

Exam Questions Databricks-Certified-Professional-Data-Engineer

Databricks Certified Data Engineer Professional Exam

<https://www.2passeasy.com/dumps/Databricks-Certified-Professional-Data-Engineer/>



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

An upstream source writes Parquet data as hourly batches to directories named with the current date. A nightly batch job runs the following code to ingest all data from the previous day as indicated by the date variable:

```
(spark.read
  .format("parquet")
  .load(f"/mnt/raw_orders/{date}")
  .dropDuplicates(["customer_id", "order_id"])
  .write
  .mode("append")
  .saveAsTable("orders")
)
```

Assume that the fields customer_id and order_id serve as a composite key to uniquely identify each order.

If the upstream system is known to occasionally produce duplicate entries for a single order hours apart, which statement is correct?

- A. Each write to the orders table will only contain unique records, and only those records without duplicates in the target table will be written.
- B. Each write to the orders table will only contain unique records, but newly written records may have duplicates already present in the target table.
- C. Each write to the orders table will only contain unique records; if existing records with the same key are present in the target table, these records will be overwritten.
- D. Each write to the orders table will only contain unique records; if existing records with the same key are present in the target table, the operation will fail.
- E. Each write to the orders table will run deduplication over the union of new and existing records, ensuring no duplicate records are present.

Answer: B

Explanation:

This is the correct answer because the code uses the dropDuplicates method to remove any duplicate records within each batch of data before writing to the orders table. However, this method does not check for duplicates across different batches or in the target table, so it is possible that newly written records may have duplicates already present in the target table. To avoid this, a better approach would be to use Delta Lake and perform an upsert operation using mergeInto. Verified References: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "DROP DUPLICATES" section.

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 user new to Databricks is trying to troubleshoot long execution times for some pipeline logic they are working on. Presently, the user is executing code cell-by-cell, using `display()` calls to confirm code is producing the logically correct results as new transformations are added to an operation. To get a measure of average time to execute, the user is running each cell multiple times interactively.

Which of the following adjustments will get a more accurate measure of how code is likely to perform in production?

A. Scala is the only language that can be accurately tested using interactive notebooks; because the best performance is achieved by using Scala code compiled to JAR

B. all PySpark and Spark SQL logic should be refactored.

C. The only way to meaningfully troubleshoot code execution times in development notebooks is to use production-sized data and production-sized clusters with Run All execution.

D. Production code development should only be done using an IDE; executing code against a local build of open source Spark and Delta Lake will provide the most accurate benchmarks for how code will perform in production.

E. Calling `display()` forces a job to trigger, while many transformations will only add to the logical query plan; because of caching, repeated execution of the same logic does not provide meaningful results.

F. The Jobs UI should be leveraged to occasionally run the notebook as a job and track execution time during incremental code development because Photon can only be enabled on clusters launched for scheduled jobs.

Answer: D

Explanation:

In Databricks notebooks, using the `display()` function triggers an action that forces Spark to execute the code and produce a result. However, Spark operations are generally divided into transformations and actions. Transformations create a new dataset from an existing one and are lazy, meaning they are not computed immediately but added to a logical plan. Actions, like `display()`, trigger the execution of this logical plan. Repeatedly running the same code cell can lead to misleading performance measurements due to caching. When a dataset is used multiple times, Spark's optimization mechanism caches it in memory, making subsequent executions faster. This behavior does not accurately represent the first-time execution performance in a production environment where data might not be cached yet.

To get a more realistic measure of performance, it is recommended to:

? Clear the cache or restart the cluster to avoid the effects of caching.

? Test the entire workflow end-to-end rather than cell-by-cell to understand the cumulative performance.

? Consider using a representative sample of the production data, ensuring it includes various cases the code will encounter in production.

References:

? Databricks Documentation on Performance Optimization: Databricks Performance Tuning

? Apache Spark Documentation: RDD Programming Guide - Understanding transformations and actions

NEW QUESTION 5

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 to power several production monitoring dashboards and a production model. At present, 45 of the 100 fields are being used in at least one of these applications.

