Databricks.Databricks-Certified-Professional-Data-Engineer.v2026-06-12.q103

Exam Code:	Databricks-Certified-Professional-Data-Engineer
Exam Name:	Databricks Certified Professional Data Engineer Exam
Certification Provider:	Databricks
Free Question Number:	103
Version:	v2026-06-12
# of views:	103
# of Questions views:	1044
https://www.freecram.net/torrent/Databricks.Databricks-Certified-Professional-Data-Engineer.v2026-06-12.q103.html	

NEW QUESTION: 1
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_ey, 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 3 table; old values are maintained as a new column alongside the current value.
B. The customers table is implemented as a Type 2 table; old values are maintained but marked as no longer current and new values are inserted.
C. The customers table is implemented as a Type 0 table; all writes are append only with no changes to existing values.
D. The customers table is implemented as a Type 1 table; old values are overwritten by new values and no history is maintained.

E. The customers table is implemented as a Type 2 table; old values are overwritten and new customers are appended.
Answer: (SHOW ANSWER)
The logic uses the MERGE INTO command to merge new records from the view updates into the table customers. The MERGE INTO command takes two arguments: a target table and a source table or view. The command also specifies a condition to match records between the target and the source, and a set of actions to perform when there is a match or not. In this case, the condition is to match records by customer_id, which is the primary key of the customers table. The actions are to update the existing record in the target with the new values from the source, and set the current_flag to false to indicate that the record is no longer current; and to insert a new record in the target with the new values from the source, and set the current_flag to true to indicate that the record is current. This means that old values are maintained but marked as no longer current and new values are inserted, which is the definition of a Type 2 table. Verified References: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Merge Into (Delta Lake on Databricks)" section.

NEW QUESTION: 2
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:

```
CREATE DATABASE finance_eda_db
LOCATION '/mnt/finance_eda_bucket';
GRANT USAGE ON DATABASE finance_eda_db TO finance;
GRANT CREATE ON DATABASE finance_eda_db TO finance;
```

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

```
CREATE TABLE finance_eda_db.tx_sales AS
SELECT *
FROM sales
WHERE state = "TX";
```

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: (SHOW ANSWER)
https://docs.databricks.com/en/lakehouse/data-objects.html

NEW QUESTION: 3
A data engineer is performing a join operating to combine values from a static userlookup table with a streaming DataFrame streamingDF.
Which code block attempts to perform an invalid stream-static join?
A. userLookup.join(streamingDF, ["userid"], how="inner")
B. streamingDF.join(userLookup, ["user_id"], how="outer")
C. streamingDF.join(userLookup, ["user_id"], how="left")
D. streamingDF.join(userLookup, ["userid"], how="inner")

E. userLookup.join(streamingDF, ["user_id"], how="right")
Answer: (SHOW ANSWER)
In Spark Structured Streaming, certain types of joins between a static DataFrame and a streaming DataFrame are not supported. Specifically, a right outer join where the static DataFrame is on the left side and the streaming DataFrame is on the right side is not valid. This is because Spark Structured Streaming cannot handle scenarios where it has to wait for new rows to arrive in the streaming DataFrame to match rows in the static DataFrame. The other join types listed (inner, left, and full outer joins) are supported in streaming-static DataFrame joins.
:
Structured Streaming Programming Guide: Join Operations
Databricks Documentation on Stream-Static Joins: Databricks Stream-Static Joins

NEW QUESTION: 4
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 ofVACUUMcommands 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: (SHOW ANSWER)
https://learn.microsoft.com/en-us/azure/databricks/delta/vacuum

NEW QUESTION: 5
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 fieldpk_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: (SHOW ANSWER)

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 bronzetable, 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: 6
The data science team has requested assistance in accelerating queries on free form text from user reviews.
The data is currently stored in Parquet with the below schema:
item_id INT, user_id INT, review_id INT, rating FLOAT, review STRING
The review column contains the full text of the review left by the user. Specifically, the data science team is looking to identify if any of 30 key words exist in this field.
A junior data engineer suggests converting this data to Delta Lake will improve query performance.
Which response to the junior data engineer s suggestion is correct?
A. Delta Lake statistics are not optimized for free text fields with high cardinality.
B. Text data cannot be stored with Delta Lake.
C. ZORDER ON review will need to be run to see performance gains.
D. The Delta log creates a term matrix for free text fields to support selective filtering.
E. Delta Lake statistics are only collected on the first 4 columns in a table.
Answer: A (LEAVE A REPLY)
Converting the data to Delta Lake may not improve query performance on free text fields with high cardinality, such as the review column. This is because Delta Lake collects statistics on the minimum and maximum values of each column, which are not very useful for filtering or skipping data on free text fields.
Moreover, Delta Lake collectsstatistics on the first 32 columns by default, which may not include the review column if the table has more columns. Therefore, the junior data engineer's suggestion is not correct. A better approach would be to use a full-text search engine, such as Elasticsearch, to index and query the review column. Alternatively, you can use natural language processing techniques, such as tokenization, stemming, and lemmatization, to preprocess the review column and create a new column with normalized terms that can be used for filtering or skipping data. References:
* Optimizations: https://docs.delta.io/latest/optimizations-oss.html
* Full-text search with Elasticsearch: https://docs.databricks.com/data/data-sources/elasticsearch.html
* Natural language processing: https://docs.databricks.com/applications/nlp/index.html

NEW QUESTION: 7
Assuming that the Databricks CLI has been installed and configured correctly, which Databricks CLI command can be used to upload a custom Python Wheel to object storage mounted with the DBFS for use with a production job?
A. configure
B. fs
C. jobs
D. libraries
E. workspace
Answer: (SHOW ANSWER)

The libraries command group allows you to install, uninstall, and list libraries on Databricks clusters. You can use the libraries install command to install a custom Python Wheel on a cluster by specifying the --whl option and the path to the wheel file. For example, you can use the following command to install a custom Python Wheel named mylib-0.1-py3-none-any.whl on a cluster with the id 1234-567890-abcde123:
databricks libraries install --cluster-id 1234-567890-abcde123 --whl dbfs:/mnt/mylib/mylib-0.1-py3-none-any.
whl
This will upload the custom Python Wheel to the cluster and make it available for use with a production job.
You can also use the libraries uninstall command to uninstall a library from a cluster, and the libraries list command to list the libraries installed on a cluster.
References:
Libraries CLI (legacy): https://docs.databricks.com/en/archive/dev-tools/cli/libraries-cli.html Library operations: https://docs.databricks.com/en/dev-tools/cli/commands.html#library-operations Install or update the Databricks CLI: https://docs.databricks.com/en/dev-tools/cli/install.html

NEW QUESTION: 8
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, usingdisplay()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 JARs. all PySpark and Spark SQL logic should be refactored.
B. 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.
C. 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.
D. 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.
E. 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: (SHOW ANSWER)
This is the correct answer because it explains which of the following adjustments will get a more accurate measure of how code is likely to perform in production. The adjustment is that 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. When developing code in Databricks notebooks, one should be aware of how Spark handles transformations and actions. Transformations are operations that create a new DataFrame or Dataset from an existing one, such as filter, select, or join. Actions are operations that trigger a computation on a DataFrame or Dataset and return a result to the driver program or write it to storage, such as count, show, or save. Calling display() on a DataFrame or Dataset is also an action that triggers a computation and displays the result in a notebook cell. Spark uses lazy evaluation for transformations, which means that they are not executed until an action is called. Spark also uses caching to store intermediate results in memory or disk for faster access in subsequent actions. Therefore, 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. To get a more accurate measure of how code is likely to perform in production, one should avoid calling display() too often or clear the cache before running each cell. Verified References: [Databricks Certified Data Engineer Professional], under "Spark Core" section; Databricks Documentation, under "Lazy evaluation" section; Databricks Documentation, under "Caching" section.

NEW QUESTION: 9

Given the following error traceback (from display(df.select(3* " heartrate "))) which shows AnalysisException: cannot resolve ' heartrateheartrateheartrate ' , which statement describes the error being raised?

A. There is a type error because a DataFrame object cannot be multiplied.

B. There is a syntax error because the heartrate column is not correctly identified as a column.

C. There is no column in the table named heartrateheartrateheartrate.

D. There is a type error because a column object cannot be multiplied.

Answer: (SHOW ANSWER)

* Exact extract: "select() expects column names or Column expressions." References: PySpark DataFrame select; Column expressions and col().

NEW QUESTION: 10

A data engineer is configuring a Databricks Asset Bundle to deploy a job with granular permissions. The requirements are:

* Grant the data-engineers group CAN_MANAGE access to the job.
* Ensure the auditors group can view the job but not modify or run it.
* Avoid granting unintended permissions to other users or groups.

How should the data engineer deploy the job while meeting the requirements?

A. resources:
jobs:
my-job:
name: data-pipeline
tasks: (...)
job_clusters: (...)
permissions:
- group_name: data-engineers
level: CAN_MANAGE
- group_name: auditors
level: CAN_VIEW

B. resources:
jobs:
my-job:
name: data-pipeline
tasks: (...)
job_clusters: (...)
permissions:
- group_name: data-engineers
level: CAN_MANAGE
- group_name: auditors
level: CAN_VIEW

C. resources:
jobs:
my-job:
name: data-pipeline

tasks: [...]
job_clusters: [...]
permissions:
- group_name: data-engineers
level: CAN_MANAGE
permissions:
- group_name: auditors
level: CAN_VIEW
D. permissions:
- group_name: data-engineers
level: CAN_MANAGE
- group_name: auditors
level: CAN_VIEW
resources:
jobs:
my-job:
name: data-pipeline
tasks: [...]
job_clusters: [...]
Answer: (SHOW ANSWER)
Databricks documents that resource-specific permissions for bundle resources can be defined under the resource itself, such as resources.jobs. < job > .permissions , using group_name and level . The documented syntax supports CAN_VIEW , CAN_MANAGE , and related permission levels, which matches option B. (Databricks Documentation) Option C is invalid because it repeats the permissions key incorrectly. Option D applies top-level permissions more broadly across bundle resources instead of scoping them specifically to the job, which does not best satisfy the "avoid unintended permissions" requirement. Option B is therefore the correct and properly scoped configuration. (Databricks Documentation)
======

NEW QUESTION: 11
A team of data engineer are adding tables to a DLT pipeline that contain repetitive expectations for many of the same data quality checks.
One member of the team suggests reusing these data quality rules across all tables defined for this pipeline.
What approach would allow them to do this?
A. Maintain data quality rules in a Delta table outside of this pipeline's target schema, providing the schema name as a pipeline parameter.
B. Use global Python variables to make expectations visible across DLT notebooks included in the same pipeline.
C. Add data quality constraints to tables in this pipeline using an external job with access to pipeline configuration files.
D. Maintain data quality rules in a separate Databricks notebook that each DLT notebook of file.
Answer: (SHOW ANSWER)
Maintaining data quality rules in a centralized Delta table allows for the reuse of these rules across multiple DLT (Delta Live Tables) pipelines. By storing these rules outside the pipeline's target schema and referencing the schema name as a pipeline parameter, the team can apply the same set of data quality checks to different tables within the pipeline. This approach ensures consistency in data quality validations and reduces redundancy in code by not having to replicate the same rules in each DLT notebook or file.
:

Databricks Documentation on Delta Live Tables: Delta Live Tables Guide

NEW QUESTION: 12
A data engineer, while designing a Pandas UDF to process financial time-series data with complex calculations that require maintaining state across rows within each stock symbol group, must ensure the function is efficient and scalable. Which approach will solve the problem with minimum overhead while preserving data integrity?
A. Use a scalar_iter Pandas UDF with iterator-based processing, implementing state management through persistent storage (Delta tables) that gets updated after each batch to maintain continuity across iterator chunks.
B. Use a scalar Pandas UDF that processes the entire dataset at once, implementing custom partitioning logic within the UDF to group by stock symbol and maintain state using global variables shared across all executor processes.
C. Use applyInPandas on a Spark DataFrame so that each stock symbol group is received as a pandas DataFrame, allowing processing within each group while maintaining state variables local to each group's processing function.
D. Use a grouped-aggregate Pandas UDF that processes each stock symbol group independently, maintaining state through intermediate aggregation results that get passed between successive UDF calls via broadcast variables.
Answer: (SHOW ANSWER)
applyInPandas is the documented grouped Pandas API for processing each group as a pandas DataFrame.
Spark passes all columns for each group together, which allows per-group state to be maintained naturally inside the function. By contrast, scalar Pandas UDFs are batch-oriented Series-to-Series operations, not group- state processing tools. (Apache Spark) This is why option C is the intended best answer among the listed choices.
Option A adds unnecessary external persistence overhead, option B relies on unsupported global executor state, and option D misuses grouped aggregation semantics for row-by-row stateful logic. Spark also documents applyInPandas specifically as a grouped operation, while scalar Pandas UDFs process row batches and concatenate results rather than preserving grouped state semantics. (Apache Spark)
======

NEW QUESTION: 13
A Structured Streaming job deployed to production has been experiencing delays during peak hours of the day. At present, during normal execution, each microbatch of data is processed in less than 3 seconds. During peak hours of the day, execution time for each microbatch becomes very inconsistent, sometimes exceeding
30 seconds. The streaming write is currently configured with a trigger interval of 10 seconds.
Holding all other variables constant and assuming records need to be processed in less than 10 seconds, which adjustment will meet the requirement?
A. Decrease the trigger interval to 5 seconds; triggering batches more frequently allows idle executors to begin processing the next batch while longer running tasks from previous batches finish.
B. Increase the trigger interval to 30 seconds; setting the trigger interval near the maximum execution time observed for each batch is always best practice to ensure no records are dropped.
C. The trigger interval cannot be modified without modifying the checkpoint directory; to maintain the current stream state, increase the number of shuffle partitions to maximize parallelism.
D. Use the trigger once option and configure a Databricks job to execute the query every 10 seconds; this ensures all backlogged records are processed with each batch.
E. Decrease the trigger interval to 5 seconds; triggering batches more frequently may prevent records from backing up and large batches from causing spill.
Answer: (SHOW ANSWER)
The scenario presented involves inconsistent microbatch processing times in a Structured Streaming job during peak hours, with the need to ensure that records are processed within 10 seconds. The trigger once option is the most suitable adjustment to address these challenges:
* Understanding Triggering Options:
* Fixed Interval Triggering (Current Setup): The current trigger interval of 10 seconds may contribute to the inconsistency during peak times as it doesn ' t adapt based on the processing time of the microbatches. If a batch takes longer to process, subsequent batches will start piling up, exacerbating the delays.

