

Databricks

Exam Questions Databricks-Certified-Professional-Data-Engineer

Databricks Certified Data Engineer Professional Exam



NEW QUESTION 1

A Databricks job has been configured with 3 tasks, each of which is a Databricks notebook. Task A does not depend on other tasks. Tasks B and C run in parallel, with each having a serial dependency on Task A.

If task A fails during a scheduled run, which statement describes the results of this run?

- A. Because all tasks are managed as a dependency graph, no changes will be committed to the Lakehouse until all tasks have successfully been completed.
- B. Tasks B and C will attempt to run as configured; any changes made in task A will be rolled back due to task failure.
- C. Unless all tasks complete successfully, no changes will be committed to the Lakehouse; because task A failed, all commits will be rolled back automatically.
- D. Tasks B and C will be skipped; some logic expressed in task A may have been committed before task failure.
- E. Tasks B and C will be skipped; task A will not commit any changes because of stage failure.

Answer: D

Explanation:

When a Databricks job runs multiple tasks with dependencies, the tasks are executed in a dependency graph. If a task fails, the downstream tasks that depend on it are skipped and marked as Upstream failed. However, the failed task may have already committed some changes to the Lakehouse before the failure occurred, and those changes are not rolled back automatically. Therefore, the job run may result in a partial update of the Lakehouse. To avoid this, you can use the transactional writes feature of Delta Lake to ensure that the changes are only committed when the entire job run succeeds.

Alternatively, you can use the Run if condition to configure tasks to run even when some or all of their dependencies have failed, allowing your job to recover from failures and

continue running. References:

? transactional writes: <https://docs.databricks.com/delta/delta-intro.html#transactional-writes>

? Run if: <https://docs.databricks.com/en/workflows/jobs/conditional-tasks.html>

NEW QUESTION 2

In order to facilitate near real-time workloads, a data engineer is creating a helper function to leverage the schema detection and evolution functionality of Databricks Auto Loader. The desired function will automatically detect the schema of the source directly, incrementally process JSON files as they arrive in a source directory, and automatically evolve the schema of the table when new fields are detected.

The function is displayed below with a blank:

Which response correctly fills in the blank to meet the specified requirements?

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

Answer: B

Explanation:

Option B correctly fills in the blank to meet the specified requirements. Option B uses the "cloudFiles.schemaLocation" option, which is required for the schema detection and evolution functionality of Databricks Auto Loader. Additionally, option B uses the "mergeSchema" option, which is required for the schema evolution functionality of Databricks Auto Loader. Finally, option B uses the "writeStream" method, which is required for the incremental processing of JSON files as they arrive in a source directory. The other options are incorrect because they either omit the required options, use the wrong method, or use the wrong format.

References:

? Configure schema inference and evolution in Auto Loader:

<https://docs.databricks.com/en/ingestion/auto-loader/schema.html>

? Write streaming data: <https://docs.databricks.com/spark/latest/structured-streaming/writing-streaming-data.html>

NEW QUESTION 3

A data ingestion task requires a one-TB JSON dataset to be written out to Parquet with a target part-file size of 512 MB. Because Parquet is being used instead of Delta Lake, built-in file-sizing features such as Auto-Optimize & Auto-Compaction cannot be used.

Which strategy will yield the best performance without shuffling data?

- A. Set spark.sql.files.maxPartitionBytes to 512 MB, ingest the data, execute the narrow transformations, and then write to parquet.
- B. Set spark.sql.shuffle.partitions to 2,048 partitions (1TB*1024*1024/512), ingest the data, execute the narrow transformations, optimize the data by sorting it (which automatically repartitions the data), and then write to parquet.
- C. Set spark.sql.adaptive.advisoryPartitionSizeInBytes to 512 MB bytes, ingest the data, execute the narrow transformations, coalesce to 2,048 partitions (1TB*1024*1024/512), and then write to parquet.
- D. Ingest the data, execute the narrow transformations, repartition to 2,048 partitions (1TB*1024*1024/512), and then write to parquet.
- E. Set spark.sql.shuffle.partitions to 512, ingest the data, execute the narrow transformations, and then write to parquet.

Answer: B

Explanation:

The key to efficiently converting a large JSON dataset to Parquet files of a specific size without shuffling data lies in controlling the size of the output files directly.

? Setting spark.sql.files.maxPartitionBytes to 512 MB configures Spark to process

data in chunks of 512 MB. This setting directly influences the size of the part-files in the output, aligning with the target file size.

? Narrow transformations (which do not involve shuffling data across partitions) can then be applied to this data.

? Writing the data out to Parquet will result in files that are approximately the size specified by spark.sql.files.maxPartitionBytes, in this case, 512 MB.

? The other options involve unnecessary shuffles or repartitions (B, C, D) or an incorrect setting for this specific requirement (E).

References:

? Apache Spark Documentation: Configuration - spark.sql.files.maxPartitionBytes

? Databricks Documentation on Data Sources: Databricks Data Sources Guide

NEW QUESTION 4

A junior data engineer seeks to leverage Delta Lake's Change Data Feed functionality to create a Type 1 table representing all of the values that have ever been

valid for all rows in a bronze table created with the property `delta.enableChangeDataFeed = true`. They plan to execute the following code as a daily job:
Which statement describes the execution and results of running the above query multiple times?

- A. Each time the job is executed, newly updated records will be merged into the target table, overwriting previous values with the same primary keys.
- B. Each time the job is executed, the entire available history of inserted or updated records will be appended to the target table, resulting in many duplicate entries.
- C. Each time the job is executed, the target table will be overwritten using the entire history of inserted or updated records, giving the desired result.
- D. Each time the job is executed, the differences between the original and current versions are calculated; this may result in duplicate entries for some records.
- E. Each time the job is executed, only those records that have been inserted or updated since the last execution will be appended to the target table giving the desired result.

Answer: B

Explanation:

Reading table's changes, captured by CDF, using `spark.read` means that you are reading them as a static source. So, each time you run the query, all table's changes (starting from the specified `startingVersion`) will be read.

NEW QUESTION 5

A Delta Lake table in the Lakehouse named `customer_parsams` is used in churn prediction by the machine learning team. The table contains information about customers derived from a number of upstream sources. Currently, the data engineering team populates this table nightly by overwriting the table with the current valid values derived from upstream data sources.

Immediately after each update succeeds, the data engineer team would like to determine the difference between the new version and the previous of the table. Given the current implementation, which method can be used?

- A. Parse the Delta Lake transaction log to identify all newly written data files.
- B. Execute `DESCRIBE HISTORY customer_churn_params` to obtain the full operation metrics for the update, including a log of all records that have been added or modified.
- C. Execute a query to calculate the difference between the new version and the previous version using Delta Lake's built-in versioning and time travel functionality.
- D. Parse the Spark event logs to identify those rows that were updated, inserted, or deleted.

Answer: C

Explanation:

Delta Lake provides built-in versioning and time travel capabilities, allowing users to query previous snapshots of a table. This feature is particularly useful for understanding changes between different versions of the table. In this scenario, where the table is overwritten nightly, you can use Delta Lake's time travel feature to execute a query comparing the latest version of the table (the current state) with its previous version. This approach effectively identifies the differences (such as new, updated, or deleted records) between the two versions. The other options do not provide a straightforward or efficient way to directly compare different versions of a Delta Lake table.

References:

? Delta Lake Documentation on Time Travel: Delta Time Travel

? Delta Lake Versioning: Delta Lake Versioning Guide

NEW QUESTION 6

The data architect has mandated that all tables in the Lakehouse should be configured as external Delta Lake tables. Which approach will ensure that this requirement is met?

- A. Whenever a database is being created, make sure that the location keyword is used
- B. When configuring an external data warehouse for all table storag
- C. leverage Databricks for all ELT.
- D. Whenever a table is being created, make sure that the location keyword is used.
- E. When tables are created, make sure that the external keyword is used in the create table statement.
- F. When the workspace is being configured, make sure that external cloud object storage has been mounted.

Answer: C

Explanation:

This is the correct answer because it ensures that this requirement is met. The requirement is that all tables in the Lakehouse should be configured as external Delta Lake tables. An external table is a table that is stored outside of the default warehouse directory and whose metadata is not managed by Databricks. An external table can be created by using the location keyword to specify the path to an existing directory in a cloud storage system, such as DBFS or S3. By creating external tables, the data engineering team can avoid losing data if they drop or overwrite the table, as well as leverage existing data without moving or copying it. Verified References: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Create an external table" section.

NEW QUESTION 7

The data engineer team is configuring environment for development testing, and production before beginning migration on a new data pipeline. The team requires extensive testing on both the code and data resulting from code execution, and the team want to develop and test against similar production data as possible.

A junior data engineer suggests that production data can be mounted to the development testing environments, allowing pre production code to execute against production data. Because all users have

Admin privileges in the development environment, the junior data engineer has offered to configure permissions and mount this data for the team.

Which statement captures best practices for this situation?

- A. Because access to production data will always be verified using passthrough credentials it is safe to mount data to any Databricks development environment.
- B. All developer, testing and production code and data should exist in a single unified workspace; creating separate environments for testing and development further reduces risks.
- C. In environments where interactive code will be executed, production data should only be accessible with read permissions; creating isolated databases for each environment further reduces risks.
- D. Because delta Lake versions all data and supports time travel, it is not possible for user error or malicious actors to permanently delete production data, as such it is generally safe to mount production data anywhere.

