Snowflake

NAS-C01

SnowPro Specialty: Native Apps

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: 6.0

Question: 1

You are developing a Snowflake Native App that allows consumers to enhance their existing customer data with enrichment data provided by your app. The app relies on secure data sharing. To minimize data duplication and maximize query performance within the consumer's account, which of the following approaches offers the MOST optimized and secure way to deliver the enrichment data?

A. Create a staging table in the provider account, load the enrichment data into it, and grant the consumer account SELECT privileges on this staging table. The consumer then creates their own table and copies the data.
B. Utilize Snowflake Secure Data Sharing to directly share the enrichment data tables from the provider account to the consumer account. The consumer can then create views on the shared tables or query them directly.
C. Develop a Snowflake UDF (User-Defined Function) that, when called in the consumer account, retrieves the enrichment data from an external API endpoint (maintained in the provider account) and returns it.
D. Create a secure view within the application package itself, which joins the application's enrichment data with consumer-provided data via an API integration. The consumer queries this secure view.
E. Use Snowflake Data Marketplace listing to provide enrichment data.

Answer: B

Explanation:
Option B (Utilize Snowflake Secure Data Sharing) is the most optimized and secure approach. Secure Data Sharing avoids data duplication, minimizes latency, and allows the consumer to directly access the enrichment data without needing to copy or move it. It leverages Snowflake's native sharing capabilities, ensuring security and governance. Option A involves data duplication. Option C introduces external API dependency and potential performance bottlenecks. Option D mixes consumer data with the application code, which is not optimal and less secure. Option E is suitable for data discoverability and monetization, not for application-specific, customized data enrichment.

Question: 2

A Snowflake Native App developer is building an application package that contains multiple versions. They want to ensure that a consumer upgrading from version 1.0 to version 2.0 MUST execute a specific set of SQL scripts as part of the upgrade process to migrate existing application data. Which Snowflake Native App feature should they leverage to achieve this?

A. Application Role-Based Access Control (RBAC)
B. A post-install script defined in the setup script.
C. A version change handler defined within the application package.

D. Snowflake Streams and Tasks to monitor the application version and trigger the migration scripts automatically.
E. Snowflake Pipes to load and transform the existing data.

Answer: C

Explanation:
Option C is the correct approach. A version change handler is a dedicated script that runs automatically when a consumer upgrades or downgrades the application. This handler can contain the necessary SQL scripts to migrate data, update schemas, or perform any other actions required during the version transition. Option B executes only during initial installation, not during upgrades. Option A, RBAC, manages permissions but doesn't execute scripts. Option D, Streams and Tasks, could be used, but it's more complex than a version change handler. Option E is used for continuous data loading.

Question: 3

You are designing a Snowflake Native App that needs to store temporary state information specific to each consumer account. This state information should not be directly accessible to the consumer but must be persistent across application sessions. Which of the following approaches are valid and recommended for storing this data?

A. Create a public schema in the application package and store the state information in tables within that schema. Grant SELECT access to the consumer's account on these tables.
B. Utilize internal stages within the application package to store the state data as files.
C. Create secured views that expose the state data but control access through application roles.
D. Store the state information in a private schema within the application package, ensuring that consumers do not have direct access.
E. Use external stages connected to a provider controlled AWS S3 bucket or similar.

Answer: B,D

Explanation:
Options B and D are the correct approaches. Option D: Private schema, Consumers should not have direct access to the app's internal data. Storing state information in a private schema within the application package ensures that consumers cannot directly query or modify this data, preserving the application's internal state and security. Option B: Internal Stages, offer a way to store file-based data that is tightly coupled with the application package. It enables you to persist the state information for each consumer securely. Option A is incorrect as it gives consumers direct access. Option C is incorrect because the requirement says the data cannot be accessible, not that it should be controlled. Option E creates dependency on external infrastructure.