* Trigger Once: This option allows the job to run a single microbatch for processing all available data and then stop. It is useful in scenarios where batch sizes are unpredictable and can vary significantly, which seems to be the case during peak hours in this scenario.
* Implementation of Trigger Once:
* Setup: Instead of continuously running, the job can be scheduled to run every 10 seconds using a Databricks job. This scheduling effectively acts as a custom trigger interval, ensuring that each execution cycle handles all available data up to that point without overlapping or queuing up additional executions.
* Advantages: This approach allows for each batch to complete processing all available data before the next batch starts, ensuring consistency in handling data surges and preventing the system from being overwhelmed.
* Rationale Against Other Options:
* Option A and E (Decrease Interval): Decreasing the trigger interval to 5 seconds might exacerbate the problem by increasing the frequency of batch starts without ensuring the completion of previous batches, potentially leading to higher overhead and less efficient processing.
* Option B (Increase Interval): Increasing the trigger interval to 30 seconds could lead to latency issues, as the data would be processed less frequently, which contradicts the requirement of processing records in less than 10 seconds.
* Option C (Modify Partitions): While increasing parallelism through more shuffle partitions can improve performance, it does not address the fundamental issue of batch scheduling and could still lead to inconsistency during peak loads.
* Conclusion:
* By using the trigger once option and scheduling the job every 10 seconds, you ensure that each microbatch has sufficient time to process all available data thoroughly before the next cycle begins, aligning with the need to handle peak loads more predictably and efficiently.
References
* Structured Streaming Programming Guide - Triggering
* Databricks Jobs Scheduling

NEW QUESTION: 14
In order to prevent accidental commits to production data, a senior data engineer has instituted a policy that all development work will reference clones of Delta Lake tables. After testing both deep and shallow clone, development tables are created using shallow clone.
A few weeks after initial table creation, the cloned versions of several tables implemented as Type 1 Slowly Changing Dimension (SCD) stop working. The transaction logs for the source tables show that vacuum was run the day before.
Why are the cloned tables no longer working?
A. The data files compacted by vacuum are not tracked by the cloned metadata; running refresh on the cloned table will pull in recent changes.
B. Because Type 1 changes overwrite existing records, Delta Lake cannot guarantee data consistency for cloned tables.
C. The metadata created by the clone operation is referencing data files that were purged as invalid by the vacuum command
D. Running vacuum automatically invalidates any shallow clones of a table; deep clone should always be used when a cloned table will be repeatedly queried.
Answer: (SHOW ANSWER)
In Delta Lake, a shallow clone creates a new table by copying the metadata of the source table without duplicating the data files. When the vacuum command is run on the source table, it removes old data files that are no longer needed to maintain the transactional log's integrity, potentially including files referenced by the shallow clone's metadata. If these files are purged, the shallow cloned tables will reference non-existent data files, causing them to stop working properly. This highlights the dependency of shallow clones on the source table's data files and the impact of data management operations like vacuum on these clones.
Databricks documentation on Delta Lake, particularly the sections on cloning tables (shallow and deep cloning) and data retention with the vacuum command (https://docs.databricks.com/delta/index.html).

NEW QUESTION: 15

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 allowed. "Can Attach To" privileges on the required cluster
C. Cluster creation allowed. "Can Attach To" privileges on the required cluster
D. "Can Restart" privileges on the required cluster
E. Cluster creation allowed. "Can Restart" privileges on the required cluster
Answer: (SHOW ANSWER)
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: 16
A data team ' s Structured Streaming job is configured to calculate running aggregates for item sales to update a downstream marketing dashboard. The marketing team has introduced a new field to track the number of times this promotion code is used for each item. A junior data engineer suggests updating the existing query as follows:
Note that proposed changes are in bold.

Which step must also be completed to put the proposed query into production?
A. Increase the shuffle partitions to account for additional aggregates
B. Specify a new checkpointlocation
C. Run REFRESH TABLE delta, /item_agg '
D. Remove .option (mergeSchema ' , true ') from the streaming write
Answer: (SHOW ANSWER)
When introducing a new aggregation or a change in the logic of a Structured Streaming query, it is generally necessary to specify a new checkpoint location. This is because the checkpoint directory contains metadata about the offsets and the state of the aggregations of a streaming query. If the logic of the query changes, such as including a new aggregation field, the state information saved in the current checkpoint would not be compatible with the new logic, potentially leading to incorrect results or failures. Therefore, to accommodate the new field and ensure the streaming job has the correct starting point and state information for aggregations, a new checkpoint location should be specified.

:
Databricks documentation on Structured Streaming: https://docs.databricks.com/spark/latest/structured- streaming/index.html Databricks documentation on streaming checkpoints: https://docs.databricks.com/spark/latest/structured- streaming/production.html#checkpointing

NEW QUESTION: 17
When scheduling Structured Streaming jobs for production, which configuration automatically recovers from query failures and keeps costs low?
A. Cluster: New Job Cluster;
Retries: Unlimited;
Maximum Concurrent Runs: Unlimited
B. Cluster: New Job Cluster;
Retries: None;
Maximum Concurrent Runs: 1
C. Cluster: Existing All-Purpose Cluster;
Retries: Unlimited;
Maximum Concurrent Runs: 1
D. Cluster: New Job Cluster;
Retries: Unlimited;
Maximum Concurrent Runs: 1
E. Cluster: Existing All-Purpose Cluster;
Retries: None;
Maximum Concurrent Runs: 1
Answer: (SHOW ANSWER)
Maximum concurrent runs: Set to 1. There must be only one instance of each query concurrently active.
Retries: Set to Unlimited. https://docs.databricks.com/en/structured-streaming/query-recovery.html

NEW QUESTION: 18
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 spar configuration property is set for an interactive to the same interactive cluster.
C. Spark configuration set within an 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: (SHOW ANSWER)

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

NEW QUESTION: 19
A nightly job ingests data into a Delta Lake table using the following code:

```
from pyspark.sql.functions import current_timestamp, input_file_name, co
from pyspark.sql.column import Column

def ingest_daily_batch(time_col: Column, year:int, month:int, day:int):
    (spark.read
        .format("parquet")
        .load(f"/mnt/daily_batch/{year}/{month}/{day}")
        .select("*",
                time_col.alias("ingest_time"),
                input_file_name().alias("source_file")
            )
        .write
        .mode("append")
        .saveAsTable("bronze")
    )
```


The next step in the pipeline requires a function that returns an object that can be used to manipulate new records that have not yet been processed to the next table in the pipeline.
Which code snippet completes this function definition?
def new_records():
A. return spark.readStream.table(" bronze ")
B. return spark.readStream.load(" bronze ")
C.

```
return (spark.read
    .table("bronze")
    .filter(col("ingest_time") == current_timestamp())
)
```

D. return spark.read.option(" readChangeFeed " , " true ").table (" bronze ")

E.
```
return (spark.read
    .table("bronze")
    .filter(col("source_file") == f"/mnt/daily_batch/{year}/{month}/{day}")
)
```

Answer: (SHOW ANSWER)

https://docs.databricks.com/en/delta/delta-change-data-feed.html

NEW QUESTION: 20

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 and 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: (SHOW ANSWER)

For this scenario where a one-TB JSON dataset needs to be converted into Parquet format without employing Delta Lake ' s auto-sizing features, the goal is to avoid unnecessary data shuffles and yet ensure optimal file sizes for the output Parquet files. Here's a breakdown of why option A is most suitable:

* Setting maxPartitionBytes: The spark.sql.files.maxPartitionBytes configuration controls the size of blocks that Spark reads from the data source (in this case, the JSON files) but also influences the output size of files when data is written without repartition or coalesce operations. Setting this parameter to 512 MB directly addresses the requirement to manage the output file size effectively.
* Data Ingestion and Processing:
* Ingesting Data: Load the JSON dataset into a DataFrame.
* Applying Transformations: Perform any required narrow transformations that do not involve shuffling data (like filtering or adding new columns).
* Writing to Parquet: Directly write the transformed DataFrame to Parquet files. The setting for maxPartitionBytes ensures that each part-file is approximately 512 MB, meeting the requirement for part-file size without additional steps to repartition or coalesce the data.
* Performance Consideration: This approach is optimal because:
* It avoids the overhead of shuffling data, which can be significant, especially with large datasets.
* It directly ties the read/write operations to a configuration that matches the target output size, making it efficient in terms of both computation and I/O operations.
* Alternative Options Analysis:
* Option B and D: Involves repartitioning, which would trigger a shuffle of the data, contradicting the requirement to avoid shuffling for performance reasons.
* Option C: Uses coalesce, which is less intensive than repartition but can still lead to uneven partition sizes and does not directly control the output file size as effectively as setting maxPartitionBytes.

* Option E: Setting shuffle partitions to 512 doesn't directly control the output file size for writing to Parquet and could lead to smaller files depending on the dataset ' s partitioning post- transformations.
References
* Apache Spark Configuration
* Writing to Parquet Files in Spark

NEW QUESTION: 21
A data engineer is designing a system to process batch patient encounter data stored in an S3 bucket, creating a Delta table (patient_encounters) with columns encounter_id, patient_id, encounter_date, diagnosis_code, and treatment_cost. The table is queried frequently by patient_id and encounter_date, requiring fast performance. Fine-grained access controls must be enforced. The engineer wants to minimize maintenance and boost performance.
How should the data engineer create the patient_encounters table?
A. Create an external table in Unity Catalog, specifying an S3 location for the data files. Enable predictive optimization through table properties, and configure Unity Catalog permissions for access controls.
B. Create a managed table in Unity Catalog . Configure Unity Catalog permissions for access controls, and rely on predictive optimization to enhance query performance and simplify maintenance.
C. Create a managed table in Unity Catalog. Configure Unity Catalog permissions for access controls, schedule jobs to run OPTIMIZE and VACUUM commands daily to achieve best performance.
D. Create a managed table in Hive Metastore. Configure Hive Metastore permissions for access controls, and rely on predictive optimization to enhance query performance and simplify maintenance.
Answer: (SHOW ANSWER)
Databricks documentation specifies that Unity Catalog managed tables are the preferred choice for secure, low-maintenance Delta Lake architectures. Managed tables provide full lifecycle management, including metadata, file storage, and access control integration with Unity Catalog. Fine-grained permissions can be enforced at the column and row level through built-in Unity Catalog governance.
Additionally, Predictive Optimization (Auto Optimize + Auto Compaction) automatically manages file sizes, metadata pruning, and layout optimization, eliminating the need for manual maintenance such as scheduling OPTIMIZE or VACUUM.
External tables (A) require manual path management, and Hive Metastore tables (D) do not support Unity Catalog access policies. Therefore, creating a managed Unity Catalog table with predictive optimization provides both the security and performance benefits needed, making B the correct solution.

NEW QUESTION: 22
A data engineer is optimizing a managed Delta table that suffers from data skew and frequently changing query filter columns . The engineer wants to avoid costly data rewrites when query patterns evolve. The table size is under 1 TB.
How should the data engineer meet this requirement?
A. Apply Z-ordering , since it allows flexible reorganization of data layout without rewriting existing files and adapts easily to new filter columns.
B. Use Hive-style partitioning , as it provides efficient data skipping and is easy to change partition columns at any time.
C. Enable liquid clustering , as it efficiently handles data skew, allows clustering keys to be changed without rewriting existing data, and adapts to evolving query patterns.
D. Combine partitioning and Z-ordering to maximize flexibility and minimize maintenance as query patterns change.
Answer: C (LEAVE A REPLY)
The Databricks documentation describes Liquid Clustering as the recommended data layout optimization for evolving workloads. Unlike traditional partitioning or Z-ordering, Liquid Clustering dynamically maintains data organization without rewriting existing files when clustering keys change. It handles data skew automatically and supports flexible re-clustering based on query patterns. Partitioning and Z-ordering require full data rewrites whenever key structures change, making them expensive for

tables with frequently evolving access patterns. For tables under a few terabytes, Liquid Clustering offers the best balance between scalability, adaptability, and maintenance efficiency. Thus, option C aligns with Databricks' best practice for modern adaptive layout optimization.

NEW QUESTION: 23
A data engineer manages a production Lakeflow Declarative Pipeline that processes customer transaction data. The pipeline includes several data quality expectations such as transaction_amount > 0 and customer_id IS NOT NULL. These expectations are defined using the EXPECT clause in SQL.
The engineer aims to monitor the pipeline's data quality by analyzing the number of records that passed or failed each expectation during the latest pipeline update. The Lakeflow Declarative Pipelines event logs are stored in a Delta table named event_log_table.
For the most recent pipeline update, determine a programmatically appropriate approach to extract information like the name of each expectation, associated dataset, count of records that passed the expectation, and count of records that failed the expectation.
Which method retrieves the desired data quality metrics from the Lakeflow Declarative Pipelines event log?
A. Access the event_log_table, filter for events where event_type = ' flow_progress ' , and parse details.
flow_progress.data_quality.expectations field to extract the required metrics.
B. Use the Lakeflow Declarative Pipelines UI to navigate to the specific pipeline, select the dataset, and view the Data Quality tab to manually retrieve the expectation metrics.
C. Query the event_log_table for events with event_type = ' data_quality ' and directly select the passed_records and failed_records fields.
D. Access the event_log_table, filter for events where event_type = ' expectation_result ' , and extract the expectation metrics from the details field.
Answer: (SHOW ANSWER)
The Databricks documentation specifies that for Lakeflow Declarative Pipelines, detailed data quality metrics are logged as events of type expectation_result within the event log. Each record of this type contains fields including expectation_name, dataset_name, passed_records, and failed_records. Filtering on event_type = ' expectation_result ' and expanding the details field allows retrieving metrics for each expectation from the most recent pipeline update. While flow_progress provides summary statistics and data_quality events aggregate results, only expectation_result events provide granular, per-expectation metrics required for audit and monitoring automation.

NEW QUESTION: 24
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: (SHOW ANSWER)
https://delta.io/blog/2023-04-19-faster-aggregations-metadata/#:~:text=You%20can%20get%20the%20number,a%20given%20Delta%20table%20version.

NEW QUESTION: 25
Incorporating unit tests into a PySpark application requires upfront attention to the design of your jobs, or a potentially significant refactoring of existing code.
Which statement describes a main benefit that offset this additional effort?
A. Ensures that all steps interact correctly to achieve the desired end result

B. Improves the quality of your data
C. Validates a complete use case of your application
D. Yields faster deployment and execution times
E. Troubleshooting is easier since all steps are isolated and tested individually
Answer: (SHOW ANSWER)

NEW QUESTION: 26
The data engineering team maintains the following code:

```
import pyspark.sql.functions as F

(spark.table("silver_customer_sales")
  .groupBy("customer_id")
  .agg(
    F.min("sale_date").alias("first_transaction_date"),
    F.max("sale_date").alias("last_transaction_date"),
    F.mean("sale_total").alias("average_sales"),
    F.countDistinct("order_id").alias("total_orders"),
    F.sum("sale_total").alias("lifetime_value")
  ).write
  .mode("overwrite")
  .table("gold_customer_lifetime_sales_summary")
)
```


Assuming that this code produces logically correct results and the data in the source table has been de- duplicated and validated, which statement describes what will occur when this code is executed?
A. The silver_customer_sales table will be overwritten by aggregated values calculated from all records in the gold_customer_lifetime_sales_summary table as a batch job.
B. A batch job will update the gold_customer_lifetime_sales_summary table, replacing only those rows that have different values than the current version of the table, using customer_id as the primary key.
C. The gold_customer_lifetime_sales_summary table will be overwritten by aggregated values calculated from all records in the silver_customer_sales table as a batch job.
D. An incremental job will leverage running information in the state store to update aggregate values in the gold_customer_lifetime_sales_summary table.
E. An incremental job will detect if new rows have been written to the silver_customer_sales table; if new rows are detected, all aggregates will be recalculated and used to overwrite the gold_customer_lifetime_sales_summary table.
Answer: (SHOW ANSWER)
This code is using the pyspark.sql.functions library to group the silver_customer_sales table by customer_id and then aggregate the data using the minimum sale date, maximum sale total, and sum of distinct order ids.

