Databricks.Databricks-Certified-Professional-Data-Engineer.v2026-04-16.q82

Exam Code:	Databricks-Certified-Professional-Data-Engineer
Exam Name:	Databricks Certified Professional Data Engineer Exam
Certification Provider:	Databricks
Free Question Number:	82
Version:	v2026-04-16
# of views:	102
# of Questions views:	836
https://www.freecram.net/torrent/Databricks.Databricks-Certified-Professional-Data-Engineer.v2026-04-16.q82.html	

NEW QUESTION: 1
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: 2
A data engineer is designing a pipeline in Databricks that processes records from a Kafka stream where late-arriving data is common.
Which approach should the data engineer use?
A. Implement a custom solution using Databricks Jobs to periodically reprocess all historical data.
B. Use batch processing and overwrite the entire output table each time to ensure late data is incorporated correctly.
C. Use an Auto CDC pipeline with batch tables to simplify late data handling.
D. Use a watermark to specify the allowed lateness to accommodate records that arrive after their expected window, ensuring correct aggregation and state management.
Answer: (SHOW ANSWER)

Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
In Structured Streaming, event-time watermarks control how long the engine waits for late-arriving data before finalizing aggregations. By setting an appropriate watermark, Databricks can handle late data gracefully - incorporating records that arrive within the defined window while discarding excessively delayed events.
This approach ensures accurate aggregations, minimizes state size, and prevents memory leaks. Manual reprocessing (A) or overwriting entire datasets (B) is inefficient and costly, while Auto CDC (C) is used for change tracking in Delta tables, not for streaming event lateness.
Thus, using watermarking is the recommended and official approach for managing late data in streaming pipelines.

NEW QUESTION: 3
An hourly batch job is configured to ingest data files from a cloud object storage container where each batch represent all records produced by the source system in a given hour. The batch job to process these records into the Lakehouse is sufficiently delayed to ensure no late-arriving data is missed. The user_id field represents a unique key for the data, which has the following schema: user_id BIGINT, username STRING, user_utc STRING, user_region STRING, last_login BIGINT, auto_pay BOOLEAN, last_updated BIGINT New records are all ingested into a table named account_history which maintains a full record of all data in the same schema as the source. The next table in the system is named account_current and is implemented as a Type 1 table representing the most recent value for each unique user_id.
Assuming there are millions of user accounts and tens of thousands of records processed hourly, which implementation can be used to efficiently update the described account_current table as part of each hourly batch job?
A. Use Auto Loader to subscribe to new files in the account history directory; configure a Structured 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
Answer: (SHOW ANSWER)
most recent value for each username.
Explanation:

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

NEW QUESTION: 4
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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 5
The following table consists of items found in user carts within an e-commerce website.

```
Carts (id LONG, items ARRAY<STRUCT<id: LONG, count: INT>>)
id  items                                                email
1001[{id: "DESK65", count: 1}]                           "u1@domain.com"
1002[{id: "KYBD48", count: 1}, {id: "M27", count: 2}]"u2@domain.com"
1003[{id: "M27", count: 1}]                              "u3@domain.com"
```

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

How would the following update be handled?

A. The update is moved to separate "restored" column because it is missing a column expected in the target schema.

B. The new restored field is added to the target schema, and dynamically read as NULL for existing unmatched records.

C. The update throws an error because changes to existing columns in the target schema are not supported.

D. The new nested field is added to the target schema, and files underlying existing records are updated to include NULL values for the new field.

Answer: (SHOW ANSWER)

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

Reference:

Databricks documentation on schema evolution in Delta Lake:

https://docs.databricks.com/delta/delta-batch.html#schema-evolution

NEW QUESTION: 6

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: A (LEAVE A REPLY)

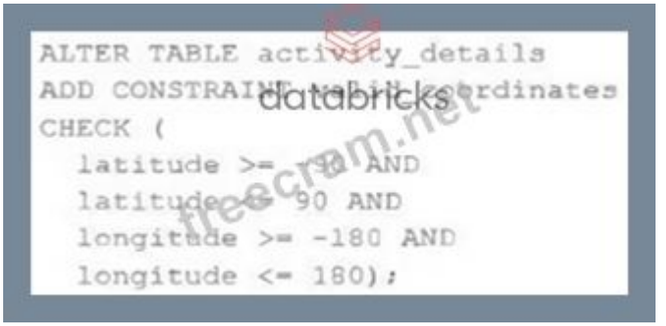
When Spark configuration properties are set for an interactive cluster using the Clusters UI in Databricks, those configurations are applied at the cluster level. This means that all notebooks

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

NEW QUESTION: 7
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. Reference:
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: 8

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:



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 Reference: [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: 9
A junior data engineer has configured a workload that posts the following JSON to the Databricks REST API endpoint 2.0/jobs/create.

Assuming that all configurations and referenced resources are available, which statement describes the result of executing this workload three times?
A. Three new jobs named "Ingest new data" will be defined in the workspace, and they will each run once daily.
B. The logic defined in the referenced notebook will be executed three times on new clusters with the configurations of the provided cluster ID.
C. Three new jobs named "Ingest new data" will be defined in the workspace, but no jobs will be executed.
D. One new job named "Ingest new data" will be defined in the workspace, but it will not be executed.
E. The logic defined in the referenced notebook will be executed three times on the referenced existing all purpose cluster.
Answer: (SHOW ANSWER)
This is the correct answer because the JSON posted to the Databricks REST API endpoint 2.0/jobs/create defines a new job with a name, an existing cluster id, and a notebook task. However, it does not specify any schedule or trigger for the job execution. Therefore, three new jobs with the same name and configuration will be created in the workspace, but none of them will be executed until they are manually triggered or scheduled. Verified Reference: [Databricks Certified Data Engineer Professional], under "Monitoring & Logging" section; [Databricks Documentation], under "Jobs API - Create" section.

NEW QUESTION: 10
A data engineer wants to ingest a large collection of image files (JPEG and PNG) from cloud object storage into a Unity Catalog-managed table for analysis and visualization.
Which two configurations and practices are recommended to incrementally ingest these images into the table? (Choose 2 answers)
A. Move files to a volume and read with SQL editor.
B. Use Auto Loader and set cloudFiles.format to "BINARYFILE".
C. Use Auto Loader and set cloudFiles.format to "TEXT".
D. Use Auto Loader and set cloudFiles.format to "IMAGE".
E. Use the pathGlobFilter option to select only image files (e.g., "*.jpg,*.png").

Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
Databricks Auto Loader supports ingestion of binary file formats using the cloudFiles.format option. For ingesting JPEG or PNG image files, the correct setting is "BINARYFILE", which loads the raw binary content and file metadata into a DataFrame. Additionally, when processing files from object storage, it is best practice to apply pathGlobFilter to limit ingestion to specific file types and reduce unnecessary scanning of non-image files. Options like "IMAGE" or "TEXT" are invalid, and using volumes with SQL editors does not provide incremental ingestion. Therefore, combining Auto Loader with cloudFiles.format="BINARYFILE" and pathGlobFilter ensures scalable, incremental ingestion of image data into Unity Catalog tables.

NEW QUESTION: 11
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: 12
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: (SHOW ANSWER)
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. Reference:
Watermarking: https://docs.databricks.com/spark/latest/structured-streaming/watermarks.html
Windowed aggregations: https://docs.databricks.com/spark/latest/structured-streaming/window-operations.html

