PassQueen
a.k.a dumpsqueen.com

SnowPro Advanced: Data Engineer Certification Exam

Snowflake DEA-C01

Version Demo

Total Demo Questions: 25

Total Premium Questions: 250

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Data Movement	69
Topic 2, Data Transformation	72
Topic 3, Data Governance	47
Topic 4, Data Storage	62
Total	250

QUESTION NO: 1

As Data Engineer, you have been asked to access data held in AWS Glacier Deep Archive storage class for Historical Data Analysis, which one is the correct statement to recommend?

A. You cannot access data held in archival cloud storage classes that requires restoration before it can be retrieved.

B. Loading data from AWS cloud storage services is supported regardless of the cloud platform that hosts your Snowflake account.

C. Data can be accessed from External stage using AWS Private link in this case.

D. We can simply access AWS Glacier Deep Archive storage External Stage data using PUT command.

E. Upload (i.e. stage) files to your cloud storage account using the tools provided by the cloud storage service.

ANSWER: A

Explanation:

You cannot access data held in archival cloud storage classes that requires restoration before it can be retrieved. Snowflake external stages access files stored in cloud object storage only when those objects are available for immediate retrieval. Amazon S3 Glacier Deep Archive is an archival storage class in which objects must first be restored through Amazon S3 before their content can be read. Consequently, files that remain in Glacier Deep Archive cannot be directly loaded into Snowflake or accessed through an external stage for analysis. A historical-data workflow must account for the restore process and its associated delay. After restoring the objects, the organization should ensure the files are available in a retrievable S3 state or copy them to a suitable accessible location, then use the relevant Snowflake stage and loading or querying workflow. Snowflake explicitly lists Amazon S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive among storage classes whose files Snowflake cannot access directly. This constraint applies to the availability of the objects themselves, rather than to the syntax used to define the stage. See Snowflake's Amazon S3 storage integration documentation and its staging data considerations for supported cloud-storage access requirements.

QUESTION NO: 2

Which methods can be used to create a DataFrame object in Snowpark? (Select THREE)

A. session.jdbc_connection()

B. session.read.json()

C. session.table()

D. DataFrame.write()

E. session.builder()

F. session.sql()

ANSWER: B C F

Explanation:

`session.read.json()`, `session.table()`, and `session.sql()` each return a Snowpark DataFrame, enabling work to be defined lazily and executed in Snowflake. The JSON reader is available through the session's `read` property and creates a DataFrame representing semi-structured JSON data from a supported source such as a staged file. This makes it possible to apply Snowpark transformations directly after reading the data. Snowflake documents this reader in the DataFrameReader.json API reference.

`session.table()` creates a DataFrame that represents a specified Snowflake table, allowing table data to be queried and transformed with the Snowpark DataFrame API. `session.sql()` similarly creates a DataFrame from a SQL statement, so SQL query results can participate in subsequent DataFrame operations. Both methods are documented as Session APIs that return DataFrame objects; see Snowflake's Session API reference. These creation methods support Snowpark's

deferred-execution model: constructing the DataFrame defines the work, while an action such as collecting results initiates execution.

QUESTION NO: 3

Which property can be used with ALTER USER command to temporarily disable MFA for the user so

that they can log in?

A. HOURS_TO_BYPASS_MFA

B. SECS_TO_BYPASS_MFA

C. MINS_TO_SKIP_MFA

D. MINS_TO_BYPASS_MFA

ANSWER: D

Explanation:

`MINS_TO_BYPASS_MFA` is the Snowflake user property intended for a temporary, time-bounded MFA bypass. An administrator can set it through `ALTER USER`, for example, `ALTER USER user_name SET MINS_TO_BYPASS_MFA = 30;`. The value specifies the number of minutes during which the designated user can sign in without completing the configured MFA challenge. This is useful for an account-recovery scenario, such as when a user has lost or replaced the device used by their authenticator application and needs to access Snowflake to register MFA again.

The bypass is deliberately measured in minutes and expires automatically after the configured period. This supports the security principle of granting only a short-lived exception rather than disabling MFA indefinitely. The setting applies to the individual user and should be used by an administrator only for the minimum period needed to restore access. Snowflake documents `MINS_TO_BYPASS_MFA` as an `ALTER USER` property and describes its role in temporarily bypassing multi-factor authentication. See the ALTER USER SQL reference and Snowflake's multi-factor authentication documentation.