The resulting aggregated data is then written to the gold_customer_lifetime_sales_summary table, overwriting any existing data in that table. This is a batch job that does not use any incremental or streaming logic, and does not perform any merge or update operations. Therefore, the code will overwrite the gold table with the aggregated values from the silver table every time it is executed. References :
* https://docs.databricks.com/spark/latest/dataframes-datasets/introduction-to-dataframes-python.html
* https://docs.databricks.com/spark/latest/dataframes-datasets/transforming-data-with-dataframes.html
* https://docs.databricks.com/spark/latest/dataframes-datasets/aggregating-data-with-dataframes.html

NEW QUESTION: 27
To identify the top users consuming compute resources, a data engineering team needs to monitor usage within their Databricks workspace for better resource utilization and cost control. The team decided to use Databricks system tables, available under the System catalog in Unity Catalog, to gain detailed visibility into workspace activity.
Which SQL query should the team run from the System catalog to achieve this?
A. SELECT sku_name,
identity_metadata.created_by AS user_email,
COUNT(usage_quantity) AS total_dbus
FROM system.billing.usage
GROUP BY user_email, sku_name
ORDER BY total_dbus DESC
LIMIT 10
B. SELECT identity_metadata.run_as AS user_email,
SUM(usage_quantity) AS total_dbus
FROM system.billing.usage
GROUP BY user_email
ORDER BY total_dbus DESC
LIMIT 10
C. SELECT sku_name,
identity_metadata.created_by AS user_email,
SUM(usage_quantity * usage_unit) AS total_dbus
FROM system.billing.usage
GROUP BY user_email, sku_name
ORDER BY total_dbus DESC
LIMIT 10
D. SELECT sku_name,
usage_metadata.run_name AS user_email,
SUM(usage_quantity) AS total_dbus
FROM system.billing.usage
GROUP BY user_email, sku_name
ORDER BY total_dbus DESC
LIMIT 10
Answer: (SHOW ANSWER)
The system.billing.usage table in the Unity Catalog System schema provides detailed usage metrics for each workload in the workspace. The field identity_metadata.run_as identifies the user or service principal under which the job or query executed. Summing usage_quantity provides total DBU (Databricks Unit)

consumption per user. According to Databricks documentation, this table is the authoritative source for monitoring workspace cost drivers, showing compute SKU, user, and DBU consumption over time. Grouping by identity_metadata.run_as and summing usage_quantity produces the correct aggregation to determine top users. Other queries use non-existent or incorrect fields (created_by, run_name, or multiplied usage quantities), which do not reflect actual billing metrics.

NEW QUESTION: 28
A data engineer is creating a data ingestion pipeline to understand where customers are taking their rented bicycles during use. The engineer noticed that over time, data being transmitted from the bicycle sensors fails to include key details like latitude and longitude. Downstream analysts need both the clean records and the quarantined records available for separate processing.
The data engineer already has this code:
import dlt
from pyspark.sql.functions import expr
rules = {
" valid_lat " : " (lat IS NOT NULL) " ,
" valid_long " : " (long IS NOT NULL) "
}
quarantine_rules = " NOT({0}) " .format(" AND " .join(rules.values()))
@dlt.view
def raw_trips_data():
return spark.readStream.table(" ride_and_go.telemetry.trips ")
How should the data engineer meet the requirements to capture good and bad data?
A. @dlt.table(partition_cols=[" is_quarantined "])
@dlt.expect_all(rules)
def trips_data_quarantine():
return (
spark.readStream.table(" raw_trips_data ")
withColumn(" is_quarantined " , expr(quarantine_rules))
)
B. @dlt.table
@dlt.expect_all_or_drop(rules)
def trips_data_quarantine():
return spark.readStream.table(" raw_trips_data ")
C. @dlt.table(name= " trips_data_quarantine ")
def trips_data_quarantine():
return (
spark.readStream.table(" raw_trips_data ")
filter(expr(quarantine_rules))
)
D. @dlt.view
@dlt.expect_or_drop(" lat_long_present " , " (lat IS NOT NULL AND long IS NOT NULL) ") def trips_data_quarantine():
return spark.readStream.table(" ride_and_go.telemetry.trips ")
Answer: (SHOW ANSWER)

Databricks documents a quarantine pattern for Lakeflow Spark Declarative Pipelines in which you create a dataset containing both valid and invalid rows, add an is_quarantined flag based on your rule set, and then use that dataset for separate downstream processing paths. The documented pattern uses a boolean quarantine expression and preserves all rows while tracking quality metrics through expectations. (Databricks Documentation) Option A is the only choice that matches that documented design: it keeps all records, marks invalid rows with is_quarantined , and applies expectations to capture data quality metrics. Option B drops invalid rows, which fails the requirement to keep quarantined records available. Option C captures only bad rows and loses the clean path. Option D also drops invalid rows and therefore does not preserve both good and quarantined records for separate downstream use. (Databricks Documentation)
======

NEW QUESTION: 29
Which of the following is true of Delta Lake and the Lakehouse?
A. Because Parquet compresses data row by row. strings will only be compressed when a character is repeated multiple times.
B. Delta Lake automatically collects statistics on the first 32 columns of each table which are leveraged in data skipping based on query filters.
C. Views in the Lakehouse maintain a valid cache of the most recent versions of source tables at all times.
D. Primary and foreign key constraints can be leveraged to ensure duplicate values are never entered into a dimension table.
E. Z-order can only be applied to numeric values stored in Delta Lake tables
Answer: (SHOW ANSWER)
https://docs.delta.io/2.0.0/table-properties.html
Delta Lake automatically collects statistics on the first 32 columns of each table, which are leveraged in data skipping based on query filters 1 . Data skipping is a performance optimization technique that aims to avoid reading irrelevant data from the storage layer 1 . By collecting statistics such as min/max values, null counts, and bloom filters, Delta Lake can efficiently prune unnecessary files or partitions from the query plan 1 . This can significantly improve the query performance and reduce the I/O cost.
The other options are false because:
* Parquet compresses data column by column, not row by row 2 . This allows for better compression ratios, especially for repeated or similar values within a column 2 .
* Views in the Lakehouse do not maintain a valid cache of the most recent versions of source tables at all times 3 . Views are logical constructs that are defined by a SQL query on one or more base tables 3
. Views are not materialized by default, which means they do not store any data, but only the query definition 3 . Therefore, views always reflect the latest state of the source tables when queried 3 .
However, views can be cached manually using the CACHE TABLE or CREATE TABLE AS SELECT commands.
* Primary and foreign key constraints can not be leveraged to ensure duplicate values are never entered into a dimension table. Delta Lake does not support enforcing primary and foreign key constraints on tables. Constraints are logical rules that define the integrity and validity of the data in a table. Delta Lake relies on the application logic or the user to ensure the data quality and consistency.
* Z-order can be applied to any values stored in Delta Lake tables, not only numeric values. Z-order is a technique to optimize the layout of the data files by sorting them on one or more columns. Z-order can improve the query performance by clustering related values together and enabling more efficient data skipping. Z-order can be applied to any column that has a defined ordering, such as numeric, string, date, or boolean values.
References: Data Skipping , Parquet Format , Views , [Caching], [Constraints] , [Z-Ordering]

NEW QUESTION: 30
The business reporting tem 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 jo to execute the pipeline once and 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 hour on a new job cluster.
D. Configure a job that executes every time new data lands in a given directory.
Answer: (SHOW ANSWER)
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.
:
Databricks documentation on jobs and job clusters: https://docs.databricks.com/jobs.html

NEW QUESTION: 31
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_ey, 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: (SHOW ANSWER)
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.
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: 32
An organization processes customer data from web and mobile applications. Data includes names, emails, phone numbers, and location history. Data arrives both as batch files (from SFTP daily) and streaming JSON events (from Kafka in real-time).
To comply with data privacy policies, the following requirements must be met:
* Personally Identifiable Information (PII) such as email, phone number, and IP address must be masked or anonymized before storage.
* Both batch and streaming pipelines must apply consistent PII handling.
* Masking logic must be auditable and reproducible.
* The masked data must remain usable for downstream analytics.
How should the data engineer design a compliant data pipeline on Databricks that supports both batch and streaming modes, applies data masking to PII, and maintains traceability for audits?
A. Allow PII to be stored unmasked in Bronze for lineage tracking, then apply masking logic in Gold tables used for reporting.
B. Load batch data with notebooks and ingest streaming data with SQL Warehouses; use Unity Catalog column masks on Silver tables to redact fields after storage.
C. Ingest both batch and streaming data using Lakeflow Declarative Pipelines, and apply masking via Unity Catalog column masks at read time to avoid modifying the data during ingestion.
D. Use Lakeflow Declarative Pipelines for batch and streaming ingestion, define a PII masking function
, and apply it during Bronze ingestion before writing to Delta Lake .
Answer: (SHOW ANSWER)
Databricks recommends applying data masking or anonymization before persisting PII to ensure compliance with privacy regulations such as GDPR and HIPAA. In a Lakeflow Declarative Pipeline , developers can define custom Python or SQL-based masking functions to standardize PII handling across both batch and streaming inputs.
This approach ensures that data entering the Delta Lake is already anonymized, guaranteeing consistent and auditable behavior. By applying masking during ingestion (in the Bronze layer), audit trails are preserved through pipeline event logs.
While Unity Catalog column masks (option C) can enforce dynamic masking at query time, they do not prevent PII storage. Thus, option D aligns with the best practice of securing PII before storage , while still supporting reproducibility and analytics usability.

NEW QUESTION: 33
A data engineer created a daily batch ingestion pipeline using a cluster with the latest DBR version to store banking transaction data, and persisted it in a MANAGED DELTA table called prod.gold.
all_banking_transactions_daily. The data engineer is constantly receiving complaints from business users who query this table ad hoc through a SQL Serverless Warehouse about poor query performance. Upon analysis, the data engineer identified that these users frequently use high-cardinality columns as filters. The engineer now seeks to implement a data layout optimization technique that is incremental, easy to maintain, and can evolve over time.

Which command should the data engineer implement?
A. Alter the table to use Hive-Style Partitions + Z-ORDER and implement a periodic OPTIMIZE command.
B. Alter the table to use Liquid Clustering and implement a periodic OPTIMIZE command.
C. Alter the table to use Hive-Style Partitions and implement a periodic OPTIMIZE command.
D. Alter the table to use Z-ORDER and implement a periodic OPTIMIZE command.
Answer: (SHOW ANSWER)
Databricks recommends Liquid Clustering for optimizing data layout in large Delta tables where query filters involve high-cardinality columns. Liquid Clustering automatically manages file organization and supports incremental maintenance without the need to rewrite data when clustering keys evolve. This is a key advantage over static partitioning or Z-ordering, which require costly file rewrites whenever optimization keys change. By combining Liquid Clustering with a periodic OPTIMIZE command, Databricks automatically compacts small files and maintains efficient data skipping performance. As stated in the Delta Lake optimization guide, Liquid Clustering is designed for scalability, minimal maintenance, and adaptability for analytical workloads with evolving query patterns-making B the correct answer.

NEW QUESTION: 34
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 tableused 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: (SHOW ANSWER)
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: 35
A transactions table has been liquid clustered on the columns product_id, user_id, and event_date.
Which operation lacks support for cluster on write?

A. spark.writestream.format('delta').mode('append')
B. CTAS and RTAS statements
C. INSERT INTO operations
D. spark.write.format('delta').mode('append')
Answer: (SHOW ANSWER)
Delta Lake'sLiquid Clusteringis an advanced feature that improves query performance by dynamically clustering data without requiring costly compaction steps like traditional Z-ordering.
When performing writes to aLiquid Clusteredtable, some write operations automatically maintain clustering, while othersdo not.
Explanation of Each Option:
* (A) spark.writestream.format('delta').mode('append') (Correct Answer)
* Reason:Streaming writes (writestream) donotsupportLiquid Clusteringbecause streaming data arrives in micro-batches.
* Since Liquid Clustering needs efficient global reorganization of files, streaming append operations don't provide sufficient data volume at a time to be effectively clustered.
* Delta Lake documentation states that Liquid Clustering is only supported for batch writes.
* (B) CTAS and RTAS statements
* Reason:CREATE TABLE AS SELECT (CTAS) and REPLACE TABLE AS SELECT (RTAS) are batch operationsand can enforce Liquid Clustering.
* These operations create or replace a table based on a query result, and since they are batch-based, Liquid Clustering applies.
* (C) INSERT INTO operations
* Reason:INSERT INTOis supportedfor Liquid Clustering because it is a batch operation.
* While it may not be as efficient as MERGE or COPY INTO, clustering is applied upon execution.
* (D) spark.write.format('delta').mode('append')
* Reason:Batch append operationsare supportedfor Liquid Clustering.
* Unlike streaming append, batch writes allow the optimizer to re-cluster data efficiently.
Conclusion:
Sincestreaming append operations do not support Liquid Clustering, option(A)is the correct answer.
References:
* Liquid Clustering in Delta Lake - Databricks Documentation

NEW QUESTION: 36
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 ail 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: (SHOW ANSWER)
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_eda_bucket/tx_sales.parquet. By using the LOCATION keyword, the querycreates 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_eda_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: 37
A data engineer is configuring a Databricks Asset Bundle to deploy a job with granular permissions. The requirements are:
* Grant the data-engineers group CAN_MANAGE access to the job.
* Ensure the auditors' group can view the job but not modify/run it.
* Avoid granting unintended permissions to other users/groups.
How should the data engineer deploy the job while meeting the requirements?
A. resources:
jobs:
my-job:
name: data-pipeline
tasks: [...]
job_clusters: [...]
permissions:
- group_name: data-engineers
level: CAN_MANAGE
- group_name: auditors
level: CAN_VIEW
- group_name: admin-team
level: IS_OWNER
B. resources:
jobs:
my-job:
name: data-pipeline
tasks: [...]
job: [...]
permissions:
- group_name: data-engineers
level: CAN_MANAGE
permissions:
- group_name: auditors
level: CAN_VIEW
C. permissions:
- group_name: data-engineers

level: CAN_MANAGE
- group_name: auditors
level: CAN_VIEW
resources:
jobs:
my-job:
name: data-pipeline
tasks: [...]
job_clusters: [...]
D. resources:
jobs:
my-job:
name: data-pipeline
tasks: [...]
job_clusters: [...]
permissions:
- group_name: data-engineers
level: CAN_MANAGE
- group_name: auditors
level: CAN_VIEW

Answer: (SHOW ANSWER)

Databricks Asset Bundles (DABs) allow jobs, clusters, and permissions to be defined as code in YAML configuration files. According to the Databricks documentation on job permissions and bundle deployment, when defining permissions within a job resource, they must be scoped directly under that specific job's definition. This ensures that permissions are applied only to the intended job resource and not inadvertently propagated to other jobs or resources.