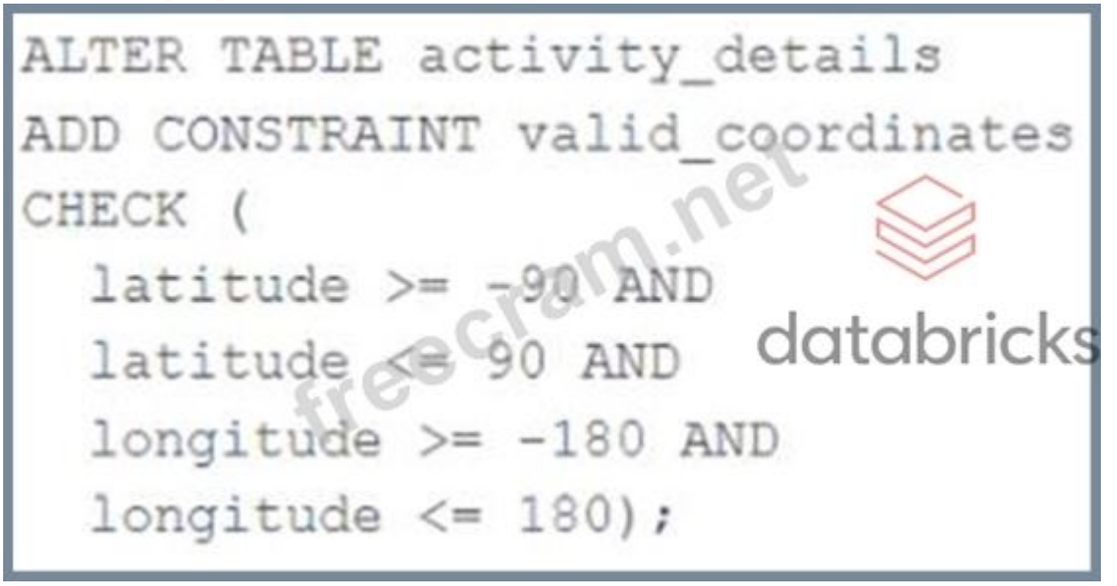
NEW QUESTION: 13
A data engineer has created a transactions Delta table on Databricks that should be used by the analytics team. The analytics team wants to use the table with another tool that requires Apache Iceberg format.
What should the data engineer do?
A. Require the analytics team to use a tool that supports Delta table.
B. Enable uniform on the transactions table to 'iceberg' so that the table can be read as an Iceberg table.
C. Create an Iceberg copy of the transactions Delta table which can be used by the analytics team.
D. Convert the transactions Delta table to Iceberg and enable uniform so that the table can be read as a Delta table.
Answer: (SHOW ANSWER)
Delta Lake introduced Delta Universal Format (Delta UniForm), which allows seamless interoperability between Delta Lake and Apache Iceberg. This means a Delta table can be converted into an Iceberg table while maintaining Delta capabilities.
Explanation of Each Option:
(A) Require the analytics team to use a tool that supports Delta table
Incorrect: While Delta Lake is widely used, requiring the team to change tools is not a flexible or scalable solution.
(B) Enable uniform on the transactions table to 'iceberg' so that the table can be read as an Iceberg table Incorrect:
The uniform feature must be enabled after conversion.
You cannot directly enable uniform without first converting the table.
(C) Create an Iceberg copy of the transactions Delta table which can be used by the analytics team Incorrect:
Creating a separate Iceberg copy would duplicate storage and increase maintenance complexity.
This is not necessary when Delta UniForm allows direct compatibility with Iceberg.
(D) Convert the transactions Delta table to Iceberg and enable uniform so that the table can be read as a Delta table Correct:
The best approach is to convert the existing Delta table to Iceberg using the Databricks Delta to Iceberg migration tools.
After conversion, enabling uniform ensures the table remains accessible in both Delta and Iceberg formats.
Conclusion:
The best practice for interoperability between Delta and Iceberg is to convert the Delta table to Iceberg and enable uniform, ensuring cross-compatibility without data duplication.
Thus, Option (D) is the correct answer.
Reference:
Delta UniForm for Apache Iceberg - Databricks Documentation
Convert Delta to Iceberg - Databricks

NEW QUESTION: 14
A data engineer is designing an append-only pipeline that needs to handle both batch and streaming data in Delta Lake. The team wants to ensure that the streaming component can efficiently track which data has already been processed.
Which configuration should be set to enable this?
A. overwriteSchema
B. partitionBy
C. checkpointLocation
D. mergeSchema
Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
When working with Delta Lake streaming ingestion, checkpointing is critical for maintaining fault tolerance and ensuring exactly-once data processing semantics.
The checkpointLocation parameter defines the directory where Spark Structured Streaming stores progress information, offsets, and metadata. This allows the engine to resume processing from the last committed offset without reprocessing previously ingested data.
Without checkpointing, each stream restart would reprocess all data, leading to duplicates.
Parameters like partitionBy or schema options (mergeSchema / overwriteSchema) affect table structure, not data lineage tracking. Therefore, the correct and required configuration for efficient streaming state management is checkpointLocation.

NEW QUESTION: 15
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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 16
A CHECK constraint has been successfully added to the Delta table named activity_details using the following logic:



A batch job is attempting to insert new records to the table, including a record where latitude = 45.50 and longitude = 212.67.
Which statement describes the outcome of this batch insert?
A. The write will fail when the violating record is reached; any records previously processed will be recorded to the target table.
B. The write will fail completely because of the constraint violation and no records will be inserted into the target table.
C. The write will insert all records except those that violate the table constraints; the violating records will be recorded to a quarantine table.
D. The write will include all records in the target table; any violations will be indicated in the boolean column named valid_coordinates.
E. The write will insert all records except those that violate the table constraints; the violating records will be reported in a warning log.
Answer: B (LEAVE A REPLY)
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. Reference:
Constraints: https://docs.delta.io/latest/delta-constraints.html

ACID Transactions: https://docs.delta.io/latest/delta-intro.html#acid-transactions

NEW QUESTION: 17
A data organization has adopted Delta Sharing to securely distribute curated datasets from a Unity Catalog-enabled workspace. The data engineering team shares large Delta tables internally via Databricks-to-Databricks and externally via Open Sharing for aggregated reports. While testing, they encounter challenges related to access control, data update visibility, and shareable object types.
What is a limitation of the Delta Sharing protocol or implementation when used with Databricks-to-Databricks or Open Sharing?
A. With Open Sharing, recipients cannot access Volumes, Models, or notebooks - only static Delta tables are supported.
B. Delta Sharing does not support Unity Catalog-enabled tables; only legacy Hive Metastore tables are shareable.
C. With Databricks-to-Databricks sharing, Unity Catalog recipients must re-ingest data manually using COPY INTO or REST APIs.
D. Delta Sharing (both Databricks-to-Databricks and Open Sharing) allows recipients to modify the source data if they have select privileges.
Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
According to Databricks' documentation, Open Sharing allows secure sharing of Delta tables to any recipient via a REST-based protocol without requiring a Databricks account. However, the Open Sharing protocol is limited to static Delta tables-it does not support sharing of Unity Catalog objects like Volumes, Machine Learning models, or notebooks. Only Databricks-to-Databricks sharing supports dynamic data sharing with update visibility and streaming reads. Thus, the inability to share non-table objects in Open Sharing represents a known limitation of the protocol. Option A accurately reflects this constraint as described in Delta Sharing design principles and documentation.

NEW QUESTION: 18
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: B (LEAVE A REPLY)
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.
Reference:
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: 19
Which statement describes Delta Lake Auto Compaction?
A. An asynchronous job runs after the write completes to detect if files could be further compacted; if yes, an optimize job is executed toward a default of 1 GB.

B. Before a Jobs cluster terminates, optimize is executed on all tables modified during the most recent job.
C. Optimized writes use logical partitions instead of directory partitions; because partition boundaries are only represented in metadata, fewer small files are written.
D. Data is queued in a messaging bus instead of committing data directly to memory; all data is committed from the messaging bus in one batch once the job is complete.
E. An asynchronous job runs after the write completes to detect if files could be further compacted; if yes, an optimize job is executed toward a default of 128 MB.
Answer: (SHOW ANSWER)
This is the correct answer because it describes the behavior of Delta Lake Auto Compaction, which is a feature that automatically optimizes the layout of Delta Lake tables by coalescing small files into larger ones. Auto Compaction runs as an asynchronous job after a write to a table has succeeded and checks if files within a partition can be further compacted. If yes, it runs an optimize job with a default target file size of 128 MB. Auto Compaction only compacts files that have not been compacted previously. Verified Reference: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Auto Compaction for Delta Lake on Databricks" section.
"Auto compaction occurs after a write to a table has succeeded and runs synchronously on the cluster that has performed the write. Auto compaction only compacts files that haven't been compacted previously."
https://learn.microsoft.com/en-us/azure/databricks/delta/tune-file-size

NEW QUESTION: 20
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: (SHOW ANSWER)
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 collects statistics 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. Reference:
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: 21
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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 22
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.
Reference: Databricks documentation on access control and permissions for notebooks within the workspace (https://docs.databricks.com/security/access-control/workspace-acl.html).