The data engineer is trying to determine the best approach for dealing with schema declaration given the highly-nested structure of the data and the numerous fields.

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

A. The 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.

B. Because Delta Lake uses Parquet for data storage, data types can be easily evolved by just modifying file footer information in place.

C. Human labor in writing code is the largest cost associated with data engineering workloads; as such, automating table declaration logic should be a priority in all migration workloads.

D. Because Databricks will infer schema using types that allow all observed data to be processed, setting types manually provides greater assurance of data quality enforcement.

E. Schema inference and evolution on Databricks ensure that inferred types will always accurately match the data types used by downstream systems.

Answer: D

Explanation:

This is the correct answer because it accurately presents information about Delta Lake and Databricks that may impact the decision-making process of a junior data engineer who is trying to determine the best approach for dealing with schema declaration given the highly-nested structure of the data and the numerous fields. Delta Lake and Databricks support schema inference and evolution, which means that they can automatically infer the schema of a table from the source data and allow adding new columns or changing column types without affecting existing queries or pipelines. However, schema inference and evolution may not always be desirable or reliable, especially when dealing with complex or nested data structures or when enforcing data quality and consistency across different systems. Therefore, setting types manually can provide greater assurance of data quality enforcement and avoid potential errors or conflicts due to incompatible or unexpected data types. Verified References: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Schema inference and partition of streaming DataFrames/Datasets" section.

NEW QUESTION 6

The business intelligence team has a dashboard configured to track various summary metrics for retail stores. This includes total sales for the previous day alongside totals and averages for a variety of time periods. The fields required to populate this dashboard have the following schema:

For Demand forecasting, the Lakehouse contains a validated table of all itemized sales updated incrementally in near real-time. This table named `products_per_order`, includes the following fields:

Because reporting on long-term sales trends is less volatile, analysts using the new dashboard only require data to be refreshed once daily. Because the dashboard will be queried interactively by many users throughout a normal business day, it should return results quickly and reduce total compute associated with

each materialization.

Which solution meets the expectations of the end users while controlling and limiting possible costs?

- A. Use the Delta Cache to persist the products_per_order table in memory to quickly refresh the dashboard with each query.
- B. Populate the dashboard by configuring a nightly batch job to save the required data to quickly update the dashboard with each query.
- C. Use Structure Streaming to configure a live dashboard against the products_per_order table within a Databricks notebook.
- D. Define a view against the products_per_order table and refresh the dashboard against this view.

Answer: D

Explanation:

Given the requirement for daily refresh of data and the need to ensure quick response times for interactive queries while controlling costs, a nightly batch job to pre-compute and save the required summary metrics is the most suitable approach.

? By pre-aggregating data during off-peak hours, the dashboard can serve queries quickly without requiring on-the-fly computation, which can be resource-intensive and slow, especially with many users.

? This approach also limits the cost by avoiding continuous computation throughout the day and instead leverages a batch process that efficiently computes and stores the necessary data.

? The other options (A, C, D) either do not address the cost and performance requirements effectively or are not suitable for the use case of less frequent data refresh and high interactivity.

References:

? Databricks Documentation on Batch Processing: [Databricks Batch Processing](#)

? Data Lakehouse Patterns: [Data Lakehouse Best Practices](#)

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 junior data engineer has configured a workload that posts the following JSON to the Databricks REST API endpoint 2.0/jobs/create.

```
{
  "name": "Ingest new data",
  "existing_cluster_id": "6015-954420-peace720",
  "notebook_task": {
    "notebook_path": "/Prod/ingest.py"
  }
}
```

Assuming that all configurations and referenced resources are available, which statement describes the result of executing this workload three times?

- A. Three new jobs named "Ingest new data" will be defined in the workspace, and they will each run once daily.
- B. The logic defined in the referenced notebook will be executed three times on new clusters with the configurations of the provided cluster ID.
- C. Three new jobs named "Ingest new data" will be defined in the workspace, but no jobs will be executed.
- D. One new job named "Ingest new data" will be defined in the workspace, but it will not be executed.
- E. The logic defined in the referenced notebook will be executed three times on the referenced existing all purpose cluster.

