Snowflake

GES-C01

SnowPro® Specialty: Gen AI Certification Exam

For More Information – Visit link below:

https://www.examsempire.com/

Product Version

1. **Up to Date products, reliable and verified.**
2. **Questions and Answers in PDF Format.**

Latest Version: 6.1

Question: 1

A data application developer is tasked with building a multi-turn conversational AI application using Streamlit in Snowflake (SiS) that leverages the COMPLETE (SNOWFLAKE. CORTEX) LLM function. To ensure the conversation flows naturally and the LLM maintains context from previous interactions, which of the following is the most appropriate method for handling and passing the conversation history?

- ○ The developer should store the entire conversation history in a temporary table in Snowflake and query it with each new turn, passing only the latest user message to the `COMPLETE` function.
- ○ Snowflake automatically manages conversational context for `COMPLETE` within the session, so the developer only needs to pass the current user prompt as a string.
- ○ The conversation history must be explicitly managed within the Streamlit application's state, typically by initializing `st.session_state.messages = []` and appending each user and assistant message as an object with `'role'` and `'content'` keys, then passing the full list to the `prompt_or_history` argument of `COMPLETE`.
- ○ The developer should concatenate all previous user prompts and assistant responses into a single, long string, and pass this as the `<prompt>` argument to `COMPLETE` for each turn.
- ○ The `COMPLETE` function has an optional `'conversation_id'` parameter that automatically retrieves and manages conversation history when provided.

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

Answer: C

Question: 2

A Streamlit application developer wants to use AI_COMPLETE (the latest version of COMPLETE (SNOWFLAKE. CORTEX)) to process customer feedback. The goal is to extract structured information, such as the customer's sentiment, product mentioned, and any specific issues, into a predictable JSON format for immediate database ingestion. Which configuration of the AI_COMPLETE function call is essential for achieving this structured output requirement?

- ○ Including detailed instructions within the prompt string, such as `'Extract sentiment, product, and issues as a JSON object: { "sentiment": "...", "product": "...", "issues": "..." }'`.
- ○ Setting the `temperature` option to `0` and `max_tokens` to a high value, which implicitly guides the LLM to produce structured output.
- ○ Utilizing the `response_format` argument with a JSON schema object that precisely defines the expected structure, data types, and required fields for the output.
- ○ Using the `AI_EXTRACT` function multiple times, once for each piece of information (sentiment, product, issues) to be extracted, and then manually combining the results into a JSON object.
- ○ Enabling the `guardrails` option with a custom validation rule to ensure the LLM's raw text output conforms to a JSON pattern.

A. Option A
B. Option B
C. Option C
D. Option D

E. Option E

Answer: C

Explanation:
'AI_COMPLETE Structured Outputs' (and its predecessor 'COMPLETE Structured Outputs') specifically allows supplying a JSON schema as the 'response_format' argument to ensure completion responses follow a predefined structure. This significantly reduces the need for post-processing in AI data pipelines and enables seamless integration with systems requiring deterministic responses. The JSON schema object defines the structure, data types, and constraints, including required fields. While prompting the model to 'Respond in JSON' can improve accuracy for complex tasks, the 'response_format' argument is the direct mechanism for enforcing the schema. Setting 'temperature' to 0 provides more consistent results for structured output tasks. Option A is a form of prompt engineering, which can help but does not guarantee strict adherence as response_format' does. Option B controls randomness and length, not output structure. Option D is less efficient for extracting multiple related fields compared to a single structured output call. Option E's 'guardrails' are for filtering unsafe or harmful content, not for enforcing output format.

Question: 3

A Snowflake developer, `AI_ENGINEER`, is creating a Streamlit in Snowflake (SiS) application that will utilize a range of Snowflake Cortex LLM functions, including `SNOWFLAKE.CORTEX.COMPLETE`, `SNOWFLAKE.CORTEX.CLASSIFY_TEXT`, and `SNOWFLAKE.CORTEX.EMBED_TEXT_768`. The application also needs to access data from tables within a specific database and schema. `AI_ENGINEER` has created a custom role, `app_dev_role`, for the application to operate under. Which of the following privileges or roles are absolutely necessary to grant to `app_dev_role` for the successful execution of these Cortex LLM functions and interaction with the specified database objects? (Select all that apply.)