QUESTION NO: 4

To troubleshoot data load failure in one of your Copy Statement, Data Engineer have Executed a COPY statement with the VALIDATION_MODE copy option set to RETURN_ALL_ERRORS with reference to the set of files he had attempted to load. Which below function can facilitate analysis of the problematic records on top of the Results produced? [Select 2]

A. RESULT_SCAN

B. LAST_QUERY_ID

C. Rejected_record

D. LOAD_ERROR

ANSWER: A B

Explanation:

`LAST_QUERY_ID` and `RESULT_SCAN` provide the standard Snowflake workflow for analyzing rows reported by a validation-only `COPY INTO` operation. When `VALIDATION_MODE = RETURN_ALL_ERRORS` is specified, Snowflake validates the files and returns error information rather than loading data. `LAST_QUERY_ID()` obtains the query ID for the preceding validation statement in the current session, allowing the validation output to be referenced reliably. That identifier can then be supplied to `RESULT_SCAN`, which exposes the prior statement's result set as queryable table data. For example, a user can run `SELECT * FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()))` immediately after the validation command, or save the ID

and use it explicitly in later analysis. This makes it possible to filter, inspect, transform, or join the returned error rows, including the record content and diagnostic details generated by the validation process. Snowflake retains query results only for a limited period, so this inspection should occur while the result remains available and under an appropriate role context. See Snowflake's documentation for LAST_QUERY_ID and RESULT_SCAN.

QUESTION NO: 5

Snowflake does not treat the inner transaction as nested; instead, the inner transaction is a separate transaction. What is term used to call these Transaction?

A. Scoped transactions

B. Inner Transaction

C. Nested Scope Transaction

D. Atomic Transaction
Enclosed Transaction

ANSWER: A

Explanation:

Scoped transactions is the Snowflake term for this behavior. Snowflake does not support traditional nested transactions, in which an inner transaction is a child of an outer transaction and its outcome is inherently governed by the outer transaction. Instead, Snowflake can have transaction scopes that are independent of one another, particularly when a transaction is started within a stored procedure while another transaction exists outside that procedure. Each transaction must be completed within the scope in which it was started; a transaction begun in a stored procedure must be committed or rolled back there, and a transaction begun outside the procedure cannot be committed or rolled back by the procedure.

This scope-based model establishes clear ownership of transaction boundaries and avoids ambiguous nested commit or rollback behavior. Consequently, an apparently inner transaction is not a nested child transaction; it is a separate transaction with its own scope and completion requirement. Snowflake documents this explicitly under its transaction guidance as "scoped transactions." See Snowflake's Scoped transactions documentation and the Stored procedure transaction management documentation for the rules governing transaction scope, commit, and rollback behavior.

QUESTION NO: 6

Which statements correctly compare the query-history interfaces available in Snowflake? [Select 3]

A. The `SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY` view retains query history for up to one year.

B. Data in the Account Usage QUERY_HISTORY view can have latency of up to 45 minutes.

C. The Information Schema QUERY_HISTORY table function provides query history for the previous 7 days.

D. The Information Schema QUERY_HISTORY table function retains query history for one year.

E. Account Usage query history is updated synchronously when every query completes.

ANSWER: A B C

Explanation:

The `SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY` view retains query history for up to one year and is appropriate for account-level reporting and longer-term monitoring. Account Usage data is not real-time and can have latency, which can be up to 45 minutes for this view.

The `INFORMATION_SCHEMA.QUERY_HISTORY` table function provides more recent query activity with lower latency, but its query-history retention is limited to the previous 7 days. Engineers commonly use the Information Schema function for operational troubleshooting and Account Usage for centralized historical reporting. See the QUERY_HISTORY Account Usage view and the QUERY_HISTORY table function.

QUESTION NO: 7

A new customer table is created by a data pipeline in a Snowflake schema where MANAGED ACCESSenabled.

…. Can gran access to the CUSTOMER table? (Select THREE.)

