Snowflake
DAA-C01
SnowPro Advanced: Data Analyst Certification Exam

Latest Version: 6.0

Question: 1

You are tasked with ingesting clickstream data from a website into Snowflake for real-time analytics. The website generates approximately 100,000 events per minute. The business requires insights into user behavior with a maximum latency of 5 minutes. Which data collection strategy would be MOST appropriate to meet these requirements, considering both cost and near real-time needs?

A. Batch ingestion using Snowpipe with file sizes of IGB uploaded every 15 minutes.
B. Near real-time ingestion using Snowpipe with auto-ingest enabled and micro-batches of data.
C. Scheduled task using a Python script to extract data from an API endpoint every 10 minutes.
D. Utilizing a third-party ETL tool to transfer data in hourly batches.
E. Directly inserting data using JDBC from the web application.

Answer: B

Explanation:
Near real-time ingestion using Snowpipe with auto-ingest is the most appropriate choice. It provides low latency (minutes), scales well with high data volume, and is cost-effective compared to maintaining a custom solution or paying for a full-fledged ETL tool for this specific scenario. Batch processing introduces too much latency (15 minutes or more), scheduled tasks can become resource intensive and JDBC inserts directly from the application can create performance bottlenecks and security concerns.

Question: 2

A financial institution needs to collect stock ticker data for intraday trading analysis. The data source provides updates every second. They need to maintain a 5-minute rolling average of stock prices for each ticker. The system needs to be highly available and resilient to data source interruptions. Considering the need for near real-time analysis and potential data source instability, which combination of technologies and approaches would be MOST effective?

A. Using a scheduled task to query the API every minute and store the data directly into a Snowflake table with a materialized view calculating the rolling average.
B. Employing a stream processing framework (e.g., Apache Kafka) to ingest the data, perform the rolling average calculation using a tumbling window, and load the aggregated results into Snowflake.
C. Storing the raw data into Snowflake using Snowpipe in micro-batches and creating a VIEW that performs the rolling average calculation on-demand.
D. Using a traditional ETL tool to extract, transform (calculate rolling average), and load the data into Snowflake in 15-minute intervals.
E. Leveraging Snowflake's dynamic data masking and data classification capabilities to maintain data security and compliance while adhering to real-time data ingestion.

Answer: B

Explanation:
A stream processing framework like Kafka is ideal for handling high-velocity data streams. Kafka provides fault tolerance and the ability to perform real-time aggregations (rolling average with tumbling window). While Snowpipe can ingest the raw data quickly, calculating the rolling average on-demand (using a VIEW) may not meet the near real-time requirement and can be inefficient. A scheduled task might not be able to handle the volume and frequency of data. The key to answering this question is understanding the need for real-time aggregation AND resilience to potential data source outages, both of which Kafka elegantly addresses.

Question: 3

You are designing a data ingestion pipeline for IoT sensor data'. Sensors transmit data in JSON format every 5 seconds. The volume of data is expected to grow exponentially. The business requires both real-time dashboards and historical trend analysis. Which of the following strategies should you employ to address these requirements, particularly focusing on optimizing for both ingestion frequency and cost?

A. Ingest data directly into a single Snowflake table using Snowpipe with JSON data type. Create separate materialized views for real-time dashboards and historical trend analysis.
B. Use Snowpipe to ingest data into a raw landing table, and then use Snowflake tasks to transform and load the data into separate tables optimized for real-time dashboards and historical analysis.
C. Utilize an external stream processing engine to pre-aggregate the data into time windows (e.g., I-minute, I-hour) before ingesting into Snowflake using Snowpipe.
D. Employ a combination of Snowpipe for near real-time data ingestion into a raw table, and then use Snowflake's Search Optimization Service for faster queries.
E. Implement Snowpipe for initial data ingestion, complemented by Snowflake's clustering feature based on timestamp to optimize historical analysis queries. And employ a stream processing engine to perform time window pre-aggregation.

Answer: B,E

Explanation:
This question has multiple correct answers. Option B and E addresses the need for optimizing both ingestion frequency and cost. Option B involves using Snowflake tasks for transformations, which allows for separating data for real-time and historical analysis. This optimizes query performance for both use cases. Using only Snowpipe directly into a single table is simple but doesn't provide optimization for the different query patterns, while the other options are less efficient or don't fully address the dual requirements. Option E complements using Snowpipe by using Snowflake clustering features and employing stream processing engine to perform pre-aggregation. Snowflake clustering optimize the speed of historical analysis, reducing cost of scanning large datasets.