- ☐ The `SNOWFLAKE.CORTEX_USER` database role, which provides the necessary permissions to call Snowflake Cortex AI functions.
- ☐ The `CREATE SNOWFLAKE.ML.DOCUMENT_INTELLIGENCE` privilege on the schema where the application resides.
- ☐ The `USAGE` privilege on the specific database and schema where the Streamlit application and its underlying data tables are located.
- ☐ The `ACCOUNTADMIN` role to ensure unrestricted access to all Snowflake Cortex features.
- ☐ The `CREATE COMPUTE POOL` privilege to provision resources for the Streamlit application.

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

Answer: A,C

Explanation:
To execute Snowflake cortex AI functions such as 'SNOWFLAKE.CORTEX.COMPLETE , 'SNOWFLAKE.CORTEX.CLASSIFY_TEXT, and 'EMBED_TEXT_768' (or their SAE prefixed counterparts), the role used by the application in this case) must be granted the 'SNOWFLAKE.CORTEX_USER database role. Additionally, for the Streamlit application to access any database or schema objects (like tables for data input/output, or for the Streamlit app itself if it is stored as a database object), the USAGE privilege must be granted on those specific database and schema objects. Option B, 'CREATE

SNOWFLAKE.ML.DOCUMENT_INTELLIGENCE, is a privilege specific to creating Document AI model builds and is not required for general Cortex LLM functions. Option D, 'ACCOUNTADMIN', grants excessive privileges and is not a best practice for application roles. Option E, 'CREATE COMPUTE POOL' , is a privilege related to Snowpark Container Services for creating compute pools, which is not directly required for running a Streamlit in Snowflake application that consumes Cortex LLM functions.

Question: 4

A data application developer is tasked with building a multi-turn conversational AI application using Streamlit in Snowflake (SiS) that leverages the COMPLETE (SNOWFLAKE. CORTEX) LLM function. To ensure the conversation flows naturally and the LLM maintains context from previous interactions, which of the following is the most appropriate method for handling and passing the conversation history?

- ○ The developer should store the entire conversation history in a temporary table in Snowflake and query it with each new turn, passing only the latest user message to the `COMPLETE` function.
- ○ Snowflake automatically manages conversational context for `COMPLETE` within the session, so the developer only needs to pass the current user prompt as a string.
- ○ The conversation history must be explicitly managed within the Streamlit application's state, typically by initializing `st.session_state.messages = []` and appending each user and assistant message as an object with `'role'` and `'content'` keys, then passing the full list to the `prompt_or_history` argument of `COMPLETE`.
- ○ The developer should concatenate all previous user prompts and assistant responses into a single, long string, and pass this as the `<prompt>` argument to `COMPLETE` for each turn.
- ○ The `COMPLETE` function has an optional `'conversation_id'` parameter that automatically retrieves and manages conversation history when provided.

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

Answer: C

Explanation:
To provide a stateful, conversational experience with the 'COMPLETE (SNOWFLAKE.CORTEX)' function (or its latest version, 'AI_COMPLETE'), all previous user prompts and model responses must be explicitly passed as part of the argument. This argument expects an array of objects, where each object represents a turn and contains a 'role' ('system', 'user', or 'assistant') and a 'content key, presented in chronological order. In Streamlit, 'st.session_state' is the standard and recommended mechanism for storing and managing data across reruns of the application, making it ideal for maintaining chat history, by initializing 'st.session_state.messages = [I' and appending messages to it. Option A is incorrect because 'COMPLETE does not inherently manage history from external tables. Option B is incorrect as 'COMPLETE does not retain state between calls; history must be explicitly managed. Option D is a less effective form of prompt engineering compared to passing structured history, as it loses the semantic role distinction and can be less accurate for LLMs. Option E describes a non- existent parameter for the 'COMPLETE function.

Question: 5

A Streamlit application developer wants to use AI_COMPLETE (the latest version of COMPLETE (SNOWFLAKE.CORTEX)) to process customer feedback. The goal is to extract structured information, such as the customer's sentiment, product mentioned, and any specific issues, into a predictable JSON format for immediate database ingestion. Which configuration of the AI_COMPLETE function call is essential for achieving this structured output requirement?

- ○ Including detailed instructions within the prompt string, such as `'Extract sentiment, product, and issues as a JSON object: { "sentiment": "...", "product": "...", "issues": "..." }'`.
- ○ Setting the `temperature` option to `0` and `max_tokens` to a high value, which implicitly guides the LLM to produce structured output.
- ○ Utilizing the `response_format` argument with a JSON schema object that precisely defines the expected structure, data types, and required fields for the output.
- ○ Using the `AI_EXTRACT` function multiple times, once for each piece of information (sentiment, product, issues) to be extracted, and then manually combining the results into a JSON object.
- ○ Enabling the `guardrails` option with a custom validation rule to ensure the LLM's raw text output conforms to a JSON pattern.