In this scenario, the data engineer must grant the data-engineers group CAN_MANAGE access, allowing them to configure, edit, and manage the job, while the auditors group should only have CAN_VIEW, giving them read-only access to see configurations and results without the ability to modify or execute. Importantly, no additional groups should be granted permissions, in order to follow the principle of least privilege.

Options A and B introduce unnecessary or unintended groups (like admin-team in A) or define permissions outside of the job scope (as in B). Option C improperly separates the permissions block outside the job resource, which is not aligned with Databricks bundle best practices.

Option D is the correct approach because it defines the job resource my-job with its name, tasks, clusters, and the exact intended permissions (CAN_MANAGE for data-engineers and CAN_VIEW for auditors). This aligns with Databricks' principle of least privilege and ensures compliance with governance standards in Unity Catalog-enabled workspaces.

Reference: Databricks Asset Bundles documentation - Managing Jobs and Permissions

NEW QUESTION: 38

A data engineering team needs to implement a tagging system for their tables as part of an automated ETL process, and needs to apply tags programmatically to tables in Unity Catalog.

Which SQL command adds tags to a table programmatically?

A. ALTER TABLE table_name SET TAGS (' key1 ' = ' value1 ' , ' key2 ' = ' value2 ');

B. APPLY TAGS ON table_name VALUES (' key1 ' = ' value1 ' , ' key2 ' = ' value2 ');

C. COMMENT ON TABLE table_name TAGS (' key1 ' = ' value1 ' , ' key2 ' = ' value2 ');

D. SET TAGS FOR table_name AS (' key1 ' = ' value1 ' , ' key2 ' = ' value2 ');
Answer: (SHOW ANSWER)
Unity Catalog in Databricks provides the ability to attach tags (key-value metadata pairs) to securable objects such as catalogs, schemas, tables, volumes, and functions. Tags are critical for governance, compliance, and automation, as they allow organizations to track metadata like sensitivity, ownership, business purpose, and retention policies directly at the object level.
According to the official Databricks SQL reference for Unity Catalog, the correct way to programmatically add tags to a table is by using the ALTER TABLE ... SET TAGS command. The syntax is:
ALTER TABLE table_name SET TAGS (' tag_name ' = ' tag_value ' , ...);
This command can be used within ETL workflows or jobs to automatically apply metadata during or after ingestion, ensuring that governance and compliance rules are embedded in the pipeline itself.
* Option A is correct because it uses the supported syntax for applying tags.
* Option B (APPLY TAGS) is not valid SQL in Unity Catalog and is not recognized by Databricks.
* Option C confuses COMMENT with TAGS. While COMMENT can add descriptive text to a table, it does not handle tags.
* Option D (SET TAGS FOR) is not a valid SQL construct in Databricks for applying tags.
Thus, Option A is the only valid and documented way to programmatically set tags on a table in Unity Catalog.
Reference: Databricks SQL Language Reference - ALTER TABLE ... SET TAGS (Unity Catalog)

NEW QUESTION: 39
A platform engineer is creating catalogs and schemas for the development team to use.
The engineer has created an initial catalog, catalog_A, and initial schema, schema_A. The engineer has also granted USE CATALOG, USE SCHEMA, and CREATE TABLE to the development team so that the engineer can begin populating the schema with new tables.
Despite being owner of the catalog and schema, the engineer noticed that they do not have access to the underlying tables in Schema_A.
What explains the engineer's lack of access to the underlying tables?
A. The platform engineer needs to execute a REFRESH statement as the table permissions did not automatically update for owners.
B. Users granted with USE CATALOG can modify the owner's permissions to downstream tables.
C. The owner of the schema does not automatically have permission to tables within the schema, but can grant them to themselves at anypoint.
D. Permissions explicitly given by the table creator are the only way the Platform Engineer could access the underlying tables in their schema.
Answer: (SHOW ANSWER)
In Databricks, catalogs, schemas (or databases), and tables are managed through the Unity Catalog or Hive Metastore, depending on the environment. Permissions and ownership within these structures are governed by access control lists (ACLs).
* Catalog and Schema Ownership:When a platform engineer creates a catalog (such as catalog_A) and schema (such as schema_A), they automatically become the owner of those entities. This ownership gives them control over granting permissions for those entities (i.e., granting the USE CATALOG and USE SCHEMA privileges to others). However, ownership of the catalog or schema doesnot automaticallyextend to ownership or permission of individual tables within that schema.
* Table Permissions:For tables within a schema, the permission model is more granular. The table creator (i.e., whoever creates the table) is automatically assigned as the owner of that table. In this case, the platform engineer owns the schema but does not automatically inherit permissions to any table created within the schema unless explicitly granted by the table's owner or unless they grant permissions to themselves.
* Why the Engineer Lacks Access:The platform engineer notices that they do not have access to the underlying tables in schema_A despite being the owner of the schema. This occurs because the schema's ownership does not cascade to the tables. The engineer must either:
* Grant permissions to themselves for the tables in schema_A, or
* Be granted permissions by whoever created the tables within the schema.

* Resolution:As the owner of the schema, the platform engineer can easily grant themselves the required permissions (such as SELECT, INSERT, etc.) for the tables in the schema. This explains why the owner of a schema may not automatically have access to the tables and must take explicit steps to acquire those permissions.
References
* Databricks Unity Catalog Documentation: Manage Permissions
* [Databricks Permissions and Ownership](https://docs.databricks.com/security/access-control
/workspace-acl.html#permissions

NEW QUESTION: 40

A data engineer, User A, has promoted a new pipeline to production by using the REST API to programmatically create several jobs. A DevOps engineer, User B, has configured an external orchestration tool to trigger job runs through the REST API. Both users authorized the REST API calls using their personal access tokens.
Which statement describes the contents of the workspace audit logs concerning these events?

A. Because the REST API was used for job creation and triggering runs, a Service Principal will be automatically used to identity these events.
B. Because User B last configured the jobs, their identity will be associated with both the job creation events and the job run events.
C. Because these events are managed separately, User A will have their identity associated with the job creation events and User B will have their identity associated with the job run events.
D. Because the REST API was used for job creation and triggering runs, user identity will not be captured in the audit logs.
E. Because User A created the jobs, their identity will be associated with both the job creation events and the job run events.
Answer: (SHOW ANSWER)
The events are that a data engineer, User A, has promoted a new pipeline to production by using the REST API to programmatically create several jobs, and a DevOps engineer, User B, has configured an external orchestration tool to trigger job runs through the REST API. Both users authorized the REST API calls using their personal access tokens. The workspace audit logs are logs that record user activities in a Databricks workspace, such as creating, updating, or deleting objects like clusters, jobs, notebooks, or tables. The workspace audit logs also capture the identity of the user who performed each activity, as well as the time and details of the activity. Because these events are managed separately, User A will have their identity associated with the job creation events and User B will have their identity associated with the job run events in the workspace audit logs. Verified References: [Databricks Certified Data Engineer Professional], under
"Databricks Workspace" section; Databricks Documentation, under "Workspace audit logs" section.

NEW QUESTION: 41
A data engineer needs to provide access to a group named manufacturing-team. The team needs privileges to create tables in the quality schema.
Which set of SQL commands will grant a group named manufacturing-team to create tables in a schema named production with the parent catalog named manufacturing with the least privileges?
A. GRANT CREATE TABLE ON SCHEMA manufacturing.quality TO manufacturing-team; GRANT CREATE SCHEMA ON SCHEMA manufacturing.quality TO manufacturing-team; GRANT CREATE CATALOG ON CATALOG manufacturing TO manufacturing-team;
B. GRANT CREATE TABLE ON SCHEMA manufacturing.quality TO manufacturing-team; GRANT CREATE SCHEMA ON SCHEMA manufacturing.quality TO manufacturing-team; GRANT USE CATALOG ON CATALOG manufacturing TO manufacturing-team;
C. GRANT CREATE TABLE ON SCHEMA manufacturing.quality TO manufacturing-team; GRANT USE SCHEMA ON SCHEMA manufacturing.quality TO manufacturing-team; GRANT USE CATALOG ON CATALOG manufacturing TO manufacturing-team;
D. GRANT USE TABLE ON SCHEMA manufacturing.quality TO manufacturing-team; GRANT USE SCHEMA ON SCHEMA manufacturing.quality TO manufacturing-team; GRANT USE CATALOG ON CATALOG manufacturing TO manufacturing-team;
Answer: (SHOW ANSWER)

To create a table within a schema, a principal must have CREATE TABLE on the schema, USE SCHEMA on that schema, and USE CATALOG on the parent catalog. This combination ensures the group has just enough privileges to create objects in that schema without excessive permissions like CREATE SCHEMA or CREATE CATALOG.
Reference Source: Databricks Unity Catalog Privilege Model - "Privileges Required to Create a Table."

NEW QUESTION: 42
A data governance team at a large enterprise is improving data discoverability across its organization. The team has hundreds of tables in their Databricks Lakehouse with thousands of columns that lack proper documentation. Many of these tables were created by different teams over several years, with missing context about column meanings and business logic. The data governance team needs to quickly generate comprehensive column descriptions for all existing tables to meet compliance requirements and improve data literacy across the organization. They want to leverage modern capabilities to automatically generate meaningful descriptions rather than manually documenting each column, which would take months to complete.
Which approach should the team use in Databricks to automatically generate column comments and descriptions for existing tables?
A. Navigate to the table in Databricks Catalog Explorer, select the table schema view, and use the AI Generate option which leverages artificial intelligence to automatically create meaningful column descriptions based on column names, data types, sample values, and data patterns.
B. Use Delta Lake's DESCRIBE HISTORY command to analyze table evolution and infer column purposes from historical changes.
C. Use the DESCRIBE TABLE command to extract existing schema information and manually write descriptions based on column names and data types.
D. Write custom PySpark code using df.describe() and df.schema to programmatically generate basic statistical descriptions for each column.
Answer: (SHOW ANSWER)
Databricks Catalog Explorer provides a feature called AI Generate that automatically produces intelligent comments for columns. This feature uses metadata such as column names, types, patterns, and sampled values to generate human-readable documentation. According to the documentation, this is the recommended method to rapidly enrich schema metadata and improve data discoverability, especially at enterprise scale. Unlike DESCRIBE HISTORY or DESCRIBE TABLE, which only surface technical schema details, AI Generate directly produces business-oriented descriptions. PySpark statistical functions (df.describe) only return numeric statistics and cannot generate descriptive metadata. Thus, AI Generate in Catalog Explorer is the correct approach.

NEW QUESTION: 43
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: (SHOW ANSWER)
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: 44
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: (SHOW ANSWER)
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-acls/index.html
* DELETE: https://docs.databricks.com/delta/delta-update.html#delete-from-a-table

NEW QUESTION: 45
Which distribution does Databricks support for installing custom Python code packages?
A. jars
B. CRAN
C. nom
D. sbt
E. CRAM
F. Wheels
Answer: (SHOW ANSWER)

NEW QUESTION: 46
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: (SHOW ANSWER)
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: 47
To identify the top users consuming compute resources, a data engineering team needs to monitor usage within their Databricks workspace for better resource utilization and cost control. The team decided to use Databricks system tables, available under the system catalog in Unity Catalog, to gain detailed visibility into workspace activity.
Which SQL query should the team run from the system catalog to achieve this?
A. SELECT
sku_name,
identity_metadata.created_by AS user_email,
SUM(usage_quantity * usage_unit) AS total_dbus
FROM system.billing.usage
GROUP BY user_email, sku_name
ORDER BY total_dbus DESC
LIMIT 10
B. SELECT
sku_name,
identity_metadata.created_by AS user_email,
COUNT(usage_quantity) AS total_dbus
FROM system.billing.usage
GROUP BY user_email, sku_name
ORDER BY total_dbus DESC
LIMIT 10
C. SELECT
identity_metadata.run_as AS user_email,
SUM(usage_quantity) AS total_dbus
FROM system.billing.usage
GROUP BY user_email
ORDER BY total_dbus DESC
LIMIT 10
D. SELECT
sku_name,
usage_metadata.run_name AS user_email,
SUM(usage_quantity) AS total_dbus
FROM system.billing.usage
GROUP BY user_email, sku_name
ORDER BY total_dbus DESC

LIMIT 10
Answer: (SHOW ANSWER)
Databricks documents system.billing.usage as the correct system table for billable usage analysis, and it documents identity_metadata.run_as as the field that records who ran supported workloads such as jobs, notebooks, and Lakeflow Spark Declarative Pipelines. For "top users consuming compute resources," summing usage_quantity by identity_metadata.run_as is the correct conceptual approach. (Databricks Documentation) The other options are not aligned with the documented schema or metric usage. identity_metadata.created_by is not the general compute-consumer identity field for jobs and notebook workloads; it applies to specific products such as Databricks Apps and certain agent workloads. usage_quantity should be summed, not counted, and usage_unit is not something you multiply into DBUs in the way shown. usage_metadata.
run_name is not the documented user identity field for this purpose. As written, option C is the only option that matches the official identity model for user-attributed compute consumption. (Databricks Documentation)
======

NEW QUESTION: 48
A data engineer is tasked with building a nightly batch ETL pipeline that processes very large volumes of raw JSON logs from a data lake into Delta tables for reporting. The data arrives in bulk once per day, and the pipeline takes several hours to complete. Cost efficiency is important, but performance and reliability of completing the pipeline are the highest priorities.
Which type of Databricks cluster should the data engineer configure?
A. A lightweight single-node cluster with low worker node count to reduce costs.
B. A high-concurrency cluster designed for interactive SQL workloads.
C. An all-purpose cluster always kept running to ensure low-latency job startup times.
D. A job cluster configured to autoscale across multiple workers during the pipeline run.
Answer: D (LEAVE A REPLY)
Job clusters are optimized for automated production workloads. They start when a job is triggered and terminate automatically once the task completes. This ensures cost control while maintaining performance and reliability for batch ETL. Autoscaling allows Databricks to add or remove workers dynamically based on workload size, ensuring large data volumes are processed efficiently.
All-purpose clusters are intended for development or ad-hoc workloads, not scheduled ETL.
Reference Source: Databricks Compute and Job Cluster Configuration Documentation - "Autoscaling and Job Clusters."

NEW QUESTION: 49
A table in the Lakehouse namedcustomer_churn_paramsis 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.
The churn prediction model used by the ML team is fairly stable in production. The team is only interested in making predictions on records that have changed in the past 24 hours.
Which approach would simplify the identification of these changed records?
A. Apply the churn model to all rows in the customer_churn_params table, but implement logic to perform an upsert into the predictions table that ignores rows where predictions have not changed.
B. Convert the batch job to a Structured Streaming job using the complete output mode; configure a Structured Streaming job to read from the customer_churn_params table and incrementally predict against the churn model.
C. Calculate the difference between the previous model predictions and the current customer_churn_params on a key identifying unique customers before making new predictions; only make predictions on those customers not in the previous predictions.

