Free Questions for ARA-C01 by braindumpscollection

Shared by Hooper on 29-01-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

A company has a source system that provides JSON records for various loT operations. The JSON Is loading directly into a persistent table with a variant field. The data Is quickly growing to 100s of millions of records and performance to becoming an issue. There is a generic access pattern that Is used to filter on the create_date key within the variant field.

What can be done to improve performance?

Options:

A- Alter the target table to Include additional fields pulled from the JSON records. This would Include a create_date field with a datatype of time stamp. When this field Is used in the filter, partition pruning will occur.

B- Alter the target table to include additional fields pulled from the JSON records. This would include a create_date field with a datatype of varchar. When this field is used in the filter, partition pruning will occur.

C- Validate the size of the warehouse being used. If the record count is approaching 100s of millions, size XL will be the minimum size required to process this amount of data.

D- Incorporate the use of multiple tables partitioned by date ranges. When a user or process needs to query a particular date range, ensure the appropriate base table Is used.

Answer:

A

Explanation:

The correct answer is A because it improves the performance of queries by reducing the amount of data scanned and processed. By adding a create_date field with a timestamp data type, Snowflake can automatically cluster the table based on this field and prune the micro-partitions that do not match the filter condition. This avoids the need to parse the JSON data and access the variant field for every record.

Option B is incorrect because it does not improve the performance of queries. By adding a create_date field with a varchar data type, Snowflake cannot automatically cluster the table based on this field and prune the micro-partitions that do not match the filter condition. This still requires parsing the JSON data and accessing the variant field for every record.

Option C is incorrect because it does not address the root cause of the performance issue. By validating the size of the warehouse being used, Snowflake can adjust the compute resources to match the data volume and parallelize the query execution. However, this does not reduce the amount of data scanned and processed, which is the main bottleneck for queries on JSON data.

Option D is incorrect because it adds unnecessary complexity and overhead to the data loading and querying process. By incorporating the use of multiple tables partitioned by date ranges, Snowflake can reduce the amount of data scanned and processed for queries that specify a date range. However, this requires creating and maintaining multiple tables, loading data into the appropriate table based on the date, and joining the tables for queries that span multiple date ranges.Reference:

Snowflake Documentation: Loading Data Using Snowpipe: This document explains how to use Snowpipe to continuously load data from external sources into Snowflake tables. It also describes the syntax and usage of the COPY INTO command, which supports various options and parameters to control the loading behavior, such as ON_ERROR, PURGE, and SKIP_FILE.

Snowflake Documentation: Date and Time Data Types and Functions: This document explains the different data types and functions for working with date and time values in Snowflake. It also describes how to set and change the session timezone and the system timezone.

Snowflake Documentation: Querying Metadata: This document explains how to query the metadata of the objects and operations in Snowflake using various functions, views, and tables. It also describes how to access the copy history information using the COPY_HISTORY function or the COPY_HISTORY view.

Snowflake Documentation: Loading JSON Data: This document explains how to load JSON data into Snowflake tables using various methods, such as the COPY INTO command, the INSERT command, or the PUT command. It also describes how to access and query JSON data using the dot notation, the FLATTEN function, or the LATERAL join.

Snowflake Documentation: Optimizing Storage for Performance: This document explains how to optimize the storage of data in Snowflake tables to improve the performance of queries. It also describes the concepts and benefits of automatic clustering, search optimization service, and materialized views.

Question 2

Question Type: MultipleChoice

An Architect needs to automate the daily Import of two files from an external stage into Snowflake. One file has Parquet-formatted data, the other has CSV-formatted data.

How should the data be joined and aggregated to produce a final result set?

Options:

A- Use Snowpipe to ingest the two files, then create a materialized view to produce the final result set.

B- Create a task using Snowflake scripting that will import the files, and then call a User-Defined Function (UDF) to produce the final result set.

C- Create a JavaScript stored procedure to read. join, and aggregate the data directly from the external stage, and then store the results in a table.

D- Create a materialized view to read, Join, and aggregate the data directly from the external stage, and use the view to produce the final result set

Answer:

B

Explanation:

According to the Snowflake documentation, tasks are objects that enable scheduling and execution of SQL statements or JavaScript user-defined functions (UDFs) in Snowflake. Tasks can be used to automate data loading, transformation, and maintenance operations. Snowflake scripting is a feature that allows writing procedural logic using SQL statements and JavaScript UDFs. Snowflake scripting can be used to create complex workflows and orchestrate tasks. Therefore, the best option to automate the daily import of two files from an external stage into Snowflake, join and aggregate the data, and produce a final result set is to create a task using Snowflake scripting that will import the files using the COPY INTO command, and then call a UDF to perform the join and aggregation logic. The UDF can

return a table or a variant value as the final result set.Reference:

Tasks

Snowflake Scripting

User-Defined Functions

Question 3

Question Type: MultipleChoice

A Snowflake Architect Is working with Data Modelers and Table Designers to draft an ELT framework specifically for data loading using Snowpipe. The Table Designers will add a timestamp column that Inserts the current tlmestamp as the default value as records are loaded into a table. The Intent is to capture the time when each record gets loaded into the table; however, when tested the timestamps are earlier than the loae_take column values returned by the copy_history function or the Copy_HISTORY view (Account Usage).