A. The role that owns the schema

B. The role that owns the database

C. The role that owns the customer table

D. The SYSADMIN role

E. The SECURITYADMIN role

F. The USERADMIN role with the manage grants privilege

ANSWER: A E F

Explanation:

In a managed access schema, grant management is centralized rather than delegated to individual object owners. Therefore, **The role that owns the schema** can grant privileges on the CUSTOMER table because ownership of the managed access schema includes authority to manage grants for objects within that schema. **The SECURITYADMIN role** can also grant access because it is granted the account-level MANAGE GRANTS privilege by default. MANAGE GRANTS enables a role to grant and revoke object privileges broadly, including privileges on objects in managed access schemas.

Likewise, **The USERADMIN role with the manage grants privilege** can grant access to the CUSTOMER table. The role's name does not restrict the scope of an explicitly granted account-level MANAGE GRANTS privilege; the effective privileges assigned to the role determine whether it can manage grants. Snowflake documents that, for managed access schemas, grants on objects are controlled by the schema owner or a role holding MANAGE GRANTS, rather than by the owners of individual objects. See Snowflake's guidance on managed access schemas and its MANAGE GRANTS privilege documentation.

QUESTION NO: 8

A table is loaded using Snowpipe and truncated afterwards Later, a Data Engineer finds that the table needs to be reloaded but the metadata of the pipe will not allow the same files to be loaded again.

How can this issue be solved using the LEAST amount of operational overhead?

A. Wait until the metadata expires and then reload the file using Snowpipe

B. Modify the file by adding a blank row to the bottom and re-stage the file

C. Set the FORCE=TRUE option in the Snowpipe COPY INTO command

D. Recreate the pipe by using the create or replace pipe command

ANSWER: A

Explanation:

Wait until the metadata expires and then reload the file using Snowpipe is the lowest-overhead approach because Snowpipe tracks successfully loaded files in its load metadata to prevent duplicate ingestion. This tracking is retained for 14 days. Once

the relevant file metadata has expired, Snowpipe can process those same staged files again without requiring an engineer to alter the data files, create replacement staged objects, or perform pipe maintenance. This approach is particularly suitable when restoring the truncated table is not time-critical and the original files remain available in the stage. The engineer can then initiate or resume the applicable loading workflow after the retention window has elapsed, allowing the normal Snowpipe process to load the files as new candidates. Snowflake documents that Snowpipe load metadata is retained for 14 days and that this history is used to prevent duplicate loads. See the Snowpipe file-loading metadata documentation and the load metadata considerations documentation for details on the retention behavior and duplicate-load protection.

QUESTION NO: 9

Which Scenario Data engineer decide Materialized views are not useful. Select All that apply.

A. Query results contain a small number of rows and/or columns relative to the base table (the table on which the view is defined).

B. Query results contain results that require significant processing.

C. The query is on an external table (i.e. data sets stored in files in an external stage), which might have slower performance compared to querying native database tables.

D. The view's base table changes frequently.

ANSWER: D

Explanation:

The view’s base table changes frequently. is the scenario in which a data engineer would generally decide that a materialized view is not useful. Snowflake automatically maintains a materialized view so that its stored results remain consistent with the data in the base table. When the base table receives frequent inserts, updates, deletes, or merges, this ongoing maintenance can consume Snowflake compute resources and increase the overall cost of the solution. The performance benefit from precomputing query results may therefore be outweighed by the maintenance cost, particularly when the materialized view is not queried frequently enough to justify that cost.

Materialized views are most valuable when they accelerate recurring, resource-intensive queries against data whose relevant portions are comparatively stable. Before creating one, engineers should evaluate both the expected query savings and the credits required for automatic maintenance. Snowflake exposes materialized-view maintenance costs through the relevant account usage views, enabling teams to assess whether the performance and cost trade-off remains favorable as data-loading and DML patterns change. For implementation guidance and limitations, see Snowflake’s Materialized Views documentation and its CREATE MATERIALIZED VIEW reference.

QUESTION NO: 10

Which are supported Programming Languages for Creating UDTFs?

A. Python

B. Node.javascript

C. Javascript

D. Java

E. Perl

ANSWER: A C D

Explanation:

Python, Javascript, and Java are supported languages for implementing Snowflake user-defined table functions (UDTFs). A UDTF invokes handler code that can emit zero, one, or multiple output rows for each input row, making it useful for transformations that expand or generate tabular results. Snowflake supports JavaScript UDTFs created with SQL and a

JavaScript handler. It also supports Java and Python UDTFs through Snowpark, where the handler class or function is registered or defined for execution in Snowflake. These languages are part of Snowflake's documented supported handler-language model for UDTFs, allowing logic to execute within Snowflake's governed execution environment while returning a relational result set that can be queried in SQL. The supported-language list also includes Scala in the broader Snowflake documentation, but it is not among the choices provided. See Snowflake's User-Defined Table Functions documentation for supported UDTF handler languages and implementation patterns, and the Snowpark Python UDTF documentation for Python-specific registration and handler guidance.