D. Modify the overwrite logic to include a field populated by calling spark.sql.functions.
current_timestamp() as data are being written; use this field to identify records written on a particular date.
E. Replace the current overwrite logic with a merge statement to modify only those records that have changed; write logic to make predictions on the changed records identified by the change data feed.
Answer: E (LEAVE A REPLY)
The approach that would simplify the identification of the changed records is to replace the current overwrite logic with a merge statement to modify only those records that have changed, and write logic to make predictions on the changed records identified by the change data feed. This approach leverages the Delta Lake features of merge and change data feed, which are designed to handle upserts and track row-level changes in a Delta table12. By using merge, the data engineering team can avoid overwriting the entire table every night, and only update or insert the records that have changed in the source data. By using change data feed, the ML team can easily access the change events that have occurred in the customer_churn_params table, and filter them by operation type (update or insert) and timestamp. This way, they can only make predictions on the records that have changed in the past 24 hours, and avoid re-processing the unchanged records.
The other options are not as simple or efficient as the proposed approach, because:
* Option A would require applying the churn model to all rows in the customer_churn_params table, which would be wasteful and redundant. It would also require implementing logic to perform an upsert into the predictions table, which would be more complex than using the merge statement.
* Option B would require converting the batch job to a Structured Streaming job, which would involve changing the data ingestion and processing logic. It would also require using the complete output mode, which would output the entire result table every time there is a change in the source data, which would be inefficient and costly.
* Option C would require calculating the difference between the previous model predictions and the current customer_churn_params on a key identifying unique customers, which would be computationally expensive and prone to errors. It would also require storing and accessing the previous predictions, which would add extra storage and I/O costs.
* Option D would require modifying the overwrite logic to include a field populated by calling spark.sql.functions.current_timestamp() as data are being written, which would add extra complexity and overhead to the data engineering job. It would also require using this field to identify records written on a particular date, which would be less accurate and reliable than using the change data feed.
References: Merge, Change data feed

NEW QUESTION: 50
What statement is true regarding the retention of job run history?
A. It is retained for 90 days or until the run-id is re-used through custom run configuration
B. It is retained for 30 days, during which time you can deliver job run logs to DBFS or S3
C. t 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 until you export or delete job run logs
Answer: (SHOW ANSWER)

NEW QUESTION: 51
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: (SHOW ANSWER)
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: 52
A data engineer is running a groupBy aggregation on a massive user activity log grouped by user_id. A few users have millions of records, causing task skew and long runtimes.
Which technique will fix the skew in this aggregation?
A. Use salting by adding a random prefix to skewed keys before aggregation, then aggregate again after removing the prefix.
B. Increase the Spark driver memory and retry.
C. Use reduceByKey instead of groupBy to avoid shuffles.
D. Filter out the skewed users before the aggregation.
Answer: (SHOW ANSWER)
Task skew occurs when a small subset of keys holds a disproportionate amount of data, causing certain tasks to process significantly more records than others.
Databricks documentation recommends salting as an effective mitigation technique.
Salting introduces a random or calculated prefix to skewed keys, distributing records across multiple partitions and balancing the workload during the shuffle stage. After aggregation, a second pass re-aggregates results by removing the prefix to restore key integrity.
Increasing memory (B) does not resolve distribution imbalance; reduceByKey (C) still triggers shuffles; and filtering (D) would remove valid business data. Hence, salting is the correct and officially recommended approach to address skew in Spark aggregations.

NEW QUESTION: 53
The data governance team is reviewing code used for deleting records for compliance with GDPR. They note the following logic is used to delete records from the Delta Lake table named users .

```
DELETE FROM users
WHERE user_id IN
   (SELECT user_id FROM delete_requests)
```

Assuming that user_id is a unique identifying key and that delete_requests contains all users that have requested deletion, which statement describes whether successfully executing the above logic guarantees that the records to be deleted are no longer accessible and why?
A. Yes; Delta Lake ACID guarantees provide assurance that the delete command succeeded fully and permanently purged these records.
B. No; the Delta cache may return records from previous versions of the table until the cluster is restarted.
C. Yes; the Delta cache immediately updates to reflect the latest data files recorded to disk.
D. No; the Delta Lake delete command only provides ACID guarantees when combined with the merge into command.
E. No; files containing deleted records may still be accessible with time travel until a vacuum command is used to remove invalidated data files.

Answer: E (LEAVE A REPLY)
The code uses the DELETE FROM command to delete records from the users table that match a condition based on a join with another table called delete_requests, which contains all users that have requested deletion. The DELETE FROM command deletes records from a Delta Lake table by creating a new version of the table that does not contain the deleted records. However, this does not guarantee that the records to be deleted are no longer accessible, because Delta Lake supports time travel, which allows querying previous versions of the table using a timestamp or version number. Therefore, files containing deleted records may still be accessible with time travel until a vacuum command is used to remove invalidated data files from physical storage. Verified References: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Delete from a table" section; Databricks Documentation, under
"Remove files no longer referenced by a Delta table" section.

NEW QUESTION: 54
A data team is implementing an append-only Delta Lake pipeline that processes both batch and streaming data . They want to ensure that schema changes in the source data are automatically incorporated without breaking the pipeline.
Which configuration should the team use when writing data to the Delta table?
A. ignoreChanges = false
B. mergeSchema = true
C. overwriteSchema = true
D. validateSchema = false
Answer: (SHOW ANSWER)
When writing data to Delta Lake tables, the option mergeSchema = true allows automatic evolution of the table schema by merging new columns or data types introduced in the incoming data with the existing table definition. This ensures pipelines remain resilient to upstream schema changes, especially in append-only or streaming ingestion scenarios. The Databricks documentation specifies that this option should be enabled for schema-on-write flexibility, while overwriteSchema replaces the existing schema entirely and ignoreChanges applies only to CDC operations. validateSchema ensures consistency and does not handle evolution.
Therefore, B (mergeSchema = true) is the correct configuration for handling automatic schema evolution safely.

NEW QUESTION: 55
A Delta Lake table representing metadata about content from user has the following schema:
user_id LONG, post_text STRING, post_id STRING, longitude FLOAT, latitude FLOAT, post_time TIMESTAMP, date DATE 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: (SHOW ANSWER)
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.

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

NEW QUESTION: 56
A Delta Lake table was created with the below query:

```
CREATE TABLE prod.sales_by_stor
USING DELTA
LOCATION "/mnt/prod/sales_by_store"
```

Realizing that the original query had a typographical error, the below code was executed:
ALTER TABLE prod.sales_by_stor RENAME TO prod.sales_by_store
Which result will occur after running the second command?
A. The table reference in the metastore is updated and no data is changed.
B. The table name change is recorded in the Delta transaction log.
C. All related files and metadata are dropped and recreated in a single ACID transaction.
D. The table reference in the metastore is updated and all data files are moved.
E. A new Delta transaction log Is created for the renamed table.
Answer: A (LEAVE A REPLY)
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_eda_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 result that will occur after running the second command is that the table reference in the metastore is updated and no data is changed. The metastore is a service that stores metadata about tables, such as their schema, location, properties, and partitions. The metastore allows users to access tables using SQL commands or Spark APIs without knowing their physical location or format. When renaming an external table using the ALTER TABLE RENAME TO command, only the table reference in the metastore is updated with the new name; no data files or directories are moved or changed in the storage system. The table will still point to the same location and use the same format as before. However, if renaming a managed table, which is a table whose metadata and data are both managed by Databricks, both the table reference in the metastore and the data files in the default warehouse directory are moved and renamed accordingly. Verified References:
[Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "ALTER TABLE RENAME TO" section; Databricks Documentation, under "Metastore" section; Databricks Documentation, under "Managed and external tables" section.

NEW QUESTION: 57
A data engineer is building a Lakeflow Declarative Pipelines pipeline to process healthcare claims data. A metadata JSON file defines data quality rules for multiple tables, including:

```
{
" claims " : [
{ " name " : " valid_patient_id " , " constraint " : " patient_id IS NOT NULL " },
{ " name " : " non_negative_amount " , " constraint " : " claim_amount > = 0 " }
]
}
```

The pipeline must dynamically apply these rules to the claims table without hardcoding the rules.

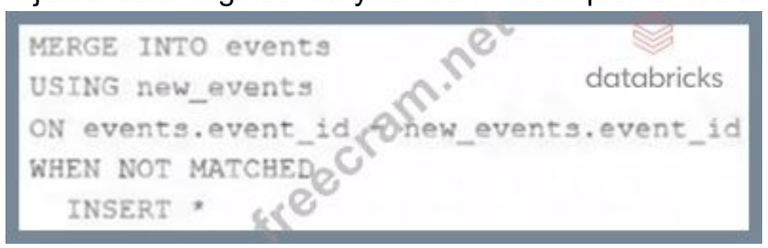
How should the data engineer achieve this?
A. Load the JSON metadata, loop through its entries, and apply expectations using dlt.expect_all.
B. Invoke an external API to validate records against the metadata rules.
C. Reference each expectation with @dlt.expect decorators in the table declaration.
D. Use a SQL CONSTRAINT block referencing the JSON file path.
Answer: (SHOW ANSWER)
Lakeflow Declarative Pipelines provide the expect_all method for programmatically applying multiple data quality expectations at once. The documentation explains that @dlt.expect_all accepts a dictionary of expectation names mapped to SQL constraints, allowing rules to be dynamically loaded from metadata such as JSON files. This ensures that pipelines remain maintainable and scalable without needing to hardcode individual @dlt.expect decorators. The event logs will track each expectation's pass and fail counts individually, making it auditable. Other options are incorrect: invoking an external API introduces unnecessary complexity, individual decorators require hardcoding, and SQL constraints cannot dynamically reference external JSON.

NEW QUESTION: 58
Which REST API call can be used to review the notebooks configured to run as tasks in a multi-task job?
A. /jobs/runs/list
B. /jobs/runs/get-output
C. /jobs/runs/get
D. /jobs/get
E. /jobs/list
Answer: (SHOW ANSWER)
This is the correct answer because it is the REST API call that can be used to review the notebooks configured to run as tasks in a multi-task job. The REST API is an interface that allows programmatically interacting with Databricks resources, such as clusters, jobs, notebooks, or tables. The REST API uses HTTP methods, such as GET, POST, PUT, or DELETE, to perform operations on these resources. The /jobs/get endpoint is a GET method that returns information about a job given its job ID. The information includes the job settings, such as the name, schedule, timeout, retries, email notifications, and tasks. The tasks are the units of work that a job executes. A task can be a notebook task, which runs a notebook with specified parameters; a jar task, which runs a JAR uploaded to DBFS with specified main class and arguments; or a python task, which runs a Python file uploaded to DBFS with specified parameters. A multi-task job is a job that has more than one task configured to run in a specific order or in parallel. By using the /jobs/get endpoint, one can review the notebooks configured to run as tasks in a multi-task job. Verified References: [Databricks Certified Data Engineer Professional], under "Databricks Jobs" section; Databricks Documentation, under "Get" section; Databricks Documentation, under "JobSettings" section.

NEW QUESTION: 59
A data engineering team is setting up deployment automation. To deploy workspace assets remotely using the Databricks CLI command, they must configure it with proper authentication.
Which authentication approach will provide the highest level of security ?
A. Use a shared user account and its OAuth client secret.
B. Use a service principal and its Personal Access Token.
C. Use a service principal ID and its OAuth client secret.
D. Use a service principal with OAuth token federation.
Answer: (SHOW ANSWER)

NEW QUESTION: 60

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



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: (SHOW ANSWER)

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: 61

How are the operational aspects of Lakeflow Declarative Pipelines different from Spark Structured Streaming ?

A. Lakeflow Declarative Pipelines manage the orchestration of multi-stage pipelines automatically, while Structured Streaming requires external orchestration for complex dependencies.
B. Structured Streaming can process continuous data streams, while Lakeflow Declarative Pipelines cannot.
C. Lakeflow Declarative Pipelines can write to Delta Lake format, while Structured Streaming cannot.
D. Lakeflow Declarative Pipelines automatically handle schema evolution, while Structured Streaming always requires manual schema management.

Answer: (SHOW ANSWER)

Databricks documentation explains that Lakeflow Declarative Pipelines build upon Structured Streaming but add higher-level orchestration and automation capabilities. They automatically manage dependencies, materialization, and recovery across multi-stage data flows without requiring external orchestration tools such as Airflow or Azure Data Factory. In contrast, Structured Streaming operates at a lower level, where developers must manually handle orchestration, retries, and dependencies between streaming jobs. Both support Delta Lake outputs and schema evolution; however, Lakeflow Declarative Pipelines simplify management by declaratively defining transformations and data quality expectations. Hence, the correct distinction is A - automated orchestration and management in Lakeflow Declarative Pipelines.

NEW QUESTION: 62
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. by noting the column statistics
E. In the Query Detail screen, by interpreting the Physical Plan
Answer: (SHOW ANSWER)
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: 63
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: (SHOW ANSWER)

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.
:
Databricks best practices for securing data: https://docs.databricks.com/security/index.html

NEW QUESTION: 64
A data engineer is designing a Lakeflow Declarative Pipeline to process streaming order data. The pipeline uses Auto Loader to ingest data and must enforce data quality by ensuring customer_id and amount are greater than zero. Invalid records should be dropped.
Which Lakeflow Declarative Pipelines configurations implement this requirement using Python?
A. @dlt.table
def silver_orders():
return (
dlt.read_stream(" bronze_orders ")
.expect_or_drop(" valid_customer " , " customer_id IS NOT NULL ")
.expect_or_drop(" valid_amount " , " amount > 0 ")
)
B. @dlt.table
@dlt.expect(" valid_customer " , " customer_id IS NOT NULL ")
@dlt.expect(" valid_amount " , " amount > 0 ")
def silver_orders():
return dlt.read_stream(" bronze_orders ")
C. @dlt.table
def silver_orders():
return (
dlt.read_stream(" bronze_orders ")
.expect(" valid_customer " , " customer_id IS NOT NULL ")
.expect(" valid_amount " , " amount > 0 ")
)
D. @dlt.table
@dlt.expect_or_drop(" valid_customer " , " customer_id IS NOT NULL ")
@dlt.expect_or_drop(" valid_amount " , " amount > 0 ")
def silver_orders():
return dlt.read_stream(" bronze_orders ")
Answer: (SHOW ANSWER)
Lakeflow Declarative Pipelines (LDP), formerly Delta Live Tables (DLT), supports enforcing data quality using expectations . Expectations can either:
* Track violations (expect) # records that do not meet conditions are flagged but still included in the pipeline.
* Drop violations (expect_or_drop) # records that do not meet conditions are excluded from downstream tables.
* Fail pipeline on violations (expect_or_fail) # records that fail conditions stop the pipeline.
In this scenario, the requirement explicitly states that invalid records (where customer_id is null or amount #

0) must be dropped . According to the official documentation, the correct method is .expect_or_drop(" expectation_name " , " SQL_predicate ") applied on the streaming input.
* Option A is correct: It uses .expect_or_drop directly within the transformation chain for both rules, ensuring records that fail are removed before writing to the silver table.
* Option B incorrectly uses @dlt.expect decorators, which only track violations but do not drop invalid rows.
* Option C uses .expect, which also only flags rows, not drop them.
* Option D uses @dlt.expect_or_drop decorator syntax, which is not supported in Python API; expect_or_drop must be applied as a method on the DataFrame, not as a decorator.
Therefore, the correct solution is Option A , which ensures compliance by enforcing data quality and dropping invalid rows programmatically during ingestion.
Reference: Databricks Lakeflow Declarative Pipelines Documentation - Expectations (expect, expect_or_drop, expect_or_fail)

NEW QUESTION: 65
A data engineer has configured their Databricks Asset Bundle with multiple targets in databricks.yml and deployed it to the production workspace. Now, to validate the deployment, they need to invoke a job named my_project_job specifically within the prod target context. Assuming the job is already deployed, they need to trigger its execution while ensuring the target-specific configuration is respected.
Which command will trigger the job execution?
A. databricks execute my_project_job -e prod
B. databricks job run my_project_job --env prod
C. databricks run my_project_job -t prod
D. databricks bundle run my_project_job -t prod
Answer: (SHOW ANSWER)
Databricks Asset Bundles (DABs) enable declarative configuration and deployment of Databricks resources such as jobs, pipelines, and dashboards across multiple environments.
Once deployed, jobs can be executed in a specific target context using the databricks bundle run command, which ensures all environment-specific configurations from the bundle definition (such as parameters, cluster settings, and workspace URLs) are respected.
The -t flag specifies the target environment (e.g., dev, staging, or prod). This ensures that the execution runs with the correct configuration defined under that target in databricks.yml.
Other options (A, B, and C) are invalid because they reference deprecated or incorrect command syntax that doesn't integrate with bundle targets. Therefore, D is the correct and verified answer.

