Actual4Dump

Choose the version that fits your needs	PDF Version	Desktop Test Engine	Online Test Engine
Latest and Up-to-Date exam dumps with real exam questions answers.	✓	✓	✓
Get 12-Months free updates without any extra charges.	✓	✓	✓
Experience same exam environment before appearing in the certification exam.	✗	✓	✓
100% exam passing guarante in the first attempt.	✓	✓	✓
20% discount on more than one license and 30% discount on 5+ license purchases.	✗	✓	✓
100% secure purchase on SSL.	✓	✓	✓
Completely private purchase without sharing your personal info with anyone.	✓	✓	✓

http://www.actual4dump.com

Superb Exam Dumps Materials lead you to get your certification easily - Actual4dump

Exam : **ARA-R01**

Title : SnowPro Advanced: Architect Recertification Exam

Vendor : Snowflake

Version : DEMO

NO.1 What built-in Snowflake features make use of the change tracking metadata for a table? (Choose two.)

A. The MERGE command

B. The UPSERT command

C. The CHANGES clause

D. A STREAM object

E. The CHANGE_DATA_CAPTURE command

Answer: A D

Explanation:

In Snowflake, the change tracking metadata for a table is utilized by the MERGE command and the STREAM object. The MERGE command uses change tracking to determine how to apply updates and inserts efficiently based on differences between source and target tables. STREAM objects, on the other hand, specifically capture and store change data, enabling incremental processing based on changes made to a table since the last stream offset was committed.References: Snowflake Documentation on MERGE and STREAM Objects.

NO.2 What built-in Snowflake features make use of the change tracking metadata for a table? (Choose two.)

A. The CHANGE_DATA_CAPTURE command

B. A STREAM object

C. The CHANGES clause

D. The MERGE command

E. The UPSERT command

Answer: B,D

Explanation:

In Snowflake, the change tracking metadata for a table is utilized by the MERGE command and the STREAM object. The MERGE command uses change tracking to determine how to apply updates and inserts efficiently based on differences between source and target tables. STREAM objects, on the other hand, specifically capture and store change data, enabling incremental processing based on changes made to a table since the last stream offset was committed.References: Snowflake Documentation on MERGE and STREAM Objects.

NO.3 How is the change of local time due to daylight savings time handled in Snowflake tasks? (Choose two.)

A. Task schedules can be designed to follow specified or local time zones to accommodate the time changes.

B. A task schedule will follow only the specified time and will fail to handle lost or duplicated hours.

C. A frequent task execution schedule like minutes may not cause a problem, but will affect the task history.

D. A task scheduled in a UTC-based schedule will have no issues with the time changes.

E. A task will move to a suspended state during the daylight savings time change.

Answer: A,D

NO.4 What are characteristics of Dynamic Data Masking? (Select TWO).

A. A masking policy that Is currently set on a table can be dropped.
B. A single masking policy can be applied to columns in different tables.
C. A masking policy can be applied to the value column of an external table.
D. The role that creates the masking policy will always see unmasked data In query results
E. A masking policy can be applied to a column with the GEOGRAPHY data type.
Answer: A B
Explanation:
Dynamic Data Masking is a feature that allows masking sensitive data in query results based on the role of the user who executes the query. A masking policy is a user-defined function that specifies the masking logic and can be applied to one or more columns in one or more tables. A masking policy that is currently set on a table can be dropped using the ALTER TABLE command. A single masking policy can be applied to columns in different tables using the ALTER TABLE command with the SET MASKING POLICY clause. The other options are either incorrect or not supported by Snowflake. A masking policy cannot be applied to the value column of an external table, as external tables do not support column-level security. The role that creates the masking policy will not always see unmasked data in query results, as the masking policy can be applied to the owner role as well. A masking policy cannot be applied to a column with the GEOGRAPHY data type, as Snowflake only supports masking policies for scalar data types. References: Snowflake Documentation:
Dynamic Data Masking, Snowflake Documentation: ALTER TABLE