Answer: C

Explanation:

The best practice in such scenarios is to ensure that production data is handled securely and with proper access controls. By granting only read access to production data in development and testing environments, it mitigates the risk of unintended data modification. Additionally, maintaining isolated databases for different environments helps to avoid accidental impacts on production data and systems. References:

? Databricks best practices for securing data:

<https://docs.databricks.com/security/index.html>

NEW QUESTION 8

A data engineer is configuring a pipeline that will potentially see late-arriving, duplicate records.

In addition to de-duplicating records within the batch, which of the following approaches allows the data engineer to deduplicate data against previously processed records as it is inserted into a Delta table?

- A. Set the configuration `delta.deduplicate = true`.
- B. VACUUM the Delta table after each batch completes.
- C. Perform an insert-only merge with a matching condition on a unique key.
- D. Perform a full outer join on a unique key and overwrite existing data.
- E. Rely on Delta Lake schema enforcement to prevent duplicate records.

Answer: C

Explanation:

To deduplicate data against previously processed records as it is inserted into a Delta table, you can use the merge operation with an insert-only clause. This allows you to insert new records that do not match any existing records based on a unique key, while ignoring duplicate records that match existing records. For example, you can use the following syntax:

```
MERGE INTO target_table USING source_table ON target_table.unique_key = source_table.unique_key WHEN NOT MATCHED THEN INSERT *
```

This will insert only the records from the source table that have a unique key that is not present in the target table, and skip the records that have a matching key.

This way, you can avoid inserting duplicate records into the Delta table.

References:

? <https://docs.databricks.com/delta/delta-update.html#upsert-into-a-table-using-merge>

? <https://docs.databricks.com/delta/delta-update.html#insert-only-merge>

NEW QUESTION 9

A Databricks job has been configured with 3 tasks, each of which is a Databricks notebook. Task A does not depend on other tasks. Tasks B and C run in parallel, with each having a serial dependency on task A.

If tasks A and B complete successfully but task C fails during a scheduled run, which statement describes the resulting state?

- A. All logic expressed in the notebook associated with tasks A and B will have been successfully completed; some operations in task C may have completed successfully.
- B. All logic expressed in the notebook associated with tasks A and B will have been successfully completed; any changes made in task C will be rolled back due to task failure.
- C. All logic expressed in the notebook associated with task A will have been successfully completed; tasks B and C will not commit any changes because of stage failure.
- D. Because all tasks are managed as a dependency graph, no changes will be committed to the Lakehouse until all tasks have successfully been completed.
- E. Unless all tasks complete successfully, no changes will be committed to the Lakehouse; because task C failed, all commits will be rolled back automatically.

Answer: A

Explanation:

The query uses the `CREATE TABLE USING DELTA` syntax to create a Delta Lake table from an existing Parquet file stored in DBFS. The query also uses the `LOCATION` keyword to specify the path to the Parquet file as `/mnt/finance_edata_bucket/tx_sales.parquet`. By using the `LOCATION` keyword, the query creates an external table, which is a table that is stored outside of the default warehouse directory and whose metadata is not managed by Databricks. An external table can be created from an existing directory in a cloud storage system, such as DBFS or S3, that contains data files in a supported format, such as Parquet or CSV. The resulting state after running the second command is that an external table will be created in the storage container mounted to `/mnt/finance_edata_bucket` with the new name `prod.sales_by_store`. The command will not change any data or move any files in the storage container; it will only update the table reference in the metastore and create a new Delta transaction log for the renamed table. Verified References: [Databricks Certified Data Engineer Professional], under “Delta Lake” section; Databricks Documentation, under “ALTER TABLE RENAME TO” section; Databricks Documentation, under “Create an external table” section.

NEW QUESTION 10

Which statement describes the default execution mode for Databricks Auto Loader?

- A. New files are identified by listing the input directory; new files are incrementally and idempotently loaded into the target Delta Lake table.
- B. Cloud vendor-specific queue storage and notification services are configured to track newly arriving files; new files are incrementally and idempotently loaded into the target Delta Lake table.
- C. Webhook trigger Databricks job to run anytime new data arrives in a source directory; new data automatically merged into target tables using rules inferred from the data.
- D. New files are identified by listing the input directory; the target table is materialized by directory querying all valid files in the source directory.

Answer: A

Explanation:

Databricks Auto Loader simplifies and automates the process of loading data into Delta Lake. The default execution mode of the Auto Loader identifies new files by listing the input directory. It incrementally and idempotently loads these new files into the target Delta Lake table. This approach ensures that files are not missed and are processed exactly once, avoiding data duplication. The other options describe different mechanisms or integrations that are not part of the default behavior of the Auto Loader.

References:

? Databricks Auto Loader Documentation: Auto Loader Guide

? Delta Lake and Auto Loader: Delta Lake Integration

NEW QUESTION 10

Which statement describes the correct use of `pyspark.sql.functions.broadcast`?

- A. It marks a column as having low enough cardinality to properly map distinct values to available partitions, allowing a broadcast join.
- B. It marks a column as small enough to store in memory on all executors, allowing a broadcast join.
- C. It caches a copy of the indicated table on attached storage volumes for all active clusters within a Databricks workspace.
- D. It marks a DataFrame as small enough to store in memory on all executors, allowing a broadcast join.
- E. It caches a copy of the indicated table on all nodes in the cluster for use in all future queries during the cluster lifetime.

Answer: D

Explanation:

<https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.functions.broadcast.html>

The broadcast function in PySpark is used in the context of joins. When you mark a DataFrame with broadcast, Spark tries to send this DataFrame to all worker nodes so that it can be joined with another DataFrame without shuffling the larger DataFrame across the nodes. This is particularly beneficial when the DataFrame is small enough to fit into the memory of each node. It helps to optimize the join process by reducing the amount of data that needs to be shuffled across the cluster, which can be a very expensive operation in terms of computation and time.

The `pyspark.sql.functions.broadcast` function in PySpark is used to hint to Spark that a DataFrame is small enough to be broadcast to all worker nodes in the cluster. When this hint is applied, Spark can perform a broadcast join, where the smaller DataFrame is sent to each executor only once and joined with the larger DataFrame on each executor. This can significantly reduce the amount of data shuffled across the network and can improve the performance of the join operation. In a broadcast join, the entire smaller DataFrame is sent to each executor, not just a specific column or a cached version on attached storage. This function is particularly useful when one of the DataFrames in a join operation is much smaller than the other, and can fit comfortably in the memory of each executor node.

References:

? Databricks Documentation on Broadcast Joins: Databricks Broadcast Join Guide

? PySpark API Reference: `pyspark.sql.functions.broadcast`

NEW QUESTION 14

A data engineer is testing a collection of mathematical functions, one of which calculates the area under a curve as described by another function.

Which kind of the test does the above line exemplify?

- A. Integration
- B. Unit
- C. Manual
- D. functional

Answer: B

Explanation:

A unit test is designed to verify the correctness of a small, isolated piece of

code, typically a single function. Testing a mathematical function that calculates the area under a curve is an example of a unit test because it is testing a specific, individual function to ensure it operates as expected.

References:

? Software Testing Fundamentals: Unit Testing

NEW QUESTION 17

An hourly batch job is configured to ingest data files from a cloud object storage container where each batch represent all records produced by the source system in a given hour. The batch job to process these records into the Lakehouse is sufficiently delayed to ensure no late-arriving data is missed. The `user_id` field represents a unique key for the data, which has the following schema:

`user_id BIGINT, username STRING, user_utc STRING, user_region STRING, last_login BIGINT, auto_pay BOOLEAN, last_updated BIGINT`

New records are all ingested into a table named `account_history` which maintains a full record of all data in the same schema as the source. The next table in the system is named `account_current` and is implemented as a Type 1 table representing the most recent value for each unique `user_id`.

Assuming there are millions of user accounts and tens of thousands of records processed hourly, which implementation can be used to efficiently update the described `account_current` table as part of each hourly batch job?

- A. Use Auto Loader to subscribe to new files in the account history directory; configure a Structured Streaming trigger once job to batch update newly detected files into the account current table.
- B. Overwrite the account current table with each batch using the results of a query against the account history table grouping by user id and filtering for the max value of last updated.
- C. Filter records in account history using the last updated field and the most recent hour processed, as well as the max last login by user id write a merge statement to update or insert the most recent value for each user id.
- D. Use Delta Lake version history to get the difference between the latest version of account history and one version prior, then write these records to account current.
- E. Filter records in account history using the last updated field and the most recent hour processed, making sure to deduplicate on username; write a merge statement to update or insert the most recent value for each username.

Answer: C

Explanation:

This is the correct answer because it efficiently updates the account current table with only the most recent value for each user id. The code filters records in account history using the last updated field and the most recent hour processed, which means it will only process the latest batch of data. It also filters by the max last login by user id, which means it will only keep the most recent record for each user id within that batch. Then, it writes a merge statement to update or insert the most recent value for each user id into account current, which means it will perform an upsert operation based on the user id column. Verified References:

[Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Upsert into a table using merge" section.

NEW QUESTION 19

The data engineering team is migrating an enterprise system with thousands of tables and views into the Lakehouse. They plan to implement the target architecture using a series of bronze, silver, and gold tables. Bronze tables will almost exclusively be used by production data engineering workloads, while silver tables will be used to support both data engineering and machine learning workloads. Gold tables will largely serve business intelligence and reporting purposes. While personal identifying information (PII) exists in all tiers of data, pseudonymization and anonymization rules are in place for all data at the silver and gold levels.

The organization is interested in reducing security concerns while maximizing the ability to collaborate across diverse teams.

Which statement exemplifies best practices for implementing this system?