Answer: E

Explanation:

This is the correct answer because the JSON posted to the Databricks REST API endpoint 2.0/jobs/create defines a new job with a name, an existing cluster id, and a notebook task. However, it does not specify any schedule or trigger for the job execution. Therefore, three new jobs with the same name and configuration will be created in the workspace, but none of them will be executed until they are manually triggered or scheduled. Verified References: [Databricks Certified Data Engineer Professional], under "Monitoring & Logging" section; [Databricks Documentation], under "Jobs API - Create" section.

NEW QUESTION 9

Which statement characterizes the general programming model used by Spark Structured Streaming?

- A. Structured Streaming leverages the parallel processing of GPUs to achieve highly parallel data throughput.
- B. Structured Streaming is implemented as a messaging bus and is derived from Apache Kafka.
- C. Structured Streaming uses specialized hardware and I/O streams to achieve sub-second latency for data transfer.
- D. Structured Streaming models new data arriving in a data stream as new rows appended to an unbounded table.
- E. Structured Streaming relies on a distributed network of nodes that hold incremental state values for cached stages.

Answer: B

Explanation:

This is the correct answer because it characterizes the general programming model used by Spark Structured Streaming, which is to treat a live data stream as a table that is being continuously appended. This leads to a new stream processing model that is very similar to a batch processing model, where users can express their streaming computation using the same Dataset/DataFrame API as they would use for static data. The Spark SQL engine will take care of running the streaming query incrementally and continuously and updating the final result as streaming data continues to arrive. Verified References: [Databricks Certified Data Engineer Professional], under "Structured Streaming" section; Databricks Documentation, under "Overview" section.

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 10

The Databricks CLI is used to trigger a run of an existing job by passing the `job_id` parameter. The response that the job run request has been submitted successfully includes a `run_id`.

Which statement describes what the number alongside this field represents?

- A. The `job_id` is returned in this field.
- B. The `job_id` and number of times the job has been are concatenated and returned.
- C. The number of times the job definition has been run in the workspace.
- D. The globally unique ID of the newly triggered run.

Answer: D

Explanation:

When triggering a job run using the Databricks CLI, the `run_id` field in the response represents a globally unique identifier for that particular run of the job. This `run_id` is distinct from the `job_id`. While the `job_id` identifies the job definition and is constant across all runs of that job, the `run_id` is unique to each execution and is used to track and query the status of that specific job run within the Databricks environment. This distinction allows users to manage and reference individual executions of a job directly.

NEW QUESTION 12

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 17

What statement is true regarding the retention of job run history?

- A. It is retained until you export or delete job run logs
- B. It is retained for 30 days, during which time you can deliver job run logs to DBFS or S3
- C. It is retained for 60 days, during which you can export notebook run results to HTML
- D. It is retained for 60 days, after which logs are archived
- E. It is retained for 90 days or until the run-id is re-used through custom run configuration

Answer: C

NEW QUESTION 22

The business reporting team requires that data for their dashboards be updated every hour. The total processing time for the pipeline that extracts transforms and load the data for their pipeline runs in 10 minutes.

Assuming normal operating conditions, which configuration will meet their service-level agreement requirements with the lowest cost?

- A. Schedule a job to execute the pipeline once an hour on a dedicated interactive cluster.
- B. Schedule a Structured Streaming job with a trigger interval of 60 minutes.
- C. Schedule a job to execute the pipeline once an hour on a new job cluster.
- D. Configure a job that executes every time new data lands in a given directory.

Answer: C

Explanation:

Scheduling a job to execute the data processing pipeline once an hour on a new job cluster is the most cost-effective solution given the scenario. Job clusters are ephemeral in nature; they are spun up just before the job execution and terminated upon completion, which means you only incur costs for the time the cluster is active. Since the total processing time is only 10 minutes, a new job cluster created for each hourly execution minimizes the running time and thus the cost, while also fulfilling the requirement for hourly data updates for the business reporting team's dashboards.