NO.5 What considerations need to be taken when using database cloning as a tool for data lifecycle management in a development environment? (Select TWO).
A. Any pipes in the source are not cloned.
B. Any pipes in the source referring to internal stages are not cloned.
C. Any pipes in the source referring to external stages are not cloned.
D. The clone inherits all granted privileges of all child objects in the source object, including the database.
E. The clone inherits all granted privileges of all child objects in the source object, excluding the database.
Answer: A C

NO.6 How does a standard virtual warehouse policy work in Snowflake?
A. It conserves credits by keeping running clusters fully loaded rather than starting additional clusters.
B. It starts only if the system estimates that there is a query load that will keep the cluster busy for at least 6 minutes.
C. It starts only f the system estimates that there is a query load that will keep the cluster busy for at least 2 minutes.
D. It prevents or minimizes queuing by starting additional clusters instead of conserving credits.
Answer: D
Explanation:
A standard virtual warehouse policy is one of the two scaling policies available for multi-cluster warehouses in Snowflake. The other policy is economic. A standard policy aims to prevent or minimize queuing by starting additional clusters as soon as the current cluster is fully loaded,

regardless of the number of queries in the queue. This policy can improve query performance and concurrency, but it may also consume more credits than an economic policy, which tries to conserve credits by keeping the running clusters fully loaded before starting additional clusters. The scaling policy can be set when creating or modifying a warehouse, and it can be changed at any time.
References:
* Snowflake Documentation: Multi-cluster Warehouses
* Snowflake Documentation: Scaling Policy for Multi-cluster Warehouses

NO.7 A company has an inbound share set up with eight tables and five secure views. The company plans to make the share part of its production data pipelines.
Which actions can the company take with the inbound share? (Choose two.)
A. Clone a table from a share.
B. Grant modify permissions on the share.
C. Create a table from the shared database.
D. Create additional views inside the shared database.
E. Create a table stream on the shared table.
Answer: A D
Explanation:
These two actions are possible with an inbound share, according to the Snowflake documentation and the web search results. An inbound share is a share that is created by another Snowflake account (the provider) and imported into your account (the consumer). An inbound share allows you to access the data shared by the provider, but not to modify or delete it. However, you can perform some actions with the inbound share, such as:
* Clone a table from a share. You can create a copy of a table from an inbound share using the CREATE TABLE ... CLONE statement. The clone will contain the same data and metadata as the original table, but it will be independent of the share. You can modify or delete the clone as you wish, but it will not reflect any changes made to the original table by the provider1.
* Create additional views inside the shared database. You can create views on the tables or views from an inbound share using the CREATE VIEW statement. The views will be stored in the shared database, but they will be owned by your account. You can query the views as you would query any other view in your account, but you cannot modify or delete the underlying objects from the share2.
The other actions listed are not possible with an inbound share, because they would require modifying the share or the shared objects, which are read-only for the consumer. You cannot grant modify permissions on the share, create a table from the shared database, or create a table stream on the shared table34.
References:
* Cloning Objects from a Share | Snowflake Documentation
* Creating Views on Shared Data | Snowflake Documentation
* Importing Data from a Share | Snowflake Documentation
* Streams on Shared Tables | Snowflake Documentation

NO.8 The following table exists in the production database:
A regulatory requirement states that the company must mask the username for events that are older than six months based on the current date when the data is queried.
How can the requirement be met without duplicating the event data and making sure it is applied when creating views using the table or cloning the table?