- A. Isolating tables in separate databases based on data quality tiers allows for easy permissions management through database ACLs and allows physical separation of default storage locations for managed tables.
- B. Because databases on Databricks are merely a logical construct, choices around database organization do not impact security or discoverability in the Lakehouse.
- C. Storing all production tables in a single database provides a unified view of all data assets available throughout the Lakehouse, simplifying discoverability by granting all users view privileges on this database.
- D. Working in the default Databricks database provides the greatest security when working with managed tables, as these will be created in the DBFS root.
- E. Because all tables must live in the same storage containers used for the database they're created in, organizations should be prepared to create between dozens and thousands of databases depending on their data isolation requirements.

Answer: A

Explanation:

This is the correct answer because it exemplifies best practices for implementing this system. By isolating tables in separate databases based on data quality tiers, such as bronze, silver, and gold, the data engineering team can achieve several benefits. First, they can easily manage permissions for different users and groups through database ACLs, which allow granting or revoking access to databases, tables, or views. Second, they can physically separate the default storage locations for managed tables in each database, which can improve performance and reduce costs. Third, they can provide a clear and consistent naming convention for the tables in each database, which can improve discoverability and usability. Verified References: [Databricks Certified Data Engineer Professional], under "Lakehouse" section; Databricks Documentation, under "Database object privileges" section.

NEW QUESTION 21

A Delta Lake table was created with the below query:

Consider the following query:

```
DROP TABLE prod.sales_by_store -
```

If this statement is executed by a workspace admin, which result will occur?

- A. Nothing will occur until a COMMIT command is executed.
- B. The table will be removed from the catalog but the data will remain in storage.
- C. The table will be removed from the catalog and the data will be deleted.
- D. An error will occur because Delta Lake prevents the deletion of production data.
- E. Data will be marked as deleted but still recoverable with Time Travel.

Answer: C

Explanation:

When a table is dropped in Delta Lake, the table is removed from the catalog and the data is deleted. This is because Delta Lake is a transactional storage layer that provides ACID guarantees. When a table is dropped, the transaction log is updated to reflect the deletion of the table and the data is deleted from the underlying storage. References:

? <https://docs.databricks.com/delta/quick-start.html#drop-a-table>

? <https://docs.databricks.com/delta/delta-batch.html#drop-table>

NEW QUESTION 25

A junior data engineer is working to implement logic for a Lakehouse table named silver_device_recordings. The source data contains 100 unique fields in a highly nested JSON structure.

The silver_device_recordings table will be used downstream for highly selective joins on a number of fields, and will also be leveraged by the machine learning team to filter on a handful of relevant fields, in total, 15 fields have been identified that will often be used for filter and join logic.

The data engineer is trying to determine the best approach for dealing with these nested fields before declaring the table schema.

Which of the following accurately presents information about Delta Lake and Databricks that may impact their decision-making process?

- A. Because Delta Lake uses Parquet for data storage, Dremel encoding information for nesting can be directly referenced by the Delta transaction log.
- B. Tungsten encoding used by Databricks is optimized for storing string data: newly-added native support for querying JSON strings means that string types are always most efficient.
- C. Schema inference and evolution on Databricks ensure that inferred types will always accurately match the data types used by downstream systems.
- D. By default Delta Lake collects statistics on the first 32 columns in a table; these statistics are leveraged for data skipping when executing selective queries.

Answer: D

Explanation:

Delta Lake, built on top of Parquet, enhances query performance through data skipping, which is based on the statistics collected for each file in a table. For tables with a large number of columns, Delta Lake by default collects and stores statistics only for the first 32 columns. These statistics include min/max values and null counts, which are used to optimize query execution by skipping irrelevant data files. When dealing with highly nested JSON structures, understanding this behavior is crucial for schema design, especially when determining which fields should be flattened or prioritized in the table structure to leverage data skipping efficiently for performance optimization. References: Databricks documentation on Delta Lake optimization techniques, including data skipping and statistics collection (<https://docs.databricks.com/delta/optimizations/index.html>).

NEW QUESTION 27

A junior member of the data engineering team is exploring the language interoperability of Databricks notebooks. The intended outcome of the below code is to register a view of all sales that occurred in countries on the continent of Africa that appear in the geo_lookup table.

Before executing the code, running SHOW TABLES on the current database indicates the database contains only two tables: geo_lookup and sales.

```
Cmd 1
%python
countries_af = [x[0] for x in
spark.table("geo_lookup").filter("continent='AF'").select("country").collect()]

Cmd 2
%sql
CREATE VIEW sales_af AS
SELECT *
FROM sales
WHERE city IN countries_af
AND CONTINENT = "AF"
```

Which statement correctly describes the outcome of executing these command cells in order in an interactive notebook?

- A. Both commands will succeed
- B. Executing show tables will show that countries at and sales at have been registered as views.
- C. Cmd 1 will succeed
- D. Cmd 2 will search all accessible databases for a table or view named countries af: if this entity exists, Cmd 2 will succeed.
- E. Cmd 1 will succeed and Cmd 2 will fail, countries at will be a Python variable representing a PySpark DataFrame.
- F. Both commands will fail
- G. No new variables, tables, or views will be created.
- H. Cmd 1 will succeed and Cmd 2 will fail, countries at will be a Python variable containing a list of strings.

Answer: E

Explanation:

This is the correct answer because Cmd 1 is written in Python and uses a list comprehension to extract the country names from the geo_lookup table and store them in a Python variable named countries af. This variable will contain a list of strings, not a PySpark DataFrame or a SQL view. Cmd 2 is written in SQL and tries to create a view named sales af by selecting from the sales table where city is in countries af. However, this command will fail because countries af is not a valid SQL entity and cannot be used in a SQL query. To fix this, a better approach would be to use spark.sql() to execute a SQL query in Python and pass the countries af variable as a parameter. Verified References: [Databricks Certified Data Engineer Professional], under “Language Interoperability” section; Databricks Documentation, under “Mix languages” section.

NEW QUESTION 31

To reduce storage and compute costs, the data engineering team has been tasked with curating a series of aggregate tables leveraged by business intelligence dashboards, customer-facing applications, production machine learning models, and ad hoc analytical queries.

The data engineering team has been made aware of new requirements from a customer-facing application, which is the only downstream workload they manage entirely. As a result, an aggregate table used by numerous teams across the organization will need to have a number of fields renamed, and additional fields will also be added.

Which of the solutions addresses the situation while minimally interrupting other teams in the organization without increasing the number of tables that need to be managed?

- A. Send all users notice that the schema for the table will be changing; include in the communication the logic necessary to revert the new table schema to match historic queries.
- B. Configure a new table with all the requisite fields and new names and use this as the source for the customer-facing application; create a view that maintains the original data schema and table name by aliasing select fields from the new table.
- C. Create a new table with the required schema and new fields and use Delta Lake's deep clone functionality to sync up changes committed to one table to the corresponding table.
- D. Replace the current table definition with a logical view defined with the query logic currently writing the aggregate table; create a new table to power the customer-facing application.
- E. Add a table comment warning all users that the table schema and field names will be changing on a given date; overwrite the table in place to the specifications of the customer-facing application.

Answer: B

Explanation:

This is the correct answer because it addresses the situation while minimally interrupting other teams in the organization without increasing the number of tables that need to be managed. The situation is that an aggregate table used by numerous teams across the organization will need to have a number of fields renamed, and additional fields will also be added, due to new requirements from a customer-facing application. By configuring a new table with all the requisite fields and new names and using this as the source for the customer-facing application, the data engineering team can meet the new requirements without affecting other teams that rely on the existing table schema and name. By creating a view that maintains the original data schema and table name by aliasing select fields from the new table, the data engineering team can also avoid duplicating data or creating additional tables that need to be managed. Verified References: [Databricks Certified Data Engineer Professional], under “Lakehouse” section; Databricks Documentation, under “CREATE VIEW” section.

NEW QUESTION 34

A developer has successfully configured credential for Databricks Repos and cloned a remote Git repository. They do not have privileges to make changes to the main branch, which is the only branch currently visible in their workspace.

Use Response to pull changes from the remote Git repository commit and push changes to a branch that appeared as a change were pulled.

- A. Use Repos to merge all differences and make a pull request back to the remote repository.
- B. Use repos to merge all difference and make a pull request back to the remote repository.
- C. Use Repos to create a new branch commit all changes and push changes to the remote Git repository.
- D. Use repos to create a fork of the remote repository commit all changes and make a pull request on the source repository

Answer: C

Explanation:

In Databricks Repos, when a user does not have privileges to make changes directly to the main branch of a cloned remote Git repository, the recommended approach is to create a new branch within the Databricks workspace. The developer can then make changes in this new branch, commit those changes, and push the new branch to the remote Git repository. This workflow allows for isolated development without affecting the main branch, enabling the developer to propose changes via a pull request from the new branch to the main branch in the remote repository. This method adheres to common Git collaboration workflows,

fostering code review and collaboration while ensuring the integrity of the main branch.

References:

? Databricks documentation on using Repos with Git: <https://docs.databricks.com/repos.html>

NEW QUESTION 39

What is a method of installing a Python package scoped at the notebook level to all nodes in the currently active cluster?

- A. Use &Pip install in a notebook cell
- B. Run source env/bin/activate in a notebook setup script
- C. Install libraries from PyPi using the cluster UI
- D. Use &sh install in a notebook cell

Answer: C

Explanation:

Installing a Python package scoped at the notebook level to all nodes in the currently active cluster in Databricks can be achieved by using the Libraries tab in the cluster UI. This interface allows you to install libraries across all nodes in the cluster. While the %pip command in a notebook cell would only affect the driver node, using the cluster UI ensures that the package is installed on all nodes.