References:

? Databricks documentation on jobs and job clusters: <https://docs.databricks.com/jobs.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 32

A data architect has designed a system in which two Structured Streaming jobs will concurrently write to a single bronze Delta table. Each job is subscribing to a

different topic from an Apache Kafka source, but they will write data with the same schema. To keep the directory structure simple, a data engineer has decided to nest a checkpoint directory to be shared by both streams.

The proposed directory structure is displayed below:

Which statement describes whether this checkpoint directory structure is valid for the given scenario and why?

- A. No; Delta Lake manages streaming checkpoints in the transaction log.
- B. Yes; both of the streams can share a single checkpoint directory.
- C. No; only one stream can write to a Delta Lake table.
- D. Yes; Delta Lake supports infinite concurrent writers.
- E. No; each of the streams needs to have its own checkpoint directory.

Answer: E

Explanation:

This is the correct answer because checkpointing is a critical feature of Structured Streaming that provides fault tolerance and recovery in case of failures. Checkpointing stores the current state and progress of a streaming query in a reliable storage system, such as DBFS or S3. Each streaming query must have its own checkpoint directory that is unique and exclusive to that query. If two streaming queries share the same checkpoint directory, they will interfere with each other and cause unexpected errors or data loss. Verified References: [Databricks Certified Data Engineer Professional], under "Structured Streaming" section; Databricks Documentation, under "Checkpointing" section.

NEW QUESTION 34

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 37

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 38

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 42

The Databricks workspace administrator has configured interactive clusters for each of the data engineering groups. To control costs, clusters are set to terminate after 30 minutes of inactivity. Each user should be able to execute workloads against their assigned clusters at any time of the day.

Assuming users have been added to a workspace but not granted any permissions, which of the following describes the minimal permissions a user would need to start and attach to an already configured cluster.

- A. "Can Manage" privileges on the required cluster
- B. Workspace Admin privileges, cluster creation allowe
- C. "Can Attach To" privileges on the required cluster
- D. Cluster creation allowe
- E. "Can Attach To" privileges on the required cluster
- F. "Can Restart" privileges on the required cluster
- G. Cluster creation allowe
- H. "Can Restart" privileges on the required cluster

Answer: D

Explanation:

<https://learn.microsoft.com/en-us/azure/databricks/security/auth-authz/access-control/cluster-acl>
<https://docs.databricks.com/en/security/auth-authz/access-control/cluster-acl.html>

NEW QUESTION 43

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 48

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

```

Cards (id LONG, items ARRAY<STRUCT<id: LONG, count: INT>>)
id items email
1001[[{"id": "DESK65", count: 1}]] "u1@domain.com"
1002[[{"id": "KYBD45", 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 cards 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: "BOGUSO"}]]

```

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 53

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 55

A data engineer wants to refactor the following DLT code, which includes multiple definition with very similar code:

```
@dlt.table(name=f"t1_dataset")
def t1_dataset():
    return spark.read.table(t1)

@dlt.table(name=f"t2_dataset")
def t2_dataset():
    return spark.read.table(t2)

@dlt.table(name=f"t3_dataset")
def t3_dataset():
    return spark.read.table(t3)

...
```

In an attempt to programmatically create these tables using a parameterized table definition, the data engineer writes the following code.

```
tables = ["t1", "t2", "t3"]

for t in tables:
    @dlt.table(name=f"{t}_dataset")
    def new_table():
```

The pipeline runs an update with this refactored code, but generates a different DAG showing incorrect configuration values for tables. How can the data engineer fix this?

- A. Convert the list of configuration values to a dictionary of table settings, using table names as keys.
- B. Convert the list of configuration values to a dictionary of table settings, using different input the for loop.
- C. Load the configuration values for these tables from a separate file, located at a path provided by a pipeline parameter.
- D. Wrap the loop inside another table definition, using generalized names and properties to replace with those from the inner table

Answer: A

Explanation:

The issue with the refactored code is that it tries to use string interpolation to dynamically create table names within the `dlt.table` decorator, which will not correctly interpret the table names. Instead, by using a dictionary with table names as keys and their configurations as values, the data engineer can iterate over the dictionary items and use the keys (table names) to properly configure the table settings. This way, the decorator can correctly recognize each table name, and the corresponding configuration settings can be applied appropriately.

