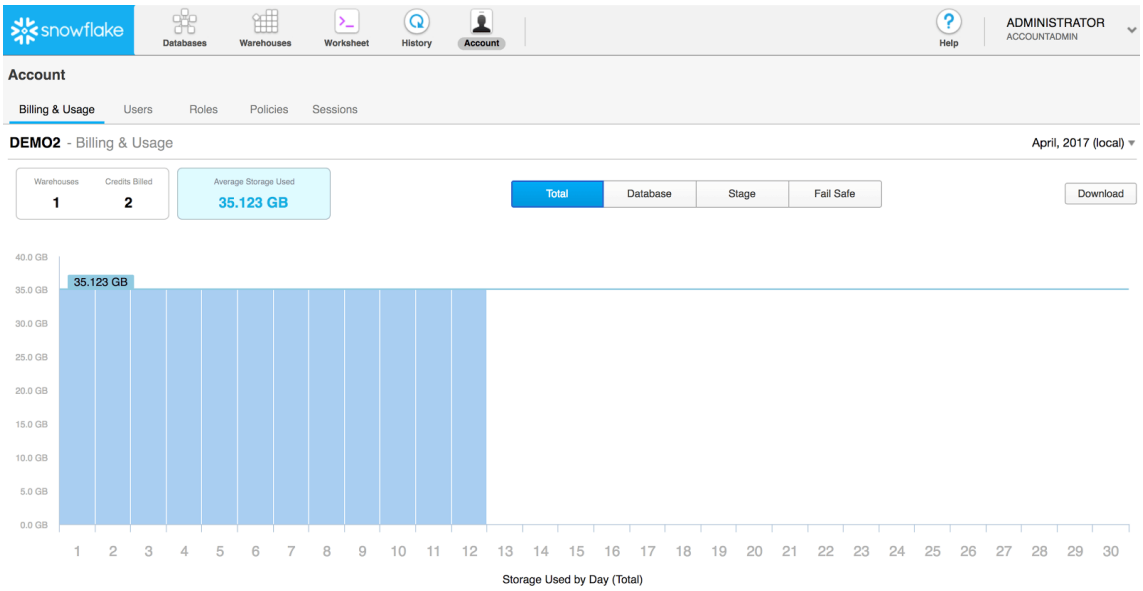
Understanding Your Snowflake Utilization, Part 2: Storage Profiling

In this lab, I provide a deep-dive into understanding how you are utilizing data storage in Snowflake at the database, stage, and table level. To do this, I will show you examples of two functions and a view provided in the Information Schema for monitoring storage usage. I will also show you a handy page in the UI that provides an account-level view of your storage. Keep in mind that you need ACCOUNTADMIN access to perform any of the tasks described in this lab.

Let's get started.

Summary Storage Profiling in the UI

Before diving into our detailed analysis of data storage, let's take a quick look at the summary, account-level storage view provided by Snowflake. As a user with the ACCOUNTADMIN role, you can navigate to the **Account** page in the Snowflake UI to get a visual overview of the data storage for your account.



This page provides a view, by month, of the average and daily storage usage across your entire account. You can use the filters on the page to filter by database, Snowflake stage, and data maintained in Fail-safe (for disaster recovery).

Detailed Storage Profiling Using the Information Schema

The Snowflake Information Schema provides two functions and one view for monitoring detailed data storage at the database, stage, and table level:

- DATABASE_STORAGE_USAGE_HISTORY (function)
- STAGE_STORAGE_USAGE_HISTORY (function)
- TABLE_STORAGE_METRICS (view)

