10 Using High-Performing Functions

10.1 Lab Introduction

10.1.1 Purpose

As you know, common functions in relational database management systems such as COUNT() and percentage/percentile functions require a scan of an entire dataset to yield a result. Although a cloud database like Snowflake is designed to handle virtually unlimited quantities of data, executing a COUNT() on a very large cloud table could take far longer than a user is willing to wait. Additionally, when working with very large datasets, absolutely precise counts are unnecessary, especially if your data is being updated in real time or near real time.

Snowflake's high performing functions are designed to give you approximate results that should satisfy your analytical needs but in a far shorter time frame than a standard COUNT(DISTINCT…). The purpose of this lab is to give you hands-on experience with a couple of these functions: HLL() or Hyperloglog, which can be used in lieu of the standard COUNT() function, and APPROX_PERCENTILE, which can be used in lieu of the standard SQL MEDIAN function.

Whether you're a data analyst or if you perform a different role, you'll become familiar with yet another important tool in your Snowflake arsenal.

10.1.2 What you'll learn

- How to leverage Hyperloglog to get an approximate count
- How to leverage APPROX_PERCENTILE to get an approximate percentile

10.1.3 How to complete this lab

In order to complete this lab, you can key the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the code file for this lab that was provided at the start of the class. You would simply need to open it in TextEdit (mac) or Notepad (Windows), and then copy and paste the code directly into a worksheet.

10.1.4 Scenario

You've just learned about a couple of Snowflake's high performing functions that you think may be useful for your analysis needs. One is Hyperloglog, and the other is APPROX_PERCENTILE. You've decided to try these out on a couple of tables that you know have anywhere from hundreds of millions of rows to even billions of rows just to see how they perform.

This is your plan:

1. Run both Hyperloglog and COUNT() to see which returns a result faster.
2. Run both APPROX_PERCENTILE and MEDIAN() to see which returns a result faster.

Let's get started!

10.1.5 Create a new folder and call it High Performing Functions.

10.1.6 Create a new worksheet inside the folder and call it Working with High Performing Functions.

10.1.7 Alter the session so it does not use cached results. This will give us an accurate reading as to the longest time the functions will take to run:

```
ALTER SESSION SET use_cached_result=false;
```

10.1.8 Set the Worksheet contexts as follows:

As you can see, we will be using a schema other than our PROMO_CATALOG_SALES OR PROMO_CATALOG_SALES_STAR schemas. This data has a very similar model to these two schemas that we've been using except that the LINEITEM table has hundreds of millions of rows! This is perfect for what we are trying to do.

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE LEARNER_WH;
USE DATABASE SNOWFLAKE_SAMPLE_DATA;
USE SCHEMA TPCH_SF100;
```

10.1.9 Change the virtual warehouse size to XSmall

Your warehouse may already be XSmall, but we want to make sure that it is so we can get a clear difference between how quickly each function will run.

```
ALTER WAREHOUSE LEARNER_WH SET WAREHOUSE_SIZE = 'XSmall';
```

10.1.10 Next, suspend and resume the warehouse to clear any data in the warehouse cache. Then use the query below to determine an approximate count with Snowflake's Hyperloglog high-performing function:

```
ALTER WAREHOUSE LEARNER_WH SUSPEND;
ALTER WAREHOUSE LEARNER_WH RESUME;

SELECT HLL(l_orderkey) FROM lineitem;
```

How long did it take to run and how many rows did it find? It should have taken fewer than 10 seconds to run and it should have counted right around 145,660,677 rows.

10.1.11 Suspend and resume the warehouse again to clear the data cache. Execute the regular COUNT version of the query so we can compare the results to the those of the Hyperloglog execution:

```
ALTER WAREHOUSE LEARNER_WH SUSPEND;
ALTER WAREHOUSE LEARNER_WH RESUME;

SELECT COUNT(DISTINCT l_orderkey) FROM lineitem;
```

How long did it take to run and how many rows did it count? It should have taken more than 20 seconds to run and it should have returned a count of exactly 150,000,000 tows.

So, the difference is approximately 4,339,323 rows, which is a variance of 2.9%. If a variance of 2.9% is not critical to whatever analysis you're doing, especially when working with counts in the hundreds of millions, then Hyperloglog can be a better choice than COUNT() in those instances.

10.2 Use Percentile Estimation Functions

Now let's try out the APPROX_PERCENTILE function. This function is a more efficient version of the regular SQL MEDIAN function.

Rather than the LINEITEM table we're going to use the ORDERS table.

10.2.1 Change your warehouse size to Large and clear your warehouse cache:

```
ALTER WAREHOUSE LEARNER_WH
    SET WAREHOUSE_SIZE = 'Large';

ALTER WAREHOUSE LEARNER_WH SUSPEND;
ALTER WAREHOUSE LEARNER_WH RESUME;
```

10.2.2 Start by using the SQL Median Function. The following statement determines the median order total in each year of data:

```
SELECT
      YEAR(O_ORDERDATE)
    , MEDIAN(O_TOTALPRICE)

FROM
    ORDERS

GROUP BY
    YEAR(O_ORDERDATE)

ORDER BY
    YEAR(O_ORDERDATE);
```

How long did it take to run and what results did you get? It should have run in 15-20 seconds, and you should have gotten the results below:

- 1992 - 144310.1
- 1993 - 144303.67
- 1994 - 144285.85
- 1995 - 144282.92
- 1996 - 144322.46
- 1997 - 144284.45
- 1998 - 144318.58

10.2.3 Run the Percentile Estimation Function on the same *store_sales* table to find the approximate 50th percentile of store sales for each store identified in the *store_sales* table:

```
ALTER WAREHOUSE LEARNER_WH SUSPEND;
ALTER WAREHOUSE LEARNER_WH RESUME;

SELECT
      YEAR(O_ORDERDATE)
    , APPROX_PERCENTILE(O_TOTALPRICE, 0.5)

FROM
    ORDERS

GROUP BY
    YEAR(O_ORDERDATE)

ORDER BY
    YEAR(O_ORDERDATE);
```

How long did it take to run and what results did you get? It should have run in fewer than 2 seconds, and you should have gotten the results that look very much like (but not exactly like) the ones below:

- 1992 - 144315.338198642
- 1993 - 144302.777148666
- 1994 - 144284.126806681
- 1995 - 144278.419806512
- 1996 - 144325.010908467
- 1997 - 144289.365240779
- 1998 - 144323.018226321

As you can see, the APPROX_PERCENTILE function does run quite a bit faster than the standard MEDIAN() function and produces almost the exact same result. Your results may not look exactly like the figures we've shown above because the function returns an approximate value each time. Regardless, for the tiny variance you get, you can get a far faster return of the result set.

10.2.4 Change your warehouse size to XSmall:

```
ALTER WAREHOUSE LEARNER_WH
```



```
    SET WAREHOUSE_SIZE = 'XSmall';

ALTER WAREHOUSE LEARNER_WH SUSPEND;
```

10.3 Key takeaways

- If small variances are not critical to you, high performing functions can help you write more efficient queries that run much faster for your business users.