NEW QUESTION: 23
An analytics team wants to run a short-term experiment in Databricks SQL on the customer transactions Delta table (about 20 billion records) created by the data engineering team. Which strategy should the data engineering team use to ensure minimal downtime and no impact on the ongoing ETL processes?
A. Create a new table for the analytics team using a CTAS statement.
B. Deep clone the table for the analytics team.
C. Give the analytics team direct access to the production table.
D. Shallow clone the table for the analytics team.
Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation From Exact Extract:
Exact extract: "A shallow clone creates a copy of the metadata that references the source data files; it is fast and inexpensive." Exact extract: "A deep clone copies the data." Exact extract: "Clones provide a point-in-time snapshot for experimentation without impacting the source." A shallow clone of the production Delta table creates an instantaneous snapshot that references the same data files, so it introduces virtually no downtime or storage overhead and avoids interfering with the ongoing ETL. A deep clone would copy all data (very expensive and slow for 20B rows). CTAS rewrites data and is unnecessary; direct access to prod risks contention and accidental changes.

NEW QUESTION: 24
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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 25
A data engineering team is migrating off its legacy Hadoop platform. As part of the process, they are evaluating storage formats for performance comparison. The legacy platform uses ORC and RCFile formats. After converting a subset of data to Delta Lake, they noticed significantly better query performance. Upon investigation, they discovered that queries reading from Delta tables leveraged a Shuffle Hash Join, whereas queries on legacy formats used Sort Merge Joins. The queries reading Delta Lake data also scanned less data.
Which reason could be attributed to the difference in query performance?
A. Delta Lake enables data skipping and file pruning using a vectorized Parquet reader.
B. The queries against the Delta Lake tables were able to leverage the dynamic file pruning optimization.
C. Shuffle Hash Joins are always more efficient than Sort Merge Joins.
D. The queries against the ORC tables leveraged the dynamic data skipping optimization but not the dynamic file pruning optimization.
Answer: A (LEAVE A REPLY)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
Delta Lake outperforms legacy Hadoop formats because it leverages Parquet-based storage, data skipping, and file pruning. According to Databricks documentation, Delta Lake automatically stores detailed statistics (min/max values and file-level metadata) in the transaction log. During query planning, the engine uses these statistics to skip entire files that do not match query filters, a process called data skipping and file pruning. Additionally, Delta uses a vectorized Parquet reader, which reduces I/O and CPU overhead. Together, these optimizations allow Delta to scan significantly less data and produce more efficient physical query plans (e.g., Shuffle Hash Join instead of Sort Merge Join). The performance gain is due to efficient data skipping, not the inherent superiority of join type.

NEW QUESTION: 26
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. Reference:
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: 27
An upstream source writes Parquet data as hourly batches to directories named with the current date. A nightly batch job runs the following code to ingest all data from the previous day as indicated by the date variable:

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

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

A. Each write to the orders table will only contain unique records, and only those records without duplicates in the target table will be written.

B. Each write to the orders table will only contain unique records, but newly written records may have duplicates already present in the target table.

C. Each write to the orders table will only contain unique records; if existing records with the same key are present in the target table, these records will be overwritten.

D. Each write to the orders table will only contain unique records; if existing records with the same key are present in the target table, the operation will tail.

E. Each write to the orders table will run deduplication over the union of new and existing records, ensuring no duplicate records are present.

Answer: (SHOW ANSWER)

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

NEW QUESTION: 28

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

A. spark.sql.files.maxPartitionBytes

B. spark.sql.autoBroadcastJoinThreshold

C. spark.sql.files.openCostInBytes

D. spark.sql.adaptive.coalescePartitions.minPartitionNum

E. spark.sql.adaptive.advisoryPartitionSizeInBytes

Answer: (SHOW ANSWER)

This is the correct answer because spark.sql.files.maxPartitionBytes is a configuration parameter that directly affects the size of a spark-partition upon ingestion of data into Spark. This parameter configures the maximum number of bytes to pack into a single partition when reading files from file-based sources such as Parquet, JSON and ORC. The default value is 128 MB, which means

each partition will be roughly 128 MB in size, unless there are too many small files or only one large file. Verified Reference: [Databricks Certified Data Engineer Professional], under "Spark Configuration" section; Databricks Documentation, under "Available Properties - spark.sql.files.maxPartitionBytes" section.

NEW QUESTION: 29
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: C (LEAVE A REPLY)
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 Reference: [Databricks Certified Data Engineer Professional], under "Databricks Workspace" section; Databricks Documentation, under "Workspace audit logs" section.

NEW QUESTION: 30

A Structured Streaming job deployed to production has been resulting in higher than expected cloud storage costs. At present, during normal execution, each micro-batch of data is processed in less than 3 seconds; at least 12 times per minute, a micro-batch is processed that contains 0 records. The streaming write was configured using the default trigger settings. The production job is currently scheduled alongside many other Databricks jobs in a workspace with instance pools provisioned to reduce start-up time for jobs with batch execution. Holding all other variables constant and assuming records need to be processed in less than 10 minutes, which adjustment will meet the requirement?

A. Set the trigger interval to 500 milliseconds; setting a small but non-zero trigger interval ensures that the source is not queried too frequently.

B. Set the trigger interval to 3 seconds; the default trigger interval is consuming too many records per batch, resulting in spill to disk that can increase volume costs.

C. Set the trigger interval to 10 minutes; each batch calls APIs in the source storage account, so decreasing trigger frequency to the maximum allowable threshold should minimize this cost.

D. Use the trigger once option and configure a Databricks job to execute the query every 10 minutes; this approach minimizes costs for both compute and storage.

Answer: (SHOW ANSWER)

Comprehensive and Detailed Explanation From Exact Extract:

Exact extract: "If no trigger is specified, the default processing-time trigger runs micro-batches as fast as possible." Exact extract: "Trigger once processes all available data once and then stops." Exact extract: "Job clusters are created for a job run and terminate when the job completes." The default "as fast as possible" trigger creates many empty micro-batches which repeatedly list/query cloud storage, inflating storage/metadata API costs. Switching to trigger(once=True) and scheduling the job to run every 10 minutes processes all available data in one batch, then stops. This both meets the <10-minute freshness requirement and minimizes compute (cluster can shut down between runs) and storage API calls (one batch per run instead of continual empty batches).

Reference:

NEW QUESTION: 31

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

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

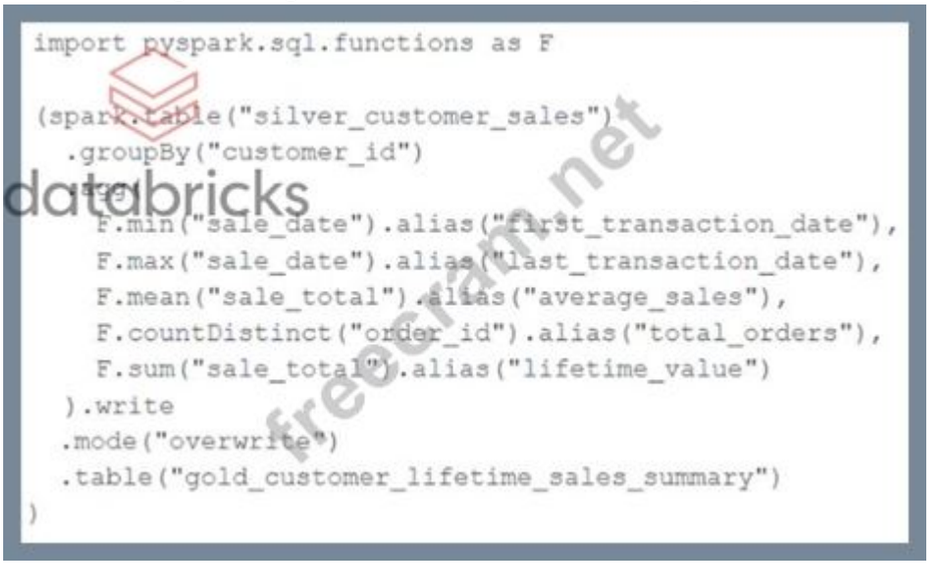
Which command allows manual confirmation that these three requirements have been met?