QUESTION NO: 11

Streams cannot be created to query change data on which of the following objects? [Select All that Apply]

A. Standard tables, including shared tables.

B. Views, including secure views

C. Directory tables

D. Query Log Tables

E. External tables

ANSWER: D

Explanation:

Query Log Tables is correct because Snowflake streams provide change data capture (CDC) only for supported database objects whose changes Snowflake can track. A stream is created on a table, external table, directory table, or view, and it exposes the changes that have occurred to the underlying supported object since the stream was last consumed or advanced. Query-log information, however, is operational metadata generated by Snowflake and exposed through system-provided interfaces such as Account Usage and Information Schema query-history views. It is not maintained as a user table or another supported stream source with change tracking that can be enabled and consumed through a stream.

This distinction is important in data-engineering designs: query history can be queried, filtered, and copied into a regular table for downstream processing if needed, but a stream cannot be directly created over the query-log metadata itself. Snowflake's documentation lists the supported source object types for `CREATE STREAM`, including standard tables, external tables, directory tables, and views. The Streams guide also explains that streams record CDC information for supported source objects rather than serving as a general mechanism for tracking changes in Snowflake system metadata. See the CREATE STREAM reference and Snowflake's Streams introduction.

QUESTION NO: 12

Search optimization works best to improve the performance of a query when the following conditions are true:[Select All that apply]

A. The table is not clustered.

B. The table is frequently queried on columns other than the primary cluster key.

C. Search Query uses Equality predicates (for example, column_name = constant) OR predicates that use IN.

D. Search Query uses Sort Operations.
Explanation:

ANSWER: A B C

Explanation:

Search optimization is most valuable when it can avoid scanning a large number of micro-partitions for a highly selective lookup. "The table is not clustered" is an appropriate condition because clustering can already provide effective partition

pruning for columns aligned with the clustering key. Where that physical ordering is absent, the search access path can provide an efficient alternative for supported predicate patterns without requiring the table to be reclustered.

“The table is frequently queried on columns other than the primary cluster key” is also an important use case. A clustering key primarily helps queries whose filters align with that key, whereas search optimization can be configured for supported non-clustering columns that are frequently used to locate a small subset of rows.

“Search Query uses Equality predicates (for example, column_name = constant) OR predicates that use IN” describes a core selective-query pattern supported by the service. Snowflake documents equality and `IN` predicates as search-optimization use cases because the access path can identify the relevant micro-partitions efficiently. Actual benefit still depends on selectivity, table size, cardinality, and workload frequency, so teams should use query profiling and cost monitoring when deciding whether to enable the service. See Snowflake’s Search Optimization Service documentation and its supported query predicates guidance.

QUESTION NO: 13

How Data Engineer can do Monitoring of Files which are Staged Internally during Continuous data pipelines loading process? [Select all that apply]

A. She Can Monitor the files using Metadata maintained by Snowflake i.e. file-name,last_modified date etc.

B. Snowflake retains historical data for COPY INTO commands executed within the pre-vious 14 days.

C. She can monitor the status of each COPY INTO operation by querying COPY_HISTORY.

D. She can use the DATA_LOAD_HISTORY Information Schema view to retrieve the history of data loaded into tables using the COPY INTO command.

E. She can use the VALIDATE table function to retrieve errors encountered while loading data files.

ANSWER: A B C E

Explanation:

Engineers can monitor an internal stage directly with the `LIST` command. Its output includes file metadata such as the staged file name, size, MD5 checksum where applicable, and last-modified timestamp, allowing a pipeline to identify newly arrived files and inspect the contents of a stage. Snowflake also records recent bulk-load activity. The Information Schema `COPY_HISTORY` table function exposes load details for `COPY INTO` operations during the preceding 14 days, including individual file names, load status, row counts, and error information. This history can support operational queries, dashboards, and alerts for continuous ingestion processes.