A. Use a masking policy on the username column using a entitlement table with valid dates.
B. Use a row level policy on the user_events table using a entitlement table with valid dates.
C. Use a masking policy on the username column with event_timestamp as a conditional column.
D. Use a secure view on the user_events table using a case statement on the username column.
Answer: C
Explanation:
A masking policy is a feature of Snowflake that allows masking sensitive data in query results based on the role of the user and the condition of the data. A masking policy can be applied to a column in a table or a view, and it can use another column in the same table or view as a conditional column. A conditional column is a column that determines whether the masking policy is applied or not based on its value1.
In this case, the requirement can be met by using a masking policy on the username column with event_timestamp as a conditional column. The masking policy can use a function that masks the username if the event_timestamp is older than six months based on the current date, and returns the original username otherwise. The masking policy can be applied to the user_events table, and it will also be applied when creating views using the table or cloning the table2.
The other options are not correct because:
* A. Using a masking policy on the username column using an entitlement table with valid dates would require creating another table that stores the valid dates for each username, and joining it with the user_events table in the masking policy function. This would add complexity and overhead to the masking policy, and it would not use the event_timestamp column as the condition for masking.
* B. Using a row level policy on the user_events table using an entitlement table with valid dates would require creating another table that stores the valid dates for each username, and joining it with the user_events table in the row access policy function. This would filter out the rows that have event_timestamp older than six months based on the valid dates, instead of masking the username
* column. This would not meet the requirement of masking the username, and it would also reduce the visibility of the event data.
* D. Using a secure view on the user_events table using a case statement on the username column would require creating a view that uses a case expression to mask the username column based on the event_timestamp column. This would meet the requirement of masking the username, but it would not be applied when cloning the table. A secure view is a view that prevents the underlying data from being exposed by queries on the view. However, a secure view does not prevent the underlying data from being exposed by cloning the table3.
References:
* 1: Masking Policies | Snowflake Documentation
* 2: Using Conditional Columns in Masking Policies | Snowflake Documentation
* 3: Secure Views | Snowflake Documentation

NO.9 When using the copy into <table> command with the CSV file format, how does the match_by_column_name parameter behave?
A. It expects a header to be present in the CSV file, which is matched to a case-sensitive table column name.
B. The parameter will be ignored.
C. The command will return an error.

D. The command will return a warning stating that the file has unmatched columns.

Answer: B

Explanation:

Option B is the best design to meet the requirements because it uses Snowpipe to ingest the data continuously and efficiently as new records arrive in the object storage, leveraging event notifications. Snowpipe is a service that automates the loading of data from external sources into Snowflake tables1. It also uses streams and tasks to orchestrate transformations on the ingested data. Streams are objects that store the change history of a table, and tasks are objects that execute SQL statements on a schedule or when triggered by another task2.

Option B also uses an external function to do model inference with Amazon Comprehend and write the final records to a Snowflake table. An external function is a user-defined function that calls an external API, such as Amazon Comprehend, to perform computations that are not natively supported by Snowflake3. Finally, option B uses the Snowflake Marketplace to make the de-identified final data set available publicly for advertising companies who use different cloud providers in different regions. The Snowflake Marketplace is a platform that enables data providers to list and share their data sets with data consumers, regardless of the cloud platform or region they use4.

Option A is not the best design because it uses copy into to ingest the data, which is not as efficient and continuous as Snowpipe. Copy into is a SQL command that loads data from files into a table in a single transaction. It also exports the data into Amazon S3 to do model inference with Amazon Comprehend, which adds an extra step and increases the operational complexity and maintenance of the infrastructure.

Option C is not the best design because it uses Amazon EMR and PySpark to ingest and transform the data, which also increases the operational complexity and maintenance of the infrastructure. Amazon EMR is a cloud service that provides a managed Hadoop framework to process and analyze large-scale data sets.

PySpark is a Python API for Spark, a distributed computing framework that can run on Hadoop.

Option C also develops a python program to do model inference by leveraging the Amazon Comprehend text analysis API, which increases the development effort.

Option D is not the best design because it is identical to option A, except for the ingestion method. It still exports the data into Amazon S3 to do model inference with Amazon Comprehend, which adds an extra step and increases the operational complexity and maintenance of the infrastructure.