References:

? Databricks Documentation on Libraries: Libraries

NEW QUESTION 40

The data science team has created and logged a production model using MLflow. The following code correctly imports and applies the production model to output the predictions as a new DataFrame named preds with the schema "customer_id LONG, predictions DOUBLE, date DATE".

```
from pyspark.sql.functions import current_date

model = mlflow.pyfunc.spark_udf(spark, model_uri="models:/churn/prod")
df = spark.table("customers")
columns = ["account_age", "time_since_last_seen", "app_rating"]
preds = (df.select(
    "customer_id",
    model(*columns).alias("predictions"),
    current_date().alias("date")
))
```

The data science team would like predictions saved to a Delta Lake table with the ability to compare all predictions across time. Churn predictions will be made at most once per day.

Which code block accomplishes this task while minimizing potential compute costs?

- A) preds.write.mode("append").saveAsTable("churn_preds")
- B) preds.write.format("delta").save("/preds/churn_preds")
- C)

```
(preds.writeStream
    .outputMode("overwrite")
    .option("checkpointPath", "/_checkpoints/churn_preds")
    .start("/preds/churn_preds")
)
```

D)

```
(preds.write
    .format("delta")
    .mode("overwrite")
    .saveAsTable("churn_preds")
)
```

E)

```
(preds.writeStream
    .outputMode("append")
    .option("checkpointPath", "/_checkpoints/churn_preds")
    .table("churn_preds")
)
```

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

Answer: A

NEW QUESTION 45

The data governance team has instituted a requirement that all tables containing Personal Identifiable Information (PII) must be clearly annotated. This includes adding column comments, table comments, and setting the custom table property "contains_pii" = true.

The following SQL DDL statement is executed to create a new table:

Which command allows manual confirmation that these three requirements have been met?

- A. DESCRIBE EXTENDED dev.pii test
- B. DESCRIBE DETAIL dev.pii test
- C. SHOW TBLPROPERTIES dev.pii test
- D. DESCRIBE HISTORY dev.pii test
- E. SHOW TABLES dev

Answer: A

Explanation:

This is the correct answer because it allows manual confirmation that these three requirements have been met. The requirements are that all tables containing Personal Identifiable Information (PII) must be clearly annotated, which includes adding column comments, table comments, and setting the custom table property “contains_pii” = true. The DESCRIBE EXTENDED command is used to display detailed information about a table, such as its schema, location, properties, and comments. By using this command on the dev.pii_test table, one can verify that the table has been created with the correct column comments, table comment, and custom table property as specified in the SQL DDL statement. Verified References: [Databricks Certified Data Engineer Professional], under “Lakehouse” section; Databricks Documentation, under “DESCRIBE EXTENDED” section.

NEW QUESTION 48

The data science team has created and logged a production using MLflow. The model accepts a list of column names and returns a new column of type DOUBLE. The following code correctly imports the production model, load the customer table containing the customer_id key column into a Dataframe, and defines the feature columns needed for the model.

```
model = mlflow.pyfunc.spark_udf (spark,
model_uri="models:/churn/prod")

df = spark.table("customers")

columns = ["account_age", "time_since_last_seen", "app_rating"]
```

Which code block will output DataFrame with the schema" customer_id LONG, predictions DOUBLE"?

- A. Model, predict (df, columns)
- B. Df, map (lambda k:midel (x [columns]) ,select ("customer_id predictions")
- C. D
- D. Select ("customer_id". Model ("columns) alias ("predictions")
- E. Df.apply(model, columns). Select ("customer_id, prediction"

Answer: A

Explanation:

Given the information that the model is registered with MLflow and assuming predict is the method used to apply the model to a set of columns, we use the model.predict() function to apply the model to the DataFrame df using the specified columns. The model.predict() function is designed to take in a DataFrame and a list of column names as arguments, applying the trained model to these features to produce a predictions column. When working with PySpark, this predictions column needs to be selected alongside the customer_id to create a new DataFrame with the schema customer_id LONG, predictions DOUBLE.

References:

? MLflow documentation on using Python function models: <https://www.mlflow.org/docs/latest/models.html#python-function-python>

? PySpark MLlib documentation on model prediction: <https://spark.apache.org/docs/latest/ml-pipeline.html#pipeline>

NEW QUESTION 53

A distributed team of data analysts share computing resources on an interactive cluster with autoscaling configured. In order to better manage costs and query throughput, the workspace administrator is hoping to evaluate whether cluster upscaling is caused by many concurrent users or resource-intensive queries.

In which location can one review the timeline for cluster resizing events?

- A. Workspace audit logs
- B. Driver's log file
- C. Ganglia
- D. Cluster Event Log
- E. Executor's log file

Answer: C

NEW QUESTION 54

A CHECK constraint has been successfully added to the Delta table named activity_details using the following logic:

A batch job is attempting to insert new records to the table, including a record where latitude = 45.50 and longitude = 212.67.

Which statement describes the outcome of this batch insert?

- A. The write will fail when the violating record is reached; any records previously processed will be recorded to the target table.
- B. The write will fail completely because of the constraint violation and no records will be inserted into the target table.
- C. The write will insert all records except those that violate the table constraints; the violating records will be recorded to a quarantine table.
- D. The write will include all records in the target table; any violations will be indicated in the boolean column named valid_coordinates.
- E. The write will insert all records except those that violate the table constraints; the violating records will be reported in a warning log.

Answer: B

Explanation:

The CHECK constraint is used to ensure that the data inserted into the table meets the specified conditions. In this case, the CHECK constraint is used to ensure that the latitude and longitude values are within the specified range. If the data does not meet the specified conditions, the write operation will fail completely and no records will be inserted into the target table. This is because Delta Lake supports ACID transactions, which means that either all the data is written or none of it

is written. Therefore, the batch insert will fail when it encounters a record that violates the constraint, and the target table will not be updated. References:
? Constraints: <https://docs.delta.io/latest/delta-constraints.html>
? ACID Transactions: <https://docs.delta.io/latest/delta-intro.html#acid-transactions>

NEW QUESTION 57

The downstream consumers of a Delta Lake table have been complaining about data quality issues impacting performance in their applications. Specifically, they have complained that invalid latitude and longitude values in the activity_details table have been breaking their ability to use other geolocation processes. A junior engineer has written the following code to add CHECK constraints to the Delta Lake table:

```
ALTER TABLE activity_details
ADD CONSTRAINT valid_coordinates
CHECK (
    latitude >= -90 AND
    latitude <= 90 AND
    longitude >= -180 AND
    longitude <= 180);
```

A senior engineer has confirmed the above logic is correct and the valid ranges for latitude and longitude are provided, but the code fails when executed. Which statement explains the cause of this failure?

- A. Because another team uses this table to support a frequently running application, two- phase locking is preventing the operation from committing.
- B. The activity details table already exists; CHECK constraints can only be added during initial table creation.
- C. The activity details table already contains records that violate the constraints; all existing data must pass CHECK constraints in order to add them to an existing table.
- D. The activity details table already contains records; CHECK constraints can only be added prior to inserting values into a table.
- E. The current table schema does not contain the field valid coordinates; schema evolution will need to be enabled before altering the table to add a constraint.

Answer: C

Explanation:

The failure is that the code to add CHECK constraints to the Delta Lake table fails when executed. The code uses ALTER TABLE ADD CONSTRAINT commands to add two CHECK constraints to a table named activity_details. The first constraint checks if the latitude value is between -90 and 90, and the second constraint checks if the longitude value is between -180 and 180. The cause of this failure is that the activity_details table already contains records that violate these constraints, meaning that they have invalid latitude or longitude values outside of these ranges. When adding CHECK constraints to an existing table, Delta Lake verifies that all existing data satisfies the constraints before adding them to the table. If any record violates the constraints, Delta Lake throws an exception and aborts the operation. Verified References: [Databricks Certified Data Engineer Professional], under “Delta Lake” section; Databricks Documentation, under “Add a CHECK constraint to an existing table” section. <https://docs.databricks.com/en/sql/language-manual/sql-ref-syntax-ddl-alter-table.html#add-constraint>

NEW QUESTION 61

A production workload incrementally applies updates from an external Change Data Capture feed to a Delta Lake table as an always-on Structured Stream job. When data was initially migrated for this table, OPTIMIZE was executed and most data files were resized to 1 GB. Auto Optimize and Auto Compaction were both turned on for the streaming production job. Recent review of data files shows that most data files are under 64 MB, although each partition in the table contains at least 1 GB of data and the total table size is over 10 TB. Which of the following likely explains these smaller file sizes?

- A. Databricks has autotuned to a smaller target file size to reduce duration of MERGE operations
- B. Z-order indices calculated on the table are preventing file compaction
- C. Bloom filter indices calculated on the table are preventing file compaction
- D. Databricks has autotuned to a smaller target file size based on the overall size of data in the table
- E. Databricks has autotuned to a smaller target file size based on the amount of data in each partition

Answer: A

Explanation:

This is the correct answer because Databricks has a feature called Auto Optimize, which automatically optimizes the layout of Delta Lake tables by coalescing small files into larger ones and sorting data within each file by a specified column. However, Auto Optimize also considers the trade-off between file size and merge performance, and may choose a smaller target file size to reduce the duration of merge operations, especially for streaming workloads that frequently update existing records. Therefore, it is possible that Auto Optimize has autotuned to a smaller target file size based on the characteristics of the streaming production job. Verified References: [Databricks Certified Data Engineer Professional], under “Delta Lake” section; Databricks Documentation, under “Auto Optimize” section. <https://docs.databricks.com/en/delta/tune-file-size.html#autotune-table> 'Autotune file size based on workload'

NEW QUESTION 64

The data engineering team maintains a table of aggregate statistics through batch nightly updates. This includes total sales for the previous day alongside totals and averages for a variety of time periods including the 7 previous days, year-to-date, and quarter-to-date. This table is named store_sales_summary and the schema is as follows:

The table daily_store_sales contains all the information needed to update store_sales_summary. The schema for this table is: store_id INT, sales_date DATE, total_sales FLOAT If daily_store_sales is implemented as a Type 1 table and the total_sales column might be adjusted after manual data auditing, which approach is the safest to generate accurate reports in the store_sales_summary table?