For a specific load, an engineer can use the query ID returned by a `COPY INTO` statement to inspect that load and its per-file outcomes through `COPY_HISTORY`. When a load completes with tolerated errors, the `VALIDATE` table function can retrieve errors for the relevant load job, enabling targeted investigation of rejected records and parsing issues. Together, stage listing, copy history, and validation provide both arrival monitoring and load-result monitoring. See Snowflake’s LIST command documentation and COPY_HISTORY documentation.

QUESTION NO: 14

Snowflake computes and adds partitions based on the defined partition column expressions when an external table metadata is refreshed.

What are the Correct Statements to configure Partition metadata refresh in case of External Tables?

A. By default, the metadata is refreshed automatically when the object is created.

B. The object owner can configure the metadata to refresh automatically when new or updated data files are available in the external stage.

C. Metadata refresh is not required as its Managed implicitly by Snowflake.

D. Partitions of External tables is managed by External Stage Cloud provider.

E. There is nothing like adding partitions on External tables.

Explanation:

ANSWER: A B

Explanation:

By default, the metadata is refreshed automatically when the object is created. This is correct because the `CREATE EXTERNAL TABLE` command performs an initial metadata refresh by default through the `REFRESH_ON_CREATE = TRUE` setting. During that refresh, Snowflake scans the files available in the external stage, registers their metadata, and evaluates the external table's partition expressions to derive and store partition values. This establishes the initial set of files and partitions that can be queried through the external table.

The object owner can configure the metadata to refresh automatically when new or updated data files are available in the external stage. This is also correct. External-table metadata can be maintained automatically by setting `AUTO_REFRESH = TRUE` and configuring the applicable cloud-event notification integration for the external stage. Snowflake then receives object-storage events and refreshes the table metadata as files arrive or are updated, allowing partition metadata derived from the defined expressions to remain current. The configuration is managed through external-table properties and the associated notification setup, with the required ownership and privileges.

See Snowflake's documentation for CREATE EXTERNAL TABLE and automatic refresh of external tables.

QUESTION NO: 15

Data Engineer Loading File named snowdata.tsv in the /datadir directory from his local machine to Snowflake stage and try to prefix the file with a folder named tablestage, please mark the correct command which helps him to load the files data into snowflake internal Table stage?

A. put file://c:\datadir\snowdata.tsv @~/tablestage;

B. put file://c:\datadir\snowdata.tsv @%tablestage;

C. put file://c:\datadir\snowdata.tsv @tablestage;

D. put file:///datadir/snowdata.tsv @%tablestage;

ANSWER: B

Explanation:

`put file://c:\datadir\snowdata.tsv @%tablestage;` is the correct command because `PUT` uploads a file from a client-local filesystem to a Snowflake stage, and the `file://` URI identifies the local source file. The Windows-style path in this command is appropriate for the stated local-file form. Snowflake uses the `@%<table_name>` stage-location syntax for the internal stage automatically associated with a table. Therefore, `@%tablestage` targets the internal table stage belonging to the table named `tablestage`. A table stage is created automatically with its table and is suitable for staging data files before loading them with a subsequent `COPY INTO` command. The command may also include a stage path after the table-stage reference when files must be organized under a specific directory-like prefix, but no such additional path is required to address the table stage itself. Snowflake documents that `PUT` transfers local files into internal stages and identifies table stages with the `@%` notation. See the PUT command reference and Snowflake's documentation for loading files from a local filesystem.

QUESTION NO: 16

Select the correct usage statements with regards to SQL UDF?

A. The body of a UDF cannot contain DDL statements or any DML statement other than SELECT.

B. Scalar functions (UDFs) have a limit of 500 input arguments.

C. When using a query expression in a SQL UDF, do not include a semicolon within the UDF body to terminate the query expression.

D. You can include only one query expression.

E. All of above are correct.

ANSWER: A B C D E

Explanation:

All of the listed statements accurately describe Snowflake SQL UDF usage and limits. A SQL UDF body is restricted to SQL expressions and query expressions; it is not a multi-statement procedural container. Consequently, the body cannot execute DDL and cannot execute DML operations other than a SELECT query used to compute the function result. Snowflake also documents that a UDF can define up to 500 input arguments, which applies to scalar UDFs. When the body uses a query expression, it must be a single query expression, and the query inside the function body must not be terminated with a semicolon. The semicolon terminates the overall CREATE FUNCTION statement rather than an embedded query in the UDF definition. These constraints enable SQL UDFs to encapsulate reusable SQL logic that produces a value or tabular result while remaining within Snowflake's function execution model. Because the statements concerning DDL/DML restrictions, the 500-argument limit, omission of an internal semicolon, and the one-query-expression limit are all correct, "All of above are correct." is correct as well. See Snowflake's SQL UDF reference and CREATE FUNCTION documentation.