A. DESCRIBE EXTENDED dev.pii test

B. DESCRIBE DETAIL dev.pii test

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

NEW QUESTION: 32
The data engineering team maintains the following code:



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: C (LEAVE A REPLY)

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. Reference:
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: 33

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

A. importlib.resource path

B. ,sys.path

C. os-path

D. pylib.source

E. pypi.path

Answer: (SHOW ANSWER)

NEW QUESTION: 34

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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 35
A data engineering team is configuring access controls in Databricks Unity Catalog. They grant the SELECT privilege on the sales catalog to the analyst_group, expecting that members of this group will automatically have SELECT access to all current and future schemas, tables, and views within the catalog.
What describes the privilege inheritance behavior in Unity Catalog?
A. Granting SELECT on a catalog automatically applies SELECT to all current and future schemas, tables, and views within that catalog.
B. Privileges in Unity Catalog do not cascade; SELECT must be explicitly granted on each schema and table, even if granted at the catalog level.
C. Granting SELECT at the catalog level applies to existing schemas and tables but not to those created in the future.
D. Privileges granted at the schema level override any catalog-level privileges and prevent access unless explicitly revoked.
Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:

In Unity Catalog, privileges are non-cascading-meaning that granting a privilege (like SELECT) on a catalog does not automatically grant the same privilege on contained objects (schemas, tables, or views). Each object type has its own independent access control hierarchy.
According to the Databricks access control documentation: "Privileges do not automatically cascade from catalog to schema or table levels." Administrators must explicitly grant privileges on each level if users need access across objects. This design ensures tighter governance and least-privilege enforcement. Therefore, option B correctly describes Unity Catalog's privilege model, while A and D incorrectly imply automatic inheritance.

NEW QUESTION: 36
A data engineer is implementing Unity Catalog governance for a multi-team environment. Data scientists need interactive clusters for basic data exploration tasks, while automated ETL jobs require dedicated processing.
How should the data engineer configure cluster isolation policies to enforce least privilege and ensure Unity Catalog compliance?
A. Use only DEDICATED access mode for both interactive workloads and automated jobs to maximize security isolation.
B. Allow all users to create any cluster type and rely on manual configuration to enable Unity Catalog access modes.
C. Configure all clusters with NO ISOLATION_SHARED access mode since Unity Catalog works with any cluster configuration.
D. Create compute policies with STANDARD access mode for interactive workloads and DEDICATED access mode for automated jobs.
Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
Unity Catalog enforces governance and data isolation through cluster access modes and compute policies. According to Databricks documentation, "Interactive clusters that multiple users share should use Standard access mode, while automated jobs and production pipelines should use Dedicated access mode for stricter isolation." Standard access mode allows multiple users to share the same compute resources but still respects Unity Catalog permissions. Dedicated access mode isolates the job run's execution environment, ensuring that data access is limited to the job's identity. Configuring these modes within compute policies enforces least privilege and ensures all compute complies with Unity Catalog security standards. Options A and C are incorrect because using only Dedicated clusters reduces resource efficiency, while "No isolation" clusters are not Unity Catalog compliant.

NEW QUESTION: 37
Each configuration below is identical to the extent that each cluster has 400 GB total of RAM, 160 total cores and only one Executor per VM.

Given a job with at least one wide transformation, which of the following cluster configurations will result in maximum performance?

A. * Total VMs; 1
* 400 GB per Executor
* 160 Cores / Executor

B. * Total VMs: 8
* 50 GB per Executor
* 20 Cores / Executor

Answer: (SHOW ANSWER)

C.
* Total VMs: 4
* 100 GB per Executor
* 40 Cores/Executor

D.
* Total VMs:2
* 200 GB per Executor
* 80 Cores / Executor

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

NEW QUESTION: 38
Which method can be used to determine the total wall-clock time it took to execute a query?

A. In the Spark UI, take the job duration of the longest-running job associated with that query.

B. In the Spark UI, take the sum of all task durations that ran across all stages for all jobs associated with that query.

C. Open the Query Profiler associated with that query and use the Total wall-clock duration metric.

D. Open the Query Profiler associated with that query and use the Aggregated task time metric.

Answer: (SHOW ANSWER)

Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
The Query Profiler in Databricks SQL and notebooks provides a detailed breakdown of query performance metrics. The "Total wall-clock duration" metric directly represents the total elapsed time from query start to completion, including all execution, planning, and waiting stages. In contrast, "Aggregated task time" reflects the cumulative duration across all parallel tasks, which does not equal the total elapsed wall time since tasks often run concurrently. Using job duration from Spark UI can underestimate or overestimate runtime when queries span multiple jobs. Therefore, the Query Profiler's total wall-clock duration is the officially documented method to determine actual query execution time.

NEW QUESTION: 39
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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 40
The data engineering team maintains a table of aggregate statistics through batch nightly updates. This includes total sales for the previous day alongside totals and averages for a variety of time periods including the 7 previous days, year-to-date, and quarter-to-date. This table is named store_saies_summary and the schema is as follows:

```
store_id INT, total_sales_qtd FLOAT, avg_daily_sales_qtd FLOAT,  total_sales_ytd FLOAT,
avg_daily_sales_ytd FLOAT, previous_day_sales FLOAT, total_sales_7d FLOAT, avg_daily_sales_7d
FLOAT, updated TIMESTAMP
```

The table daily_store_sales contains all the information needed to update store_sales_summary. The schema for this table is:
store_id INT, sales_date DATE, total_sales FLOAT
If daily_store_sales is implemented as a Type 1 table and the total_sales column might be adjusted after manual data auditing, which approach is the safest to generate accurate reports in the store_sales_summary table?
A. Implement the appropriate aggregate logic as a batch read against the daily_store_sales table and overwrite the store_sales_summary table with each Update.
B. Implement the appropriate aggregate logic as a batch read against the daily_store_sales table and append new rows nightly to the store_sales_summary table.
C. Implement the appropriate aggregate logic as a batch read against the daily_store_sales table and use upsert logic to update results in the store_sales_summary table.
D. Implement the appropriate aggregate logic as a Structured Streaming read against the daily_store_sales table and use upsert logic to update results in the store_sales_summary table.
E. Use Structured Streaming to subscribe to the change data feed for daily_store_sales and apply changes to the aggregates in the store_sales_summary table with each update.
Answer: (SHOW ANSWER)
The daily_store_sales table contains all the information needed to update store_sales_summary. The schema of the table is:
store_id INT, sales_date DATE, total_sales FLOAT
The daily_store_sales table is implemented as a Type 1 table, which means that old values are overwritten by new values and no history is maintained. The total_sales column might be adjusted after manual data auditing, which means that the data in the table may change over time.
The safest approach to generate accurate reports in the store_sales_summary table is to use Structured Streaming to subscribe to the change data feed for daily_store_sales and apply changes to the aggregates in the store_sales_summary table with each update. Structured Streaming is a scalable and fault-tolerant stream processing engine built on Spark SQL. Structured Streaming allows processing data streams as if they were tables or DataFrames, using familiar operations such as select, filter, groupBy, or join. Structured Streaming also supports output modes that specify how to write the results of a streaming query to a sink, such as append, update, or complete. Structured Streaming can handle both streaming and batch data sources in a unified manner.
The change data feed is a feature of Delta Lake that provides structured streaming sources that can subscribe to changes made to a Delta Lake table. The change data feed captures both data changes and schema changes as ordered events that can be processed by downstream applications or services. The change data feed can be configured with different options, such as starting from a specific version or timestamp, filtering by operation type or partition values, or excluding no-op changes.
By using Structured Streaming to subscribe to the change data feed for daily_store_sales, one can capture and process any changes made to the total_sales column due to manual data auditing. By applying these changes to the aggregates in the store_sales_summary table with

each update, one can ensure that the reports are always consistent and accurate with the latest data. Verified Reference: [Databricks Certified Data Engineer Professional], under "Spark Core" section; Databricks Documentation, under "Structured Streaming" section; Databricks Documentation, under "Delta Change Data Feed" section.

NEW QUESTION: 41
A user new to Databricks is trying to troubleshoot long execution times for some pipeline logic they are working on. Presently, the user is executing code cell-by-cell, using display() calls to confirm code is producing the logically correct results as new transformations are added to an operation. To get a measure of average time to execute, the user is running each cell multiple times interactively.
Which of the following adjustments will get a more accurate measure of how code is likely to perform in production?
A. Scala is the only language that can be accurately tested using interactive notebooks; because the best performance is achieved by using Scala code compiled to 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 Reference: [Databricks Certified Data Engineer Professional], under "Spark Core" section; Databricks Documentation, under "Lazy evaluation" section; Databricks Documentation, under "Caching" section.