References: 1: Snowpipe Overview 2: Using Streams and Tasks to Automate Data Pipelines 3: External Functions Overview 4: Snowflake Data Marketplace Overview : [Loading Data Using COPY INTO] : [What is Amazon EMR?] : [PySpark Overview]

* The copy into <table> command is used to load data from staged files into an existing table in Snowflake. The command supports various file formats, such as CSV, JSON, AVRO, ORC, PARQUET, and XML1.

* The match_by_column_name parameter is a copy option that enables loading semi-structured data into separate columns in the target table that match corresponding columns represented in the source data. The parameter can have one of the following values2:

* CASE_SENSITIVE: The column names in the source data must match the column names in the target table exactly, including the case. This is the default value.

* CASE_INSENSITIVE: The column names in the source data must match the column names in the target table, but the case is ignored.

* NONE: The column names in the source data are ignored, and the data is loaded based on the order of the columns in the target table.

* The match_by_column_name parameter only applies to semi-structured data, such as JSON, AVRO, ORC, PARQUET, and XML. It does not apply to CSV data, which is considered structured data2.
* When using the copy into <table> command with the CSV file format, the match_by_column_name parameter behaves as follows2:
* It expects a header to be present in the CSV file, which is matched to a case-sensitive table column name. This means that the first row of the CSV file must contain the column names, and they must match the column names in the target table exactly, including the case. If the header is missing or does not match, the command will return an error.
* The parameter will not be ignored, even if it is set to NONE. The command will still try to match the column names in the CSV file with the column names in the target table, and will return an error if they do not match.
* The command will not return a warning stating that the file has unmatched columns. It will either load the data successfully if the column names match, or return an error if they do not match.
References:
* 1: COPY INTO <table> | Snowflake Documentation
* 2: MATCH_BY_COLUMN_NAME | Snowflake Documentation

NO.10 A company has built a data pipeline using Snowpipe to ingest files from an Amazon S3 bucket. Snowpipe is configured to load data into staging database tables. Then a task runs to load the data from the staging database tables into the reporting database tables.
The company is satisfied with the availability of the data in the reporting database tables, but the reporting tables are not pruning effectively. Currently, a size 4X-Large virtual warehouse is being used to query all of the tables in the reporting database.
What step can be taken to improve the pruning of the reporting tables?

A. Eliminate the use of Snowpipe and load the files into internal stages using PUT commands.

B. Increase the size of the virtual warehouse to a size 5X-Large.

C. Use an ORDER BY <cluster_key (s) > command to load the reporting tables.

D. Create larger files for Snowpipe to ingest and ensure the staging frequency does not exceed 1 minute.

Answer: C
Explanation:
Effective pruning in Snowflake relies on the organization of data within micro-partitions. By using an ORDER BY clause with clustering keys when loading data into the reporting tables, Snowflake can better organize the data within micro-partitions. This organization allows Snowflake to skip over irrelevant micro-partitions during a query, thus improving query performance and reducing the amount of data scanned12.
References =
*Snowflake Documentation on micro-partitions and data clustering2
*Community article on recognizing unsatisfactory pruning and improving it1

NO.11 A table for IOT devices that measures water usage is created. The table quickly becomes large and contains more than 2 billion rows.

```
create table water_iot (
UniqueId number,
DeviceId varchar(20),
DeviceManufacturer varchar(50)
CustomerId varchar(20),
IOT_timestamp timestamp_ntz,
City varchar(80),
Location varchar(50)
)
```

The general query patterns for the table are:
1. DeviceId, IOT_timestamp and CustomerId are frequently used in the filter predicate for the select statement
2. The columns City and DeviceManuf acturer are often retrieved
3. There is often a count on UniqueId
Which field(s) should be used for the clustering key?