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

Answer: C

Explanation:
'AI_COMPLETE Structured OutputS (and its predecessor 'COMPLETE Structured OutputS) specifically allows supplying a JSON schema as the 'response_format' argument to ensure completion responses follow a predefined structure. This significantly reduces the need for post-processing in AI data pipelines and enables seamless integration with systems requiring deterministic responses. The JSON schema object defines the structure, data types, and constraints, including required fields. For complex tasks, prompting the model to respond in JSON can improve accuracy, but the 'response_format' argument is the direct mechanism for enforcing the schema. Setting 'temperature to 0 provides more consistent results for structured output tasks. Option A is a form of prompt engineering, which can help but does not guarantee strict adherence as 'response_format does. Option B controls randomness and length, not output structure. Option D, while 'AI_EXTRACT (or EXTRACT ANSWER) can extract information, using it multiple times and then manually combining results is less efficient and less robust than a single 'AI_COMPLETE call with a structured output schema for multiple related fields. Option E's 'guardrails' are for filtering unsafe or harmful content, not for enforcing output format.

Question: 6

A Snowflake developer, `AI_ENGINEER`, is creating a Streamlit in Snowflake (SiS) application that will utilize a range of Snowflake Cortex LLM functions, including `SNOWFLAKE.CORTEX.COMPLETE`, `SNOWFLAKE.CORTEX.CLASSIFY_TEXT`, and `SNOWFLAKE.CORTEX.EMBED_TEXT_768`. The application also needs to access data from tables within a specific database and schema. `AI_ENGINEER` has created a custom role, `app_dev_role`, for the application to operate under. Which of the following privileges or roles are absolutely necessary to grant to `app_dev_role` for the successful execution of these Cortex LLM functions and interaction with the specified database objects? (Select all that apply.)

A.
The `SNOWFLAKE.CORTEX_USER` database role, which provides the necessary permissions to call Snowflake Cortex AI functions.
B.
The `CREATE SNOWFLAKE.ML.DOCUMENT_INTELLIGENCE` privilege on the schema where the application resides.

C. The USAGE privilege on the specific database and schema where the Streamlit application and its underlying data tables are located.
D. The ACCOUNTADMIN role to ensure unrestricted access to all Snowflake Cortex features.
E. The CREATE COMPUTE POOL privilege to provision resources for the Streamlit application.

Answer: A,C

Explanation:
To execute Snowflake Cortex AI functions such as 'SNOWFLAKE.CORTEX.COMPLETE, 'CLASSIFY TEXT (SNOWFLAKE.CORTEX)' , and (or their prefixed counterparts like 'AI_COMPLETE' , 'AI_CLASSIFY , 'AI_EMBED), the role used by the application in this case) must be granted the database role. This role includes the privileges to call these functions. Additionally, for the Streamlit application to access any database or schema objects (like tables for data input/output, or for the Streamlit app itself if it is stored as a database object), the 'USAGE' privilege must be granted on those specific database and schema objects. Option B, 'CREATE SNOWFLAKE.ML.DOCUMENT_INTELLIGENCE , is a privilege specific to creating DocumentAl model builds and is not required for general Cortex LLM functions. Option D, 'ACCOUNTADMIN'', grants excessive privileges and is not a best practice for application roles. Option E, 'CREATE COMPUTE POOL' , is a privilege related to Snowpark Container Services for creating compute pools, which is generally not directly required for running a Streamlit in Snowflake application that consumes Cortex LLM functions via SQL, unless the LLMs themselves were deployed as services on compute pools using Model Serving in Snowpark Container Services, which is not explicitly stated as the method of LLM usage here.

Question: 7

A data engineer is building a Snowflake data pipeline to ingest customer reviews from a raw staging table into a processed table. For each review, they need to determine the overall sentiment (positive, neutral, negative) and store this as a distinct column. The pipeline is implemented using SQL with streams and tasks to process new dat
a. Which Snowflake Cortex LLM function, when integrated into the SQL task, is best suited for this sentiment classification and ensures a structured, single-label output for each review?