The DATABASE_STORAGE_USAGE_HISTORY table function shows your database status and usage for all databases in your account or a specified database. Here's an example of the usage over the last 10 days for a database named `sales`:

```
use warehouse mywarehouse;

select * from
table(sales.information_schema.database_storage_usage_history(dateadd('days',-10,current
 'SALES'));
```

row#	USAGE_DATE	DATABASE_NAME	AVERAGE_DATABASE_BYTES	AVERAGE_FAILSAFE_BYTES
1	2017-03-31	SALES	5856246781	0
2	2017-04-01	SALES	5856246782	0
3	2017-04-02	SALES	5856246781	0
4	2017-04-03	SALES	5856246782	0
5	2017-04-04	SALES	5856246782	0
6	2017-04-05	SALES	5856246781	0
7	2017-04-06	SALES	5856246782	0
8	2017-04-07	SALES	5856246781	0
9	2017-04-08	SALES	5856246782	0
10	2017-04-09	SALES	5856246781	0
11	2017-04-10	SALES	5856246782	0

Note that the above screenshot only displays some of the output columns. Also, per the Snowflake documentation:

> *If a database has been dropped and its data retention period has passed (i.e. database cannot be recovered using Time Travel), then the database name is reported as DROPPED_id.*

At its core, the most useful insight from this function is the average growth in your database. Keep in mind, the output includes both AVERAGE_DATABASE_BYTES and AVERAGE_FAILSAFE_BYTES. Leveraging these data points to derive a percentage of `Fail-safe` over actual database size should give you an idea of how much you should be investing towards your Fail-safe storage. If certain data is not mission critical and doesn't require Fail-safe, try setting these tables to `transient`. More granular information about Fail-safe data is provided in TABLE_STORAGE_METRICS, which we will look at more closely later in this lab.

Next, let's look at STAGE_STORAGE_USAGE_HSTORY. This function shows you how much storage is being used for staged files across ***all*** your Snowflake staging locations, including named, internal *stages*. Note that this function does not allow querying storage on individual stages.

Here's an example of staged file usage for the last 10 days (using the same database, `sales`, from the previous example):

```
select * from
table(sales.information_schema.stage_storage_usage_history(dateadd('days',-10,current_da
```

row#	USAGE_DATE	AVERAGE_STAGE_BYTES
1	2017-03-31	448284316
2	2017-04-01	448284317
3	2017-04-02	448284316
4	2017-04-03	448284316
5	2017-04-04	448284317
6	2017-04-05	448284316
7	2017-04-06	448284317
8	2017-04-07	448284317
9	2017-04-08	448284316
10	2017-04-09	448284317
11	2017-04-10	448284318

Note that the above screenshot only displays some of the output columns.

Also note that you can only query up to 6 months worth of data using this function. Some of our users like to use Snowflake stages to store their raw data. For example, one user leverages table staging locations for their raw data storage just in case they need to access the data in the future. There's nothing wrong with this approach, and since Snowflake compresses your staged data files, it certainly makes sense; however, only the last 6 months of staged data storage is available.

Finally, the TABLE_STORAGE_METRICS view shows your table-level storage at runtime. This is a snapshot of your table storage which includes your active and Fail-safe storage. Additionally, you can derive cloned storage as well utilizing the CLONE_GROUP_ID column. As of today, this is the most granular level of storage detail available to users.

Here's a general use example (using the `sales` database):

```
select *
from sales.information_schema.table_storage_metrics
where table_catalog = 'SALES';
```