NEW QUESTION: 66
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: (SHOW ANSWER)

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.
:
Delta Lake Documentation on Time Travel: Delta Time Travel
Delta Lake Versioning: Delta Lake Versioning Guide

NEW QUESTION: 67
A junior data engineer has been asked to develop a streaming data pipeline with a grouped aggregation using DataFrame df . The pipeline needs to calculate the average humidity and average temperature for each non- overlapping five-minute interval. Events are recorded once per minute per device.
Streaming DataFrame df has the following schema:
" device_id INT, event_time TIMESTAMP, temp FLOAT, humidity FLOAT "
Code block:

```
df.withWatermark("event_time", "10 minutes")
    .groupBy(
        _____,
        "device_id"
    )
    .agg(
        avg("temp").alias("avg_temp"),
        avg("humidity").alias("avg_humidity")
    )
    .writeStream
    .format("delta")
    .saveAsTable("sensor_avg")
```

Choose the response that correctly fills in the blank within the code block to complete this task.
A. to_interval(" event_time " , " 5 minutes ").alias(" time ")
B. window(" event_time " , " 5 minutes ").alias(" time ")
C. " event_time "
D. window(" event_time " , " 10 minutes ").alias(" time ")
E. lag(" event_time " , " 10 minutes ").alias(" time ")
Answer: (SHOW ANSWER)
This is the correct answer because the window function is used to group streaming data by time intervals. The window function takes two arguments: a time column and a window duration. The window duration specifies how long each window is, and must be a multiple of 1 second. In this case, the window duration is "5 minutes", which means each window will cover a non-overlapping five-minute interval. The window function also returns a struct column with two fields: start and end, which represent the

start and end time of each window. The alias function is used to rename the struct column as "time". Verified References: [Databricks Certified Data Engineer Professional], under "Structured Streaming" section; Databricks Documentation , under "WINDOW" section. https://www.databricks.com/blog/2017/05/08/event-time-aggregation- watermarking-apache-sparks-structured-streaming.html

NEW QUESTION: 68
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 approval 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 jon 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: (SHOW ANSWER)
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.
:
Databricks documentation on using SQL views in Delta Lake: https://docs.databricks.com/delta/quick-start.
html#sql-views

NEW QUESTION: 69
A facilities-monitoring team is building a near-real-time Power BI dashboard off the Delta table device_readings :
* device_id STRING - unique sensor ID
* event_ts TIMESTAMP - ingestion timestamp (UTC)
* temperature_c DOUBLE - temperature in °C
* notes STRING
For each sensor, the team needs one row per non-overlapping 5-minute interval, offset by 2 minutes (for example, intervals like 00:02-00:07 , 00:07-00:12 , and so on), showing the average temperature in that slice.
The result must include each interval's start and end timestamps so downstream tools can plot time-series bars correctly. Which query satisfies the requirement?
A. WITH buckets AS (
SELECT
device_id,
window(event_ts, ' 5 minutes ' , ' 5 minutes ' , ' 2 minutes ') AS win, temperature_c FROM device_readings) SELECT device_id, win.start AS bucket_start, win.end AS bucket_end, AVG(temperature_c) AS avg_temp_5m FROM buckets GROUP BY device_id, win ORDER BY device_id, bucket_start;
B. SELECT
device_id,
window.start AS bucket_start,
window.end AS bucket_end,
AVG(temperature_c) AS avg_temp_5m
FROM device_readings

GROUP BY device_id, window(event_ts, ' 5 minutes ' , ' 5 minutes ' , ' 2 minutes ') ORDER BY device_id, bucket_start;
C. SELECT
device_id,
date_trunc(' minute ' , event_ts - INTERVAL 2 MINUTES) + INTERVAL 2 MINUTES AS bucket_start, date_trunc(' minute ' , event_ts - INTERVAL 2 MINUTES) + INTERVAL 7 MINUTES AS bucket_end, AVG(temperature_c) AS avg_temp_5m FROM device_readings GROUP BY device_id, date_trunc(' minute ' , event_ts - INTERVAL 2 MINUTES) ORDER BY device_id, bucket_start;
D. SELECT
device_id,
event_ts,
AVG(temperature_c) OVER (
PARTITION BY device_id
ORDER BY event_ts
RANGE BETWEEN INTERVAL 5 MINUTES PRECEDING AND CURRENT ROW
) AS avg_temp_5m
FROM device_readings
WINDOW w AS (window(event_ts, ' 5 minutes ' , ' 2 minutes '));
Answer: (SHOW ANSWER)
Spark documents window(timeColumn, windowDuration, slideDuration=None, startTime=None) for time bucketing. The startTime argument is specifically the offset from the epoch used to align window boundaries, and the output is a window struct with start and end fields. That exactly matches the requirement for 5-minute non-overlapping intervals offset by 2 minutes. (Apache Spark) Option A correctly uses window(event_ts, ' 5 minutes ' , ' 5 minutes ' , ' 2 minutes ') , which creates tumbling 5-minute windows offset by 2 minutes and then exposes win.start and win.end . Option B is malformed in how it references the generated window column, option C creates minute-aligned groupings rather than true 5- minute tumbling windows, and option D computes a rolling window average instead of one row per non- overlapping time bucket. (Apache Spark)

NEW QUESTION: 70
A data engineer is configuring Delta Sharing for a Databricks-to-Databricks scenario to optimize read performance. The recipient needs to perform time travel queries and streaming reads on shared sales data.
Which configuration will provide the optimal performance while enabling these capabilities?

A. Share tables WITH HISTORY , ensure tables don't have partitioning enabled, and enable CDF before sharing.
B. Share tables WITHOUT HISTORY and enable partitioning for better query performance.
C. Share the entire schema WITHOUT HISTORY and rely on recipient-side caching for performance.
D. Use the open sharing protocol instead of Databricks-to-Databricks sharing for better performance.
Answer: (SHOW ANSWER)
The official Delta Sharing guidance specifies that in order for recipients to use time travel queries and streaming reads , providers must share Delta tables WITH HISTORY . Sharing history ensures the Delta log is included, which enables efficient access to table snapshots and incremental data streams. Additionally, Change Data Feed (CDF) must be enabled prior to sharing if downstream consumers require streaming CDC queries. Without history, recipients cannot perform time travel or streaming queries. Open sharing supports static Delta tables but lacks streaming support. Therefore, sharing tables WITH HISTORY and enabling CDF is the required configuration for both performance and functionality.

NEW QUESTION: 71

A company wants to implement Lakehouse Federation across multiple data sources but is concerned about data consistency and ensuring that all teams access the same authoritative version of their data.
Which statement is applicable for Lakehouse Federations to maintain data consistency?
A. Federation provides read-only access that reflects the current state of source systems.
B. Federation implements change data capture (CDC) from all sources.
C. A separate data synchronization service must be deployed.
D. Federation creates local copies that must be manually refreshed.
Answer: (SHOW ANSWER)
Lakehouse Federation allows Databricks to query and manage external data sources through a single governance layer, without moving or copying data . The documentation specifies that "Federated queries provide read-only access to data, reflecting the current state of the underlying source system." This ensures consistency across teams since all users access the same source of truth directly from the external system through Unity Catalog. Federation does not perform CDC replication or local caching; it queries live data on demand. Hence, option A accurately represents how Lakehouse Federation maintains consistency across federated sources.

NEW QUESTION: 72
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. Executor's log file
E. Cluster Event Log
Answer: (SHOW ANSWER)

NEW QUESTION: 73
A query is taking too long to run. After investigating the Spark UI, the data engineer discovered a significant amount of disk spill . The compute instance being used has a core-to-memory ratio of 1:2.
What are the two steps the data engineer should take to minimize spillage? (Choose 2 answers)
A. Choose a compute instance with a higher core-to-memory ratio.
B. Choose a compute instance with more disk space.
C. Increase spark.sql.files.maxPartitionBytes.
D. Reduce spark.sql.files.maxPartitionBytes.
E. Choose a compute instance with more network bandwidth.
Answer: (SHOW ANSWER)
Databricks recommends addressing disk spilling -which occurs when Spark tasks run out of memory-by increasing memory per core and controlling partition size. Selecting an instance type with a higher memory- to-core ratio (A) provides each task with more available RAM, directly reducing the chance of spilling to disk. Additionally, reducing spark.sql.files.maxPartitionBytes (D) creates smaller partitions, preventing any single task from holding too much data in memory. Increasing partition size (C) or disk capacity (B) does not solve memory bottlenecks, and bandwidth (E) affects network I/O, not spill behavior. Therefore, the correct actions are A and D .

NEW QUESTION: 74

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 monitizable clicks.

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

Answer: (SHOW ANSWER)

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: 75

A data engineer wants to automate job monitoring and recovery in Databricks using the Jobs API. They need to list all jobs, identify a failed job, and rerun it.

Which sequence of API actions should the data engineer perform?

A. Use the jobs/list endpoint to list jobs, check job run statuses with jobs/runs/list, and rerun a failed job using jobs/run-now.

B. Use the jobs/get endpoint to retrieve job details, then use jobs/update to rerun failed jobs.

C. Use the jobs/list endpoint to list jobs, then use the jobs/create endpoint to create a new job, and run the new job using jobs/run-now.

D. Use the jobs/cancel endpoint to remove failed jobs, then recreate them with jobs/create and run the new ones.

Answer: (SHOW ANSWER)

The Databricks Jobs REST API provides several endpoints for automation. The correct monitoring and rerun flow uses three specific calls:

* GET /api/2.1/jobs/list - Lists all available jobs within the workspace.

* GET /api/2.1/jobs/runs/list - Returns all runs for a specific job, including their current state (e.g., TERMINATED: FAILED).

* POST /api/2.1/jobs/run-now - Immediately triggers a rerun of the specified job.

This sequence aligns with Databricks' prescribed automation model for job observability and recovery. Using jobs/update modifies metadata but does not rerun jobs, and jobs/create is only used for creating new jobs, not rerunning failed ones. Cancelling and recreating jobs introduces unnecessary duplication. Therefore, option A is the correct automated recovery workflow.

NEW QUESTION: 76

A junior data engineer is migrating a workload from a relational database system to the Databricks Lakehouse. The source system uses a star schema, leveraging foreign key constrains and multi-table inserts to validate records on write.
Which consideration will impact the decisions made by the engineer while migrating this workload?
A. All Delta Lake transactions are ACID compliance against a single table, and Databricks does not enforce foreign key constraints.
B. Databricks only allows foreign key constraints on hashed identifiers, which avoid collisions in highly- parallel writes.
C. Foreign keys must reference a primary key field; multi-table inserts must leverage Delta Lake ' s upsert functionality.
D. Committing to multiple tables simultaneously requires taking out multiple table locks and can lead to a state of deadlock.
Answer: (SHOW ANSWER)
In Databricks and Delta Lake, transactions are indeed ACID-compliant, but this compliance is limited to single table transactions. Delta Lake does not inherently enforce foreign key constraints, which are a staple in relational database systems for maintaining referential integrity between tables. This means that when migrating workloads from a relational database system to Databricks Lakehouse, engineers need to reconsider how to maintain data integrity and relationships that were previously enforced by foreign key constraints.
Unlike traditional relational databases where foreign key constraints help in maintaining the consistency across tables, in Databricks Lakehouse, the data engineer has to manage data consistency and integrity at the application level or through careful design of ETL processes.
:
Databricks Documentation on Delta Lake: Delta Lake Guide
Databricks Documentation on ACID Transactions in Delta Lake: ACID Transactions in Delta Lake

NEW QUESTION: 77
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. Theuser_idfield 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 namedaccount_historywhich maintains a full record of all data in the same schema as the source. The next table in the system is namedaccount_currentand is implemented as a Type 1 table representing the most recent value for each uniqueuser_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 describedaccount_currenttable as part of each hourly batch job?
A. Use Auto Loader to subscribe to new files in the account history directory; configure a Structured Streaminq 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 iogin 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: (SHOW ANSWER)
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: 78
Review the following error traceback:

```
AnalysisException                         Traceback (most recent call last)
<command-3293767849433948> in <module>
----> 1 display(df.select(3*"heartrate"))

/databricks/spark/python/pyspark/sql/dataframe.py in select(self, *cols)
   1690         [Row(name='Alice', age=12), Row(name='Bob', age=15)]
   1691         """
-> 1692         jdf = self._jdf.select(self._jcols(*cols))
   1693         return DataFrame(jdf, self.sql_ctx)
   1694

/databricks/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py in __call__(self, *args)
   1302
   1303         answer = self.gateway_client.send_command(command)
-> 1304         return_value = get_return_value(
   1305             answer, self.gateway_client, self.target_id, self.name)
   1306

/databricks/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
    121                 # Hide where the exception came from that shows a non-Pythonic
    122                 # JVM exception message.
--> 123                 raise converted from None
    124             else:
    125                 raise

AnalysisException: cannot resolve '`heartrateheartrateheartrate`' given input columns:
[spark_catalog.database.table.device_id, spark_catalog.database.table.heartrate,
spark_catalog.database.table.mrn, spark_catalog.database.table.time];
'Project ['heartrateheartrateheartrate]
+- SubqueryAlias spark_catalog.database.table
   +- Relation[device_id#75L,heartrate#76,mrn#77L,time#78] parquet
```

Which statement describes the error being raised?