- ○ Use `SNOWFLAKE.CORTEX.SENTIMENT()` to get a numerical score, then apply a separate SQL CASE statement or UDF for classification into 'positive', 'neutral', 'negative'.
- ○ Use `SNOWFLAKE.CORTEX.CLASSIFY_TEXT()` with the input text and a list of categories like `['positive', 'negative', 'neutral']` to directly obtain the classification label.
- ○ Use `AI_COMPLETE()` with a prompt such as `'Classify the sentiment of this review: [review_text] into positive, neutral, or negative.'` and configure the `response_format` to be a string.
- ○ Use `SNOWFLAKE.CORTEX.EXTRACT_ANSWER()` asking the question `'What is the sentiment of this review?'` to get a descriptive answer that then needs parsing.
- ○ Use multiple `AI_FILTER()` calls, one for each sentiment category (e.g., `AI_FILTER(prompt('Is this review positive?'))`), and combine the boolean results.

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

Answer: B

Explanation:
To classify text into predefined categories, the function (or its updated version, is purpose-built and directly returns the classification label. This approach is more direct and efficient than using 'SENTIMENT()' which returns a score, which extracts an answer to a question, or multiple calls which return Boolean values. While could be prompted for classification, is a more specific task-specific function designed for this exact use case within Cortex LLM functions.

Question: 8

A financial services company is developing an automated data pipeline in Snowflake to process Federal Reserve Meeting Minutes, which are initially loaded as PDF documents. The pipeline needs to extract specific entities like the FED's stance on interest rates ('hawkish', 'dovish', or 'neutral') and the reasoning behind it, storing these as structured JSON objects within a Snowflake table. The goal is to ensure the output is always a valid JSON object with predefined keys. Which AI_COMPLETE configuration, used within an in-line SQL statement in a task, is most effective for achieving this structured extraction directly in the pipeline?

- ○ Using a simple prompt like `'Extract FED stance on interest rates and the reasoning as JSON from the document: [document_text]'` and expecting the LLM to format it correctly without further configuration.
- ○ Setting the `temperature` parameter to 0 and `max_tokens` to a sufficiently large value, along with a prompt to output JSON, to inherently guide the LLM to a structured format.
- ○ Employing the `response_format` argument within `AI_COMPLETE` with a JSON schema that explicitly defines the `stance` (enum: 'hawkish', 'dovish', 'neutral') and `reasoning` (string) fields, ensuring strict adherence to the output structure.
- ○ Utilizing multiple calls to `SNOWFLAKE.CORTEX.EXTRACT_ANSWER()` or `AI_EXTRACT()`, one for the stance and another for the reasoning, then manually constructing the JSON in a subsequent SQL step.
- ○ Relying on default `AI_COMPLETE` behavior, as Snowflake Cortex automatically detects the need for JSON output when entity extraction with specific fields is implied by the prompt.

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

Answer: C

Explanation:
To ensure that LLM responses adhere to a predefined JSON structure, the 'AI_COMPLETE function's 'response_format' argument, which accepts a JSON schema, is the most effective and direct method. This mechanism enforces the structure, data types, and required fields, significantly reducing the need for post-processing and ensuring deterministic, high-quality output. The AI-Infused Data Pipelines with Snowflake Cortex blog highlights asking the LLM to create a JSON object for maximizing utility. While setting 'temperature' to 0 can improve consistency, it does not enforce a specific schema. Prompt engineering (Option A) can help but does not guarantee strict adherence. Using multiple extraction calls (Option D) is less efficient and robust for extracting multiple related fields than a single 'AI_COMPLETE call with a structured output schema. Snowflake Cortex does not automatically infer and enforce a JSON schema without explicit configuration (Option E).

Question: 9

A data engineering team is building a pipeline in Snowflake that uses a SQL task to call various Snowflake Cortex LLM functions (e.g., AI_COMPLETE, AI EMBED) on large datasets of customer interaction logs. The team observes fluctuating costs and occasional query failures, which sometimes halt the pipeline. To address these issues and ensure an efficient, robust, and monitorable pipeline, which of the following actions or considerations are essential? (Select all that apply.)

