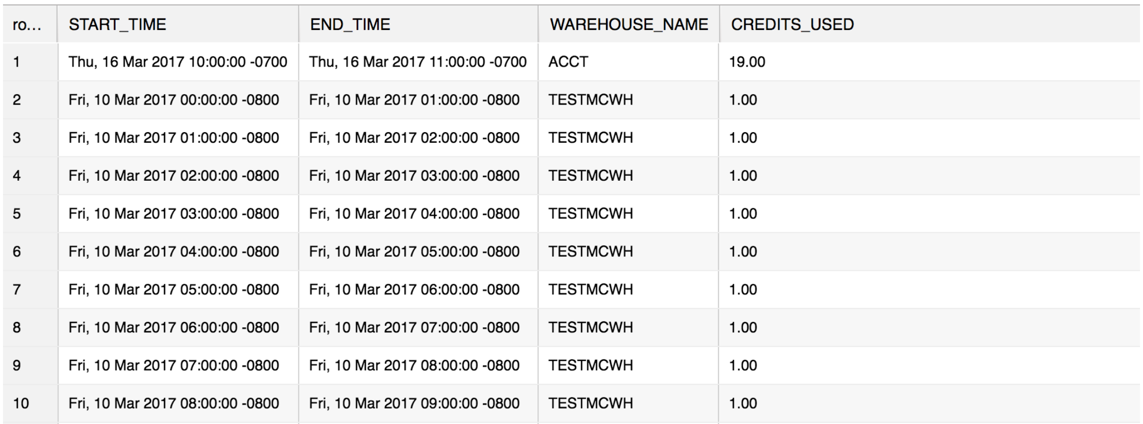


row#	START_TIME	END_TIME	WAREHOUSE_NAME	AVG_RUNNING	AVG_QUEUED_LOAD	AVG_QUEUED_PROVISIONING	AVG_BLOCKED
1	2017-03-20 06:33:20.000 +0000	2017-03-20 06:33:25.000 +0000	SNOWHOUSE	0.02	0.00	0.00	0.00
2	2017-03-20 06:33:25.000 +0000	2017-03-20 06:33:30.000 +0000	SNOWHOUSE	1.12	0.00	0.00	0.00
3	2017-03-20 06:33:30.000 +0000	2017-03-20 06:33:35.000 +0000	SNOWHOUSE	2.02	0.00	0.00	0.00
4	2017-03-20 06:33:35.000 +0000	2017-03-20 06:33:40.000 +0000	SNOWHOUSE	4.29	0.00	0.00	0.00
5	2017-03-20 06:33:40.000 +0000	2017-03-20 06:33:45.000 +0000	SNOWHOUSE	4.54	0.00	0.00	0.00
6	2017-03-20 06:33:45.000 +0000	2017-03-20 06:33:50.000 +0000	SNOWHOUSE	4.97	0.00	0.00	0.00

Per our documentation:

- AVG_RUNNING -- Average number of queries executed.
- AVG_QUEUE_LOAD -- Average number of queries queued because the warehouse was overloaded.
- AVG_QUEUE_PROVISION -- Average number of queries queued because the warehouse was being provisioned.
- AVG_BLOCKED -- Average number of queries blocked by a transaction lock.

And here's an example of the output from the WAREHOUSE_METERING_HISTORY query against SNOWHOUSE:



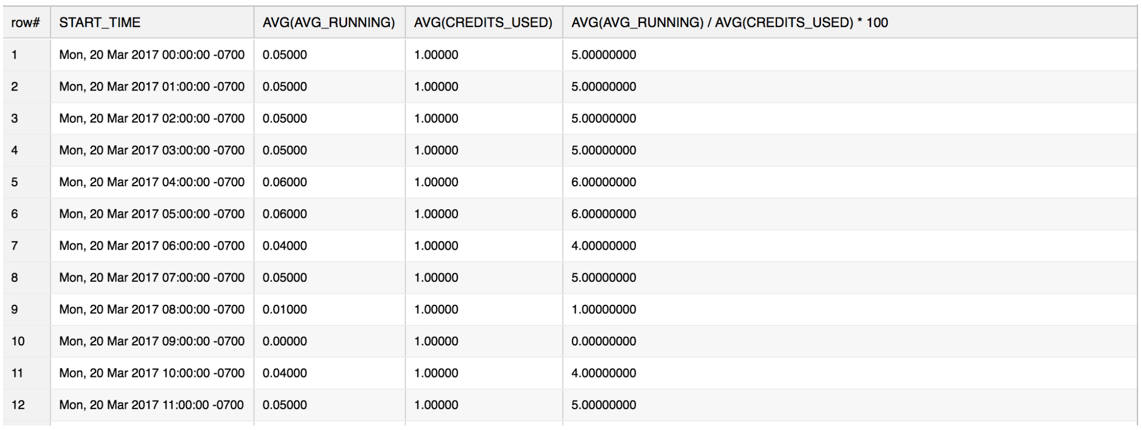
ro...	START_TIME	END_TIME	WAREHOUSE_NAME	CREDITS_USED
1	Thu, 16 Mar 2017 10:00:00 -0700	Thu, 16 Mar 2017 11:00:00 -0700	ACCT	19.00
2	Fri, 10 Mar 2017 00:00:00 -0800	Fri, 10 Mar 2017 01:00:00 -0800	TESTMCWH	1.00
3	Fri, 10 Mar 2017 01:00:00 -0800	Fri, 10 Mar 2017 02:00:00 -0800	TESTMCWH	1.00
4	Fri, 10 Mar 2017 02:00:00 -0800	Fri, 10 Mar 2017 03:00:00 -0800	TESTMCWH	1.00
5	Fri, 10 Mar 2017 03:00:00 -0800	Fri, 10 Mar 2017 04:00:00 -0800	TESTMCWH	1.00
6	Fri, 10 Mar 2017 04:00:00 -0800	Fri, 10 Mar 2017 05:00:00 -0800	TESTMCWH	1.00
7	Fri, 10 Mar 2017 05:00:00 -0800	Fri, 10 Mar 2017 06:00:00 -0800	TESTMCWH	1.00
8	Fri, 10 Mar 2017 06:00:00 -0800	Fri, 10 Mar 2017 07:00:00 -0800	TESTMCWH	1.00
9	Fri, 10 Mar 2017 07:00:00 -0800	Fri, 10 Mar 2017 08:00:00 -0800	TESTMCWH	1.00
10	Fri, 10 Mar 2017 08:00:00 -0800	Fri, 10 Mar 2017 09:00:00 -0800	TESTMCWH	1.00

Now that we know the amount of work that was performed during the time period (via WAREHOUSE_LOAD_HISTORY) and the cost per time period (via WAREHOUSE_METERING_HISTORY), we can perform a simple efficiency ratio calculation for a particular warehouse. This example returns this information for a warehouse named XSMALL:

```
with cte as (
  select date_trunc('hour', start_time) as start_time, end_time, warehouse_name,
credits_used
  from
table(information_schema.warehouse_metering_history(dateadd('days',-1,current_date()),cu

  where warehouse_name = 'XSMALL')
select date_trunc('hour', a.start_time) as start_time, avg(AVG_RUNNING),
avg(credits_used), avg(AVG_RUNNING) / avg(credits_used) * 100
from
table(information_schema.warehouse_load_history(dateadd('days',-1,current_date()),curren
 a
join cte b on a.start_time = date_trunc('hour', a.start_time)
where a.warehouse_name = 'XSMALL'
```

```
group by 1
order by 1;
```

In the above query, we are treating the average of AVG_RUNNING as work and the average of CREDITS_USED as cost and we apply a simple efficiency ratio on both of these values. Feel free to experiment any way you like.