NEW QUESTION: 42
A Delta Lake table in the Lakehouse named customer_parsams is used in churn prediction by the machine learning team. The table contains information about customers derived from a number of upstream sources. Currently, the data engineering team populates this table nightly by overwriting the table with the current valid values derived from upstream data sources.
Immediately after each update succeeds, the data engineer team would like to determine the difference between the new version and the previous of the table.
Given the current implementation, which method can be used?
A. Parse the Delta Lake transaction log to identify all newly written data files.
B. Execute DESCRIBE HISTORY customer_churn_params to obtain the full operation metrics for the update, including a log of all records that have been added or modified.
C. Execute a query to calculate the difference between the new version and the previous version using Delta Lake's built-in versioning and time travel functionality.
D. Parse the Spark event logs to identify those rows that were updated, inserted, or deleted.
Answer: C (LEAVE A REPLY)
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.
Reference:
Delta Lake Documentation on Time Travel: Delta Time Travel
Delta Lake Versioning: Delta Lake Versioning Guide

NEW QUESTION: 43
The data architect has mandated that all tables in the Lakehouse should be configured as external Delta Lake tables.
Which approach will ensure that this requirement is met?
A. Whenever a database is being created, make sure that the location keyword is used
B. When configuring an external data warehouse for all table storage. leverage Databricks for all ELT.

C. Whenever a table is being created, make sure that the location keyword is used.
D. When tables are created, make sure that the external keyword is used in the create table statement.
E. When the workspace is being configured, make sure that external cloud object storage has been mounted.
Answer: (SHOW ANSWER)
This is the correct answer because it ensures that this requirement is met. The requirement is that all tables in the Lakehouse should be configured as external Delta Lake tables. An external table is a table that is stored outside of the default warehouse directory and whose metadata is not managed by Databricks. An external table can be created by using the location keyword to specify the path to an existing directory in a cloud storage system, such as DBFS or S3. By creating external tables, the data engineering team can avoid losing data if they drop or overwrite the table, as well as leverage existing data without moving or copying it. Verified Reference: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Create an external table" section.

NEW QUESTION: 44
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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 45

Which of the following technologies can be used to identify key areas of text when parsing Spark Driver log4j output?

A. Regex

B. Julia

C. pyspsark.ml.feature

D. Scala Datasets

E. C++

Answer: (SHOW ANSWER)

Regex, or regular expressions, are a powerful way of matching patterns in text. They can be used to identify key areas of text when parsing Spark Driver log4j output, such as the log level, the timestamp, the thread name, the class name, the method name, and the message. Regex can be applied in various languages and frameworks, such as Scala, Python, Java, Spark SQL, and Databricks notebooks. Reference:

https://docs.databricks.com/notebooks/notebooks-use.html#use-regular-expressions

https://docs.databricks.com/spark/latest/spark-sql/udf-scala.html#using-regular-expressions-in-udfs

https://docs.databricks.com/spark/latest/sparkr/functions/regexp_extract.html

https://docs.databricks.com/spark/latest/sparkr/functions/regexp_replace.html

NEW QUESTION: 46

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)

Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:

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: 47
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 Reference: [Databricks Certified Data Engineer Professional], under "Databricks Workspace" section; Databricks Documentation, under "Notebook permissions" section.

NEW QUESTION: 48

A junior developer complains that the code in their notebook isn't producing the correct results in the development environment. A shared screenshot reveals that while they're using a notebook versioned with Databricks Repos, they're using a personal branch that contains old logic. The desired branch named dev-2.3.9 is not available from the branch selection dropdown.
Which approach will allow this developer to review the current logic for this notebook?
A. Use Repos to make a pull request use the Databricks REST API to update the current branch to dev-2.3.9
B. Use Repos to pull changes from the remote Git repository and select the dev-2.3.9 branch.
C. Use Repos to checkout the dev-2.3.9 branch and auto-resolve conflicts with the current branch
D. Merge all changes back to the main branch in the remote Git repository and clone the repo again
E. Use Repos to merge the current branch and the dev-2.3.9 branch, then make a pull request to sync with the remote repository
Answer: B (LEAVE A REPLY)
This is the correct answer because it will allow the developer to update their local repository with the latest changes from the remote repository and switch to the desired branch. Pulling changes will not affect the current branch or create any conflicts, as it will only fetch the changes and not merge them. Selecting the dev-2.3.9 branch from the dropdown will checkout that branch and display its contents in the notebook. Verified Reference: [Databricks Certified Data Engineer Professional], under "Databricks Tooling" section; Databricks Documentation, under "Pull changes from a remote repository" section.

NEW QUESTION: 49
A data engineer is using Lakeflow Declarative Pipelines Expectations feature to track the data quality of their incoming sensor data. Periodically, sensors send bad readings that are out of range, and they are currently flagging those rows with a warning and writing them to the silver table along with the good data. They've been given a new requirement - the bad rows need to be quarantined in a separate quarantine table and no longer included in the silver table.
This is the existing code for their silver table:
@dlt.table
@dlt.expect("valid_sensor_reading", "reading < 120")
def silver_sensor_readings():
return spark.readStream.table("bronze_sensor_readings")
What code will satisfy the requirements?
A. @dlt.table
@dlt.expect("valid_sensor_reading", "reading < 120")
def silver_sensor_readings():
return spark.readStream.table("bronze_sensor_readings")
@dlt.table
@dlt.expect("invalid_sensor_reading", "reading >= 120")
def quarantine_sensor_readings():

return spark.readStream.table("bronze_sensor_readings")
B. @dlt.table
@dlt.expect_or_drop("valid_sensor_reading", "reading < 120")
def silver_sensor_readings():
return spark.readStream.table("bronze_sensor_readings")
@dlt.table
@dlt.expect("invalid_sensor_reading", "reading < 120")
def quarantine_sensor_readings():
return spark.readStream.table("bronze_sensor_readings")
C. @dlt.table
@dlt.expect_or_drop("valid_sensor_reading", "reading < 120")
def silver_sensor_readings():
return spark.readStream.table("bronze_sensor_readings")
@dlt.table
@dlt.expect_or_drop("invalid_sensor_reading", "reading >= 120")
def quarantine_sensor_readings():
return spark.readStream.table("bronze_sensor_readings")
D. @dlt.table
@dlt.expect_or_drop("valid_sensor_reading", "reading < 120")
def silver_sensor_readings():
return spark.readStream.table("bronze_sensor_readings")
@dlt.table
@dlt.expect("invalid_sensor_reading", "reading >= 120")
def quarantine_sensor_readings():
return spark.readStream.table("bronze_sensor_readings")
Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation from Databricks Documentation:
Lakeflow Declarative Pipelines (DLT) supports data quality enforcement using @dlt.expect, @dlt.expect_or_drop, and @dlt.expect_all.
@dlt.expect applies a rule and records whether rows pass or fail the condition but does not drop failing rows. Instead, failing rows can be written to a quarantine table.
@dlt.expect_or_drop enforces that only rows passing the condition flow downstream, dropping bad records automatically.
In this case, the requirement is:
Good rows (reading < 120) go to the silver table.
Bad rows (reading >= 120) go to a quarantine table.
Bad rows should not be included in silver.
The correct implementation is Option A, where:
The silver table uses @dlt.expect to validate reading < 120. These rows flow normally.
The quarantine table applies an expectation for reading >= 120, ensuring bad records are captured separately.

Other options are incorrect:
Option B/D: These either use expect_or_drop incorrectly or apply wrong conditions, leading to dropped rows without quarantining properly.
Option C: Uses expect_or_drop for both tables, which would discard bad rows instead of persisting them into a quarantine table.
Thus, Option A meets the business requirement to split good and bad data streams while ensuring both are captured for auditing and processing.

NEW QUESTION: 50

A data engineer is developing a Lakeflow Declarative Pipeline (LDP) using a Databricks notebook directly connected to their pipeline. After adding new table definitions and transformation logic in their notebook, they want to check for any syntax errors in the pipeline code without actually processing data or running the pipeline.
How should the data engineer perform this syntax check?