- ☐ Implement `TRY_COMPLETE()` instead of `AI_COMPLETE()` in the SQL task to prevent pipeline failures caused by LLM errors, allowing for graceful error handling and continuous processing.
- ☐ Regularly monitor the `SNOWFLAKE.ACCOUNT_USAGE.CORTEX_FUNCTIONS_USAGE_HISTORY` view to track token consumption and identify which LLM function calls are the primary drivers of cost.
- ☐ Increase the virtual warehouse size associated with the SQL task (e.g., to a 2XL or 3XL) to significantly improve the performance of Cortex LLM functions and reduce overall cost due to faster execution.
- ☐ Set the `temperature` parameter to 0 for all `AI_COMPLETE()` calls to guarantee the most concise and deterministic responses, thereby minimizing the number of output tokens and associated costs.
- ☐ Utilize user-defined functions (UDFs) to encapsulate complex prompt engineering logic for `AI_COMPLETE()` calls, making the prompts easier to manage and reuse programmatically within the SQL pipeline.

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

Answer: A,B,E

Explanation:
A. "Correct." The 'TRY function is designed to perform the same operation as but returns ‘NULL ‘ instead of raising an error when the LLM operation cannot be performed. This is critical for building robust data pipelines, as it prevents pipeline halts due to transient or specific LLM failures, allowing for more resilient data processing. B. ‘ The view provides detailed information on token consumption and credit usage for Snowflake Cortex LLM functions. Monitoring this view is essential for understanding cost drivers and optimizing expenditure within AI pipelines. C. "Incorrect." Snowflake recommends executing queries that call Cortex AISQL functions with a smaller warehouse (no larger than MEDIUM), as larger warehouses do not necessarily increase performance but can lead to unnecessary costs. The LLM inference itself runs on Snowflake-managed compute, not solely on the user's virtual warehouse compute size. D. ‘ Setting the 'temperature' parameter to 0 makes the LLM's output more deterministic and focused. While this can be beneficial for consistency in certain tasks, it does not directly minimize token usage. Token usage is primarily determined by the length of the input prompt and the length of the generated output, which can vary regardless of ‘temperature'. E. "Correct." Encapsulating complex and potentially lengthy prompt logic within a UDF CUSER DEFINED FUNCTION') makes the prompts more manageable, reusable, and easier to integrate programmatically into SQL statements within a data pipeline. This improves code organization and maintainability.

Question: 10

A data engineering team is setting up an automated pipeline in Snowflake to process call center transcripts. These transcripts, once loaded into a raw table, need to be enriched by extracting specific entities like the customer's name, the primary issue reported, and the proposed resolution. The extracted data must be stored in a structured JSON format in a processed table. The pipeline leverages a

SQL task that processes new records from a stream. Which of the following SQL snippets and approaches, utilizing Snowflake Cortex LLM functions, would most effectively extract this information and guarantee a structured JSON output for each transcript?

- Use `SNOWFLAKE.CORTEX.EXTRACT_ANSWER()` multiple times with separate questions for 'customer name', 'issue', and 'resolution', then use SQL JSON functions to combine the results into a single JSON object.
- Use `SNOWFLAKE.CORTEX.COMPLETE()` with a prompt like `'Extract customer name, issue, and resolution as a JSON object: [transcript_text]'` and rely on the LLM's natural ability to generate JSON.
- Use `AI_COMPLETE()` with a single prompt and explicitly define a `response_format` argument containing a JSON schema for `customer_name`, `issue`, and `resolution` fields.
- Use `SNOWFLAKE.CORTEX.SUMMARIZE()` on the transcript and then manually parse the summary text using regular expressions within SQL to extract the desired entities.
- Create a Python UDF that calls an external LLM API to extract the entities and return a JSON string, then integrate this UDF into the SQL task.

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

Answer: C

Explanation:
To guarantee a structured JSON output for entity extraction, (the updated version of 'COMPLETE()') with the response_format' argument and a specified JSON schema is the most effective approach. This mechanism enforces that the LLM's output strictly conforms to the predefined structure, including data types and required fields, significantly reducing the need for post-processing and improving data quality within the pipeline. Option A requires multiple calls and manual JSON assembly, which is less efficient. Option B relies on the LLM's 'natural ability' to generate JSON, which might not be consistently structured without explicit 'response_format' . Option D uses , which is for generating summaries, not structured entity extraction. Option E involves external LLM API calls and Python UDFs, which, while possible, is less direct than using native 'AI_COMPLETE structured outputs within a SQL pipeline in Snowflake Cortex for this specific goal.

Thank You for Trying Our Product

Special 16 USD Discount Coupon: NSZUBG3X

Email: support@examsempire.com

Check our Customer Testimonials and ratings available on every product page.

Visit our website.

https://examsempire.com/