row#	TABLE_CAT...	TABLE_SC...	TABLE_NA...	ID	CLONE_GR...	IS_TRANSI...	ACTIVE_RO..	ACTIVE_BY...	TIME_TRAV...	FAILSAFE_...	OWNED_A...	TABLE_CR...	TABLE_DR...	TABLE_ENT..	CATALOG_...	CATALOG_...
1	SALES	PUBLIC	NATION	69646	69646	NO	0	0	0	0	0	2016-10-06 1...	NULL	NULL	2016-10-06 1...	NULL
2	SALES	PUBLIC	CUSTOMER	69648	69648	NO	15050000	1089571840	0	0	1089571840	2016-10-06 1...	NULL	NULL	2016-10-06 1...	NULL
3	SALES	PUBLIC	SUPPLIER	69650	69650	NO	0	0	0	0	0	2016-10-06 1...	NULL	NULL	2016-10-06 1...	NULL
4	SALES	PUBLIC	PRODUCT	69652	69652	NO	1	3584	0	0	3584	2016-10-06 1...	NULL	NULL	2016-10-06 1...	NULL
5	SALES	PUBLIC	PRODUCTH...	70669	70669	NO	1	1024	0	0	1024	2016-10-06 1...	NULL	NULL	2016-10-06 1...	NULL
6	SALES	PUBLIC	ORDERS	70671	70671	NO	150000000	4766642688	0	0	4766642688	2016-10-06 1...	NULL	NULL	2016-10-06 1...	NULL
7	SALES	PUBLIC	LINEITEM	70673	70673	NO	5302	26624	0	0	26624	2016-10-06 1...	NULL	NULL	2016-10-06 1...	NULL
8	SALES	PUBLIC	_after_reset	70675	70675	NO	1	1024	0	0	1024	2016-10-06 1...	NULL	NULL	2016-10-06 1...	NULL
9	SALES	PUBLIC	REGION	71688	71688	NO	0	0	0	0	0	2016-10-06 1...	NULL	NULL	2016-10-06 1...	NULL
10	SALES	PUBLIC	PRODUCTS...	71690	71690	NO	0	0	0	0	0	2016-10-06 1...	NULL	NULL	2016-10-06 1...	NULL

Note that the above screenshot only shows a portion of the output columns.

In Snowflake, cloning data has no additional costs (until the data is modified or deleted) and it's done very quickly. All users benefit from "zero-copy cloning", but some are curious to know exactly what percentage of their table storage actually came from cloned data. To determine this, we'll leverage the CLONE_GROUP_ID column in TABLE_STORAGE_METRICS.

For example (using a database named `concurrency_wh`):

```
with storage_sum as (
  select clone_group_id,
         sum(owned_active_and_time_travel_bytes) as owned_bytes,
```

```
        sum(active_bytes) + sum(time_travel_bytes) as referred_bytes
  from concurrency_wh.information_schema.table_storage_metrics
  where active_bytes > 0
  group by 1)
select * , referred_bytes / owned_bytes as ratio
from storage_sum
where referred_bytes > 0 and ratio > 1
order by owned_bytes desc;
```

row#	CLONE_GROUP_ID	OWNED_BYTES	REFERRED_BYTES	RATIO
1	12292	41606396613632	49438168488960	1.188234803
2	1146387	6983569222656	10792538451456	1.545418697
3	186519	4700058698240	9124382190592	1.941333668
4	186542	1377472353280	2699929145344	1.960060497
5	184490	1077942566912	2155885121536	1.999999989
6	149508	465707046400	820531732480	1.761905341
7	186536	213844349440	328041306112	1.534019052
8	184454	107014242304	205778106368	1.922903923
9	746268	51754352128	121072992768	2.339378
10	15366	48444371456	82108175872	1.694896092
11	184452	31855170560	57569825792	1.807236464
12	349800	30342430720	91027292160	3
13	75778	20327801856	36357701120	1.788570224
14	163232	15405162496	15405174784	1.000000798
15	164389	14270820352	21313835008	1.493525564
16	349844	11844087808	35532263424	3

The ratio in the above query gives you an idea of how much of the original data is being "referred to" by the clone. In general, when you make a clone of a table, the CLONE_GROUP_ID for the original table is assigned to the new, cloned table. As you perform DML on the new table, your REFERRED_BYTES value gets updated. If you join the CLONE_GROUP_ID back into the original view, you get the output of the original table along with the cloned table. A ratio of 1 in the above example means the table data is not cloned.

If you need to find out the exact table name from the above query, then simply join the CTE back to the TABLE_STORAGE_METRICS view and ask for the TABLE_NAME column.