- A. Implement the appropriate aggregate logic as a batch read against the daily_store_sales table and overwrite the store_sales_summary table with each Update.
- B. Implement the appropriate aggregate logic as a batch read against the daily_store_sales table and append new rows nightly to the store_sales_summary table.
- C. Implement the appropriate aggregate logic as a batch read against the daily_store_sales table and use upsert logic to update results in the

store_sales_summary table.

D. Implement the appropriate aggregate logic as a Structured Streaming read against the daily_store_sales table and use upsert logic to update results in the store_sales_summary table.

E. Use Structured Streaming to subscribe to the change data feed for daily_store_sales and apply changes to the aggregates in the store_sales_summary table with each update.

Answer: E

Explanation:

The daily_store_sales table contains all the information needed to update store_sales_summary. The schema of the table is:

store_id INT, sales_date DATE, total_sales FLOAT

The daily_store_sales table is implemented as a Type 1 table, which means that old values are overwritten by new values and no history is maintained. The total_sales column might be adjusted after manual data auditing, which means that the data in the table may change over time.

The safest approach to generate accurate reports in the store_sales_summary table is to use Structured Streaming to subscribe to the change data feed for daily_store_sales and apply changes to the aggregates in the store_sales_summary table with each update. Structured Streaming is a scalable and fault-tolerant stream processing engine built on Spark SQL. Structured Streaming allows processing data streams as if they were tables or DataFrames, using familiar operations such as select, filter, groupBy, or join. Structured Streaming also supports output modes that specify how to write the results of a streaming query to a sink, such as append, update, or complete. Structured Streaming can handle both streaming and batch data sources in a unified manner.

The change data feed is a feature of Delta Lake that provides structured streaming sources that can subscribe to changes made to a Delta Lake table. The change data feed captures both data changes and schema changes as ordered events that can be processed by downstream applications or services. The change data feed can be configured with different options, such as starting from a specific version or timestamp, filtering by operation type or partition values, or excluding no-op changes.

By using Structured Streaming to subscribe to the change data feed for daily_store_sales, one can capture and process any changes made to the total_sales column due to manual data auditing. By applying these changes to the aggregates in the store_sales_summary table with each update, one can ensure that the reports are always consistent and accurate with the latest data. Verified References: [Databricks Certified Data Engineer Professional], under “Spark Core” section; Databricks Documentation, under “Structured Streaming” section; Databricks Documentation, under “Delta Change Data Feed” section.

NEW QUESTION 69

A Data engineer wants to run unit's tests using common Python testing frameworks on python functions defined across several Databricks notebooks currently used in production.

How can the data engineer run unit tests against function that work with data in production?

- A. Run unit tests against non-production data that closely mirrors production
- B. Define and unit test functions using Files in Repos
- C. Define units test and functions within the same notebook
- D. Define and import unit test functions from a separate Databricks notebook

Answer: A

Explanation:

The best practice for running unit tests on functions that interact with data is to use a dataset that closely mirrors the production data. This approach allows data engineers to validate the logic of their functions without the risk of affecting the actual production data. It's important to have a representative sample of production data to catch edge cases and ensure the functions will work correctly when used in a production environment.

References:

? Databricks Documentation on Testing: Testing and Validation of Data and Notebooks

NEW QUESTION 71

The following table consists of items found in user carts within an e-commerce website.

```
Carts (id LONG, items ARRAY<STRUCT<id: LONG, count: INT>>)  
id  items                                     email  
1001[[{"id: "DESK65", count: 1}]]           "u1@domain.com"  
1002[[{"id: "HYBD45", count: 1}, {"id: "M27", count: 2}]] "u2@domain.com"  
1003[[{"id: "M27", count: 1}]]              "u3@domain.com"
```

The following MERGE statement is used to update this table using an updates view, with schema evaluation enabled on this table.

```
MERGE INTO carts c  
USING updates u  
ON c.id = u.id  
WHEN MATCHED  
THEN UPDATE SET *
```

How would the following update be handled?

```
(new nested field, missing existing column)  
id  items  
1001[[{"id: "DESK65", count: 2, coupon: "BOGO50"}]]
```

How would the following update be handled?

- A. The update is moved to separate "restored" column because it is missing a column expected in the target schema.
- B. The new restored field is added to the target schema, and dynamically read as NULL for existing unmatched records.
- C. The update throws an error because changes to existing columns in the target schema are not supported.
- D. The new nested field is added to the target schema, and files underlying existing records are updated to include NULL values for the new field.

Answer: D

Explanation:

With schema evolution enabled in Databricks Delta tables, when a new field is added to a record through a MERGE operation, Databricks automatically modifies the table schema to include the new field. In existing records where this new field is not present, Databricks will insert NULL values for that field. This ensures that the schema remains consistent across all records in the table, with the new field being present in every record, even if it is NULL for records that did not originally include it.

References:

? Databricks documentation on schema evolution in Delta Lake: <https://docs.databricks.com/delta/delta-batch.html#schema-evolution>

NEW QUESTION 72

A data engineer wants to join a stream of advertisement impressions (when an ad was shown) with another stream of user clicks on advertisements to correlate when impression led to monetizable clicks.

```
In the code below, Impressions is a streaming DataFrame with a watermark ("event_time", "10 minutes")
.groupBy(
  window("event_time", "5 minutes"),
  "id")
.count()
).      withWatermark("event_time", 2 hours)
impressions.join(clicks, expr("clickAdId = impressionAdId"), "inner")
```

Which solution would improve the performance?

- A)
Joining on event time constraint: clickTime == impressionTime using a leftOuter join
- B)
Joining on event time constraint: clickTime >= impressionTime - interval 3 hours and removing watermarks
- C)
Joining on event time constraint: clickTime + 3 hours < impressionTime - 2 hours
- D)
Joining on event time constraint: clickTime >= impressionTime AND clickTime <= impressionTime + interval 1 hour

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

Answer: A

Explanation:

When joining a stream of advertisement impressions with a stream of user clicks, you want to minimize the state that you need to maintain for the join. Option A suggests using a left outer join with the condition that clickTime == impressionTime, which is suitable for correlating events that occur at the exact same time. However, in a real-world scenario, you would likely need some leeway to account for the delay between an impression and a possible click. It's important to design the join condition and the window of time considered to optimize performance while still capturing the relevant user interactions. In this case, having the watermark can help with state management and avoid state growing unbounded by discarding old state data that's unlikely to match with new data.

NEW QUESTION 73

A junior developer complains that the code in their notebook isn't producing the correct results in the development environment. A shared screenshot reveals that while they're using a notebook versioned with Databricks Repos, they're using a personal branch that contains old logic. The desired branch named dev-2.3.9 is not available from the branch selection dropdown.

Which approach will allow this developer to review the current logic for this notebook?

- A. Use Repos to make a pull request use the Databricks REST API to update the current branch to dev-2.3.9
- B. Use Repos to pull changes from the remote Git repository and select the dev-2.3.9 branch.
- C. Use Repos to checkout the dev-2.3.9 branch and auto-resolve conflicts with the current branch
- D. Merge all changes back to the main branch in the remote Git repository and clone the repo again
- E. Use Repos to merge the current branch and the dev-2.3.9 branch, then make a pull request to sync with the remote repository

Answer: B

Explanation:

This is the correct answer because it will allow the developer to update their local repository with the latest changes from the remote repository and switch to the desired branch. Pulling changes will not affect the current branch or create any conflicts, as it will only fetch the changes and not merge them. Selecting the dev-2.3.9 branch from the dropdown will checkout that branch and display its contents in the notebook. Verified References: [Databricks Certified Data Engineer Professional], under "Databricks Tooling" section; Databricks Documentation, under "Pull changes from a remote repository" section.

NEW QUESTION 74

Spill occurs as a result of executing various wide transformations. However, diagnosing spill requires one to proactively look for key indicators. Where in the Spark UI are two of the primary indicators that a partition is spilling to disk?

- A. Stage's detail screen and Executor's files
- B. Stage's detail screen and Query's detail screen
- C. Driver's and Executor's log files
- D. Executor's detail screen and Executor's log files

Answer: B

Explanation:

In Apache Spark's UI, indicators of data spilling to disk during the execution of wide transformations can be found in the Stage's detail screen and the Query's detail screen. These screens provide detailed metrics about each stage of a Spark job, including information about memory usage and spill data. If a task is spilling data to disk, it indicates that the data being processed exceeds the available memory, causing Spark to spill data to disk to free up memory. This is an important performance metric as excessive spill can significantly slow down the processing.

References:

? Apache Spark Monitoring and Instrumentation: Spark Monitoring Guide

? Spark UI Explained: Spark UI Documentation

NEW QUESTION 75

The data engineer is using Spark's MEMORY_ONLY storage level.

Which indicators should the data engineer look for in the spark UI's Storage tab to signal that a cached table is not performing optimally?