A. IOT_timestamp

B. City and DeviceManuf acturer

C. DeviceId and CustomerId

D. UniqueId

Answer: C

Explanation:
A clustering key is a subset of columns or expressions that are used to co-locate the data in the same micro-partitions, which are the units of storage in Snowflake. Clustering can improve the performance of queries that filter on the clustering key columns, as it reduces the amount of data that needs to be scanned. The best choice for a clustering key depends on the query patterns and the data distribution in the table. In this case, the columns DeviceId, IOT_timestamp, and CustomerId are frequently used in the filter predicate for the select statement, which means they are good candidates for the clustering key. The columns City and DeviceManufacturer are often retrieved, but not filtered on, so they are not as important for the clustering key.
The column UniqueId is used for counting, but it is not a good choice for the clustering key, as it is likely to have a high cardinality and a uniform distribution, which means it will not help to co-locate the data.
Therefore, the best option is to use DeviceId and CustomerId as the clustering key, as they can help to prune the micro-partitions and speed up the queries. References: Clustering Keys & Clustered Tables, Micro-partitions & Data Clustering, A Complete Guide to Snowflake Clustering

NO.12 The Business Intelligence team reports that when some team members run queries for their dashboards in parallel with others, the query response time is getting significantly slower What can a Snowflake Architect do to identify what is occurring and troubleshoot this issue?

A.

Use larger warehouses to speed up the queries running in parallel. Identify the queries running in parallel using this query:

```
SELECT QUERY_ID
,USER_NAME
,WAREHOUSE_NAME
,WAREHOUSE_SIZE
,BYTES_SCANNED
,BYTES_SPILLED_TO_REMOTE_STORAGE
,BYTES_SPILLED_TO_REMOTE_STORAGE / BYTES_SCANNED AS SPILLING_READ_RATIO
FROM "SNOWFLAKE"."ACCOUNT_USAGE"."QUERY_HISTORY"
WHERE BYTES_SPILLED_TO_REMOTE_STORAGE > BYTES_SCANNED * 5
ORDER BY SPILLING_READ_RATIO DESC ;
```

B.

Increase the size of the warehouse cache to speed up concurrent queries. Identify the concurrent queries using this query:

```
SELECT WAREHOUSE_NAME ,COUNT(*) AS QUERY_COUNT ,SUM(BYTES_SCANNED) AS BYTES_SCANNED
,SUM(BYTES_SCANNED*PERCENTAGE_SCANNED_FROM_CACHE) AS BYTES_SCANNED_FROM_CACHE ,SUM(BYTES_SCANNED*PERCENTAGE_SCANNED_FROM_CACHE)
/ SUM(BYTES_SCANNED) AS PERCENT_SCANNED_FROM_CACHE
FROM "SNOWFLAKE"."ACCOUNT_USAGE"."QUERY_HISTORY"
WHERE START_TIME >= DATEADD(month,-1,current_timestamp()) AND BYTES_SCANNED > 0 GROUP BY 1 ORDER BY 5 ;
```

C.

Introduce multi-cluster warehouses to help with concurrent queries. Identify the concurrent queries by running this query:

```
SELECT TO_DATE(START_TIME) AS DATE
,WAREHOUSE_NAME ,SUM(AVG_RUNNING) AS SUM_RUNNING ,SUM(AVG_QUEUED_LOAD) AS SUM_QUEUED
FROM "SNOWFLAKE"."ACCOUNT_USAGE"."WAREHOUSE_LOAD_HISTORY" WHERE TO_DATE(START_TIME) >=
DATEADD(month,-1,CURRENT_TIMESTAMP()) GROUP BY 1,2 HAVING SUM(AVG_QUEUED_LOAD) >0 ;
```

D.