Question: 4

You are developing a Snowflake Native App that leverages shared data from the consumer's account. The app needs to dynamically determine the existence of a specific table (e.g., 'CUSTOMER DATA) in the consumer's shared database before attempting to query it. Which of the following SQL commands or

approaches within the application package provides the MOST reliable way to programmatically check for the existence of the table?

A. Use the ‘ SHOW TABLES LIKE 'CUSTOMER_DATA" command and check if any rows are returned.
B. Execute a 'SELECT 1 FROM CUSTOMER_DATA LIMIT 1' query within a 'TRY...CATCH' block and handle the exception if the table does not exist.
C. Use the 'INFORMATION SCHEMA.TABLES' view and query it to check if a table with the name 'CUSTOMER DATA' exists in the shared database.
D. Use the 'DESCRIBE TABLE CUSTOMER_DATX command within a 'TRY...CATCH' block and handle the exception if the table does not exist.
E. Attempt to grant SELECT privilege on the table and check if an error occurs

Answer: C

Explanation:
Option C (Use the ‘INFORMATION SCHEMATABLES view) is the most reliable and recommended approach. The INFORMATION_SCHEMR provides metadata about database objects, and querying it to check for the existence of a table is a standard and efficient practice. Option A ('SHOW TABLES) might be affected by the session context or user's privileges. Options B and D (using 'TRY...CATCH' blocks) rely on exception handling, which can be less efficient and less readable. Also, consuming exceptions can be more expensive than directly querying system views. Option E is inefficient since it requires an attempt to change a permission.

Question: 5

A data science company is developing a Snowflake Native App that provides advanced fraud detection for financial institutions. The app requires secure access to sensitive transaction data within the consumer's Snowflake account. Which of the following approaches BEST aligns with Snowflake's recommended practices for managing data access and securing data within a Native App?

A. The app should request full ACCOUNTADMIN privileges in the consumer account to access all necessary data for fraud detection.
B. The app should use secure views created by the consumer that grant the app read-only access to only the specific transaction data needed for fraud detection.
C. The app should directly access tables containing transaction data using the consumer's own service account credentials, which are securely stored within the app's code.
D. The app should create and manage its own temporary tables in the consumer account and load transaction data into them for processing.
E. The app should directly access transaction data through data sharing from provider account.

Answer: B

Explanation:
Using secure views is the best approach. Secure views allow the consumer to control precisely whatdata the application can access without granting overly broad permissions. This minimizes the risk of unintended data exposure and adheres to the principle of least privilege. ACCOUNTADMIN is excessive

and insecure. Storing credentials is also insecure and against best practices. Creating temporary tables and loading data duplicates data and can be inefficient.

Question: 6

You are developing a Snowflake Native App that requires specific privileges to be granted to the consumer account. These privileges are necessary for the app to access and process data within the consumer's Snowflake environment. Which section of the manifest file is primarily used to declare these required consumer-side privileges?

A.
```
application.privileges
```
B.
```
application.behavior
```
C.
```
application.resources
```
D.
```
application.setup
```
E.
```
application.initialization
```

Answer: A.

Explanation:
The application.privileges section in the manifest file is specifically designed to declare the privileges required by the Native App in application . privileges the consumer's account. This ensures that the app has the necessary permissions to function correctly within the consumer's Snowflake environment. The other options are related to different aspects of the app's behavior, resources, and setup but not the declaration of required privileges.

Question: 7

A Snowflake Native App is designed to perform complex data transformations on large datasets provided by the consumer. To optimize performance and manage resource consumption, you need to choose the appropriate execution context for your application logic. Which of the following options should be considered to accomplish this?

A. Executing all data transformations within a single stored procedure to minimize overhead.
B. Distributing the transformations across multiple SQL UDFs running in parallel using Snowflake's compute grid.
C. Leveraging Snowflake's external functions to offload data transformations to a separate cloud-based compute environment.

D. Using a combination of stored procedures, SQL UDFs, and external functions to orchestrate the data transformation process, depending on the specific requirements of each step.
E. Run entire process by calling API from provider side.