- A. Size on Disk is > 0
- B. The number of Cached Partitions > the number of Spark Partitions
- C. The RDD Block Name included the " annotation signaling failure to cache
- D. On Heap Memory Usage is within 75% of off Heap Memory usage

Answer: C

Explanation:

In the Spark UI's Storage tab, an indicator that a cached table is not performing optimally would be the presence of the _disk annotation in the RDD Block Name. This annotation indicates that some partitions of the cached data have been spilled to disk because there wasn't enough memory to hold them. This is suboptimal because accessing data from disk is much slower than from memory. The goal of caching is to keep data in memory for fast access, and a spill to disk means that this goal is not fully achieved.

NEW QUESTION 77

The data engineer team has been tasked with configured connections to an external database that does not have a supported native connector with Databricks. The external database already has data security configured by group membership. These groups map directly to user group already created in Databricks that represent various teams within the company.

A new login credential has been created for each group in the external database. The Databricks Utilities Secrets module will be used to make these credentials available to Databricks users.

Assuming that all the credentials are configured correctly on the external database and group membership is properly configured on Databricks, which statement describes how teams can be granted the minimum necessary access to using these credentials?

- A. "Read" permissions should be set on a secret key mapped to those credentials that will be used by a given team.
- B. No additional configuration is necessary as long as all users are configured as administrators in the workspace where secrets have been added.
- C. "Read" permissions should be set on a secret scope containing only those credentials that will be used by a given team.
- D. "Manage" permission should be set on a secret scope containing only those credentials that will be used by a given team.

Answer: C

Explanation:

In Databricks, using the Secrets module allows for secure management of sensitive information such as database credentials. Granting 'Read' permissions on a secret key that maps to database credentials for a specific team ensures that only members of that team can access these credentials. This approach aligns with the principle of least privilege, granting users the minimum level of access required to perform their jobs, thus enhancing security.

References:

? Databricks Documentation on Secret Management: Secrets

NEW QUESTION 79

A DLT pipeline includes the following streaming tables:

Raw_iot ingest raw device measurement data from a heart rate tracking device. Bgm_stats incrementally computes user statistics based on BPM measurements from raw_iot.

How can the data engineer configure this pipeline to be able to retain manually deleted or updated records in the raw_iot table while recomputing the downstream table when a pipeline update is run?

- A. Set the skipChangeCommits flag to true on bpm_stats
- B. Set the SkipChangeCommits flag to true raw_iot
- C. Set the pipelines, reset, allowed property to false on bpm_stats
- D. Set the pipelines, reset, allowed property to false on raw_iot

Answer: D

Explanation:

In Databricks Lakehouse, to retain manually deleted or updated records in the raw_iot table while recomputing downstream tables when a pipeline update is run, the property pipelines.reset.allowed should be set to false. This property prevents the system from resetting the state of the table, which includes the removal of the history of changes, during a pipeline update. By keeping this property as false, any changes to the raw_iot table, including manual deletes or updates, are retained, and recomputation of downstream tables, such as bpm_stats, can occur with the full history of data changes intact. References:

? Databricks documentation on DLT pipelines: <https://docs.databricks.com/data-engineering/delta-live-tables/delta-live-tables-overview.html>

NEW QUESTION 80

The marketing team is looking to share data in an aggregate table with the sales organization, but the field names used by the teams do not match, and a number of marketing specific fields have not been approved for the sales org.

Which of the following solutions addresses the situation while emphasizing simplicity?

- A. Create a view on the marketing table selecting only these fields approved for the sales team alias the names of any fields that should be standardized to the sales naming conventions.
- B. Use a CTAS statement to create a derivative table from the marketing table configure a production job to propagation changes.
- C. Add a parallel table write to the current production pipeline, updating a new sales table that varies as required from marketing table.
- D. Create a new table with the required schema and use Delta Lake's DEEP CLONE functionality to sync up changes committed to one table to the corresponding table.

Answer: A

Explanation:

Creating a view is a straightforward solution that can address the need for field name standardization and selective field sharing between departments. A view allows for presenting a transformed version of the underlying data without duplicating it. In this scenario, the view would only include the approved fields for the sales team and rename any fields as per their naming conventions.

References:

? Databricks documentation on using SQL views in Delta Lake: <https://docs.databricks.com/delta/quick-start.html#sql-views>

NEW QUESTION 85

The DevOps team has configured a production workload as a collection of notebooks scheduled to run daily using the Jobs UI. A new data engineering hire is onboarding to the team and has requested access to one of these notebooks to review the production logic.

What are the maximum notebook permissions that can be granted to the user without allowing accidental changes to production code or data?

- A. Can manage
- B. Can edit
- C. Can run
- D. Can Read

Answer: D

Explanation:

Granting a user 'Can Read' permissions on a notebook within Databricks allows them to view the notebook's content without the ability to execute or edit it. This level of permission ensures that the new team member can review the production logic for learning or auditing purposes without the risk of altering the notebook's code or affecting production data and workflows. This approach aligns with best practices for maintaining security and integrity in production environments, where strict access controls are essential to prevent unintended modifications. References: Databricks documentation on access control and permissions for notebooks within the workspace (<https://docs.databricks.com/security/access-control/workspace-acl.html>).

NEW QUESTION 87

The data architect has mandated that all tables in the Lakehouse should be configured as external (also known as "unmanaged") Delta Lake tables.

Which approach will ensure that this requirement is met?

- A. When a database is being created, make sure that the LOCATION keyword is used.
- B. When configuring an external data warehouse for all table storage, leverage Databricks for all ELT.
- C. When data is saved to a table, make sure that a full file path is specified alongside the Delta format.
- D. When tables are created, make sure that the EXTERNAL keyword is used in the CREATE TABLE statement.
- E. When the workspace is being configured, make sure that external cloud object storage has been mounted.

Answer: D

Explanation:

To create an external or unmanaged Delta Lake table, you need to use the EXTERNAL keyword in the CREATE TABLE statement. This indicates that the table is not managed by the catalog and the data files are not deleted when the table is dropped. You also need to provide a LOCATION clause to specify the path where the data files are stored. For example:

```
CREATE EXTERNAL TABLE events ( date DATE, eventId STRING, eventType STRING, data STRING) USING DELTA LOCATION '/mnt/delta/events';
```

This creates an external Delta Lake table named events that references the data files in the '/mnt/delta/events' path. If you drop this table, the data files will remain intact and you can recreate the table with the same statement.

References:

? <https://docs.databricks.com/delta/delta-batch.html#create-a-table>

? <https://docs.databricks.com/delta/delta-batch.html#drop-a-table>

NEW QUESTION 90

What is the first of a Databricks Python notebook when viewed in a text editor?

- A. %python
- B. % Databricks notebook source
- C. -- Databricks notebook source
- D. //Databricks notebook source

Answer: B

Explanation:

When viewing a Databricks Python notebook in a text editor, the first line indicates the format and source type of the notebook. The correct option is % Databricks notebook source, which is a magic command that specifies the start of a Databricks notebook source file.

NEW QUESTION 92

An upstream system is emitting change data capture (CDC) logs that are being written to a cloud object storage directory. Each record in the log indicates the change type (insert, update, or delete) and the values for each field after the change. The source table has a primary key identified by the field pk_id.

For auditing purposes, the data governance team wishes to maintain a full record of all values that have ever been valid in the source system. For analytical purposes, only the most recent value for each record needs to be recorded. The Databricks job to ingest these records occurs once per hour, but each individual record may have changed multiple times over the course of an hour.

Which solution meets these requirements?

- A. Create a separate history table for each pk_id resolve the current state of the table by running a union all filtering the history tables for the most recent state.
- B. Use merge into to insert, update, or delete the most recent entry for each pk_id into a bronze table, then propagate all changes throughout the system.
- C. Iterate through an ordered set of changes to the table, applying each in turn; rely on Delta Lake's versioning ability to create an audit log.
- D. Use Delta Lake's change data feed to automatically process CDC data from an external system, propagating all changes to all dependent tables in the Lakehouse.
- E. Ingest all log information into a bronze table; use merge into to insert, update, or delete the most recent entry for each pk_id into a silver table to recreate the current table state.

Answer: B

Explanation:

This is the correct answer because it meets the requirements of maintaining a full record of all values that have ever been valid in the source system and recreating the current table state with only the most recent value for each record. The code ingests all log information into a bronze table, which preserves the raw

CDC data as it is. Then, it uses merge into to perform an upsert operation on a silver table, which means it will insert new records or update or delete existing records based on the change type and the pk_id columns. This way, the silver table will always reflect the current state of the source table, while the bronze table will keep the history of all changes. Verified References: [Databricks Certified Data Engineer Professional], under “Delta Lake” section; Databricks Documentation, under “Upsert into a table using merge” section.

NEW QUESTION 97

The following code has been migrated to a Databricks notebook from a legacy workload:

```
%sh
git clone https://github.com/foo/data_loader;
python ./data_loader/run.py;
mv ./output /dbfs/mnt/new_data
```

The code executes successfully and provides the logically correct results, however, it takes over 20 minutes to extract and load around 1 GB of data. Which statement is a possible explanation for this behavior?

- A. %sh triggers a cluster restart to collect and install Gi
- B. Most of the latency is related to cluster startup time.
- C. Instead of cloning, the code should use %sh pip install so that the Python code can get executed in parallel across all nodes in a cluster.
- D. %sh does not distribute file moving operations; the final line of code should be updated to use %fs instead.
- E. Python will always execute slower than Scala on Databrick
- F. The run.py script should be refactored to Scala.
- G. %sh executes shell code on the driver nod
- H. The code does not take advantage of the worker nodes or Databricks optimized Spark.