QUESTION NO: 17

Which query will show a list of the 20 most recent executions of a specified task kttask, that have been scheduled within the last hour that have ended or are stillrunning's.

A)

B)

C)

D)

A. Option A

B. Option B

C. Option C

D. Option D

E. SELECT * FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY(TASK_NAME => 'KTTASK', SCHEDULED_TIME_RANGE_START => DATEADD('hour', -1, CURRENT_TIMESTAMP()), SCHEDULED_TIME_RANGE_END => CURRENT_TIMESTAMP(), RESULT_LIMIT => 20)) ORDER BY SCHEDULED_TIME DESC;

ANSWER: E

Explanation:

`SELECT * FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY(TASK_NAME => 'KTTASK', SCHEDULED_TIME_RANGE_START => DATEADD('hour', -1, CURRENT_TIMESTAMP()), SCHEDULED_TIME_RANGE_END => CURRENT_TIMESTAMP(), RESULT_LIMIT => 20)) ORDER BY SCHEDULED_TIME DESC;` is correct because the `TASK_HISTORY` Information Schema table function is designed to retrieve task-run history. Supplying `TASK_NAME` scopes the results to `KTTASK`. The scheduled-time start and end arguments restrict results to runs scheduled during the preceding hour. `RESULT_LIMIT => 20` limits the returned history to twenty records, and sorting by `SCHEDULED_TIME` in descending order places the most recently scheduled runs first. The function includes runs that have completed as well as executions that are currently in progress, which satisfies the requirement to show

executions that have ended or are still running. The task name should use the appropriate identifier form for the task being queried; an unquoted name such as `KTTASK` is interpreted as uppercase by Snowflake. See Snowflake's TASK_HISTORY documentation for the supported arguments, returned execution states, and time-range behavior. Snowflake also documents task monitoring and history retrieval in its Tasks guide.

QUESTION NO: 18

Which Function would Data engineer used to recursively resume all tasks in Chain of Tasks rather than resuming each task individually (using ALTER TASK *€¦ RESUME)?

A. SYSTEM$TASK_DEPENDENTS

B. SYSTEM$TASK_DEPENDENTS_ENABLE

C. SYSTEM$TASK_DEPENDENTS_RESUME

D. SYSTEM$TASK_RECURSIVE_ENABLE

ANSWER: B

Explanation:

`SYSTEM$TASK_DEPENDENTS_ENABLE` is the Snowflake system function designed to enable a task and all of its dependent child tasks in a task graph. It is intended for a directed acyclic graph of tasks, including a simple chain in which each downstream task depends on the preceding task. Calling the function with the fully qualified name of the root task recursively enables the eligible dependent tasks, eliminating the operational need to issue a separate `ALTER TASK ... RESUME` command for every task.

This is particularly useful when task graphs are deployed, migrated, or restarted after a maintenance activity. Snowflake requires task dependencies to be managed consistently: a root task can be resumed directly, and dependent tasks must be enabled before they can execute when their predecessor completes. The function automates that dependency traversal and enablement process, helping ensure that downstream tasks in the graph are made ready together. The task owner must have the necessary privileges, and the supplied task name identifies the root of the dependent-task hierarchy to process. Snowflake documents the function's syntax, behavior, and return value at SYSTEM$TASK_DEPENDENTS_ENABLE. For task-graph concepts and dependency behavior, see Snowflake's Introduction to tasks.

QUESTION NO: 19

A Data Engineer is creating an external stage that accesses a cloud storage location by using a storage integration. Which statements are correct? [Select 3]

A. A storage integration is a securable Snowflake object.

B. A role needs the applicable `USAGE` privilege on the storage integration to use it with a stage.

C. Allowed and blocked storage locations can restrict the cloud locations that stages use through the integration.

D. The cloud access credentials must be embedded directly in every stage that uses the integration.

E. A storage integration can be used only with internal Snowflake stages.

ANSWER: A B C

Explanation:

A storage integration is a Snowflake securable object that stores the cloud-storage identity and permitted storage locations. This separates cloud authentication details from individual stage definitions and allows centralized administration of approved external locations.

