



CERTSWARRIOR

# Snowflake DEA-C02

**SnowPro Advanced: Data Engineer (DEA-C02)**

**Questions&AnswersPDF**

**ForMoreInformation:**

**<https://www.certswarrior.com/>**

## **Features:**

- 90DaysFreeUpdates
- 30DaysMoneyBackGuarantee
- InstantDownloadOncePurchased
- 24/7OnlineChat Support
- ItsLatestVersion

# Latest Version: 6.0

## Question: 1

You are tasked with loading a large dataset (50TB) of JSON files into Snowflake. The JSON files are complex, deeply nested, and irregularly structured. You want to maximize loading performance while minimizing storage costs and ensuring data integrity. You have a dedicated Snowflake virtual warehouse (X-Large).

Which combination of approaches would be MOST effective?

- A. Use Snowpipe with auto-ingest, create a single VARIANT column in your target table, and rely solely on Snowflake's automatic schema detection.
- B. Pre-process the JSON data using a Python script with Pandas to flatten the structure and convert it into a relational format like CSV. Then, load the CSV files using the COPY INTO command with gzip compression.
- C. Use Snowpipe with auto-ingest, create a raw VARIANT column alongside projected relational columns for frequently accessed fields, and use search optimization on those projected columns.
- D. Load the JSON data using the COPY INTO command with no pre-processing. Create a VIEW on top of the raw VARIANT column to flatten the data for querying.
- E. Load the JSON data using the COPY INTO command with gzip compression. Create a raw VARIANT column alongside projected relational columns for frequently accessed fields, and use materialized views to improve query performance.

**Answer: C**

Explanation:

Option C is the most effective. Snowpipe provides continuous loading. A raw VARIANT column captures all data, and projecting commonly accessed fields into relational columns optimizes query performance. Search optimization on the projected columns allows for faster filtering and lookups. Options A, B, D, and E have trade-offs. A lacks optimized querying and can lead to expensive computations on the variant column. B requires pre-processing and may lose data fidelity. D impacts query performance due to runtime flattening. E introduces complexities with materialized view maintenance.

## Question: 2

You are loading data from an S3 bucket into a Snowflake table using the COPY INTO command. The source data contains dates in various formats (e.g., 'YYYY-MM-DD', 'MM/DD/YYYY', 'DD-Mon-YYYY'). You want to ensure that all dates are loaded correctly and consistently into a DATE column in Snowflake. Which of the following COPY INTO options and commands is the MOST appropriate to handle this?

- A. Use the 'DATE FORMAT' option in the COPY INTO command with a single format string that covers all possible date formats.
- B. Use the 'ON\_ERROR = 'SKIP FILE'' option to skip files with invalid date formats.

- C. Use the 'VALIDATE(O)' command before the COPY INTO command to identify files with invalid date formats and then process them separately.
- D. Use the 'STRTOC TO DATE' function within a SELECT statement in a Snowpipe transformation to dynamically parse the dates based on different patterns.
- E. Utilize the 'DATE' function with explicit format strings inside a Snowpipe transformation pipeline. This involves pattern matching using 'CASE WHEN' statements to identify date formats before converting to the DATE data type.

**Answer: E**

Explanation:

Option E is the most robust. It uses regular expressions for pattern matching to accurately determine the format of each date and then uses TO DATE with the correct format string. This ensures consistent conversion to Snowflake's DATE data type. While Option D aims to use STRTOC TO DATE, Snowflake doesn't offer dynamic date parsing during data loading, so each date pattern will be handled manually using CASE statement approach mentioned in the option E, making it more suitable for a varied array of patterns. Options A, B, and C are inadequate. A single 'DATE FORMAT' won't handle multiple formats. B skips files, leading to data loss. C requires manual intervention and doesn't automate the loading process.

### Question: 3

You are designing a data loading process for a high-volume streaming data source. The data arrives as Avro files in an AWS S3 bucket. You need to load this data into a Snowflake table with minimal latency and operational overhead. Which of the following combinations of Snowflake features and configurations would be MOST suitable for this scenario? (Select TWO)

- A. Use the 'COPY INTO' command with a scheduled task that runs every 5 minutes to load new files from the S3 bucket.
- B. Implement Snowpipe with auto-ingest configured to listen for S3 event notifications whenever a new Avro file is added to the bucket.
- C. Use a Kafka connector to stream data directly from the Kafka topic to Snowflake.
- D. Create a custom Spark application that reads Avro files from S3, transforms the data, and then writes it to Snowflake using the Snowflake Spark connector.
- E. Configure an external table pointing to the S3 bucket and query the Avro files directly from Snowflake.