A. Use the "Validate" option in the notebook to check for syntax errors.
B. Open the web terminal from the notebook and run a shell command to validate the pipeline code.
C. Disconnect the notebook from the pipeline and reconnect it to a compute cluster to access code validation features.
D. Switch to a workspace file instead of a notebook to access validation and diagnostics tools.
Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
Databricks provides a "Validate" option within the Lakeflow Declarative Pipeline development interface that checks pipeline configurations, transformations, and syntax errors before actual execution.
This feature parses and validates the pipeline logic defined in notebooks or workspace files to ensure correctness and consistency of table dependencies, DLT (Delta Live Table) syntax, and schema references.
The validation process does not process or move any data, making it ideal for testing new configurations before deployment.
Using the shell terminal (B) or workspace files (D) does not perform integrated pipeline-level validation, while reconnecting to compute clusters (C) is unrelated to syntax checks. Therefore, the verified and correct approach is A.

NEW QUESTION: 51

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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 52
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.
Ensure that notifications are not spammed during sustained issues.
Options:
A. 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
B. 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
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)
The key requirement is to detect when the daily average of percent_null is greater than 5% for three consecutive days.
Option A only checks the last 24 hours, not consecutive days. It would trigger too frequently and cause spam.
Option C calculates an average across all records in the last 3 days, but this could be skewed by one high or low day - it does not ensure consecutive daily violations.
Option D simply counts days where the threshold was exceeded, but it does not guarantee that those days were consecutive. This could incorrectly trigger on non-adjacent violations.
Option B is correct:
It aggregates hourly values into daily averages.
It checks that the last 3 consecutive days all had averages above 5%.
It avoids redundant alerts by using Notification Frequency: Just once.
This matches Databricks Lakehouse Monitoring best practices, where SQL alerts should be designed to aggregate metrics to the correct granularity (daily here) and ensure consecutive threshold violations before triggering.
Reference (Databricks Lakehouse Monitoring, SQL Alerts Best Practices):
Use DATE_TRUNC to compute metrics at the correct time granularity.
To detect consecutive-day issues, filter the last N daily aggregates and check conditions across all rows.
Always configure alerts with controlled notification frequency to prevent alert fatigue.

NEW QUESTION: 53
An upstream system is emitting change data capture (CDC) logs that are being written to a cloud object storage directory. Each record in the log indicates the change type (insert, update, or delete) and the values for each field after the change. The source table has a primary key identified by the field pk_id.
For auditing purposes, the data governance team wishes to maintain a full record of all values that have ever been valid in the source system. For analytical purposes, only the most recent value for each record needs to be recorded. The Databricks job to ingest these records occurs once per hour, but each individual record may have changed multiple times over the course of an hour.
Which solution meets these requirements?

A. Create a separate history table for each pk_id resolve the current state of the table by running a union all filtering the history tables for the most recent state.
B. Use merge into to insert, update, or delete the most recent entry for each pk_id into a bronze table, then propagate all changes throughout the system.
C. Iterate through an ordered set of changes to the table, applying each in turn; rely on Delta Lake's versioning ability to create an audit log.
D. Use Delta Lake's change data feed to automatically process CDC data from an external system, propagating all changes to all dependent tables in the Lakehouse.
E. Ingest all log information into a bronze table; use merge into to insert, update, or delete the most recent entry for each pk_id into a silver table to recreate the current table state.
Answer: (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 bronze table, which preserves the raw CDC data as it is. Then, it uses merge into to perform an upsert operation on a silver table, which means it will insert new records or update or delete existing records based on the change type and the pk_id columns. This way, the silver table will always reflect the current state of the source table, while the bronze table will keep the history of all changes. Verified Reference: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Upsert into a table using merge" section.

NEW QUESTION: 54
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 that receives all rows for each stock symbol as a Pandas DataFrame, allowing processing within each group while maintaining state variables local to each group's processing function.
D. Use a grouped_agg 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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:

The Databricks documentation recommends applyInPandas() for complex per-group operations where maintaining internal state within each group is necessary. When using applyInPandas(), Spark provides all records for each grouping key as a Pandas DataFrame to the function, allowing efficient vectorized operations with local state management. This approach ensures high performance and scalability while maintaining logical isolation between groups. In contrast, SCALAR and SCALAR_ITER UDFs operate on individual rows or batches and cannot maintain inter-row state effectively. grouped_agg UDFs are limited to computing aggregates and do not support complex multi-row transformations. Therefore, applyInPandas() is the correct and Databricks-recommended solution for stateful per-group time-series computations.

NEW QUESTION: 55
A Spark job is taking longer than expected. Using the Spark UI, a data engineer notes that the Min, Median, and Max Durations for tasks in a particular stage show the minimum and median time to complete a task as roughly the same, but the max duration for a task to be roughly 100 times as long as the minimum.
Which situation is causing increased duration of the overall job?
A. Task queueing resulting from improper thread pool assignment.
B. Spill resulting from attached volume storage being too small.
C. Network latency due to some cluster nodes being in different regions from the source data
D. Skew caused by more data being assigned to a subset of spark-partitions.
E. Credential validation errors while pulling data from an external system.
Answer: (SHOW ANSWER)
This is the correct answer because skew is a common situation that causes increased duration of the overall job. Skew occurs when some partitions have more data than others, resulting in uneven distribution of work among tasks and executors. Skew can be caused by various factors, such as skewed data distribution, improper partitioning strategy, or join operations with skewed keys. Skew can lead to performance issues such as long-running tasks, wasted resources, or even task failures due to memory or disk spills. Verified Reference: [Databricks Certified Data Engineer Professional], under "Performance Tuning" section; Databricks Documentation, under "Skew" section.

NEW QUESTION: 56
A user wants to use DLT expectations to validate that a derived table report contains all records from the source, included in the table validation_copy.
The user attempts and fails to accomplish this by adding an expectation to the report table definition.
Which approach would allow using DLT expectations to validate all expected records are present in this table?
A. Define a SQL UDF that performs a left outer join on two tables, and check if this returns null values for report key values in a DLT expectation for the report table.

B. Define a function that performs a left outer join on validation_copy and report and report, and check against the result in a DLT expectation for the report table
C. Define a temporary table that perform a left outer join on validation_copy and report, and define an expectation that no report key values are null
D. Define a view that performs a left outer join on validation_copy and report, and reference this view in DLT expectations for the report table
Answer: D (LEAVE A REPLY)
To validate that all records from the source are included in the derived table, creating a view that performs a left outer join between the validation_copy table and the report table is effective. The view can highlight any discrepancies, such as null values in the report table's key columns, indicating missing records. This view can then be referenced in DLT (Delta Live Tables) expectations for the report table to ensure data integrity. This approach allows for a comprehensive comparison between the source and the derived table.
Reference:
Databricks Documentation on Delta Live Tables and Expectations: Delta Live Tables Expectations

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: A (LEAVE A REPLY)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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
What is a method of installing a Python package scoped at the notebook level to all nodes in the currently active cluster?
A. Use &Pip install in a notebook cell
B. Run source env/bin/activate in a notebook setup script
C. Install libraries from PyPi using the cluster UI
D. Use &sh install in a notebook cell
Answer: (SHOW ANSWER)
In Databricks notebooks, you can use the %pip install command in a notebook cell to install a Python package. This will install the package on all nodes in the currently active cluster at the notebook level. It is a feature provided by Databricks to facilitate the installation of Python libraries for the notebook environment specifically.

NEW QUESTION: 59
A data engineer wants to create a cluster using the Databricks CLI for a big ETL pipeline. The cluster should have five workers, one driver of type i3.xlarge, and should use the '14.3.x-scala2.12' runtime.
Which command should the data engineer use?
A. databricks clusters create 14.3.x-scala2.12 --num-workers 5 --node-type-id i3.xlarge --cluster-name DataEngineer_cluster
B. databricks clusters add 14.3.x-scala2.12 --num-workers 5 --node-type-id i3.xlarge --cluster-name Data Engineer_cluster
C. databricks compute add 14.3.x-scala2.12 --num-workers 5 --node-type-id i3.xlarge --cluster-name Data Engineer_cluster
D. databricks compute create 14.3.x-scala2.12 --num-workers 5 --node-type-id i3.xlarge --cluster-name Data Engineer_cluster
Answer: (SHOW ANSWER)
The Databricks CLI allows users to manage clusters using command-line commands. The correct command for creating a cluster follows a specific format.
Key Components in the Command:
Command Type: databricks compute create is the correct syntax for creating a new compute resource (cluster).
Runtime Version: '14.3.x-scala2.12' specifies the Databricks runtime to use.
Workers: --num-workers 5 sets the number of worker nodes to 5.
Node Type: --node-type-id i3.xlarge defines the hardware configuration.
Cluster Name: --cluster-name DataEngineer_cluster assigns a recognizable name to the cluster.
Evaluation of Options:

Option A (databricks clusters create ...)
Incorrect: databricks clusters create is not a valid command in the Databricks CLI v0.205.
The correct CLI command for cluster creation is databricks compute create.
Option B (databricks clusters add ...)
Incorrect: databricks clusters add is not a valid CLI command.
Option C (databricks compute add ...)
Incorrect: databricks compute add is not a valid CLI command.
Option D (databricks compute create ...)
Correct: databricks compute create is the correct command for creating a cluster.
Conclusion:
The correct command to create a cluster with five workers, an i3.xlarge node type, and Databricks runtime 14.3.x-scala2.12 is:
databricks compute create 14.3.x-scala2.12 --num-workers 5 --node-type-id i3.xlarge --cluster-name Data Engineer_cluster Thus, the correct answer is D.
Reference:
Databricks CLI Documentation

NEW QUESTION: 60

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: D (LEAVE A REPLY)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 61
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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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.