Identify which queries are spilled to remote storage and change the warehouse parameters to address this issue. Identify the issue by running this query:

```
SELECT QUERY_ID
,SUBSTR(QUERY_TEXT, 1, 50) PARTIAL_QUERY_TEXT
,USER_NAME
,WAREHOUSE_NAME
,WAREHOUSE_SIZE
,BYTES_SPILLED_TO_REMOTE_STORAGE
,START_TIME, END_TIME
,TOTAL_ELAPSED_TIME/1000 TOTAL_ELAPSED_TIME
FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY WHERE BYTES_SPILLED_TO_REMOTE_STORAGE > 0 AND START_TIME::DATE > DATEADD('DAYS', -45,
CURRENT_DATE) ORDER BY BYTES_SPILLED_TO_REMOTE_STORAGE DESC LIMIT 10 ;
```

Answer: A

Explanation:

The image shows a SQL query that can be used to identify which queries are spilled to remote storage and suggests changing the warehouse parameters to address this issue. Spilling to remote storage occurs when the memory allocated to a warehouse is insufficient to process a query, and Snowflake uses disk or cloud storage as a temporary cache. This can significantly slow down the query performance and increase the cost. To troubleshoot this issue, a Snowflake Architect can run the query shown in the image to find out which queries are spilling, how much data they are spilling, and which warehouses they are using. Then, the architect can adjust the warehouse size, type, or scaling policy to provide enough memory for the queries and avoid spilling12. References:

* Recognizing Disk Spilling
* Managing the Kafka Connector

NO.13 An Architect needs to automate the daily Import of two files from an external stage into Snowflake. One file has Parquet-formatted data, the other has CSV-formatted data.
How should the data be joined and aggregated to produce a final result set?

A. Use Snowpipe to ingest the two files, then create a materialized view to produce the final result set.
B. Create a task using Snowflake scripting that will import the files, and then call a User-Defined Function (UDF) to produce the final result set.
C. Create a JavaScript stored procedure to read. join, and aggregate the data directly from the external stage, and then store the results in a table.
D. Create a materialized view to read, Join, and aggregate the data directly from the external stage, and use the view to produce the final result set
Answer: B
Explanation:
According to the Snowflake documentation, tasks are objects that enable scheduling and execution of SQL statements or JavaScript user-defined functions (UDFs) in Snowflake. Tasks can be used to automate data loading, transformation, and maintenance operations. Snowflake scripting is a feature that allows writing procedural logic using SQL statements and JavaScript UDFs. Snowflake scripting can be used to create complex workflows and orchestrate tasks. Therefore, the best option to automate the daily import of two files from an external stage into Snowflake, join and aggregate the data, and produce a final result set is to create a task using Snowflake scripting that will import the files using the COPY INTO command, and then call a UDF to perform the join and aggregation logic. The UDF can return a table or a variant value as the final result set. References:
* Tasks
* Snowflake Scripting
* User-Defined Functions

NO.14 A company is trying to Ingest 10 TB of CSV data into a Snowflake table using Snowpipe as part of Its migration from a legacy database platform. The records need to be ingested in the MOST performant and cost-effective way.
How can these requirements be met?
A. Use ON_ERROR = continue in the copy into command.
B. Use purge = TRUE in the copy into command.
C. Use FURGE = FALSE in the copy into command.
D. Use on error = SKIP_FILE in the copy into command.
Answer: D
Explanation:
For ingesting a large volume of CSV data into Snowflake using Snowpipe, especially for a substantial amount like 10 TB, the on error = SKIP_FILE option in the COPY INTO command can be highly effective. This approach allows Snowpipe to skip over files that cause errors during the ingestion process, thereby not halting or significantly slowing down the overall data load. It helps in maintaining performance and cost-effectiveness by avoiding the reprocessing of problematic files and continuing with the ingestion of other data.

NO.15 An Architect is troubleshooting a query with poor performance using the QUERY_HISTORY function. The Architect observes that the COMPILATIONJHME is greater than the EXECUTIONJTIME.
What is the reason for this?
A. The query is processing a very large dataset.
B. The query has overly complex logic.