A. The code executed was PvSoark but was executed in a Scala notebook.

B. There is no column in the table named heartrateheartrateheartrate

C. There is a type error because a column object cannot be multiplied.
D. There is a type error because a DataFrame object cannot be multiplied.
E. There is a syntax error because the heartrate column is not correctly identified as a column.
Answer: (SHOW ANSWER)
The error being raised is an AnalysisException, which is a type of exception that occurs when Spark SQL cannot analyze or execute a query due to some logical or semantic error 1 . In this case, the error message indicates that the query cannot resolve the column name 'heartrateheartrateheartrate' given the input columns 'heartrate' and 'age'. This means that there is no column in the table named 'heartrateheartrateheartrate', and the query is invalid. A possible cause of this error is a typo or a copy-paste mistake in the query. To fix this error, the query should use a valid column name that exists in the table, such as 'heartrate'. References: AnalysisException

NEW QUESTION: 79
A data engineer needs to install the PyYAML Python package within an air-gapped Databricks environment . The workspace has no direct internet access to PyPI. The engineer has downloaded the .whl file locally and wants it available automatically on all new clusters.
Which approach should the data engineer use?
A. Upload the PyYAML .whl file to the user home directory and create a cluster-scoped init script to install it.
B. Upload the PyYAML .whl file to a Unity Catalog Volume, ensure it's allow-listed, and create a cluster-scoped init script that installs it from that path.
C. Set up a private PyPI repository and install via pip index URL.
D. Add the .whl file to Databricks Git Repos and assume automatic installation.
Answer: (SHOW ANSWER)
For secure, air-gapped Databricks deployments , the recommended practice is to host dependency files such as .whl packages in Unity Catalog Volumes - a managed storage layer governed by Unity Catalog.
Once stored in a volume, these files can be safely referenced from cluster-scoped init scripts , which automatically execute installation commands (e.g., pip install /Volumes/catalog/schema/path/PyYAML.whl) during cluster startup.
This ensures consistent environment setup across clusters and compliance with data governance rules.
User directories (A) lack enterprise security controls; private repositories (C) are not viable in air-gapped setups; and Git repos (D) do not trigger package installation.
Therefore, B is the correct and officially approved method.

NEW QUESTION: 80
The data architect has decided that once data has been ingested from external sources into the Databricks Lakehouse, table access controls will be leveraged to manage permissions for all production tables and views.
The following logic was executed to grant privileges for interactive queries on a production database to the core engineering group.
GRANT USAGE ON DATABASE prod TO eng;
GRANT SELECT ON DATABASE prod TO eng;
Assuming these are the only privileges that have been granted to the eng group and that these users are not workspace administrators, which statement describes their privileges?

A. Group members have full permissions on the prod database and can also assign permissions to other users or groups.
B. Group members are able to list all tables in the prod database but are not able to see the results of any queries on those tables.
C. Group members are able to query and modify all tables and views in the prod database, but cannot create new tables or views.
D. Group members are able to query all tables and views in the prod database, but cannot create or edit anything in the database.
E. Group members are able to create, query, and modify all tables and views in the prod database, but cannot define custom functions.

Answer: (SHOW ANSWER)
The GRANT USAGE ON DATABASE prod TO eng command grants the eng group the permission to use the prod database, which means they can list and access the tables and views in the database. The GRANT SELECT ON DATABASE prod TO eng command grants the eng group the permission to select data from the tables and views in the prod database, which means they can query the data using SQL or DataFrame API.
However, these commands do not grant the eng group any other permissions, such as creating, modifying, or deleting tables and views, or defining custom functions. Therefore, the eng group members are able to query all tables and views in the prod database, but cannot create or edit anything in the database. References:
* Grant privileges on a database: https://docs.databricks.com/en/security/auth-authz/table-acls/grant- privileges-database.html
* Privileges you can grant on Hive metastore objects: https://docs.databricks.com/en/security/auth-authz
/table-acls/privileges.html

NEW QUESTION: 81
An upstream system has been configured to pass the date for a given batch of data to the Databricks Jobs API as a parameter. The notebook to be scheduled will use this parameter to load data with the following code:
df = spark.read.format("parquet").load(f"/mnt/source/(date)")
Which code block should be used to create the date Python variable used in the above code block?
A. date = spark.conf.get("date")
B. input_dict = input()
date= input_dict["date"]
C. import sys
date = sys.argv[1]
D. date = dbutils.notebooks.getParam("date")
E. dbutils.widgets.text("date", "null")
date = dbutils.widgets.get("date")
Answer: (SHOW ANSWER)
The code block that should be used to create the date Python variable used in the above code block is:
dbutils.widgets.text("date", "null") date = dbutils.widgets.get("date") This code block uses the dbutils.widgets API to create and get a text widget named "date" that can accept a string value as a parameter1. The default value of the widget is "null", which means that if no parameter is passed, the date variable will be "null". However, if a parameter is passed through the Databricks Jobs API, the date variable will be assigned the value of the parameter. For example, if the parameter is "2021-11-01", the date variable will be "2021-11-01". This way, the notebook can use the date variable to load data from the specified path.
The other options are not correct, because:
* Option A is incorrect because spark.conf.get("date") is not a valid way to get a parameter passed through the Databricks Jobs API. The spark.conf API is used to get or set Spark configuration properties, not notebook parameters2.
* Option B is incorrect because input() is not a valid way to get a parameter passed through the Databricks Jobs API. The input() function is used to get user input from the standard input stream, not from the API request3.
* Option C is incorrect because sys.argv1 is not a valid way to get a parameter passed through the Databricks Jobs API. The sys.argv list is used to get the command-line arguments passed to a Python script, not to a notebook4.
* Option D is incorrect because dbutils.notebooks.getParam("date") is not a valid way to get a parameter passed through the Databricks Jobs API. The dbutils.notebooks API is used to get or set notebook parameters when running a notebook as a job or as a subnotebook, not when passing parameters through the API5.
References: Widgets, Spark Configuration, input(), sys.argv, Notebooks

NEW QUESTION: 82

A data engineer has created a new cluster using shared access mode with default configurations. The data engineer needs to allow the development team access to view the driver logs if needed.
What are the minimal cluster permissions that allow the development team to accomplish this?
A. CAN ATTACH TO
B. CAN MANAGE
C. CAN VIEW
D. CAN RESTART
Answer: (SHOW ANSWER)
Databricks provides different permission levels to control access to clusters. The correct minimal permission required for viewing driver logs is CAN VIEW .
Databricks Cluster Permission Levels:
* CAN ATTACH TO:
* Allows users to attach notebooks to a cluster but does not allow them to view logs .
* Not sufficient for viewing driver logs.
* CAN MANAGE:
* Grants full control over the cluster, including starting, stopping, and editing configurations.
* Too broad for this requirement .
* CAN VIEW (Correct Answer):
* Allows users to view cluster details, logs, and status but not modify any configurations.
* Minimal required permission for viewing logs .
* CAN RESTART:
* Grants permission to restart the cluster , but does not include log access .
* Not sufficient for viewing logs.
Conclusion:
The minimal permission needed to allow the development team to view driver logs is CAN VIEW .
References:
Databricks Cluster Permissions Documentation

NEW QUESTION: 83
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: (SHOW ANSWER)
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.: Databricks documentation on access control and permissions for notebooks within the workspace (https://docs.databricks.com/security/access-control/workspace-acl.html).

NEW QUESTION: 84
A data team is automating a daily multi-task ETL pipeline in Databricks. The pipeline includes a notebook for ingesting raw data, a Python wheel task for data transformation, and a SQL query to update aggregates. They want to trigger the pipeline programmatically and see previous runs in the GUI. They need to ensure tasks are retried on failure and stakeholders are notified by email if any task fails.
Which two approaches will meet these requirements? (Choose 2 answers)
A. Use the REST API endpoint /jobs/runs/submit to trigger each task individually as separate job runs and implement retries using custom logic in the orchestrator.
B. Create a multi-task job using the UI, Databricks Asset Bundles (DABs), or the Jobs REST API (/jobs
/create) with notebook, Python wheel, and SQL tasks. Configure task-level retries and email notifications in the job definition.
C. Trigger the job programmatically using the Databricks Jobs REST API (/jobs/run-now), the CLI (databricks jobs run-now), or one of the Databricks SDKs.
D. Create a single orchestrator notebook that calls each step with dbutils.notebook.run(), defining a job for that notebook and configuring retries and notifications at the notebook level.
E. Use Databricks Asset Bundles (DABs) to deploy the workflow, then trigger individual tasks directly by referencing each task's notebook or script path in the workspace.
Answer: (SHOW ANSWER)
Databricks Jobs supports defining multi-task workflows that include notebooks, SQL statements, and Python wheel tasks. These can be configured with retry policies, dependency chains, and failure notifications. The correct practice, as stated in the documentation, is to use the Jobs REST API (/jobs/create) or Databricks Asset Bundles to define multi-task jobs, and then trigger them programmatically using /jobs/run-now, CLI, or SDK. This allows the team to maintain full job history, handle retries automatically, and receive alerts via configured email notifications. Using /jobs/runs/submit creates one-off ad hoc runs without maintaining dependency visibility. Therefore, options B and C together satisfy the operational, automation, and governance requirements.

NEW QUESTION: 85
A facilities-monitoring team is building a near-real-time PowerBI dashboard off the Delta table device_readings:
Columns:
* device_id (STRING, unique sensor ID)
* event_ts (TIMESTAMP, ingestion timestamp UTC)
* temperature_c (DOUBLE, temperature in °C)
Requirement:
* For each sensor, generate one row per non-overlapping 5-minute interval , offset by 2 minutes (e.g.,
00:02-00:07, 00:07-00:12, ...).
* Each row must include interval start, interval end, and average temperature in that slice.
* Downstream BI tools (e.g., Power BI) must use the interval timestamps to plot time-series bars.
Options:
A. WITH buckets AS (
SELECT device_id,
window(event_ts, ' 5 minutes ' , ' 2 minutes ' , ' 5 minutes ') AS win, temperature_c FROM device_readings) SELECT device_id, win.start AS bucket_start, win.end AS bucket_end, AVG(temperature_c) AS avg_temp_5m FROM buckets GROUP BY device_id, win ORDER BY device_id, bucket_start;
B. SELECT device_id,
event_ts,
AVG(temperature_c) OVER (
PARTITION BY device_id
ORDER BY event_ts

RANGE BETWEEN INTERVAL 5 MINUTES PRECEDING AND CURRENT ROW
) AS avg_temp_5m
FROM device_readings
WINDOW w AS (window(event_ts, ' 5 minutes ' , ' 2 minutes '));
C. SELECT device_id,
date_trunc(' minute ' , event_ts - INTERVAL 2 MINUTES) + INTERVAL 2 MINUTES AS bucket_start, date_trunc(' minute ' , event_ts - INTERVAL 2 MINUTES) + INTERVAL 7 MINUTES AS bucket_end, AVG(temperature_c) AS avg_temp_5m FROM device_readings GROUP BY device_id, date_trunc(' minute ' , event_ts - INTERVAL 2 MINUTES) ORDER BY device_id, bucket_start;
D. SELECT device_id,
window.start AS bucket_start,
window.end AS bucket_end,
AVG(temperature_c) AS avg_temp_5m
FROM device_readings
GROUP BY device_id, window(event_ts, ' 5 minutes ' , ' 5 minutes ' , ' 2 minutes ') ORDER BY device_id, bucket_start;
Answer: (SHOW ANSWER)
The correct way to satisfy non-overlapping windows with an offset in Databricks SQL is to use the window function with three parameters: window duration, slide duration, and start offset .
* In option A , the function call:
* window(event_ts, ' 5 minutes ' , ' 2 minutes ' , ' 5 minutes ')
creates 5-minute windows that slide every 5 minutes , with a 2-minute offset , which exactly matches the requirement (intervals like 00:02-00:07, 00:07-00:12, ...).
* Option B is incorrect because it uses a windowed aggregation with RANGE , which produces overlapping sliding averages , not discrete non-overlapping buckets.
* Option C manually constructs bucket boundaries with date_trunc and offsets, but this is brittle and less efficient than the built-in window function.
* Option D incorrectly passes four parameters to window but with the wrong ordering (5 minutes, 5 minutes, 2 minutes). This creates a sliding window every 5 minutes with overlap , rather than true non-overlapping shifted windows.
Reference (Databricks SQL Windowing Functions):
Databricks documentation specifies that:
window(time_col, windowDuration, slideDuration, startTime)
produces tumbling or sliding windows . When slideDuration = windowDuration, it produces non- overlapping tumbling windows . The startTime argument allows for offset windows , which is why ' 2 minutes ' ensures alignment at 00:02, 00:07, etc.
Thus, A is the only correct solution as it directly implements non-overlapping, offset-based tumbling windows .

NEW QUESTION: 86
A data engineer is creating a daily reporting job. There are two reporting notebooks-one for weekdays and one for weekends. An "if/else condition" task is configured as {{job.start_time.is_weekday}} == true to route the job to either the weekday or weekend notebook tasks. The same job would be used across multiple time zones.
Which action should a senior data engineer take upon reviewing the job to merge or reject the pull request?
A. Reject, as the {{job.start_time.is_weekday}} is for the UTC timezone .
B. Reject, as the {{job.start_time.is_weekday}} is not a valid value reference.
C. Merge, as the job configuration looks good.
D. Reject, as they should use {{job.trigger_time.is_weekday}} instead.
Answer: (SHOW ANSWER)

Databricks parameter templates like {{job.start_time.is_weekday}} evaluate in UTC time by default, not in local workspace or regional time zones. Therefore, when jobs are configured to run across different time zones, relying on is_weekday using UTC may cause scheduling and task routing mismatches (for example, triggering the weekday notebook in one region while it's still the weekend locally).
Databricks recommends adjusting conditional logic or pipeline parameters explicitly to handle time zone conversions if business requirements depend on local times.
Because the engineer's configuration does not account for this behavior, a senior data engineer should reject the pull request and suggest time-zone-aware logic before merging.

NEW QUESTION: 87
A Data Engineer is building a simple data pipeline using Lakeflow Declarative Pipelines (LDP) in Databricks to ingest customer data. The raw customer data is stored in a cloud storage location in JSON format. The task is to create Lakeflow Declarative Pipelines that read the raw JSON data and write it into a Delta table for further processing.
Which code snippet will correctly ingest the raw JSON data and create a Delta table using LDP?
A. import dlt
@dlt.table
def raw_customers():
return spark.read.format(" csv ").load(" s3://my-bucket/raw-customers/ ")
B. import dlt
@dlt.table
def raw_customers():
return spark.read.json(" s3://my-bucket/raw-customers/ ")
C. import dlt
@dlt.table
def raw_customers():
return spark.read.format(" parquet ").load(" s3://my-bucket/raw-customers/ ")
D. import dlt
@dlt.view
def raw_customers():
return spark.format.json(" s3://my-bucket/raw-customers/ ")
Answer: (SHOW ANSWER)
The correct method to define a table using Lakeflow Declarative Pipelines (LDP) is with the @dlt.table decorator, which persists the output as a managed Delta table.
When ingesting raw JSON data, spark.read.
json() or spark.read.format(" json ").load() is the standard approach. This reads JSON-formatted files from the source and stores them in Delta format automatically managed by Databricks.
Reference Source: Databricks Lakeflow Declarative Pipelines Developer Guide - "Create tables from raw JSON and Delta sources."

NEW QUESTION: 88
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: (SHOW ANSWER)

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 and Governance" section; Databricks Documentation, under
"Secrets" section.

NEW QUESTION: 89

A healthcare analytics team is implementing a dimensional model in Delta Lake for patient care analysis.

They have a date dimension table and are evaluating design options to ensure it supports a wide range of time- based analyses.

Which design approach for the date dimension will support efficient time-based querying and aggregation?