Question: 4

You are responsible for collecting server log data from multiple geographically distributed data centers. The logs are generated at a high velocity and variety of formats (JSON, CSV, plain text). The requirement is to ensure minimal data loss and efficient ingestion into Snowflake, while also handling potential

schema variations across different log sources. Which of the following is the MOST robust and scalable solution, considering potential schema drift and data volume?

A. Use a centralized file server to collect logs and then use Snowpipe with schema detection enabled on a single variant column in Snowflake.
B. Employ a distributed log aggregation system (e.g., Fluentd or Logstash) to standardize the log format and then use Snowpipe to ingest the data into Snowflake.
C. Write a custom Python script to pull logs from each data center, transform them into a consistent CSV format, and then upload the CSV files to Snowflake using Snowpipe.
D. Configure each data center to directly stream logs to Snowflake using the Snowflake JDBC driver.
E. Utilize a message queue (e.g., Kafka) to collect logs from all data centers and create an external table pointing to the message queue. Use Snowflake streams to ingest the data from the message queue into Snowflake.

Answer: B

Explanation:
A distributed log aggregation system (Fluentd/Logstash) is the best choice here. These systems are designed for handling high- velocity, varied log formats, and schema variations. They can buffer data to prevent data loss and transform the data into a consistent format before ingestion into Snowflake. Snowpipe provides efficient data loading from cloud storage. This combination provides scalability, reliability, and flexibility. Message queues require more configuration overhead and external tables can be slower for querying. Custom scripts are less scalable and harder to maintain. Direct streaming using JDBC is not recommended for high-volume data.

Question: 5

You are designing a data ingestion pipeline to collect clickstream data from a high-traffic e-commerce website. You anticipate a daily data volume that fluctuates significantly, ranging from 5 TB to 20 TB. To optimize Snowflake costs and ensure efficient data loading, which combination of Snowflake features and ingestion methods would be MOST effective for identifying the accurate data volume before the data is fully loaded into a production table?

A. Use Snowflake's Resource Monitors to track credit usage during data loading. Configure a data pipeline using Snowpipe with auto-ingest enabled. Rely solely on Snowflake's cost estimation tools to predict volume.
B. Utilize a cloud-based message queue (e.g., AWS SQS, Azure Queue Storage) to buffer incoming data. Implement a separate Snowflake task to periodically query the message queue's approximate message count and size. Combine this with Snowsight monitoring of storage costs for the raw data stage.
C. Implement Snowpipe with pre-validation using a data transformation tool like dbt Core. Create an external function to execute a sampling query against a percentage of files in cloud storage and estimate the total volume. Load a smaller sample data to a staging table and then use ‘TABLE SIZE' to get the actual size and scale it up.
D. Use Snowflake Tasks to schedule a Python script that reads file metadata (size, number of files) from the cloud storage location using the cloud provider's SDK (e.g., bot03 for AWS S3, azure-storage-blob for Azure Blob Storage). Store the collected metadata in a Snowflake table. Then, use SQL queries to calculate the total data volume.

E. Rely solely on Snowsight's query history and data loading metrics after the data has been loaded into Snowflake, adjusting future ingestion parameters based on historical data.

Answer: D

Explanation:
Option D is the most effective because it directly measures the volume of data before it is fully loaded into Snowflake, allowing for accurate volume identification and cost optimization. Using cloud provider SDK to check the data volume will help to estimate cost and volume.

Question: 6

You are tasked with ingesting data from a REST API that provides daily sales reports in JSON format. The API has rate limits (100 requests per minute) and returns a large dataset (approximately 5GB per day). The data needs to be processed within 2 hours of its availability. You want to leverage Snowflake external functions and tasks. Which approach balances efficiency, cost, and adherence to rate limits?

A. Create a single Snowflake task that calls an external function which iterates through all API pages sequentially in a single execution to retrieve and load all the data. Rely on Snowflake's automatic scaling to handle the load.
B. Create a Snowflake task that triggers an external function which retrieves only the metadata (e.g., total records, page count) from the API. Then, create a dynamic number of child tasks, each responsible for retrieving a subset of the data, using the metadata to respect the API rate limits.
C. Create a Snowflake external function that directly connects to the API and loads data into a staging table. Use Snowpipe with auto-ingest to continuously load the data as it arrives. Ignore the API Rate limits; assume they will be handled by API itself.
D. Create a Snowflake task that calls an external function. This external function calls an intermediate service (e.g., AWS Lambda, Azure Function) which is responsible for fetching the data in batches, respecting the API rate limits, and storing the data in cloud storage. Snowpipe then loads the data from cloud storage into Snowflake.
E. Call API directly from a scheduled task without considering Rate Limit. Persist the data using COPY INTO command.