C. The query is queued for execution.

D. The query is reading from remote storage.

Answer: B

Explanation:

Compilation time is the time it takes for the optimizer to create an optimal query plan for the efficient execution of the query. It also involves some pruning of partition files, making the query execution efficient2 If the compilation time is greater than the execution time, it means that the optimizer spent more time analyzing the query than actually running it. This could indicate that the query has overly complex logic, such as multiple joins, subqueries, aggregations, or expressions. The complexity of the query could also affect the size and quality of the query plan, which could impact the performance of the query3 To reduce the compilation time, the Architect can try to simplify the query logic, use views or common table expressions (CTEs) to break down the query into smaller parts, or use hints to guide the optimizer. The Architect can also use the EXPLAIN command to examine the query plan and identify potential bottlenecks or inefficiencies4 References:

* 1: SnowPro Advanced: Architect | Study Guide 5
* 2: Snowflake Documentation | Query Profile Overview 6
* 3: Understanding Why Compilation Time in Snowflake Can Be Higher than Execution Time 7
* 4: Snowflake Documentation | Optimizing Query Performance 8
* : SnowPro Advanced: Architect | Study Guide
* : Query Profile Overview
* : Understanding Why Compilation Time in Snowflake Can Be Higher than Execution Time
* : Optimizing Query Performance

NO.16 A healthcare company is deploying a Snowflake account that may include Personal Health Information (PHI).

The company must ensure compliance with all relevant privacy standards.

Which best practice recommendations will meet data protection and compliance requirements? (Choose three.)

A. Use, at minimum, the Business Critical edition of Snowflake.

B. Create Dynamic Data Masking policies and apply them to columns that contain PHI.

C. Use the Internal Tokenization feature to obfuscate sensitive data.

D. Use the External Tokenization feature to obfuscate sensitive data.

E. Rewrite SQL queries to eliminate projections of PHI data based on current_role().

F. Avoid sharing data with partner organizations.

Answer: A B D

Explanation:

* A healthcare company that handles PHI data must ensure compliance with relevant privacy standards, such as HIPAA, HITRUST, and GDPR. Snowflake provides several features and best practices to help customers meet their data protection and compliance requirements1.
* One best practice recommendation is to use, at minimum, the Business Critical edition of Snowflake. This edition provides the highest level of data protection and security, including end-to-end encryption with customer-managed keys, enhanced object-level security, and HIPAA and HITRUST compliance2. Therefore, option A is correct.
* Another best practice recommendation is to create Dynamic Data Masking policies and apply them to columns that contain PHI. Dynamic Data Masking is a feature that allows masking or redacting

sensitive data based on the current user's role. This way, only authorized users can view the unmasked data, while others will see masked values, such as NULL, asterisks, or random characters3. Therefore, option B is correct.
* A third best practice recommendation is to use the External Tokenization feature to obfuscate sensitive data. External Tokenization is a feature that allows replacing sensitive data with tokens that are generated and stored by an external service, such as Protegrity. This way, the original data is never stored or processed by Snowflake, and only authorized users can access the tokenized data through the external service4. Therefore, option D is correct.
* Option C is incorrect, because the Internal Tokenization feature is not available in Snowflake. Snowflake does not provide any native tokenization functionality, but only supports integration with external tokenization services4.
* Option E is incorrect, because rewriting SQL queries to eliminate projections of PHI data based on current_role() is not a best practice. This approach is error-prone, inefficient, and hard to maintain. A better alternative is to use Dynamic Data Masking policies, which can automatically mask data based on the user's role without modifying the queries3.
* Option F is incorrect, because avoiding sharing data with partner organizations is not a best practice.
Snowflake enables secure and governed data sharing with internal and external consumers, such as business units, customers, or partners. Data sharing does not involve copying or moving data, but only granting access privileges to the shared objects. Data sharing can also leverage Dynamic Data Masking and External Tokenization features to protect sensitive data5.
References: : Snowflake's Security & Compliance Reports : Snowflake Editions : Dynamic Data Masking : External Tokenization : Secure Data Sharing