A. Store the date as a string in the format YYYY-MM-DD for readability.

B. Create separate dimension tables for different calendar systems (fiscal, academic, etc.).

C. Store only the date value and calculate all time attributes dynamically in queries.

D. Pre-calculate attributes like fiscal_period, quarter, month_name, day_of_week, and holiday.

Answer: (SHOW ANSWER)

In dimensional modeling, Databricks recommends denormalized, attribute-rich dimension tables for performance and usability. A date dimension should include all commonly used derived time attributes such as fiscal period, quarter, month, weekday, and holiday flags. Precomputing these attributes ensures consistent business logic, eliminates repeated calculations during query time, and enables efficient filtering and aggregation. The documentation for Delta Lake and Lakehouse design explicitly advises precomputing these attributes for analytical workloads that depend heavily on time-based slicing. Options A and C degrade performance and consistency, while maintaining multiple calendar-specific dimension tables (B) complicates the model unnecessarily.

NEW QUESTION: 90

A data engineer wants to reflector 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)
```

databricks

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():
```

databricks

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: (SHOW ANSWER)

The issue with the refactored code is that it tries to use string interpolation to dynamically create table names within the dlc.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.

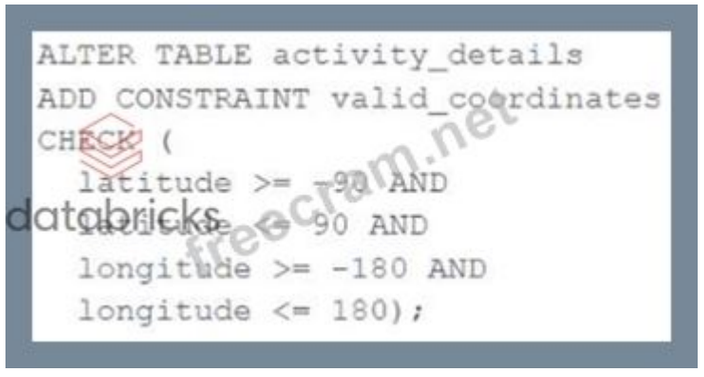
NEW QUESTION: 91

A Delta Lake table was created with the below query:

```
CREATE TABLE prod.sales_by_store
AS (
  SELECT *
  FROM prod.sales a
  INNER JOIN prod.store b
  ON a.store_id = b.store_id
)
```


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: (SHOW ANSWER)
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: 92
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: (SHOW ANSWER)

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: 93

A workspace admin has created a new catalog called finance_data and wants to delegate permission management to a finance team lead without giving them full admin rights.

Which privilege should be granted to the finance team lead?

A. ALL PRIVILEGES on the finance_data catalog.

B. Make the finance team lead a metastore admin.

C. GRANT OPTION privilege on the finance_data catalog.

D. MANAGE privilege on the finance_data catalog.

Answer: (SHOW ANSWER)

The MANAGE privilege in Unity Catalog provides the ability to grant and revoke privileges on the specified object (in this case, a catalog) without giving full administrative access or ownership.

This is the Databricks-recommended approach for delegating governance responsibilities while preserving the principle of least privilege .

By contrast, the ALL PRIVILEGES option grants excessive access (including read and write permissions), and metastore admin status provides global control over all catalogs-far exceeding the requirement. The MANAGE privilege enables the finance team lead to control access to objects within finance_data responsibly while limiting overall administrative exposure.

NEW QUESTION: 94

A junior data engineer has been asked to develop a streaming data pipeline with a grouped aggregation using DataFrame df. The pipeline needs to calculate the average humidity and average temperature for each non- overlapping five-minute interval. Incremental state information should be maintained for 10 minutes for late- arriving data.
Streaming DataFrame df has the following schema:
" device_id INT, event_time TIMESTAMP, temp FLOAT, humidity FLOAT "
Code block:

```
df._____
    .groupBy(
        window("event_time", "5 minutes").alias("time"),
        "device_id"
    )
    .agg(
        avg("temp").alias("avg_temp"),
        avg("humidity").alias("avg_humidity")
    )
    .writeStream
    .format("delta")
    .saveAsTable("sensor_avg")
```


Choose the response that correctly fills in the blank within the code block to complete this task.
A. withWatermark(" event_time " , " 10 minutes ")
B. awaitArrival(" event_time " , " 10 minutes ")
C. await(" event_time + '10 minutes ' ")
D. slidingWindow(" event_time " , " 10 minutes ")
E. delayWrite(" event_time " , " 10 minutes ")
Answer: A (LEAVE A REPLY)
The correct answer is A. withWatermark("event_time", "10 minutes"). This is because the question asks for incremental state information to be maintained for 10 minutes for late-arriving data. The withWatermark method is used to define the watermark for late data. The watermark is a timestamp column and a threshold that tells the system how long to wait for late data. In this case, the watermark is set to 10 minutes. The other options are incorrect because they are not valid methods or syntax for watermarking in Structured Streaming. References:
* Watermarking : https://docs.databricks.com/spark/latest/structured-streaming/watermarks.html
* Windowed aggregations : https://docs.databricks.com/spark/latest/structured-streaming/window-operations.html

NEW QUESTION: 95
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: (SHOW ANSWER)

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: 96
A data engineer is creating a data ingestion pipeline to understand where customers are taking their rented bicycles during use. The engineer noticed that, over time, data being transmitted from the bicycle sensors fail to include key details like latitude and longitude. Downstream analysts need both the clean records and the quarantined records available for separate processing.
The data engineer already has this code:
import dlt
from pyspark.sql.functions import expr
rules = {
" valid_lat " : " (lat IS NOT NULL) " ,
" valid_long " : " (long IS NOT NULL) "
}
quarantine_rules = " NOT({}) " .format(" AND " .join(rules.values()))
@dlt.view
def raw_trips_data():
return spark.readStream.table(" ride_and_go.telemetry.trips ")
How should the data engineer meet the requirements to capture good and bad data?
A. @dlt.table(name= " trips_data_quarantine ")
def trips_data_quarantine():
return (
spark.readStream.table(" raw_trips_data ")
.filter(expr(quarantine_rules))
)
B. @dlt.view
@dlt.expect_or_drop(" lat_long_present " , " (lat IS NOT NULL AND long IS NOT NULL) ") def trips_data_quarantine():
return spark.readStream.table(" ride_and_go.telemetry.trips ")
C. @dlt.table
@dlt.expect_all_or_drop(rules)
def trips_data_quarantine():
return spark.readStream.table(" raw_trips_data ")
D. @dlt.table(partition_cols=[" is_quarantined " ,])
@dlt.expect_all(rules)
def trips_data_quarantine():
return (

spark.readStream.table(" raw_trips_data ")
.withColumn(" is_quarantined " , expr(quarantine_rules))
)

Answer: (SHOW ANSWER)

The requirement is that both valid (good) and invalid (bad) records must be captured and available separately for downstream processing. Invalid records should not simply be dropped; they must be quarantined in a dedicated table.

In Databricks Lakeflow Declarative Pipelines (DLT) , this is achieved by creating separate output tables :

* One table for valid records (Silver table) that pass the expectations.
* Another quarantine table that explicitly captures records failing the expectations.

Option A correctly implements this by:

* Declaring a DLT table trips_data_quarantine.
* Using .filter(expr(quarantine_rules)) to isolate invalid records (records where latitude or longitude is NULL).
* This ensures analysts can query both good records (from the main Silver pipeline table) and bad records (from the quarantine table).

Why not the others?

* B : Uses @dlt.expect_or_drop, which drops invalid records instead of quarantining them. This violates the requirement that quarantined data should be available.
* C : Same as B, but applies expectations in bulk with expect_all_or_drop. Again, bad data is dropped, not quarantined.
* D : Adds an is_quarantined flag in the same table. While it marks bad records, it does not separate them into a distinct quarantine table as required by the business use case.

Therefore, Option A is the only solution aligned with Databricks documentation for quarantining invalid data into a dedicated table while keeping valid data in the main pipeline.

NEW QUESTION: 97

A data engineering team uses Databricks Lakehouse Monitoring to track the percent_null metric for a critical column in their Delta table. The profile metrics table (prod_catalog.prod_schema.
customer_data_profile_metrics) stores hourly percent_null values. The team wants to trigger an alert when the daily average of percent_null exceeds 5% for three consecutive days, while ensuring notifications are not spammed during sustained issues. Which SQL alert configuration achieves this goal while minimizing false positives and redundant notifications?

A. WITH daily_avg AS (
SELECT
DATE_TRUNC(' DAY ' , window.end) AS day,
AVG(percent_null) AS avg_null
FROM prod_catalog.prod_schema.customer_data_profile_metrics
GROUP BY DATE_TRUNC(' DAY ' , window.end)
)
SELECT day, avg_null
FROM daily_avg
ORDER BY day DESC
LIMIT 3
Alert Condition: ALL avg_null > 5 for the latest 3 rows
Notification Frequency: Just once

B. SELECT percent_null

FROM prod_catalog.prod_schema.customer_data_profile_metrics
WHERE window.end > = CURRENT_TIMESTAMP - INTERVAL ' 1 ' DAY
Alert Condition: percent_null > 5
Notification Frequency: At most every 24 hours
C. SELECT AVG(percent_null) AS daily_avg
FROM prod_catalog.prod_schema.customer_data_profile_metrics
WHERE window.end > = CURRENT_TIMESTAMP - INTERVAL ' 3 ' DAY
Alert Condition: daily_avg > 5
Notification Frequency: Each time alert is evaluated
D. SELECT SUM(CASE WHEN percent_null > 5 THEN 1 ELSE 0 END) AS violation_days FROM prod_catalog.prod_schema.customer_data_profile_metrics WHERE window.end > = CURRENT_TIMESTAMP - INTERVAL ' 3 ' DAY Alert Condition: violation_days > = 3 Notification Frequency: Just once
Answer: (SHOW ANSWER)
Databricks SQL alerts support alert conditions over aggregated query results, including AVG , and they support notification-frequency behavior that avoids repeated alerts during a sustained triggered state.
Databricks documents that with Just Once , a notification is sent when the alert changes from OK to TRIGGERED , but not repeatedly while it remains triggered.
(Databricks Documentation) Option A is the only choice that correctly computes a daily average, checks the latest three daily rows, and pairs that with the anti-spam Just Once notification behavior. Option B checks only raw hourly values over one day, option C averages across the entire three-day span rather than requiring three consecutive daily breaches, and option D counts hourly threshold violations instead of true daily-average violations. (Databricks Documentation)
======
NEW QUESTION: 98
A data engineer is attempting to execute the following PySpark code:
df = spark.read.table(" sales ")
result = df.groupBy(" region ").agg(sum(" revenue "))
However, upon inspecting the execution plan and profiling the Spark job, they observe excessive data shuffling during the aggregation phase.
Which technique should be applied to reduce shuffling during the groupBy aggregation operation?
A. Caching the DataFrame df.
B. Repartition by region before aggregation.
C. Use coalesce() after the aggregation.
D. Use broadcast join.
Answer: (SHOW ANSWER)
Databricks documents that shuffle occurs when Spark redistributes data across partitions for grouping or joining. To optimize aggregation performance, repartitioning by the grouping key (region) ensures rows with the same key are co-located in the same partition, thus minimizing shuffle movement. Caching improves reuse of DataFrames but does not reduce shuffle volume. coalesce() reduces the number of partitions after computation and cannot prevent shuffle. Broadcast joins are unrelated to single-table aggregations. The recommended practice for reducing shuffle in aggregation is explicit repartitioning by the grouping column.

NEW QUESTION: 99
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: (SHOW ANSWER)

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: 100

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

A. pylib.source

B. pypi.path

C. importlib.resource path

D. os-path

E. ,sys.path

Answer: (SHOW ANSWER)

NEW QUESTION: 101

A platform team is creating a standardized template for Databricks Asset Bundles to support CI/CD. The template must specify defaults for artifacts, workspace root paths, and a run identity, while allowing a "dev" target to be the default and override specific paths.
How should the team use databricks.yml to satisfy these requirements?

A. Use deployment, builds, context, identity, and environments; set dev as default environment and override paths under builds.

B. Use roots, modules, profiles, actor, and targets; where profiles contain workspace and artifacts defaults and actor sets run identity.

C. Use project, packages, environment, identity, and stages; set dev as default stage and override workspace under environment.
D. Use bundle, artifacts, workspace, run_as, and targets at the top level; set one target with default: true and override workspace paths or artifacts under that target.
Answer: (SHOW ANSWER)
In Databricks Asset Bundles, the databricks.yml file defines all top-level configuration keys, including bundle, artifacts, workspace, run_as, and targets. The targets section defines specific deployment contexts (for example, dev, test, prod). Setting default: true for a target marks it as the default environment. Overrides for workspace paths and artifact configurations can be defined inside each target while keeping defaults at the top level.
Reference Source: Databricks Asset Bundle Configuration Guide - "Structure of databricks.yml and target overrides."

NEW QUESTION: 102
A data engineer needs to implement column masking for a sensitive column in a Unity Catalog-managed table. The masking logic must dynamically check if users belong to specific groups defined in a separate table (group_access) that maps groups to allowed departments.
Which approach should the engineer use to efficiently enforce this requirement?
A. Create a UDF that hardcodes allowed groups and apply it as a column mask.
B. Create a view without selecting the sensitive column.
C. Apply a column mask that references the group_access mapping table in its UDF.
D. Use a row filter to restrict access based on the user's group.
Answer: (SHOW ANSWER)
Databricks Unity Catalog supports dynamic column masking , where masking logic can be implemented using SQL functions or UDFs that reference external mapping tables or metadata for context-aware access control.
By referencing the group_access table inside the masking function, the mask dynamically evaluates whether a requesting user belongs to an authorized group. If permitted, the original column value is returned; otherwise, a masked value (such as NULL or asterisks) is shown.
This method enables fine-grained, data-driven masking policies while maintaining a single authoritative access mapping source.
Hardcoding values (A) reduces flexibility, and row filters (D) apply to entire rows rather than specific columns. Therefore, C correctly aligns with Databricks best practices for dynamic masking.

NEW QUESTION: 103
Given the following PySpark code snippet in a Databricks notebook:
filtered_df = spark.read.format(" delta ").load(" /mnt/data/large_table ") \
.filter(" event_date > ' 2024-01-01 ' ")
filtered_df.count()
The data engineer notices from the Query Profiler that the scan operator for filtered_df is reading almost all files, despite the filter being applied.
What is the probable reason for poor data skipping?
A. The Delta table lacks optimization that enables dynamic file pruning.
B. The filter is executed only after the full data scan, preventing data skipping.
C. The event_date column is outside the table's partitioning and Z-ordering scheme.
D. The filter condition involves a data type excluded from data skipping support.
Answer: (SHOW ANSWER)
Delta Lake's data skipping and file pruning optimizations rely on metadata about columns used in partitioning or Z-ordering. If a filter column (e.g., event_date) is not included in the partition or Z-ordering keys, Spark cannot effectively prune files at query time, resulting in full table scans. The Databricks optimization guide states that "File pruning and data skipping are most effective when queries filter on partition or Z-order columns." This explains why the filter was applied but had no impact on the

amount of data read. Options A and B are incorrect because Delta automatically applies file pruning when possible; D is less likely, as date columns are fully supported for skipping.