Answer: D

Explanation:
Option D is the most balanced approach. It leverages an intermediate service to handle the complexities of API interaction (rate limits, pagination), decouples the data retrieval from Snowflake compute, and uses Snowpipe for efficient bulk loading. This approach addresses I both the rate limits and processing time requirements effectively.

Question: 7

You are building a real-time data pipeline to ingest IoT sensor data into Snowflake. The data arrives in Avro format via Kafka topics. You need to ensure data integrity, minimize latency, and accurately

determine the data volume ingested for billing and monitoring purposes. Which of the following options provide the BEST combination of techniques to achieve these goals? (Select TWO)

A. Use Snowflake's Kafka connector to directly load data from Kafka into a raw data table. After loading, run scheduled tasks to perform data quality checks and transformations.
B. Implement a stream processing framework (e.g., Apache Flink, Spark Streaming) to consume data from Kafka, perform data quality checks and transformations, and then load the processed data into Snowflake using the Snowflake JDBC driver.
C. Use Snowpipe with auto-ingest to continuously load data from a cloud storage location (e.g., AWS S3, Azure Blob Storage) where Kafka Connect is writing the Avro data. Configure Snowpipe error notifications to capture data quality issues. Do not perform Transformation.
D. Implement a custom Kafka consumer application that validates and transforms the Avro data before loading it into a staging table in Snowflake using the Snowflake Python connector. Use a Snowflake Task to move data from the staging to the final table.
E. Utilize Snowflake Streams and Tasks to create a change data capture (CDC) pipeline within Snowflake. Initially, load all data into a raw table. Then, use a stream to track changes, validate the data, and apply transformations incrementally.

Answer: B,E

Explanation:
Options B and E provide the best combination. Option B uses a stream processing framework for real-time validation and transformation before loading into Snowflake, minimizing latency and ensuring data quality. Option E leverages Snowflake Streams and Tasks for CDC, enabling incremental data validation and transformation within Snowflake, ensuring data integrity and volume tracking.

Question: 8

Your team is building a data pipeline to ingest data from a REST API that returns JSON payloads. Due to API rate limits, you need to implement a backoff strategy to avoid overwhelming the API. You are using Python and the 'requests' library for data ingestion. Which code snippet BEST demonstrates a robust backoff strategy with exponential backoff and jitter?

A.

```
import requests
import time

def fetch_data(url):
    try:
        response = requests.get(url)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error fetching data: {e}")
        return None
```

B.

```
import requests
import time
import random

def fetch_data(url, max_retries=5):
    retries = 0
    while retries < max_retries:
        try:
            response = requests.get(url)
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException as e:
            retries += 1
            wait_time = (2   retries) + random.uniform(0, 1)
            print(f"Retry {retries} after {wait_time:.2f} seconds")
            time.sleep(wait_time)
    print(f"Max retries reached. Failed to fetch data from {url}")
    return None
```

C.

```
import requests
import time

def fetch_data(url):
    while True:
        try:
            response = requests.get(url)
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException:
            time.sleep(5)
```

D.

```
import requests

def fetch_data(url):
    response = requests.get(url)
    return response.json()
```

E.

```
import requests
import time
import random

def fetch_data(url):
    try:
        response = requests.get(url, timeout=10)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        time.sleep(random.randint(1,5))
        return fetch_data(url)
```

Answer: B

Explanation:
Option B provides the most robust backoff strategy. It implements exponential backoff (wait time increases exponentially with each retry) and adds jitter (a random amount of time) to the wait time to avoid synchronized retries from multiple clients. also handles error codes correctly. Option A doesn't have a backoff strategy. Option C has a simple retry mechanism but lacks exponential backoff and a limit

on retries. Option D lacks error handling and retry logic. Option E uses recursion, which is generally not recommended and does not feature exponential backoff.

Question: 9

A data analyst needs to ingest data from various sources into Snowflake, cleanse it, and load it into target tables. Which of the following actions are MOST crucial for ensuring data quality and consistency during the ingestion and preparation phases?