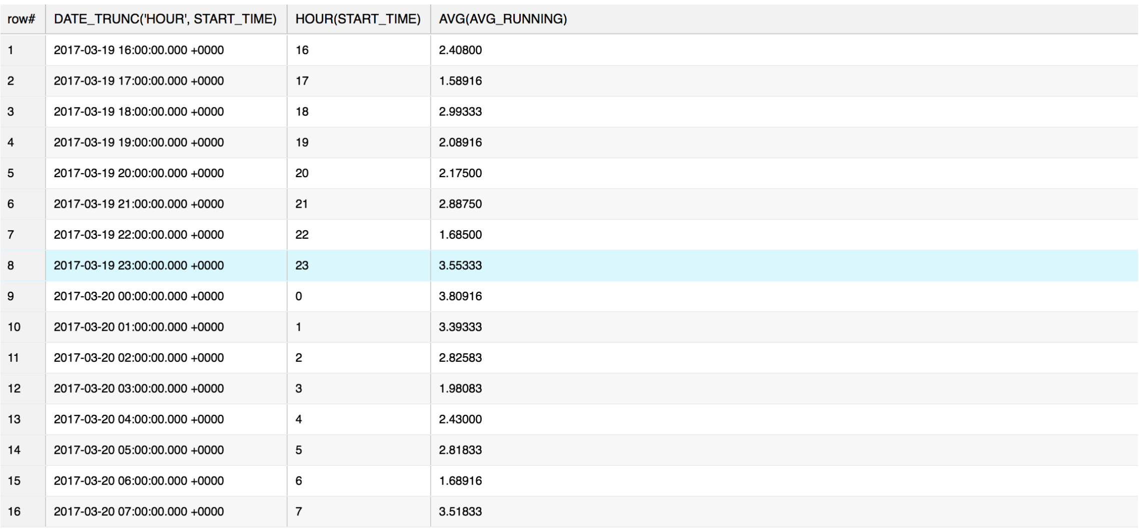


row#	START_TIME	AVG(AVG_RUNNING)	AVG(CREDITS_USED)	AVG(AVG_RUNNING) / AVG(CREDITS_USED) * 100
1	Mon, 20 Mar 2017 00:00:00 -0700	0.05000	1.00000	5.00000000
2	Mon, 20 Mar 2017 01:00:00 -0700	0.05000	1.00000	5.00000000
3	Mon, 20 Mar 2017 02:00:00 -0700	0.05000	1.00000	5.00000000
4	Mon, 20 Mar 2017 03:00:00 -0700	0.05000	1.00000	5.00000000
5	Mon, 20 Mar 2017 04:00:00 -0700	0.06000	1.00000	6.00000000
6	Mon, 20 Mar 2017 05:00:00 -0700	0.06000	1.00000	6.00000000
7	Mon, 20 Mar 2017 06:00:00 -0700	0.04000	1.00000	4.00000000
8	Mon, 20 Mar 2017 07:00:00 -0700	0.05000	1.00000	5.00000000
9	Mon, 20 Mar 2017 08:00:00 -0700	0.01000	1.00000	1.00000000
10	Mon, 20 Mar 2017 09:00:00 -0700	0.00000	1.00000	0.00000000
11	Mon, 20 Mar 2017 10:00:00 -0700	0.04000	1.00000	4.00000000
12	Mon, 20 Mar 2017 11:00:00 -0700	0.05000	1.00000	5.00000000

Next, let's talk about the specific use of WAREHOUSE_LOAD_HISTORY in our example above:

```
select date_trunc('hour', start_time), hour(start_time), avg(avg_running)
from
table(information_schema.warehouse_load_history(date_range_start=>dateadd('day',-1,curre

group by date_trunc('hour', start_time), hour(start_time)
order by date_trunc('hour', start_time) asc;
```

Here is the output:

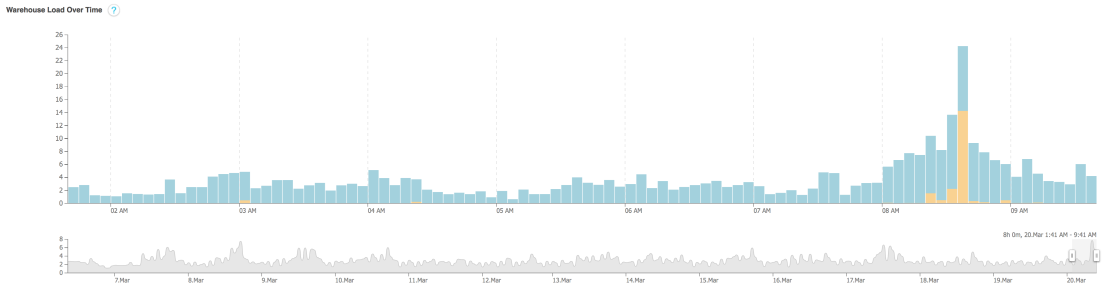


row#	DATE_TRUNC('HOUR', START_TIME)	HOUR(START_TIME)	AVG(AVG_RUNNING)
1	2017-03-19 16:00:00.000 +0000	16	2.40800
2	2017-03-19 17:00:00.000 +0000	17	1.58916
3	2017-03-19 18:00:00.000 +0000	18	2.99333
4	2017-03-19 19:00:00.000 +0000	19	2.08916
5	2017-03-19 20:00:00.000 +0000	20	2.17500
6	2017-03-19 21:00:00.000 +0000	21	2.88750
7	2017-03-19 22:00:00.000 +0000	22	1.68500
8	2017-03-19 23:00:00.000 +0000	23	3.55333
9	2017-03-20 00:00:00.000 +0000	0	3.80916
10	2017-03-20 01:00:00.000 +0000	1	3.39333
11	2017-03-20 02:00:00.000 +0000	2	2.82583
12	2017-03-20 03:00:00.000 +0000	3	1.98083
13	2017-03-20 04:00:00.000 +0000	4	2.43000
14	2017-03-20 05:00:00.000 +0000	5	2.81833
15	2017-03-20 06:00:00.000 +0000	6	1.68916
16	2017-03-20 07:00:00.000 +0000	7	3.51833

In this case, I'm indeed asking for an average of an average. I'm grouping the values by hours so I can get a general overview of my warehouse workload. I can see my warehouse is working almost a full day. However, if I see some time gaps in this output, then I might do some additional investigation around those times and see if the warehouse should be doing work.

Another thing you can see in the output from this function is whether these time gaps repeat over a few days. If they do, then I would recommend that you script the warehouse to sleep when not in use (i.e. to save money), or enable AUTO_SUSPEND and AUTO_RESUME for that warehouse.

The Snowflake web interface also has a nice visual representation of this function (under the **Warehouse** tab):



Whether you use the visual chart or the manual query, for the four available metrics, pay particular attention to AVG_RUNNING. This should give you an idea how each warehouse performs. If you have split your workload across several different warehouses, it should tell you how well your queries are distributed.

AVG_QUEUE_LOAD and AVG_BLOCKED are also interesting and should provide you with good insight about how well your warehouses are sized. Keep in mind that queuing is not necessarily a bad thing and you shouldn't expect zero queuing. The idea is to accept a certain amount of queuing per time period based on your usage requirements.

Using these metrics, you can determine what to do:

- Increasing the warehouse size will provide more throughput in processing the queries and thereby can help reduce the queuing time.
- Increasing the cluster count (if using a multi-cluster warehouse) will allow more concurrency, which should also help reduce queuing and blocking.

Finding an Underutilized Warehouse

Is there a warehouse that's underutilized? For example, any similar sized warehouses being shared across several users could potentially be consolidated to a single warehouse. You can surface this information by comparing your AVG_RUNNING and AVG_QUEUE_LOAD scores across your warehouses:

- If you see a warehouse with a very low number of queries running, you may want to turn that warehouse off and redirect the queries to another less used warehouse.
- If a warehouse is running queries and queuing, perhaps it's time to review your workflow to increase your warehouse sizes.
- If you have built your own client application to interface with Snowflake, reviewing your client scripts / application code should also reveal any biases towards one warehouse over another.