Answer: D

Explanation:
Choosing the correct execution context for your application logic requires a combination of stored procedures, SQL UDFs, and external functions to orchestrate the data transformation process, depending on the specific requirements of each step, offering the best balance of performance, scalability, and resource management. A single stored procedure may not be efficient for complex transformations. SQL UDFs in parallel can be effective but might have limitations depending on the complexity. External functions offer flexibility for offloading but introduce latency and complexity. API call will be more appropriate to transfer the data.

Question: 8

You are developing a Snowflake Native App that needs to persist state information between different invocations. The app requires tracking of user preferences, processing progress, and other runtime data. Which of the following options are viable and secure methods for persisting this type of state information within the context of a Snowflake Native App?

A. Using temporary tables within the consumer's account to store state information. The tables are dropped when the app is uninstalled.
B. Using the application provider's own Snowflake account to store state information associated with each consumer.
C. Storing state information within the application package itself by updating the package version with each state change.
D. Using internal stages and secured views in consumer account to persist state information.
E. Using secure external stages managed by the application provider to store the consumer-specific state information.

Answer: B,D

Explanation:
Persisting state information requires careful consideration of security, scalability, and isolation. Using the application provider's own Snowflake account is a viable approach if designed carefully, but requires robust security measures to isolate consumer data. Internal stages and secure views offer a more secure and controlled way to persist state within the consumer's account. Temporary tables are ephemeral and unsuitable for long-term state persistence. Storing state within the application package is impractical and would lead to versioning issues. External stages managed by the provider introduce complexity and potential security risks.

Question: 9

You are developing a Snowflake Native App that requires accessing sensitive user data within the consumer's account. To ensure data privacy and security, you need to implement appropriate access controls. Which of the following approaches is the MOST secure and recommended way to grant your app access to this data, assuming the data is stored in a table named ‘user_data’ ?

A. Granting 'USAGE privilege on the 'user_data' table directly to the application role provided by Snowflake Native App Framework.
B. Creating a custom role in the consumer account with 'SELECT privilege on the ‘user_data’ table and granting that role to the application.
C. Defining a secure view that filters the table based on the application role and granting ‘SELECT privilege on the secure view to the application role provided by Snowflake Native App Framework.
D. Using a stored procedure with 'EXECUTE AS CALLER that accesses the 'user_data' table. Grant 'EXECUTE privilege on the stored procedure to the application role.
E. Granting 'SELECT privilege directly on the 'user_data' table to the application role provided by Snowflake Native App Framework.

Answer: C

Explanation:
The most secure approach is to use a secure view. This allows you to control which rows and columns the application can access. Granting 'SELECT directly to the application role is generally discouraged as it exposes the entire table. Using 'EXECUTE AS CALLER has security implications and might not be desirable. Creating custom roles in the consumer account adds complexity to the application deployment.

Question: 10

A Native App provider wants to use a parameter in their MANIFEST.yml to dynamically control the number of worker threads a stored procedure uses. The desired behavior is to allow the consumer to configure this value. Considering security best practices, which of the following parameter definitions in the 'MANIFEST.yml' is the MOST appropriate?

A.
```yaml parameters: WORKER_THREADS: type: INT default: 4 secure: false ```
B.
```yaml parameters: WORKER_THREADS: type: STRING default: '4' secure: true ```
C.
```yaml parameters: WORKER_THREADS: type: INT default: 4 secure: true ```
D.
```yaml parameters: WORKER_THREADS: type: VARCHAR default: '4' secure: false ```
E.
```yaml parameters: WORKER_THREADS: type: INT default: 4 secure: true validation_rules: - type: range min: 1 max: 16 ```
```

Answer: E

Explanation:
Option E is the MOST appropriate. While ‘secure: true' is important to encrypt and protect the parameter value, ‘validation_rules’ provide an extra layer of security and data integrity by ensuring the consumer provides a valid value within a defined range. It prevents potentially harmful or unexpected values from being used, enhancing the app's robustness. Type INT also correctly defines the expected data type.

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/