A. Enforcing data type constraints on Snowflake table columns.
B. Implementing data validation checks using Snowflake user-defined functions (UDFs) or stored procedures to detect and handle invalid data.
C. Storing all data in a single large table to avoid data silos.
D. Using Snowflake's data masking policies to protect sensitive data during ingestion.
E. Implementing data profiling techniques to identify data quality issues and inconsistencies before loading data into target tables and creating detailed data dictionaries.

Answer: A,B,E

Explanation:
Options A, B, and E are most crucial for data quality and consistency. Enforcing data type constraints (A) ensures that only data of the correct type is loaded into the tables. Implementing data validation checks (B) using UDFs or stored procedures allows for detecting and handling invalid data, preventing it from corrupting the data warehouse. Data profiling (E) helps to understand data quality issues before the load and guides data cleansing efforts. Storing all data in a single table (C) is generally not recommended as it can lead to performance issues and make it harder to manage data. While data masking (D) is important for data security, it is not directly related to data quality and consistency during ingestion and preparation. Data dictionaries promote data quality and usability.

Question: 10

You have identified inconsistencies in the data type of the 'ORDER DATE' column across several tables within your Snowflake database. Some tables store it as DATE, while others store it as VARCHAR. You need to create a unified view that presents 'ORDER DATE' consistently as DATE, handling potential conversion errors gracefully. You have to use safe aggregate operations. Which of the following approaches provides the most robust and error-tolerant solution?

A.
```
CREATE OR REPLACE VIEW UNIFIED_ORDERS AS SELECT TRY_TO_DATE(ORDER_DATE) AS ORDER_DATE, other_columns FROM table1 UNION ALL SELECT TRY_TO_DATE(ORDER_DATE) AS ORDER_DATE, other_columns FROM table2;
```
B.
```
CREATE OR REPLACE VIEW UNIFIED_ORDERS AS SELECT CASE WHEN IS_DATE(ORDER_DATE) THEN TO_DATE(ORDER_DATE) ELSE NULL END AS ORDER_DATE, other_columns FROM table1 UNION ALL SELECT CASE WHEN IS_DATE(ORDER_DATE) THEN TO_DATE(ORDER_DATE) ELSE NULL END AS ORDER_DATE, other_columns FROM table2;
```
C.
```
CREATE OR REPLACE VIEW UNIFIED_ORDERS AS SELECT TO_DATE(ORDER_DATE) AS ORDER_DATE, other_columns FROM table1 UNION ALL SELECT TO_DATE(ORDER_DATE) AS ORDER_DATE, other_columns FROM table2;
```
D.

○ `CREATE OR REPLACE VIEW UNIFIED_ORDERS AS SELECT SAFE_CAST(ORDER_DATE AS DATE) AS ORDER_DATE, other_columns FROM table1 UNION ALL SELECT SAFE_CAST(ORDER_DATE AS DATE) AS ORDER_DATE, other_columns FROM table2;`

E.

○ `CREATE OR REPLACE VIEW UNIFIED_ORDERS AS SELECT TRY_CAST(ORDER_DATE AS DATE) AS ORDER_DATE, other_columns FROM table1 UNION ALL SELECT TRY_CAST(ORDER_DATE AS DATE) AS ORDER_DATE, other_columns FROM table2;`

Answer: E

Explanation:
The most robust and error-tolerant solution is option E, using 'TRY CAST(ORDER DATE AS DATE)'. This function attempts to convert the 'ORDER_DATE' to a DATE data type, and if the conversion fails (e.g., the VARCHAR value cannot be parsed as a date), it returns NULL without raising an error. This ensures that the view creation and queries against it will not fail due to data type conversion issues. Option A, , is an older function, and ‘ TRY_CAST' is the preferred, more general function. Option B, 'IS_DATE is not a valid snowflake Function. Option C, ‘ TO DATE will fail the query if the data in column is not a valid date. Option D 'SAFE_CAST is not a valid snowflake function.

Thank You for Trying Our Product

For More Information – Visit link below:

https://www.examsboost.com/

15 USD Discount Coupon Code:

G74JA8UF

FEATURES

- ✓ 90 Days Free Updates
- ✓ Money Back Pass Guarantee
- ✓ Instant Download or Email Attachment
- ✓ 24/7 Live Chat Support
- ✓ PDF file could be used at any Platform
- ✓ 50,000 Happy Customer