**Answer: B,C**

Explanation: Options B and C offer the best combination of low latency and operational efficiency for streaming data. Snowpipe with auto-ingest provides near real-time loading triggered by S3 events. A Kafka connector provides a direct data stream to Snowflake. Option A introduces latency due to the scheduled task interval and doesn't scale well for high-volume streams. Option D adds operational overhead with Spark application management. Option E is suitable for ad-hoc querying but not ideal for continuous data loading.

### Question: 4

You're loading data into a Snowflake table using 'COPY INTO'. You notice that some rows are being rejected due to data validation errors (e.g., data type mismatch, uniqueness constraint violations). You want to implement a strategy to capture these rejected rows for further analysis and correction. Which of the following approaches offers the MOST efficient and reliable method for capturing and storing the rejected rows, minimizing performance impact during the data loading process? Assume no staging table exists before loading data to production table.

- Enable the 'VALIDATE' function with a large number of rows (e.g., 'VALIDATE(1000)') before running the 'COPY INTO' command. Analyze the validation results and then use 'COPY INTO' with 'ON\_ERROR = 'SKIP\_FILE'' to avoid rejected rows.
- Set 'ON\_ERROR = 'CONTINUE'' in the 'COPY INTO' command. After the load, query the 'VALIDATE' function on the target table with 'SAMPLE' to identify potentially problematic rows.
- Utilize the 'COPY INTO ... REJECTED\_RECORD\_COUNT' functionality after each load, then implement a query to retrieve the rejected records from 'SYSTEM\$LAST\_COPY\_LOAD\_SUMMARY()'.
- Implement 'COPY INTO' with the 'ON\_ERROR = 'ABORT\_STATEMENT'' parameter to rollback the entire load if any errors are encountered, ensuring data consistency at the cost of potential downtime.
- Use the 'COPY INTO ... ERROR\_INTEGRATION' parameter to automatically capture rejected rows in a separate error table or stage. Define an error notification integration to alert when errors occur.

- A. Option A
- B. Option B
- C. Option C
- D. Option D
- E. Option E

**Answer: E**

Explanation:

Option E, utilizing 'ERROR INTEGRATION', is the most efficient and reliable. It automatically captures rejected rows during the 'COPY INTO' process and stores them in a designated error table or stage, minimizing performance impact and providing a structured way to analyze and correct errors. Options A, B, C, and D have drawbacks. A requires pre-validation, adding overhead. B uses sampling, which might not identify all errors. C only provides a record count, not the actual rejected rows. D aborts the entire statement, impacting availability.

## Question: 5

You are tasked with loading a large CSV file (1 T B) into Snowflake. The file contains data for the past 5 years, partitioned by year in the filename (e.g., 'data 2019.csv', 'data 2020.csv', etc.). You need to minimize data loading time and ensure data quality. You have a Snowflake virtual warehouse 'XSMALL' and a stage 'my\_stage'. Which of the following strategies would be MOST effective?

- A. Load each file individually using a separate 'COPY' command with 'VALIDATION MODE = RETURN ERRORS' to check for data quality issues before loading the next file. Use the 'XSMALL' warehouse for all loads.
- B. Increase the virtual warehouse size to 'LARGE', use a single 'COPY' command to load all files with the 'ERROR = CONTINUE' option. Implement data quality checks post-load using SQL queries.
- C. Increase the virtual warehouse size to 'LARGE', use a single 'COPY' command to load all files with the 'ERROR = ABORT STATEMENT' option. Create a file format with 'SKIP HEADER = 1' and 'TRIM SPACE = TRUE'.

- D. Use Snowpipe with auto-ingest enabled. Ensure your cloud storage event notifications are properly configured. Create a file format with 'SKIP HEADER = 1' and 'TRIM SPACE = TRUE' Leave the warehouse as 'XSMALL' to control costs.
- E. Create multiple named file formats each with a unique 'SKIP HEADER' value matching the number of header rows in each file. Load using a single 'COPY' command referencing each file format specifically.

**Answer: B**

Explanation:

Option B is the most effective. Increasing the warehouse size to 'LARGE' allows for parallel processing and faster loading. ERROR = CONTINUE ensures that the load process doesn't halt on minor errors, and post-load data quality checks are more efficient. A allows validation during load which slows down the process significantly. C will halt the entire process upon encountering an error. D is not suitable because it will be throttled by the XSMALL warehouse, which is not good for initial data loading. E isn't realistic as files should have a standard header

### Question: 6

You are loading JSON data into a Snowflake table with a 'VARIANT' column. The JSON data contains nested arrays with varying depths. You need to extract specific values from the nested arrays and load them into separate columns in your Snowflake table. Which approach would provide the BEST performance and flexibility?