NEW QUESTION 59

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 60

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 64

Which statement regarding spark configuration on the Databricks platform is true?

- A. Spark configuration properties set for an interactive cluster with the Clusters UI will impact all notebooks attached to that cluster.
- B. When the same spark configuration property is set for an interactive to the same interactive cluster.
- C. Spark configuration set within a notebook will affect all SparkSession attached to the same interactive cluster
- D. The Databricks REST API can be used to modify the Spark configuration properties for an interactive cluster without interrupting jobs.

Answer: A

Explanation:

When Spark configuration properties are set for an interactive cluster using the Clusters UI in Databricks, those configurations are applied at the cluster level. This means that all notebooks attached to that cluster will inherit and be affected by these configurations. This approach ensures consistency across all executions within that cluster, as the Spark configuration properties dictate aspects such as memory allocation, number of executors, and other vital execution parameters. This centralized configuration management helps maintain standardized execution environments across different notebooks, aiding in debugging and performance optimization.

References:

? Databricks documentation on configuring clusters: <https://docs.databricks.com/clusters/configure.html>

NEW QUESTION 66

A member of the data engineering team has submitted a short notebook that they wish to schedule as part of a larger data pipeline. Assume that the commands provided below produce the logically correct results when run as presented.

```
Cmd 1
rawDF = spark.table("raw_data")

Cmd 2
rawDF.printSchema()

Cmd 3
flattenedDF = rawDF.select("col", "values.*")

Cmd 4
finalDF = flattenedDF.drop("values")

Cmd 5
display(finalDF)

Cmd 6
finalDF.write.mode("append").saveAsTable("flat_data")
```

Which command should be removed from the notebook before scheduling it as a job?

- A. Cmd 2
- B. Cmd 3
- C. Cmd 4
- D. Cmd 5
- E. Cmd 6

Answer: E

Explanation:

Cmd 6 is the command that should be removed from the notebook before scheduling it as a job. This command is selecting all the columns from the finalDF dataframe and displaying them in the notebook. This is not necessary for the job, as the finalDF dataframe is already written to a table in Cmd 7. Displaying the dataframe in the notebook will only consume resources and time, and it will not affect the output of the job. Therefore, Cmd 6 is redundant and should be removed. The other commands are essential for the job, as they perform the following tasks:

? Cmd 1: Reads the raw_data table into a Spark dataframe called rawDF.

? Cmd 2: Prints the schema of the rawDF dataframe, which is useful for debugging and understanding the data structure.

? Cmd 3: Selects all the columns from the rawDF dataframe, as well as the nested columns from the values struct column, and creates a new dataframe called flattenedDF.

? Cmd 4: Drops the values column from the flattenedDF dataframe, as it is no longer needed after flattening, and creates a new dataframe called finalDF.

? Cmd 5: Explains the physical plan of the finalDF dataframe, which is useful for optimizing and tuning the performance of the job.

? Cmd 7: Writes the finalDF dataframe to a table called flat_data, using the append mode to add new data to the existing table.

NEW QUESTION 68

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 72

A Delta Lake table representing metadata about content from user has the following schema:
 Based on the above schema, which column is a good candidate for partitioning the Delta Table?

- A. Date
- B. Post_id
- C. User_id
- D. Post_time

Answer: A

Explanation:

Partitioning a Delta Lake table improves query performance by organizing data into partitions based on the values of a column. In the given schema, the date column is a good candidate for partitioning for several reasons:

? Time-Based Queries: If queries frequently filter or group by date, partitioning by the date column can significantly improve performance by limiting the amount of data scanned.

? Granularity: The date column likely has a granularity that leads to a reasonable number of partitions (not too many and not too few). This balance is important for optimizing both read and write performance.

? Data Skew: Other columns like post_id or user_id might lead to uneven partition sizes (data skew), which can negatively impact performance.

Partitioning by post_time could also be considered, but typically date is preferred due to its more manageable granularity.