A role creating or using an external stage with a storage integration requires the applicable `USAGE` privilege on that integration. The integration can also restrict access through allowed and blocked storage locations, helping prevent stages from referencing unauthorized cloud-storage paths. See Snowflake documentation for Storage integrations and CREATE STORAGE INTEGRATION.

QUESTION NO: 20

A Data Engineer has created a row access policy named `REGION_ACCESS_POLICY` and needs to apply it to an existing table using the `REGION` column as the policy argument. Which command is correct?

A. `ALTER TABLE sales ADD ROW ACCESS POLICY REGION_ACCESS_POLICY ON (region);`

B. `GRANT ROW ACCESS POLICY REGION_ACCESS_POLICY TO TABLE sales;`

C. `ALTER TABLE sales SET ROW ACCESS POLICY = REGION_ACCESS_POLICY;`

D. `CREATE ROW ACCESS POLICY REGION_ACCESS_POLICY ON TABLE sales (region);`

ANSWER: A

Explanation:

`ALTER TABLE sales ADD ROW ACCESS POLICY REGION_ACCESS_POLICY ON (region)` is correct. A row access policy is applied to a table by using the `ADD ROW ACCESS POLICY` clause and specifying the table column or columns that correspond to the policy signature.

After the policy is attached, Snowflake evaluates the policy at query time and filters rows according to the policy expression and the querying context. See the ALTER TABLE reference and Snowflake guidance on row access policies.

QUESTION NO: 21

Michael, a Data Engineer Running a Data query to achieve Union of Data sets coming from Multi-ple data sources, later he figured out that Data processing query is taking more time than expected. He started analyzing the Query performance using query profile interface. He discovered & realized that he used UNION when the UNION ALL semantics was sufficient.

Which Extra Data Processing Operator Michael figured out while doing query profile analysis in this case which helps him to identify this performance bottlenecks?

A. Aggregate

B. UNION ALL

C. Flatten

D. Join

E. Filter **Explanation:**

ANSWER: A

Explanation:

`Aggregate` is correct because a Snowflake `UNION` returns only distinct rows across its input result sets. Therefore, unlike `UNION ALL`, which can concatenate the inputs directly, `UNION` requires duplicate elimination after combining the results. In the Query Profile, Snowflake represents that additional distinct-processing work with an Aggregate operator. This operator processes the combined rows to identify and remove duplicates, effectively applying the set semantics required by `UNION`.

Seeing an Aggregate operator above or following the union-related processing is an important diagnostic signal when reviewing a slow query. Duplicate elimination may require substantial compute and memory, particularly when the inputs contain many rows or high-cardinality values. It can also increase the amount of intermediate data processed by the

execution plan. If the business logic does not require duplicate removal, changing the query to `UNION ALL` avoids this aggregation work while retaining all rows from each source. Snowflake documents that `UNION` eliminates duplicates and `UNION ALL` does not; its Query Profile documentation also describes Aggregate as an operator type used for aggregation-related processing. See Snowflake set operators and Query Profile operator types.

QUESTION NO: 22

The JSON below is stored in a variant column named v in a table named jCustRaw:

Which query will return one row per team member (stored in the teamMembers array) along all of the attributes of each team member?

A)

B)

C)

D)

A. Option A

B. SELECT f.value AS team_member
FROM jCustRaw,
LATERAL FLATTEN(input => v:teamMembers) f;

C. Option C

D. Option D

ANSWER: B

Explanation:

`SELECT f.value AS team_member FROM jCustRaw, LATERAL FLATTEN(input => v:teamMembers) f;` is the appropriate query because `FLATTEN` transforms the elements of a semi-structured array into a set of rows. The `input` expression navigates from the `VARIANT` column to the `teamMembers` array using Snowflake path notation. Applying `LATERAL` makes the table function run for each source row in `jCustRaw`, so its output remains associated with the JSON document from which the array originated.

For each array element, the `VALUE` output column produced by `FLATTEN` contains the full team-member object as a `VARIANT` value. Consequently, every member becomes one result row and all attributes belonging to that member are retained together in `team_member`. Those attributes can subsequently be accessed with path expressions, such as `team_member:name`, if individual relational columns are needed. This approach is the standard Snowflake pattern for expanding an array while preserving each array element's complete structure. See Snowflake's FLATTEN documentation and its guide to querying semi-structured data.