- A. Use a stored procedure to parse the JSON data and insert values into the table row by row.
- B. Load the entire JSON into a 'VARIANT' column and then use SQL with nested 'FLATTEN' functions to extract the desired values during query time.
- C. Create a view with nested 'FLATTEN' functions to extract the values from the 'VARIANT' column. The view serves as the source for further transformations.
- D. Use a 'COPY' command with a 'TRANSFORM' clause that uses JavaScript UDFs to parse the JSON and extract the values during the load process. Load the extracted values directly into the target columns.
- E. Use Snowpipe with auto-ingest, loading directly into the table with the 'VARIANT' column. Define data quality checks with pre-load data transformation.

**Answer: D**

Explanation:

Using a 'COPY' command with a 'TRANSFORM' clause and JavaScript UDFs allows for efficient parsing and extraction of values during the load process. This minimizes the amount of data stored in the 'VARIANT' column and avoids expensive query-time parsing. Stored procedures perform row by row operations which are inefficient. Using Flatten functions could be useful to denormalise json, but javascript parsing during load is better. Snowpipe and auto-ingest just move the challenge to a real-time streaming scenario, which may not be optimized for transforming data into a relational structure.

### Question: 7

You are using Snowpipe to load data from an AWS S3 bucket into Snowflake. The data files are compressed using GZIP and are being delivered frequently. You have observed that the pipe's backlog is increasing and data latency is becoming unacceptable. Which of the following actions could you take to improve Snowpipe's performance? (Select all that apply)

- A. Increase the virtual warehouse size associated with the pipe.
- B. Optimize the file size of the data files in S3. Smaller files are processed faster by Snowpipe.
- C. Ensure that the S3 event notifications are correctly configured and that there are no errors in the event delivery mechanism.
- D. Reduce the number of columns in the target Snowflake table. Fewer columns reduce the overhead of data loading.
- E. Check if the target table has any active clustering keys defined which could be causing slow down

**Answer: A,C,E**

Explanation:

Increasing the warehouse size allows Snowpipe to process more data in parallel (A). Correct S3 event notification setup ensures that Snowpipe is promptly notified of new files (C). Snowpipe picks up data only when notified using SNS service, if notifications delay, data loading latency will increase. Active clustering keys can slow down ingest when data is not well-sorted. It needs to re-arrange the table constantly as data comes in (E). While optimizing file size can help to some extent, drastically reducing the number of columns in a target table is usually not a practical approach to improve Snowpipe performance (D). While small files may seem to be better, small files also can cause problems related to too many files to be loaded. Having larger files that can be split to several chunks of parallel data loading will be better.

## Question: 8

You are tasked with loading data from a set of highly nested JSON files into Snowflake. Some files contain an inconsistent structure where a particular field might be a string in some records and an object in others. You want to avoid data loss and ensure that you capture both string and object representations of the field. What is the most efficient approach to achieve this, minimizing data transformation outside of Snowflake?

- A. Create two separate external tables, one with the field defined as VARCHAR and another with the field defined as VARIANT. Load data into both, then UNION the results in a view.
- B. Use a single external table with the field defined as VARIANT. During data loading, use the TRY CAST function within a SELECT statement to convert the field to VARCHAR when possible, V otherwise retain the VARIANT representation. Handle further processing in subsequent views or queries.
- C. Define the field in the external table as VARCHAR. During data loading, use a UDF written in Python or Java to handle the different data types, transforming objects to strings. This approach requires deploying the UDF to Snowflake.
- D. Pre-process the JSON files using a scripting language (e.g., Python) to transform object representations to string representations before loading them into Snowflake. This ensures consistent data type for the field.

E. Define the field as a VARCHAR in an internal stage and use a COPY INTO statement with the VALIDATE function to identify records with object representations. Load the valid VARCHAR values. Create a separate table for the invalid object representations identified during validation.

**Answer: B**

Explanation:

Option B is the most efficient. Defining the field as VARIANT allows Snowflake to handle different data types within the same column. TRY CAST attempts to convert the field to VARCHAR if it's a string, and retains the VARIANT representation if it's an object, avoiding data loss. This approach minimizes the need for separate tables or external data processing. A, C, D and E involve either creating multiple objects, or external stage which are not efficient.

### Question: 9

You are tasked with loading Parquet files into Snowflake from an AWS S3 bucket. The Parquet files are compressed using Snappy compression and contain a complex nested schema. Some of the columns contain timestamps with nanosecond precision. You want to create a Snowflake table that preserves the timestamp precision. Which COPY INTO statement options and table definition are MOST appropriate?