References:

? Delta Lake Documentation on Table Partitioning: Optimizing Layout with Partitioning

NEW QUESTION 76

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. No permissions
- D. Can Read
- E. Can Run

Answer: C

Explanation:

This is the correct answer because it is the maximum notebook permissions that can be granted to the user without allowing accidental changes to production code or data. Notebook permissions are used to control access to notebooks in Databricks workspaces. There are four types of notebook permissions: Can

Manage, Can Edit, Can Run, and Can Read. Can Manage allows full control over the notebook, including editing, running, deleting, exporting, and changing permissions. Can Edit allows modifying and running the notebook, but not changing permissions or deleting it. Can Run allows executing commands in an existing cluster attached to the notebook, but not modifying or exporting it. Can Read allows viewing the notebook content, but not running or modifying it. In this case, granting Can Read permission to the user will allow them to review the production logic in the notebook without allowing them to make any changes to it or run any commands that may affect production data. Verified References: [Databricks Certified Data Engineer Professional], under "Databricks Workspace" section; Databricks Documentation, under "Notebook permissions" section.

NEW QUESTION 80

The security team is exploring whether or not the Databricks secrets module can be leveraged for connecting to an external database. After testing the code with all Python variables being defined with strings, they upload the password to the secrets module and configure the correct permissions for the currently active user. They then modify their code to the following (leaving all other variables unchanged).

```
password = dbutils.secrets.get(scope="db_creds", key="jdbc_password")

print(password)

df = (spark
      .read
      .format("jdbc")
      .option("url", connection)
      .option("dbtable", tablename)
      .option("user", username)
      .option("password", password)
      )
```

Which statement describes what will happen when the above code is executed?

- A. The connection to the external table will fail; the string "redacted" will be printed.
- B. An interactive input box will appear in the notebook; if the right password is provided, the connection will succeed and the encoded password will be saved to DBFS.
- C. An interactive input box will appear in the notebook; if the right password is provided, the connection will succeed and the password will be printed in plain text.
- D. The connection to the external table will succeed; the string value of password will be printed in plain text.
- E. The connection to the external table will succeed; the string "redacted" will be printed.

Answer: E

Explanation:

This is the correct answer because the code is using the `dbutils.secrets.get` method to retrieve the password from the secrets module and store it in a variable. The secrets module allows users to securely store and access sensitive information such as passwords, tokens, or API keys. The connection to the external table will succeed because the password variable will contain the actual password value. However, when printing the password variable, the string "redacted" will be displayed instead of the plain text password, as a security measure to prevent exposing sensitive information in notebooks. Verified References: [Databricks Certified Data Engineer Professional], under "Security & Governance" section; Databricks Documentation, under "Secrets" section.

NEW QUESTION 82

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 85

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 86

The data engineering team maintains the following code:

```
accountDF = spark.table("accounts")
orderDF = spark.table("orders")
itemDF = spark.table("items")

orderWithItemDF = (orderDF.join(
    itemDF,
    orderDF.itemID == itemDF.itemID)
    .select(
        orderDF.accountID,
        orderDF.itemID,
        itemDF.itemName))

finalDF = (accountDF.join(
    orderWithItemDF,
    accountDF.accountID == orderWithItemDF.accountID)
    .select(
        orderWithItemDF["*"],
        accountDF.city))

(finalDF.write
    .mode("overwrite")
    .table("enriched_itemized_orders_by_account"))
```

Assuming that this code produces logically correct results and the data in the source tables has been de-duplicated and validated, which statement describes what will occur when this code is executed?

- A. A batch job will update the enriched_itemized_orders_by_account table, replacing only those rows that have different values than the current version of the table, using accountID as the primary key.
- B. The enriched_itemized_orders_by_account table will be overwritten using the current valid version of data in each of the three tables referenced in the join logic.
- C. An incremental job will leverage information in the state store to identify unjoined rows in the source tables and write these rows to the enriched_itemized_orders_by_account table.
- D. An incremental job will detect if new rows have been written to any of the source tables; if new rows are detected, all results will be recalculated and used to overwrite the enriched_itemized_orders_by_account table.
- E. No computation will occur until enriched_itemized_orders_by_account is queried; upon query materialization, results will be calculated using the current valid version of data in each of the three tables referenced in the join logic.