For example (using the same database, `concurrency_wh`, from the previous example):

```
with storage_sum as (
  select clone_group_id,
         sum(owned_active_and_time_travel_bytes) as owned_bytes,
         sum(active_bytes) + sum(time_travel_bytes) as referred_bytes
  from concurrency_wh.information_schema.table_storage_metrics
  where active_bytes > 0
  group by 1)
select b.table_name, a.* , referred_bytes / owned_bytes as ratio
from storage_sum a
join concurrency_wh.information_schema.table_storage_metrics b
on a.clone_group_id = b.clone_group_id
where referred_bytes > 0 and ratio > 1
order by owned_bytes desc;
```

row#	TABLE_NAME	CLONE_GROUP_ID	OWNED_BYTES	REFERRED_BYTES	RATIO
1	1cf01dc61ac4925e6326943f6c2faf73cc85b3f9	12292	43295438332928	51127210208256	1.180891387
2	1cf01dc61ac4925e6326943f6c2faf73cc85b3f9	12292	43295438332928	51127210208256	1.180891387
3	1cf01dc61ac4925e6326943f6c2faf73cc85b3f9	12292	43295438332928	51127210208256	1.180891387
4	1cf01dc61ac4925e6326943f6c2faf73cc85b3f9	12292	43295438332928	51127210208256	1.180891387
5	4d4971b5c2ca8eee185377de5378e7a8d4c44f05	186519	4756115311104	9180438803456	1.930238904
6	4d4971b5c2ca8eee185377de5378e7a8d4c44f05	186519	4756115311104	9180438803456	1.930238904
7	9398b7b93fba9ae62c3ae0b16a75a2b1b67dac09	1146387	4202746140672	8011715369472	1.906304854
8	5d4cb56f00f9ed3d6d30b223193eb39586b02b5f	1146387	4202746140672	8011715369472	1.906304854
9	45d9ebb3a323f9d945a1cfd737972e8faa02e63c	1146387	4202746140672	8011715369472	1.906304854
10	81f29e546c5bfbdfa1b95ed4c529fc6dc0024f4b	1146387	4202746140672	8011715369472	1.906304854
11	dd8d495f2ba9c65728c7904f85193e88dec5b225	186542	1389598218752	2712055010816	1.951682849
12	dd8d495f2ba9c65728c7904f85193e88dec5b225	186542	1389598218752	2712055010816	1.951682849
13	dd8d495f2ba9c65728c7904f85193e88dec5b225	186542	1389598218752	2712055010816	1.951682849
14	dd8d495f2ba9c65728c7904f85193e88dec5b225	186542	1389598218752	2712055010816	1.951682849
15	512d1121d7f781cb718336db4cb270d4f98a45bf	184490	1077942566912	2155885121536	1.999999989
16	765ec9e6af322a586f906c985902fb2d7558db4a	184490	1077942566912	2155885121536	1.999999989
17	712461a72b4df482a42b89dd765fb5a9fd6d02f9	149508	468484634112	823309320192	1.757388098
18	712461a72b4df482a42b89dd765fb5a9fd6d02f9	149508	468484634112	823309320192	1.757388098
19	a144cc44ed2bb4e14c6736caf5fb195f3b82ce39	186536	217674690560	331871647232	1.524622116

Conclusion

By utilizing the UI and the Information Schema functions and views described in this lab, you can profile your data storage to help you keep your storage costs under control and understand how your business is growing over time. It's a good idea to take regular snapshots of your storage so that you can analyze your growth month-over-month. This will help you both formulate usage insight and take actions.

I hope this lab has given you some good ideas for how to manage your Snowflake instance. Look for Part 3 of this series in coming weeks where I will show you how to analyze your query performance. As already shown in Parts 1 and 2, there are a lot of options to play with in Snowflake and they're all intended to give you the flexibility and control you need to best use Snowflake. Please share your thoughts with us!