A. Table Definition: CREATE TABLE my\_table (ts TIMESTAMP NTZ(9), other\_col VARCHAR); COPY INTO my\_table FROM FILE FORMAT = (TYPE = PARQUET COMPRESSION = SNAPPY) ON\_ERROR = 'SKIP\_FILE';

B. Table Definition: CREATE TABLE my\_table (ts TIMESTAMP NTZ, other\_col VARCHAR); COPY INTO my\_table FROM FILE FORMAT = (TYPE = PARQUET COMPRESSION = SNAPPY) ON\_ERROR = 'SKIP\_FILE';

C. Table Definition: CREATE TABLE my\_table (ts VARCHAR, other\_col VARCHAR); COPY INTO my\_table FROM FILE FORMAT = (TYPE = PARQUET COMPRESSION = SNAPPY) ON\_ERROR = 'SKIP\_FILE' = PARSE TIMESTAMP(ts));

D. Table Definition: CREATE TABLE my\_table (ts TIMESTAMP NTZ(9), other\_col VARCHAR); COPY INTO my\_table FROM FILE FORMAT = (TYPE = PARQUET COMPRESSION = AUTO) ON\_ERROR = 'SKIP\_FILE';

E. Table Definition: CREATE TABLE my\_table (ts TIMESTAMP NTZ(9), other\_col VARCHAR); COPY INTO my\_table FROM FILE FORMAT = (TYPE = PARQUET COMPRESSION = SNAPPY) ON\_ERROR = 'SKIP\_FILE' VALIDATION\_MODE = RETURN\_ERRORS;

**Answer: D**

Explanation:

The correct approach is to define the timestamp column with TIMESTAMP NTZ(9) to preserve nanosecond precision. Also, setting COMPRESSION = AUTO is a good practice to let Snowflake automatically detect and handle the compression type, even though Snappy is explicitly mentioned. Option A is close, but AUTO compression is preferred for robustness. B would lose precision as timestamp\_ntz defaults to (0), C converts TIMESTAMP to VARCHAR which causes issues with ordering. E will throw errors but does not solve the problem.

### Question:10

A data engineer observes that a daily data transformation pipeline in Snowflake, which processes data from external stage 's3://my-bucket/raw\_dataP', is consistently taking longer to complete. Upon investigation, the engineer finds that the COPY INTO statement is the bottleneck. The COPY INTO statement is as follows:

```
COPY INTO MY_DATABASE.MY_SCHEMA.TARGET_TABLE
FROM @MY_DATABASE.MY_SCHEMA.MY_STAGE
FILE_FORMAT = (TYPE = CSV FIELD_DELIMITER = ',' SKIP_HEADER = 1)
ON_ERROR = 'CONTINUE'
PATTERN = '. .csv';
```

Which of the following could be the root cause of the performance degradation and how would you address them? Select two options.

- A. The 'ON\_ERROR = 'CONTINUE'' option is causing the COPY INTO statement to perform additional error handling, slowing down the process. Remove the 'ON\_ERROR' clause to improve performance.
- B. The external stage contains a large number of small files. Snowflake's COPY INTO statement performs best with fewer, larger files. Consolidate the small files into larger files before loading.
- C. The virtual warehouse used for the COPY INTO operation is undersized. Increase the virtual warehouse size to improve performance.
- D. Snowflake automatically optimizes COPY INTO operations. No specific action is needed.
- E. The PATTERN '. .csv' is inefficient. Refine the PATTERN to be more specific, targeting only the necessary files, potentially using date-based partitioning. If date partitioning is used, leverage partition pruning by including the appropriate date criteria in the COPY INTO statement.

<b>Answer: B,E</b>
--------------------

Explanation:

Option B is correct because a large number of small files can significantly degrade COPY INTO performance. Snowflake performs best with fewer, larger files. Option E is also correct because an overly broad pattern can force Snowflake to evaluate unnecessary files. Refining the pattern and leveraging partition pruning can improve performance. Option A is incorrect as 'ON\_ERROR = CONTINUE' is not the primary cause for performance issues, it just handles the error instead of failing the job. Option C might help in some cases but it's not the primary cause, more focusing on the data loading part. Option D is incorrect as Snowflake does not fully automate COPY INTO operations and some level of tuning might be needed.





# CERTSWARRIOR

## *FULL PRODUCT INCLUDES:*

Money Back Guarantee



Instant Download after Purchase



90 Days Free Updates



PDF Format Digital Download



24/7 Live Chat Support



Latest Syllabus Updates



**For More Information – Visit link below:**

**<https://www.certswarrior.com>**

**16 USD Discount Coupon Code: U89DY2AQ**