Answer: E

Explanation:

<https://www.databricks.com/blog/2020/08/31/introducing-the-databricks-web-terminal.html>

The code is using %sh to execute shell code on the driver node. This means that the code is not taking advantage of the worker nodes or Databricks optimized Spark. This is why the code is taking longer to execute. A better approach would be to use Databricks libraries and APIs to read and write data from Git and DBFS, and to leverage the parallelism and performance of Spark. For example, you can use the Databricks Connect feature to run your Python code on a remote Databricks cluster, or you can use the Spark Git Connector to read data from Git repositories as Spark DataFrames.

NEW QUESTION 99

A data pipeline uses Structured Streaming to ingest data from kafka to Delta Lake. Data is being stored in a bronze table, and includes the Kafka_generated timesamp, key, and value. Three months after the pipeline is deployed the data engineering team has noticed some latency issued during certain times of the day. A senior data engineer updates the Delta Table's schema and ingestion logic to include the current timestamp (as recoded by Apache Spark) as well the Kafka topic and partition. The team plans to use the additional metadata fields to diagnose the transient processing delays: Which limitation will the team face while diagnosing this problem?

- A. New fields not be computed for historic records.
- B. Updating the table schema will invalidate the Delta transaction log metadata.
- C. Updating the table schema requires a default value provided for each file added.
- D. Spark cannot capture the topic partition fields from the kafka source.

Answer: A

Explanation:

When adding new fields to a Delta table's schema, these fields will not be retrospectively applied to historical records that were ingested before the schema change. Consequently, while the team can use the new metadata fields to investigate transient processing delays moving forward, they will be unable to apply this diagnostic approach to past data that lacks these fields.

References:

? Databricks documentation on Delta Lake schema management: <https://docs.databricks.com/delta/delta-batch.html#schema-management>

NEW QUESTION 101

A table is registered with the following code:

Both users and orders are Delta Lake tables. Which statement describes the results of querying recent_orders?

- A. All logic will execute at query time and return the result of joining the valid versions of the source tables at the time the query finishes.
- B. All logic will execute when the table is defined and store the result of joining tables to the DBFS; this stored data will be returned when the table is queried.
- C. Results will be computed and cached when the table is defined; these cached results will incrementally update as new records are inserted into source tables.
- D. All logic will execute at query time and return the result of joining the valid versions of the source tables at the time the query began.
- E. The versions of each source table will be stored in the table transaction log; query results will be saved to DBFS with each query.

Answer: B

NEW QUESTION 105

Which Python variable contains a list of directories to be searched when trying to locate required modules?

- A. importlib.resource path
- B. ,sys.path
- C. os-path
- D. pypi.path
- E. pylib.source

Answer: B

NEW QUESTION 109

Which statement describes integration testing?

- A. Validates interactions between subsystems of your application
- B. Requires an automated testing framework
- C. Requires manual intervention
- D. Validates an application use case
- E. Validates behavior of individual elements of your application

Answer: D

Explanation:

This is the correct answer because it describes integration testing. Integration testing is a type of testing that validates interactions between subsystems of your application, such as modules, components, or services. Integration testing ensures that the subsystems work together as expected and produce the correct outputs or results. Integration testing can be done at different levels of granularity, such as component integration testing, system integration testing, or end-to-end testing. Integration testing can help detect errors or bugs that may not be found by unit testing, which only validates behavior of individual elements of your application. Verified References: [Databricks Certified Data Engineer Professional], under “Testing” section; Databricks Documentation, under “Integration testing” section.

NEW QUESTION 110

A user wants to use DLT expectations to validate that a derived table report contains all records from the source, included in the table validation_copy.

The user attempts and fails to accomplish this by adding an expectation to the report table definition.

Which approach would allow using DLT expectations to validate all expected records are present in this table?

- A. Define a SQL UDF that performs a left outer join on two tables, and check if this returns null values for report key values in a DLT expectation for the report table.
- B. Define a function that performs a left outer join on validation_copy and report and report, and check against the result in a DLT expectation for the report table
- C. Define a temporary table that perform a left outer join on validation_copy and report, and define an expectation that no report key values are null
- D. Define a view that performs a left outer join on validation_copy and report, and reference this view in DLT expectations for the report table

Answer: D

Explanation:

To validate that all records from the source are included in the derived table, creating a view that performs a left outer join between the validation_copy table and the report table is effective. The view can highlight any discrepancies, such as null values in the report table's key columns, indicating missing records. This view can then be referenced in DLT (Delta Live Tables) expectations for the report table to ensure data integrity. This approach allows for a comprehensive comparison between the source and the derived table.

References:

? Databricks Documentation on Delta Live Tables and Expectations: Delta Live Tables Expectations

NEW QUESTION 111

A Delta table of weather records is partitioned by date and has the below schema: date DATE, device_id INT, temp FLOAT, latitude FLOAT, longitude FLOAT

To find all the records from within the Arctic Circle, you execute a query with the below filter:

latitude > 66.3

Which statement describes how the Delta engine identifies which files to load?

- A. All records are cached to an operational database and then the filter is applied
- B. The Parquet file footers are scanned for min and max statistics for the latitude column
- C. All records are cached to attached storage and then the filter is applied
- D. The Delta log is scanned for min and max statistics for the latitude column
- E. The Hive metastore is scanned for min and max statistics for the latitude column

Answer: D

Explanation:

This is the correct answer because Delta Lake uses a transaction log to store metadata about each table, including min and max statistics for each column in each data file. The Delta engine can use this information to quickly identify which files to load based on a filter condition, without scanning the entire table or the file footers. This is called data skipping and it can improve query performance significantly. Verified References: [Databricks Certified Data Engineer Professional], under “Delta Lake” section; [Databricks Documentation], under “Optimizations - Data Skipping” section.

In the Transaction log, Delta Lake captures statistics for each data file of the table. These statistics indicate per file:

- Total number of records
- Minimum value in each column of the first 32 columns of the table
- Maximum value in each column of the first 32 columns of the table
- Null value counts for in each column of the first 32 columns of the table

When a query with a selective filter is executed against the table, the query optimizer uses these statistics to generate the query result. it leverages them to identify data files that may contain records matching the conditional filter.

For the SELECT query in the question, The transaction log is scanned for min and max statistics for the price column

NEW QUESTION 113

Two of the most common data locations on Databricks are the DBFS root storage and external object storage mounted with dbutils.fs.mount().

Which of the following statements is correct?

- A. DBFS is a file system protocol that allows users to interact with files stored in object storage using syntax and guarantees similar to Unix file systems.
- B. By default, both the DBFS root and mounted data sources are only accessible to workspace administrators.
- C. The DBFS root is the most secure location to store data, because mounted storage volumes must have full public read and write permissions.
- D. Neither the DBFS root nor mounted storage can be accessed when using %sh in a Databricks notebook.
- E. The DBFS root stores files in ephemeral block volumes attached to the driver, while mounted directories will always persist saved data to external storage between sessions.

Answer: A

Explanation:

DBFS is a file system protocol that allows users to interact with files stored in object storage using syntax and guarantees similar to Unix file systems¹. DBFS is not a physical file system, but a layer over the object storage that provides a unified view of data across different data sources¹. By default, the DBFS root is accessible to all users in the workspace, and the access to mounted data sources depends on the permissions of the storage account or container². Mounted storage volumes do not need to have full public read and write permissions, but they do require a valid connection string or access key to be provided when mounting³. Both the DBFS root and mounted storage can be accessed when using %sh in a Databricks notebook, as long as the cluster has FUSE enabled⁴. The DBFS root does not store files in ephemeral block volumes attached to the driver, but in the object storage associated with the workspace¹. Mounted directories will persist saved data to external storage between sessions, unless they are unmounted or deleted³. References: DBFS, Work with files on Azure Databricks, Mounting cloud object storage on Azure Databricks, Access DBFS with FUSE

NEW QUESTION 115

Which configuration parameter directly affects the size of a spark-partition upon ingestion of data into Spark?

- A. spark.sql.files.maxPartitionBytes
- B. spark.sql.autoBroadcastJoinThreshold
- C. spark.sql.files.openCostInBytes
- D. spark.sql.adaptive.coalescePartitions.minPartitionNum
- E. spark.sql.adaptive.advisoryPartitionSizeInBytes

Answer: A

Explanation:

This is the correct answer because spark.sql.files.maxPartitionBytes is a configuration parameter that directly affects the size of a spark-partition upon ingestion of data into Spark. This parameter configures the maximum number of bytes to pack into a single partition when reading files from file-based sources such as Parquet, JSON and ORC. The default value is 128 MB, which means each partition will be roughly 128 MB in size, unless there are too many small files or only one large file. Verified References: [Databricks Certified Data Engineer Professional], under “Spark Configuration” section; Databricks Documentation, under “Available Properties - spark.sql.files.maxPartitionBytes” section.

NEW QUESTION 118

Which distribution does Databricks support for installing custom Python code packages?

- A. sbt
- B. CRAN
- C. CRAM
- D. nom
- E. Wheels
- F. jars

Answer: D

NEW QUESTION 122

A junior data engineer on your team has implemented the following code block.

```
MERGE INTO events
USING new_events
ON events.event_id = new_events.event_id
WHEN NOT MATCHED
  INSERT *
```

The view new_events contains a batch of records with the same schema as the events Delta table. The event_id field serves as a unique key for this table. When this query is executed, what will happen with new records that have the same event_id as an existing record?

- A. They are merged.
- B. They are ignored.
- C. They are updated.
- D. They are inserted.
- E. They are deleted.