Why Is this occurring?

Options:

A- The timestamps are different because there are parameter setup mismatches. The parameters need to be realigned

B- The Snowflake timezone parameter Is different from the cloud provider's parameters causing the mismatch.

C- The Table Designer team has not used the localtimestamp or systimestamp functions in the Snowflake copy statement.

D- The CURRENT_TIMEis evaluated when the load operation is compiled in cloud services rather than when the record is inserted into the table.

Answer:

D

Explanation:

The correct answer is D because the CURRENT_TIME function returns the current timestamp at the start of the statement execution, not at the time of the record insertion. Therefore, if the load operation takes some time to complete, the CURRENT_TIME value may be earlier than the actual load time.

Option A is incorrect because the parameter setup mismatches do not affect the timestamp values. The parameters are used to control the behavior and performance of the load operation, such as the file format, the error handling, the purge option, etc.

Option B is incorrect because the Snowflake timezone parameter and the cloud provider's parameters are independent of each other. The Snowflake timezone parameter determines the session timezone for displaying and converting timestamp values, while the cloud provider's parameters determine the physical location and configuration of the storage and compute resources.

Option C is incorrect because the localtimestamp and systimestamp functions are not relevant for the Snowpipe load operation. The localtimestamp function returns the current timestamp in the session timezone, while the systimestamp function returns the current timestamp in the system timezone. Neither of them reflect the actual load time of the records.Reference:

Snowflake Documentation: Loading Data Using Snowpipe: This document explains how to use Snowpipe to continuously load data from external sources into Snowflake tables. It also describes the syntax and usage of the COPY INTO command, which supports various options and parameters to control the loading behavior.

Snowflake Documentation: Date and Time Data Types and Functions: This document explains the different data types and functions for working with date and time values in Snowflake. It also describes how to set and change the session timezone and the system timezone.

Snowflake Documentation: Querying Metadata: This document explains how to query the metadata of the objects and operations in Snowflake using various functions, views, and tables. It also describes how to access the copy history information using the COPY_HISTORY function or the COPY_HISTORY view.

Question 4

Question Type: MultipleChoice

A user is executing the following command sequentially within a timeframe of 10 minutes from start to finish:

```
use role sysadmin;
use warehouse compute_wh;
use schema sales.public;
create table t_sales (numeric integer) data_retention_time_in_days=1;
create or replace table t_sales_clone clone t_sales at(offset => -60*30);
```

What would be the output of this query?

Options:

A- Table T_SALES_CLONE successfully created.

B- Time Travel data is not available for table T_SALES.

C- The offset -> is not a valid clause in the clone operation.

D- Syntax error line 1 at position 58 unexpected 'at'.

Answer:

A

Explanation:

The query is executing a clone operation on an existing table t_sales with an offset to account for the retention time. The syntax used is correct for cloning a table in Snowflake, and the use of the at(offset => -60*30) clause is valid. This specifies that the clone should be based on the state of the table 30 minutes prior (60 seconds * 30). Assuming the table t_sales exists and has been modified within the

last 30 minutes, and considering the data_retention_time_in_days is set to 1 day (which enables time travel queries for the past 24 hours), the table t_sales_clone would be successfully created based on the state of t_sales 30 minutes before the clone command was issued.

Question 5

Question Type: MultipleChoice

An Architect is troubleshooting a query with poor performance using the QUERY function. The Architect observes that the COMPILATION_TIME Is greater than the EXECUTION_TIME.

What is the reason for this?

Options:

A- The query is processing a very large dataset.

B- The query has overly complex logic.

C- The query Is queued for execution.

D- The query Is reading from remote storage

Answer:

B

Explanation:

The correct answer is B because the compilation time is the time it takes for the optimizer to create an optimal query plan for the efficient execution of the query. The compilation time depends on the complexity of the query, such as the number of tables, columns, joins, filters, aggregations, subqueries, etc. The more complex the query, the longer it takes to compile.