QUESTION NO: 23

A SQL UDF evaluates an arbitrary SQL expression and returns the result(s) of the expression. Which value type it can returns?

A. Single Value

B. A Set of Rows

C. Scalar or tabular depending on the input SQL expression

D. Regex

ANSWER: C

Explanation:

Scalar or tabular depending on the input SQL expression is correct. Snowflake supports SQL UDFs in two return forms. A scalar SQL UDF evaluates a SQL expression and returns one value for each invocation; its declared return type is a Snowflake scalar data type, such as NUMBER, VARCHAR, BOOLEAN, DATE, or VARIANT. A tabular SQL UDF, also called a user-defined table function, returns a result set consisting of zero or more rows and one or more declared output columns. The appropriate form is determined when the function is defined: scalar functions use a scalar return type and an expression body, while tabular functions declare a table-shaped return signature and use a query body that produces rows. Therefore, SQL UDF functionality is not limited to only one individual value type; it can produce either a scalar value or tabular data, according to the kind of SQL UDF and SQL body being created. Snowflake documents the syntax and behavior of scalar SQL UDFs at Introduction to SQL UDFs and documents table-returning functions at Tabular SQL UDFs.

QUESTION NO: 24

Mark the correct Statements with respect to Secure views & its creation in the SnowFlake Account?

A. For a secure view, internal optimizations can indirectly expose data & the view defini-tion is visible to other users.

B. Secure views should not be used for views that are defined solely for query convenience, such as views created to simplify queries for which users need to understand the underlying data representation.

C. To convert an existing view to a secure view and back to a regular view, set/unset the SECURE keyword in the ALTER VIEW or ALTER MATERIALIZED VIEW com-mand.

D. For non-materialized views, the IS_SECURE column in the Information Schema and Account Usage views identifies whether a view is secure.

E. The internals of a secure view are not exposed in Query Profile (in the web interface). This is the case even for the owner of the secure view, because non-owners might have access to an owner's Query Profile.

ANSWER: B C D E

Explanation:

Secure views are intended to protect sensitive data and implementation details when a view is shared with roles that should not have visibility into its underlying query logic or base objects. They are therefore appropriate when the view acts as a governed interface over data, rather than merely as a convenience abstraction for users who need to understand the underlying representation. Snowflake supports changing the security status of an existing standard or materialized view by using `ALTER VIEW ... SET SECURE` or `UNSET SECURE`, and the corresponding `ALTER MATERIALIZED VIEW` commands. For non-materialized views, the `IS_SECURE` metadata field is available through both Information Schema and Account Usage view metadata, allowing administrators to inventory and govern secure views. Snowflake also intentionally omits secure-view internals from Query Profile, including for the owner, because exposing execution details could reveal protected logic or data characteristics to users who can access profile information. These protections help prevent unintended disclosure through metadata, optimizer behavior, and diagnostic interfaces. See Snowflake's Secure views documentation and the ALTER VIEW SQL reference.

QUESTION NO: 25

Which Snowflake objects does the Snowflake Kafka connector use? (Select THREE).

A. Pipe

B. Serverless task

C. Internal user stage

D. Internal table stage

E. Internal named stage

F. Storage integration

ANSWER: A D E

Explanation:

The Snowflake Kafka connector uses Snowpipe-based ingestion. A **Pipe** is the Snowpipe object that defines how staged files are loaded into the destination Snowflake table. The connector writes buffered Kafka records into files and uses Snowpipe processing to ingest those files continuously and efficiently. An **Internal table stage** can serve as Snowflake-managed staging storage associated with a target table, avoiding a requirement to configure customer-managed external cloud storage for this ingestion workflow. An **Internal named stage** is also a Snowflake-managed stage object that can be used to hold files independently of a particular table. Named internal stages are appropriate when the connector or its loading process requires a reusable, explicitly addressable staging location. Together, the pipe and internal staging objects provide the Snowflake-side mechanism for staging connector-produced files and loading them into the target table. Snowflake documents that the Kafka connector relies on Snowpipe for data loading, and Snowflake stages are the locations from which Snowpipe loads files. See the Snowflake Kafka connector overview and the Snowpipe documentation.