Answer: B

Explanation:

This is the correct answer because it describes what will occur when this code is executed. The code uses three Delta Lake tables as input sources: accounts, orders, and order_items. These tables are joined together using SQL queries to create a view called new_enriched_itemized_orders_by_account, which contains information about each order item and its associated account details. Then, the code uses write.format("delta").mode("overwrite") to overwrite a target table called enriched_itemized_orders_by_account using the data from the view. This means that every time this code is executed, it will replace all existing data in the target table with new data based on the current valid version of data in each of the three input tables. Verified References: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Write to Delta tables" section.

NEW QUESTION 91

The data engineering team has configured a job to process customer requests to be forgotten (have their data deleted). All user data that needs to be deleted is stored in Delta Lake tables using default table settings.

The team has decided to process all deletions from the previous week as a batch job at 1am each Sunday. The total duration of this job is less than one hour.

Every Monday at 3am, a batch job executes a series of VACUUM commands on all Delta Lake tables throughout the organization.

The compliance officer has recently learned about Delta Lake's time travel functionality. They are concerned that this might allow continued access to deleted data.

Assuming all delete logic is correctly implemented, which statement correctly addresses this concern?

- A. Because the vacuum command permanently deletes all files containing deleted records, deleted records may be accessible with time travel for around 24 hours.
- B. Because the default data retention threshold is 24 hours, data files containing deleted records will be retained until the vacuum job is run the following day.
- C. Because Delta Lake time travel provides full access to the entire history of a table, deleted records can always be recreated by users with full admin privileges.
- D. Because Delta Lake's delete statements have ACID guarantees, deleted records will be permanently purged from all storage systems as soon as a delete job completes.

E. Because the default data retention threshold is 7 days, data files containing deleted records will be retained until the vacuum job is run 8 days later.

Answer: E

Explanation:

<https://learn.microsoft.com/en-us/azure/databricks/delta/vacuum>

NEW QUESTION 93

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 98

A Databricks SQL dashboard has been configured to monitor the total number of records present in a collection of Delta Lake tables using the following query pattern:

```
SELECT COUNT (*) FROM table -
```

Which of the following describes how results are generated each time the dashboard is updated?

- A. The total count of rows is calculated by scanning all data files
- B. The total count of rows will be returned from cached results unless REFRESH is run
- C. The total count of records is calculated from the Delta transaction logs
- D. The total count of records is calculated from the parquet file metadata
- E. The total count of records is calculated from the Hive metastore

Answer: C

Explanation:

<https://delta.io/blog/2023-04-19-faster-aggregations-metadata/#:~:text=You%20can%20get%20the%20number,a%20given%20Delta%20table%20version.>

NEW QUESTION 100

Although the Databricks Utilities Secrets module provides tools to store sensitive credentials and avoid accidentally displaying them in plain text users should still be careful with which credentials are stored here and which users have access to using these secrets.

Which statement describes a limitation of Databricks Secrets?

- A. Because the SHA256 hash is used to obfuscate stored secrets, reversing this hash will display the value in plain text.
- B. Account administrators can see all secrets in plain text by logging on to the Databricks Accounts console.
- C. Secrets are stored in an administrators-only table within the Hive Metastore; database administrators have permission to query this table by default.
- D. Iterating through a stored secret and printing each character will display secret contents in plain text.
- E. The Databricks REST API can be used to list secrets in plain text if the personal access token has proper credentials.

Answer: E

Explanation:

This is the correct answer because it describes a limitation of Databricks Secrets. Databricks Secrets is a module that provides tools to store sensitive credentials and avoid accidentally displaying them in plain text. Databricks Secrets allows creating secret scopes, which are collections of secrets that can be accessed by users or groups. Databricks Secrets also allows creating and managing secrets using the Databricks CLI or the Databricks REST API. However, a limitation of Databricks Secrets is that the Databricks REST API can be used to list secrets in plain text if the personal access token has proper credentials. Therefore, users should still be careful with which credentials are stored in Databricks Secrets and which users have access to using these secrets. Verified References: [Databricks Certified Data Engineer Professional], under "Databricks Workspace" section; Databricks Documentation, under "List secrets" section.