Option A is incorrect because the query processing time is not affected by the size of the dataset, but by the size of the virtual warehouse. Snowflake automatically scales the compute resources to match the data volume and parallelizes the query execution. The size of the dataset may affect the execution time, but not the compilation time.

Option C is incorrect because the query queue time is not part of the compilation time or the execution time. It is a separate metric that indicates how long the query waits for a warehouse slot before it starts running. The query queue time depends on the warehouse load, concurrency, and priority settings.

Option D is incorrect because the query remote IO time is not part of the compilation time or the execution time. It is a separate metric that indicates how long the query spends reading data from remote storage, such as S3 or Azure Blob Storage. The query remote IO time depends on the network latency, bandwidth, and caching efficiency.Reference:

Understanding Why Compilation Time in Snowflake Can Be Higher than Execution Time: This article explains why the total duration (compilation + execution) time is an essential metric to measure query performance in Snowflake. It discusses the reasons for the long compilation time, including query complexity and the number of tables and columns.

Exploring Execution Times: This document explains how to examine the past performance of queries and tasks using Snowsight or by writing queries against views in the ACCOUNT_USAGE schema. It also describes the different metrics and dimensions that affect query performance, such as duration, compilation, execution, queue, and remote IO time.

What is the "compilation time" and how to optimize it?: This community post provides some tips and best practices on how to reduce the compilation time, such as simplifying the query logic, using views or common table expressions, and avoiding unnecessary columns or joins.

Question 6

Question Type: MultipleChoice

A company is trying to Ingest 10 TB of CSV data into a Snowflake table using Snowpipe as part of Its migration from a legacy database platform. The records need to be ingested in the MOST performant and cost-effective way.

How can these requirements be met?

Options:

A- Use ON_ERROR = continue in the copy into command.

B- Use purge = TRUE in the copy into command.

C- Use FURGE = FALSE in the copy into command.

D- Use on error = SKIP_FILE in the copy into command.

Answer:

D

Explanation:

For ingesting a large volume of CSV data into Snowflake using Snowpipe, especially for a substantial amount like 10 TB, the on error = SKIP_FILE option in the COPY INTO command can be highly effective. This approach allows Snowpipe to skip over files that cause errors during the ingestion process, thereby not halting or significantly slowing down the overall data load. It helps in maintaining performance and cost-effectiveness by avoiding the reprocessing of problematic files and continuing with the ingestion of other data.

Question 7

Question Type: MultipleChoice

Consider the following scenario where a masking policy is applied on the CREDICARDND column of the CREDITCARDINFO table. The masking policy definition Is as follows:

```
create or replace masking policy creditcardno_mask as (val string) returns string ->
case
when is_role_in_session('PI_ANALYTICS') then
right(val,4)
else '***MASKED***'
end;
```

Sample data for the CREDITCARDINFO table is as follows:

NAME EXPIRYDATE CREDITCARDNO

JOHN DOE 2022-07-23 4321 5678 9012 1234

if the Snowflake system rotes have not been granted any additional roles, what will be the result?

Options:

A- The sysadmin can see the CREDICARDND column data in clear text.

B- The owner of the table will see the CREDICARDND column data in clear text.

C- Anyone with the Pl_ANALYTICS role will see the last 4 characters of the CREDICARDND column data in dear text.

D- Anyone with the Pl_ANALYTICS role will see the CREDICARDND column as*** 'MASKED* **'.

Answer:

D

Explanation:

The masking policy defined in the image indicates that if a user has the PI_ANALYTICS role, they will be able to see the last 4 characters of the CREDITCARDNO column data in clear text. Otherwise, they will see 'MASKED'. Since Snowflake system roles have not been granted any additional roles, they won't have the PI_ANALYTICS role and therefore cannot view the last 4 characters of credit card numbers.

To apply a masking policy on a column in Snowflake, you need to use the ALTER TABLE ... ALTER COLUMN command or the ALTER VIEW command and specify the policy name. For example, to apply the creditcardno_mask policy on the CREDITCARDNO column of the CREDITCARDINFO table, you can use the following command:

ALTER TABLE CREDITCARDINFO ALTER COLUMN CREDITCARDNO SET MASKING POLICY creditcardno_mask;

For more information on how to create and use masking policies in Snowflake, you can refer to the following resources:

CREATE MASKING POLICY: This document explains the syntax and usage of the CREATE MASKING POLICY command, which allows you to create a new masking policy or replace an existing one.

Using Dynamic Data Masking: This guide provides instructions on how to configure and use dynamic data masking in Snowflake, which is a feature that allows you to mask sensitive data based on the execution context of the user.

ALTER MASKING POLICY: This document explains the syntax and usage of the ALTER MASKING POLICY command, which allows you to modify the properties of an existing masking policy.