Mid-Training Assignment

Section A - Data Loading: Foundations of Data Movement

[8 Questions]

1. Your retail analytics team needs to load 2 years of POS data (CSV format, GZIP compressed) into Snowflake. Design the complete loading pipeline using `CREATE STAGE`, `CREATE FILE FORMAT`, and `COPY INTO`.
2. Explain how file size and compression affect Snowflake's loading performance. Give examples of optimal CSV chunking.
3. Create an internal stage, upload a CSV file, and verify the files loaded using `LIST @stage_name`.
4. Write a SQL statement to skip headers and handle missing values while loading CSVs.
5. Your CSV contains date fields in multiple formats. How can you define a `FILE FORMAT` that standardizes date parsing?
6. Demonstrate how to track the success or failure of a `COPY INTO` operation using the query history.
7. Compare internal and external stages in terms of cost, control, and typical enterprise usage.
8. Write a SQL query that validates record counts before and after a bulk load and handles duplicates.

Section B - Advanced & Semi-Structured Ingestion

[8 Questions]

9. Load a nested JSON file containing product catalog details (product_id, name, price, and reviews array) into a `VARIANT` column.
10. Extract only the `product_name` and `price` fields from the VARIANT column using dot notation.
11. Use `LATERAL FLATTEN` to unnest all customer reviews from the JSON data and count how many reviews each product has.
12. Write a query that finds all products with an average rating greater than 4.5 from the JSON review data.
13. Compare schema-on-read vs. schema-on-write in the context of Snowflake VARIANT data.
14. A JSON file contains null values for `price`. Demonstrate how to replace these with a default value during ingestion.
15. How would you transform nested attributes (like `supplier -> address -> city`) into a structured table for analytics?
16. Design a COPY INTO command to load Parquet data with automatic column mapping.

Section C - Continuous & Real-time Data Pipelines

[8 Questions]

17. Create a Snowpipe on an external S3 stage that automatically ingests new sales files as they arrive.
18. Explain the difference between **auto-ingest** and **Snowpipe REST API** ingestion models.
19. Demonstrate how to use `COPY_HISTORY` and `PIPE_USAGE_HISTORY` to verify whether a file was ingested successfully.
20. How would you handle a scenario where Snowpipe fails due to a corrupted CSV file?
21. Set up notification integration with AWS S3 to trigger Snowpipe automatically.
22. Compare **Snowpipe Streaming** vs **Snowpipe auto-ingest** in terms of latency and cost.
23. Write a script to reprocess data files that were not ingested due to format mismatch.
24. What are the typical enterprise use cases for using Snowpipe in e-commerce or IoT data ingestion?

Section D - SQL Transformations & User-Defined Functions

[8 Questions]

25. Create a structured `FCT_SALES` table from the semi-structured JSON data you loaded earlier.
26. Write a SQL UDF to calculate the final selling price after applying a 10% discount and 5% tax.
27. Build a UDTF that returns all customer reviews for a given `PRODUCT_ID`.
28. Create a pipeline that transforms raw sales data into a clean star schema: `DIM_CUSTOMER`, `DIM_PRODUCT`, `FCT_SALES`.
29. Explain how UDFs and UDTFs improve code reusability and performance in Snowflake.
30. How do you handle UDF versioning and changes in enterprise data models?
31. Write a query using your UDF and join it with a dimension table to show final sale prices by region.
32. Compare UDF vs. Stored Procedure use cases for data transformation.

Section E - Snowpark

[8 Questions]

33. Write a Snowpark Python script to connect to Snowflake and read data from the `FCT_SALES` table.
34. Perform DataFrame transformations to calculate the top 10 selling products by revenue.

35. Use Snowpark to join two DataFrames: `sales_df` and `product_df`, and compute total quantity per category.
36. Demonstrate lazy vs. eager evaluation in Snowpark using `.collect()` and `.show()`.
37. Explain how Snowpark's architecture differs from pandas (client vs. server execution).
38. Write a Snowpark function that calculates the RFM (Recency, Frequency, Monetary Value) for each customer.
39. Compare performance between SQL aggregation and Snowpark DataFrame aggregation for the same dataset.
40. Write a Python Stored Procedure in Snowpark that updates a table based on a condition.

Section F - Query Optimization & Performance Tuning

[10 Questions]

41. Run an inefficient query that joins large tables with a `LIKE '%value%'` condition. Analyze its Query Profile and identify bottlenecks.
42. Rewrite the query to improve performance using `LEFT JOIN` and proper filtering.
43. Explain what "spilling to disk" means in Snowflake and how to prevent it.
44. How do result, metadata, and warehouse caches differ? Give one example for each.
45. Demonstrate the impact of **Result Cache** by running the same query twice and comparing execution times.
46. Use `SYSTEM$CLUSTERING_INFORMATION` to analyze table clustering depth.
47. Describe how clustering keys can help optimize query performance on large time-series tables.
48. Your query runs slow despite caching — what are possible causes (e.g., micro-partition pruning)?
49. Write a query that filters early and avoids `SELECT *`. Measure execution improvement.
50. Propose 3 best practices to ensure query performance consistency in a high-concurrency environment.

🧮 Total Summary

- **Section A (Foundations of Loading)** → 8 Qs
- **Section B (Semi-Structured Ingestion)** → 8 Qs
- **Section C (Snowpipe)** → 8 Qs
- **Section D (SQL Transformations/UDFs)** → 8 Qs
- **Section E (Snowpark)** → 8 Qs
- **Section F (Optimization)** → 10 Qs

  ✅ **Total = 50 Case Study Questions**