NEW QUESTION: 62
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 Reference: [Databricks Certified Data Engineer Professional], under "Databricks Workspace" section; Databricks Documentation, under "List secrets" section.

NEW QUESTION: 63
A Delta Lake table representing metadata about content posts from users has the following schema:
user_id LONG, post_text STRING, post_id STRING, longitude FLOAT, latitude FLOAT, post_time TIMESTAMP, date DATE This table is partitioned by the date column. A query is run with the following filter:
longitude < 20 & longitude > -20
Which statement describes how data will be filtered?
A. Statistics in the Delta Log will be used to identify partitions that might Include files in the filtered range.
B. No file skipping will occur because the optimizer does not know the relationship between the partition column and the longitude.
C. The Delta Engine will use row-level statistics in the transaction log to identify the flies that meet the filter criteria.
D. Statistics in the Delta Log will be used to identify data files that might include records in the filtered range.
E. The Delta Engine will scan the parquet file footers to identify each row that meets the filter criteria.
Answer: (SHOW ANSWER)
This is the correct answer because it describes how data will be filtered when a query is run with the following filter: longitude < 20 & longitude > -20. The query is run on a Delta Lake table that has the following schema: user_id LONG, post_text STRING, post_id STRING, longitude FLOAT, latitude FLOAT, post_time TIMESTAMP, date DATE. This table is partitioned by the date column.

When a query is run on a partitioned Delta Lake table, Delta Lake uses statistics in the Delta Log to identify data files that might include records in the filtered range. The statistics include information such as min and max values for each column in each data file. By using these statistics, Delta Lake can skip reading data files that do not match the filter condition, which can improve query performance and reduce I/O costs. Verified Reference: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Data skipping" section.

NEW QUESTION: 64
The data engineer is using Spark's MEMORY_ONLY storage level.
Which indicators should the data engineer look for in the spark UI's Storage tab to signal that a cached table is not performing optimally?
A. Size on Disk is> 0
B. The number of Cached Partitions> the number of Spark Partitions
C. The RDD Block Name included the " annotation signaling failure to cache
D. On Heap Memory Usage is within 75% of off Heap Memory usage
Answer: (SHOW ANSWER)
In the Spark UI's Storage tab, an indicator that a cached table is not performing optimally would be the presence of the _disk annotation in the RDD Block Name. This annotation indicates that some partitions of the cached data have been spilled to disk because there wasn't enough memory to hold them. This is suboptimal because accessing data from disk is much slower than from memory. The goal of caching is to keep data in memory for fast access, and a spill to disk means that this goal is not fully achieved.

NEW QUESTION: 65
What is true for Delta Lake?
A. Views in the Lakehouse maintain a valid cache of the most recent versions of source tables at all 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. Z-ORDERcan only be applied to numeric values stored in Delta Lake tables.
D. Primary and foreign key constraints can be leveraged to ensure duplicate values are never entered into a dimension table.
Answer: (SHOW ANSWER)
* Delta Lake automaticallycollects statisticson the first32 columnsof each table. These statistics helpoptimize query performancethroughdata skipping, which allows Databricks to scan only relevant parts of a table.
* This feature significantly improves query efficiency, especially when dealing with large datasets.
Why Other Options Are Incorrect:
* Option A:Views do not cache the most recent versions of the source table; they are recomputed when queried.

* Option C:Z-ORDERcan be applied to any data type, including strings, to optimize read performance.
* Option D:Delta Lake does not enforce primary or foreign key constraints.

NEW QUESTION: 66
A junior data engineer seeks to leverage Delta Lake's Change Data Feed functionality to create a Type 1 table representing all of the values that have ever been valid for all rows in a bronze table created with the property delta.enableChangeDataFeed = true. They plan to execute the following code as a daily job:

```
from pyspark.sql.functions import col

(spark.read.format("delta")
  .option("readChangeFeed", "true")
  .option("startingVersion", 0)
  .table("bronze")
  .filter(col("_change_type").isin(["update_postimage", "insert"]))
  .write
  .mode("append")
  .table("bronze_history_type1")
)
```


Which statement describes the execution and results of running the above query multiple times?
A. Each time the job is executed, newly updated records will be merged into the target table, overwriting previous values with the same primary keys.
B. Each time the job is executed, the entire available history of inserted or updated records will be appended to the target table, resulting in many duplicate entries.
C. Each time the job is executed, the target table will be overwritten using the entire history of inserted or updated records, giving the desired result.
D. Each time the job is executed, the differences between the original and current versions are calculated; this may result in duplicate entries for some records.
E. Each time the job is executed, only those records that have been inserted or updated since the last execution will be appended to the target table giving the desired result.
Answer: B (LEAVE A REPLY)
Reading table's changes, captured by CDF, using spark.read means that you are reading them as a static source. So, each time you run the query, all table's changes (starting from the specified startingVersion) will be read.

NEW QUESTION: 67
A Delta Lake table with Change Data Feed (CDF) enabled in the Lakehouse named customer_churn_params 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. 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. Modify the overwrite logic to include a field populated by calling current_timestamp() as data are being written; use this field to identify records written on a particular date.

C. 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.

D. 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.

Answer: (SHOW ANSWER)

Comprehensive and Detailed Explanation From Exact Extract:

Exact extract: "Change data feed (CDF) provides row-level change information for Delta tables." Exact extract: "Use table_changes to query the set of rows that were inserted, updated, or deleted between two versions (or timestamps)." Exact extract: "MERGE INTO updates and inserts only the rows that changed." Overwriting the table nightly makes it difficult to isolate just the changed rows. With CDF enabled, if you update the table using MERGE so only changed rows are touched, you can read exactly those changed rows from CDF for the last 24 hours and score only them, which is simpler and more efficient.

Reference:

NEW QUESTION: 68

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

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

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: 69

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 service principal with OAuth token federation.

B. Use a service principal ID and its OAuth client secret.

C. Use a service principal and its Personal Access Token.

D. Use a shared user account and its OAuth client secret.

Answer: (SHOW ANSWER)

Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:

The most secure and enterprise-recommended authentication method for Databricks automation is OAuth token federation with service principals.

This configuration allows service principals (non-human identities) to authenticate using temporary OAuth access tokens from a trusted identity provider (such as Azure AD or AWS IAM federation). These tokens are short-lived and scoped, significantly reducing credential exposure risks.

By contrast, static client secrets (B) or PATs (C) are long-lived and require periodic manual rotation, increasing security vulnerability. Shared user accounts (D) violate least-privilege and auditability principles. Therefore, A provides the strongest, most compliant authentication model for automated CLI and CI/CD workflows.

NEW QUESTION: 70

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

A. Structured Streaming leverages the parallel processing of GPUs to achieve highly parallel data throughput.

B. Structured Streaming is implemented as a messaging bus and is derived from Apache Kafka.

C. Structured Streaming uses specialized hardware and I/O streams to achieve sub-second latency for data transfer.

D. Structured Streaming models new data arriving in a data stream as new rows appended to an unbounded table.

E. Structured Streaming relies on a distributed network of nodes that hold incremental state values for cached stages.

Answer: (SHOW ANSWER)

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

Professional], under "Structured Streaming" section; Databricks Documentation, under "Overview" section.