Answer: B

Explanation:

This is the correct answer because it describes what will happen with new records that have the same event_id as an existing record when the query is executed. The query uses the INSERT INTO command to append new records from the view new_events to the table events. However, the INSERT INTO command does not check for duplicate values in the primary key column (event_id) and does not perform any update or delete operations on existing records. Therefore, if there are new records that have the same event_id as an existing record, they will be ignored and not inserted into the table events. Verified References: [Databricks Certified Data Engineer Professional], under “Delta Lake” section; Databricks Documentation, under “Append data using INSERT INTO” section. "If none of the WHEN MATCHED conditions evaluate to true for a source and target row pair that matches the merge_condition, then the target row is left unchanged." https://docs.databricks.com/en/sql/language-manual/delta-merge-into.html#:~:text=If%20none%20of%20the%20WHEN%20MATCHED%20conditions%20evaluate%20to%20true%20for%20a%20source%20and%20target%20row%20pair%20that%20matches%20the%20merge_condition%2C%20then%20the%20target%20row%20is%20left%20unchanged.

NEW QUESTION 123

Where in the Spark UI can one diagnose a performance problem induced by not leveraging predicate push-down?

- A. In the Executor's log file, by gripping for "predicate push-down"
- B. In the Stage's Detail screen, in the Completed Stages table, by noting the size of data read from the Input column
- C. In the Storage Detail screen, by noting which RDDs are not stored on disk
- D. In the Delta Lake transaction log
- E. by noting the column statistics
- F. In the Query Detail screen, by interpreting the Physical Plan

Answer: E

Explanation:

This is the correct answer because it is where in the Spark UI one can diagnose a performance problem induced by not leveraging predicate push-down. Predicate push-down is an optimization technique that allows filtering data at the source before loading it into memory or processing it further. This can improve performance and reduce I/O costs by avoiding reading unnecessary data. To leverage predicate push-down, one should use supported data sources and formats, such as Delta Lake, Parquet, or JDBC, and use filter expressions that can be pushed down to the source. To diagnose a performance problem induced by not leveraging predicate push-down, one can use the Spark UI to access the Query Detail screen, which shows information about a SQL query executed on a Spark cluster. The Query Detail screen includes the Physical Plan, which is the actual plan executed by Spark to perform the query. The Physical Plan shows the physical operators used by Spark, such as Scan, Filter, Project, or Aggregate, and their input and output statistics, such as rows and bytes. By interpreting the Physical Plan, one can see if the filter expressions are pushed down to the source or not, and how much data is read or processed by each operator. Verified References: [Databricks Certified Data Engineer Professional], under "Spark Core" section; Databricks Documentation, under "Predicate pushdown" section; Databricks Documentation, under "Query detail page" section.

NEW QUESTION 128

A small company based in the United States has recently contracted a consulting firm in India to implement several new data engineering pipelines to power artificial intelligence applications. All the company's data is stored in regional cloud storage in the United States.

The workspace administrator at the company is uncertain about where the Databricks workspace used by the contractors should be deployed.

Assuming that all data governance considerations are accounted for, which statement accurately informs this decision?

- A. Databricks runs HDFS on cloud volume storage; as such, cloud virtual machines must be deployed in the region where the data is stored.
- B. Databricks workspaces do not rely on any regional infrastructure; as such, the decision should be made based upon what is most convenient for the workspace administrator.
- C. Cross-region reads and writes can incur significant costs and latency; whenever possible, compute should be deployed in the same region the data is stored.
- D. Databricks leverages user workstations as the driver during interactive development; as such, users should always use a workspace deployed in a region they are physically near.
- E. Databricks notebooks send all executable code from the user's browser to virtual machines over the open internet; whenever possible, choosing a workspace region near the end users is the most secure.

Answer: C

Explanation:

This is the correct answer because it accurately informs this decision. The decision is about where the Databricks workspace used by the contractors should be deployed. The contractors are based in India, while all the company's data is stored in regional cloud storage in the United States. When choosing a region for deploying a Databricks workspace, one of the important factors to consider is the proximity to the data sources and sinks. Cross-region reads and writes can incur significant costs and latency due to network bandwidth and data transfer fees. Therefore, whenever possible, compute should be deployed in the same region the data is stored to optimize performance and reduce costs. Verified References: [Databricks Certified Data Engineer Professional], under "Databricks Workspace" section; Databricks Documentation, under "Choose a region" section.

NEW QUESTION 130

An external object storage container has been mounted to the location /mnt/finance_eda_bucket.

The following logic was executed to create a database for the finance team:

After the database was successfully created and permissions configured, a member of the finance team runs the following code:

If all users on the finance team are members of the finance group, which statement describes how the tx_sales table will be created?

- A. A logical table will persist the query plan to the Hive Metastore in the Databricks control plane.
- B. An external table will be created in the storage container mounted to /mnt/finance_eda_bucket.
- C. A logical table will persist the physical plan to the Hive Metastore in the Databricks control plane.
- D. An managed table will be created in the storage container mounted to /mnt/finance_eda_bucket.
- E. A managed table will be created in the DBFS root storage container.

Answer: A

Explanation:

<https://docs.databricks.com/en/lakehouse/data-objects.html>

NEW QUESTION 135

The view updates represents an incremental batch of all newly ingested data to be inserted or updated in the customers table.

The following logic is used to process these records.

```
MERGE INTO customers USING (  
  SELECT updates.customer_id as merge_key, updates.* FROM updates  
  UNION ALL  
  SELECT NULL as merge_key, updates.* FROM updates JOIN customers  
  ON updates.customer_id = customers.customer_id  
  WHERE customers.current = true AND updates.address <> customers.address  
) staged_updates  
ON customers.customer_id = mergekey  
WHEN MATCHED AND customers.current = true AND customers.address <> staged_updates.address THEN  
  UPDATE SET current = false, end_date = staged_updates.effective_date  
WHEN NOT MATCHED THEN  
  INSERT (customer_id, address, current, effective_date, end_date)  
  VALUES (staged_updates.customer_id, staged_updates.address, true, staged_updates.effective_date, null)
```

Which statement describes this implementation?

- A. The customers table is implemented as a Type 2 table; old values are overwritten and new customers are appended.
- B. The customers table is implemented as a Type 1 table; old values are overwritten by new values and no history is maintained.
- C. The customers table is implemented as a Type 2 table; old values are maintained but marked as no longer current and new values are inserted.
- D. The customers table is implemented as a Type 0 table; all writes are append only with no changes to existing values.

Answer: C

Explanation:

The provided MERGE statement is a classic implementation of a Type 2 SCD in a data warehousing context. In this approach, historical data is preserved by keeping old records (marking them as not current) and adding new records for changes. Specifically, when a match is found and there's a change in the address, the existing record in the customers table is updated to mark it as no longer current (current = false), and an end date is assigned (end_date = staged_updates.effective_date). A new record for the customer is then inserted with the updated information, marked as current. This method ensures that the full history of changes to customer information is maintained in the table, allowing for time-based analysis of customer data. References: Databricks documentation on implementing SCDs using Delta Lake and the MERGE statement (<https://docs.databricks.com/delta/delta-update.html#upsert-into-a-table-using-merge>).

NEW QUESTION 136

All records from an Apache Kafka producer are being ingested into a single Delta Lake table with the following schema:
key BINARY, value BINARY, topic STRING, partition LONG, offset LONG, timestamp LONG

There are 5 unique topics being ingested. Only the "registration" topic contains Personal Identifiable Information (PII). The company wishes to restrict access to PII. The company also wishes to only retain records containing PII in this table for 14 days after initial ingestion. However, for non-PII information, it would like to retain these records indefinitely.

Which of the following solutions meets the requirements?

- A. All data should be deleted biweekly; Delta Lake's time travel functionality should be leveraged to maintain a history of non-PII information.
- B. Data should be partitioned by the registration field, allowing ACLs and delete statements to be set for the PII directory.
- C. Because the value field is stored as binary data, this information is not considered PII and no special precautions should be taken.
- D. Separate object storage containers should be specified based on the partition field, allowing isolation at the storage level.
- E. Data should be partitioned by the topic field, allowing ACLs and delete statements to leverage partition boundaries.

Answer: B

Explanation:

Partitioning the data by the topic field allows the company to apply different access control policies and retention policies for different topics. For example, the company can use the Table Access Control feature to grant or revoke permissions to the registration topic based on user roles or groups. The company can also use the DELETE command to remove records from the registration topic that are older than 14 days, while keeping the records from other topics indefinitely. Partitioning by the topic field also improves the performance of queries that filter by the topic field, as they can skip reading irrelevant partitions. References:

? Table Access Control: [https://docs.databricks.com/security/access-control/table-](https://docs.databricks.com/security/access-control/table-acls/index.html)

acls/index.html

? DELETE: <https://docs.databricks.com/delta/delta-update.html#delete-from-a-table>

NEW QUESTION 140

.....

Thank You for Trying Our Product

We offer two products:

1st - We have Practice Tests Software with Actual Exam Questions

2nd - Questions and Answers in PDF Format

Databricks-Certified-Professional-Data-Engineer Practice Exam Features:

- * Databricks-Certified-Professional-Data-Engineer Questions and Answers Updated Frequently
- * Databricks-Certified-Professional-Data-Engineer Practice Questions Verified by Expert Senior Certified Staff
- * Databricks-Certified-Professional-Data-Engineer Most Realistic Questions that Guarantee you a Pass on Your FirstTry
- * Databricks-Certified-Professional-Data-Engineer Practice Test Questions in Multiple Choice Formats and Updatesfor 1 Year

100% Actual & Verified — Instant Download, Please Click
[Order The Databricks-Certified-Professional-Data-Engineer Practice Test Here](#)