NEW QUESTION 101

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 104

The data governance team is reviewing user for deleting records for compliance with GDPR. The following logic has been implemented to propagate deleted requests from the user_lookup table to the user_aggregate table.

```
(spark.read
  .format("delta")
  .option("readChangeData", True)
  .option("startingTimestamp", '2021-08-22 00:00:00')
  .option("endingTimestamp", '2021-08-29 00:00:00')
  .table("user_lookup")
  .createOrReplaceTempView("changes"))

spark.sql("""
DELETE FROM user_aggregates
WHERE user_id IN (
  SELECT user_id
  FROM changes
  WHERE _change_type='delete'
)
""")
```

Assuming that user_id is a unique identifying key and that all users have requested deletion have been removed from the user_lookup table, which statement describes whether successfully executing the above logic guarantees that the records to be deleted from the user_aggregates table are no longer accessible and why?

- A. No: files containing deleted records may still be accessible with time travel until a VACUUM command is used to remove invalidated data files.
- B. Yes: Delta Lake ACID guarantees provide assurance that the DELETE command succeeded fully and permanently purged these records.
- C. No: the change data feed only tracks inserts and updates not deleted records.
- D. No: the Delta Lake DELETE command only provides ACID guarantees when combined with the MERGE INTO command

Answer: A

Explanation:

The DELETE operation in Delta Lake is ACID compliant, which means that once the operation is successful, the records are logically removed from the table. However, the underlying files that contained these records may still exist and be accessible via time travel to older versions of the table. To ensure that these records are physically removed and compliance with GDPR is maintained, a VACUUM command should be used to clean up these data files after a certain retention period. The VACUUM command will remove the files from the storage layer, and after this, the records will no longer be accessible.

NEW QUESTION 108

Each configuration below is identical to the extent that each cluster has 400 GB total of RAM, 160 total cores and only one Executor per VM. Given a job with at least one wide transformation, which of the following cluster configurations will result in maximum performance?

- A. • Total VMs: 1 • 400 GB per Executor • 160 Cores / Executor
- B. • Total VMs: 8 • 50 GB per Executor • 20 Cores / Executor
- C. • Total VMs: 4 • 100 GB per Executor • 40 Cores/Executor
- D. • Total VMs: 2 • 200 GB per Executor • 80 Cores / Executor

Answer: B

Explanation:

This is the correct answer because it is the cluster configuration that will result in maximum performance for a job with at least one wide transformation. A wide transformation is a type of transformation that requires shuffling data across partitions, such as join, groupBy, or orderBy. Shuffling can be expensive and time-consuming, especially if there are too many or too few partitions. Therefore, it is important to choose a cluster configuration that can balance the trade-off between parallelism and network overhead. In this case, having 8 VMs with 50 GB per executor and 20 cores per executor will create 8 partitions, each with enough memory and CPU resources to handle the shuffling efficiently. Having fewer VMs with more memory and cores per executor will create fewer partitions, which will reduce parallelism and increase the size of each shuffle block. Having more VMs with less memory and cores per executor will create more partitions, which will increase parallelism but also increase the network overhead and the number of shuffle files. Verified References: [Databricks Certified Data Engineer Professional], under “Performance Tuning” section; Databricks Documentation, under “Cluster configurations” section.

NEW QUESTION 111

.....

THANKS FOR TRYING THE DEMO OF OUR PRODUCT

Visit Our Site to Purchase the Full Set of Actual Databricks-Certified-Professional-Data-Engineer Exam Questions With Answers.

We Also Provide Practice Exam Software That Simulates Real Exam Environment And Has Many Self-Assessment Features. Order the Databricks-Certified-Professional-Data-Engineer Product From:

<https://www.2passeasy.com/dumps/Databricks-Certified-Professional-Data-Engineer/>

Money Back Guarantee

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