NEW QUESTION: 71
A data architect has heard about lake's built-in versioning and time travel capabilities. For auditing purposes they have a requirement to maintain a full of all valid street addresses as they appear in the customers table.
The architect is interested in implementing a Type 1 table, overwriting existing records with new values and relying on Delta Lake time travel to support long-term auditing. A data engineer on the project feels that a Type 2 table will provide better performance and scalability.
Which piece of information is critical to this decision?
A. Delta Lake time travel does not scale well in cost or latency to provide a long-term versioning solution.
B. Delta Lake time travel cannot be used to query previous versions of these tables because Type 1 changes modify data files in place.
C. Shallow clones can be combined with Type 1 tables to accelerate historic queries for long-term versioning.
D. Data corruption can occur if a query fails in a partially completed state because Type 2 tables requires
Answer: (SHOW ANSWER)
Setting multiple fields in a single update.
Explanation:
Delta Lake's time travel feature allows users to access previous versions of a table, providing a powerful tool for auditing and versioning. However, using time travel as a long-term versioning solution for auditing purposes can be less optimal in terms of cost and performance, especially as the volume of data and the number of versions grow. For maintaining a full history of valid street addresses as they appear in a customers table, using a Type 2 table (where each update creates a new record with versioning) might provide better scalability and performance by avoiding the overhead associated with accessing older versions of a large table. While Type 1 tables, where existing records are overwritten with new values, seem simpler and can leverage time travel for auditing, the critical piece of information is that time travel might not scale well in cost or latency for long-term versioning needs, making a Type 2 approach more viable for performance and scalability.
Reference:
Databricks Documentation on Delta Lake's Time Travel: Delta Lake Time Travel Databricks Blog on Managing Slowly Changing Dimensions in Delta Lake: Managing SCDs in Delta Lake

NEW QUESTION: 72
A data engineer is configuring a pipeline that will potentially see late-arriving, duplicate records.

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

A. Set the configuration delta.deduplicate = true.

B. VACUUM the Delta table after each batch completes.

C. Perform an insert-only merge with a matching condition on a unique key.

D. Perform a full outer join on a unique key and overwrite existing data.

E. Rely on Delta Lake schema enforcement to prevent duplicate records.

Answer: (SHOW ANSWER)

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

MERGE INTO target_table USING source_table ON target_table.unique_key = source_table.unique_key WHEN NOT MATCHED THEN INSERT * This will insert only the records from the source table that have a unique key that is not present in the target table, and skip the records that have a matching key. This way, you can avoid inserting duplicate records into the Delta table.

Reference:

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

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

NEW QUESTION: 73

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

```
from pyspark.sql.functions import current_date

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


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

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

A. reds.write.format("delta").save("/preds/churn_preds")

B. preds.write.mode("append").saveAsTable("churn_preds")

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

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

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

Answer: (SHOW ANSWER)

NEW QUESTION: 74

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

A. It marks a column as having low enough cardinality to properly map distinct values to available partitions, allowing a broadcast join.

B. It marks a column as small enough to store in memory on all executors, allowing a broadcast join.

C. It caches a copy of the indicated table on attached storage volumes for all active clusters within a Databricks workspace.

D. It marks a DataFrame as small enough to store in memory on all executors, allowing a broadcast join.

E. It caches a copy of the indicated table on all nodes in the cluster for use in all future queries during the cluster lifetime.

Answer: (SHOW ANSWER)

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

NEW QUESTION: 75

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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 76
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)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
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: 77

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

```
from pyspark.sql.functions import current_timestamp, input_file_name, col
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. eturn spark.readStream.load("bronze")

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

C.

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: 78

Given the following error traceback:

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] The code snippet was: display(df.select(3*"heartrate"))

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)

Comprehensive and Detailed Explanation From Exact Extract:

Exact extract: "select() expects column names or Column expressions."

Exact extract: "When using strings directly, Spark SQL interprets them as literal column names."

Exact extract: "Python string operations, such as "colname"*3, return repeated strings, not column expressions." The expression 3*"heartrate" is Python string multiplication, which evaluates to "heartrateheartrateheartrate". The select() method interprets this as a literal column name. Since there is no column with that name in the DataFrame schema, Spark raises AnalysisException saying it cannot resolve that column. To correctly multiply a column by a scalar, one must use the column expression form:

from pyspark.sql.functions import col

df.select((col("heartrate") * 3).alias("heartrate_x3"))

This ensures Spark evaluates the arithmetic operation on the column instead of misinterpreting the string.

NEW QUESTION: 79
A data engineer is using Auto Loader to read incoming JSON data as it arrives. They have configured Auto Loader to quarantine invalid JSON records but notice that over time, some records are being quarantined even though they are well-formed JSON.
The code snippet is:
df = (spark.readStream
.format("cloudFiles")
.option("cloudFiles.format", "json")
.option("badRecordsPath", "/tmp/somewhere/badRecordsPath")
.schema("a int, b int")
.load("/Volumes/catalog/schema/raw_data/"))
What is the cause of the missing data?
A. At some point, the upstream data provider switched everything to multi-line JSON.
B. The badRecordsPath location is accumulating many small files.
C. The source data is valid JSON but does not conform to the defined schema in some way.
D. The engineer forgot to set the option "cloudFiles.quarantineMode" = "rescue".
Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
Databricks Auto Loader quarantines records that fail schema validation, even if they are syntactically valid JSON. The documentation explains that "records that do not match the defined schema are considered corrupt and written to the bad records path." In this case, the data likely contains valid JSON fields whose structure or data types differ from the declared schema (a int, b int). For example, if a field expected as an integer arrives as a string, Auto Loader will quarantine it. The presence of badRecordsPath confirms this behavior. The cloudFiles.quarantineMode option controls quarantine behavior but is unrelated to valid data that fails schema validation. Thus, the correct cause is C.

NEW QUESTION: 80
A junior data engineer has manually configured a series of jobs using the Databricks Jobs UI. Upon reviewing their work, the engineer realizes that they are listed as the "Owner" for each job. They attempt to transfer "Owner" privileges to the "DevOps" group, but cannot successfully accomplish this task. Which statement explains what is preventing this privilege transfer?

A. Databricks jobs must have exactly one owner; "Owner" privileges cannot be assigned to a group.
B. The creator of a Databricks job will always have "Owner" privileges; this configuration cannot be changed.
C. Other than the default "admins" group, only individual users can be granted privileges on jobs.
D. A user can only transfer job ownership to a group if they are also a member of that group.
E. Only workspace administrators can grant "Owner" privileges to a group.
Answer: (SHOW ANSWER)

The reason why the junior data engineer cannot transfer "Owner" privileges to the "DevOps" group is that Databricks jobs must have exactly one owner, and the owner must be an individual user, not a group. A job cannot have more than one owner, and a job cannot have a group as an owner. The owner of a job is the user who created the job, or the user who was assigned the ownership by another user. The owner of a job has the highest level of permission on the job, and can grant or revoke permissions to other users or groups. However, the owner cannot transfer the ownership to a group, only to another user. Therefore, the junior data engineer's attempt to transfer "Owner" privileges to the "DevOps" group is not possible. Reference:
Jobs access control: https://docs.databricks.com/security/access-control/table-acls/index.html Job permissions: https://docs.databricks.com/security/access-control/table-acls/privileges.html#job-permissions

NEW QUESTION: 81
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 Reference: [Databricks Certified Data Engineer Professional], under "Security & Governance" section; Databricks Documentation, under "Secrets" section.

NEW QUESTION: 82
A data engineer is designing an append-only pipeline that needs to handle both batch and streaming data in Delta Lake. The team wants to ensure that the streaming component can efficiently track which data has already been processed.
Which configuration should be set to enable this?
A. checkpointLocation
B. overwriteSchema
C. partitionBy
D. mergeSchema
Answer: (SHOW ANSWER)
Comprehensive and Detailed Explanation From Exact Extract of Databricks Data Engineer Documents:
When working with Delta Lake streaming ingestion, checkpointing is critical for maintaining fault tolerance and ensuring exactly-once data processing semantics.
The checkpointLocation parameter defines the directory where Spark Structured Streaming stores progress information, offsets, and metadata. This allows the engine to resume processing from the last committed offset without reprocessing previously ingested data.
Without checkpointing, each stream restart would reprocess all data, leading to duplicates.
Parameters like partitionBy or schema options (mergeSchema / overwriteSchema) affect table structure, not data lineage tracking. Therefore, the correct and required configuration for efficient streaming state management is checkpointLocation.