

Databricks-Certified-Professional-Data-Engineer Dumps

Databricks Certified Data Engineer Professional Exam

<https://www.certleader.com/Databricks-Certified-Professional-Data-Engineer-dumps.html>



NEW QUESTION 1

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

Which strategy will yield the best performance without shuffling data?

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

Answer: B

Explanation:

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

? Setting `spark.sql.files.maxPartitionBytes` to 512 MB configures Spark to process data in chunks of 512 MB. This setting directly influences the size of the part-files in the output, aligning with the target file size.

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

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

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

References:

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

? Databricks Documentation on Data Sources: Databricks Data Sources Guide

NEW QUESTION 2

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

The `silver_device_recordings` table will be used downstream to power several production monitoring dashboards and a production model. At present, 45 of the 100 fields are being used in at least one of these applications.

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

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

- A. The Tungsten encoding used by Databricks is optimized for storing string data; newly-added native support for querying JSON strings means that string types are always most efficient.
- B. Because Delta Lake uses Parquet for data storage, data types can be easily evolved by just modifying file footer information in place.
- C. Human labor in writing code is the largest cost associated with data engineering workloads; as such, automating table declaration logic should be a priority in all migration workloads.
- D. Because Databricks will infer schema using types that allow all observed data to be processed, setting types manually provides greater assurance of data quality enforcement.
- E. Schema inference and evolution on Databricks ensure that inferred types will always accurately match the data types used by downstream systems.

Answer: D

Explanation:

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

NEW QUESTION 3

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

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

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

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

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

Answer: D

Explanation:

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

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

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

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

References:

? Databricks Documentation on Batch Processing: Databricks Batch Processing

? Data Lakehouse Patterns: Data Lakehouse Best Practices

NEW QUESTION 4

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

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

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

Answer: C

Explanation:

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

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

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

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

References:

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

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

NEW QUESTION 5

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

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

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

Answer: A

Explanation:

The query uses the `CREATE TABLE USING DELTA` syntax to create a Delta Lake table from an existing Parquet file stored in DBFS. The query also uses the `LOCATION` keyword to specify the path to the Parquet file as `/mnt/finance_eda_bucket/tx_sales.parquet`. By using the `LOCATION` keyword, the query creates an external table, which is a table that is stored outside of the default warehouse directory and whose metadata is not managed by Databricks. An external table can be created from an existing directory in a cloud storage system, such as DBFS or S3, that contains data files in a supported format, such as Parquet or CSV.

The resulting state after running the second command is that an external table will be created in the storage container mounted to `/mnt/finance_eda_bucket` with the new name `prod.sales_by_store`. The command will not change any data or move any files in the storage container; it will only update the table reference in the metastore and create a new Delta transaction log for the renamed table. Verified References: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "ALTER TABLE RENAME TO" section; Databricks Documentation, under "Create an external table" section.

NEW QUESTION 6

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

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

Answer: A

Explanation:

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

References:

? Databricks Auto Loader Documentation: Auto Loader Guide

? Delta Lake and Auto Loader: Delta Lake Integration

NEW QUESTION 7

A Delta Lake table was created with the below query:

Realizing that the original query had a typographical error, the below code was executed: ALTER TABLE prod.sales_by_stor RENAME TO prod.sales_by_store
Which result will occur after running the second command?

- A. The table reference in the metastore is updated and no data is changed.
- B. The table name change is recorded in the Delta transaction log.
- C. All related files and metadata are dropped and recreated in a single ACID transaction.
- D. The table reference in the metastore is updated and all data files are moved.
- E. A new Delta transaction log is created for the renamed table.

Answer: A

Explanation:

The query uses the CREATE TABLE USING DELTA syntax to create a Delta Lake table from an existing Parquet file stored in DBFS. The query also uses the LOCATION keyword to specify the path to the Parquet file as /mnt/finance_eda_bucket/tx_sales.parquet. By using the LOCATION keyword, the query creates an external table, which is a table that is stored outside of the default warehouse directory and whose metadata is not managed by Databricks. An external table can be created from an existing directory in a cloud storage system, such as DBFS or S3, that contains data files in a supported format, such as Parquet or CSV. The result that will occur after running the second command is that the table reference in the metastore is updated and no data is changed. The metastore is a service that stores metadata about tables, such as their schema, location, properties, and partitions. The metastore allows users to access tables using SQL commands or Spark APIs without knowing their physical location or format. When renaming an external table using the ALTER TABLE RENAME TO command, only the table reference in the metastore is updated with the new name; no data files or directories are moved or changed in the storage system. The table will still point to the same location and use the same format as before. However, if renaming a managed table, which is a table whose metadata and data are both managed by Databricks, both the table reference in the metastore and the data files in the default warehouse directory are moved and renamed accordingly. Verified References: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "ALTER TABLE RENAME TO" section; Databricks Documentation, under "Metastore" section; Databricks Documentation, under "Managed and external tables" section.

NEW QUESTION 8

In order to prevent accidental commits to production data, a senior data engineer has instituted a policy that all development work will reference clones of Delta Lake tables. After testing both deep and shallow clone, development tables are created using shallow clone.

A few weeks after initial table creation, the cloned versions of several tables implemented as Type 1 Slowly Changing Dimension (SCD) stop working. The transaction logs for the source tables show that vacuum was run the day before.

Why are the cloned tables no longer working?

- A. The data files compacted by vacuum are not tracked by the cloned metadata; running refresh on the cloned table will pull in recent changes.
- B. Because Type 1 changes overwrite existing records, Delta Lake cannot guarantee data consistency for cloned tables.
- C. The metadata created by the clone operation is referencing data files that were purged as invalid by the vacuum command
- D. Running vacuum automatically invalidates any shallow clones of a table; deep clone should always be used when a cloned table will be repeatedly queried.

Answer: C

Explanation:

In Delta Lake, a shallow clone creates a new table by copying the metadata of the source table without duplicating the data files. When the vacuum command is run on the source table, it removes old data files that are no longer needed to maintain the transactional log's integrity, potentially including files referenced by the shallow clone's metadata. If these files are purged, the shallow cloned tables will reference non-existent data files, causing them to stop working properly. This highlights the dependency of shallow clones on the source table's data files and the impact of data management operations like vacuum on these clones. References: Databricks documentation on Delta Lake, particularly the sections on cloning tables (shallow and deep cloning) and data retention with the vacuum command (<https://docs.databricks.com/delta/index.html>).

NEW QUESTION 9

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

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

Which statement exemplifies best practices for implementing this system?

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

Answer: A

Explanation:

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

NEW QUESTION 10

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

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

Answer: C

NEW QUESTION 10

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

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

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

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

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

Answer: D

Explanation:

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

NEW QUESTION 14

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

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

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

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

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

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

Answer: E

Explanation:

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

NEW QUESTION 15

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

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

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

- A. Send all users notice that the schema for the table will be changing; include in the communication the logic necessary to revert the new table schema to match historic queries.

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

Answer: B

Explanation:

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

NEW QUESTION 16

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

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

Answer: C

NEW QUESTION 18

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

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

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

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

Answer: C

Explanation:

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

NEW QUESTION 22

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

least 1 GB of data and the total table size is over 10 TB.
Which of the following likely explains these smaller file sizes?

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

Answer: A

Explanation:

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

NEW QUESTION 26

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.

Original query:

```
df.groupBy("item")
  .agg(count("item").alias("total_count"),
       mean("sale_price").alias("avg_price"))
  .writeStream
  .outputMode("complete")
  .option("checkpointLocation", "~/item_agg/__checkpoint")
  .start("~/item_agg")
```

Proposed query:

```
df.groupBy("item")
  .agg(count("item").alias("total_count"),
       mean("sale_price").alias("avg_price"),
       count("promo_code = 'NEW_MEMBER') .alias("new_member_promo"))
  .writeStream
  .outputMode("complete")
  .option('mergeSchema', 'true')
  .option("checkpointLocation", "~/item_agg/__checkpoint")
  .start("~/item_agg")
```

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

Explanation:

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. References:

? 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 31

The data engineering team maintains the following code:

```
import pyspark.sql.functions as F

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

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

- A. The silver_customer_sales table will be overwritten by aggregated values calculated from all records in the gold_customer_lifetime_sales_summary table as a batch job.
- B. A batch job will update the gold_customer_lifetime_sales_summary table, replacing only those rows that have different values than the current version of the table, using customer_id as the primary key.
- C. The gold_customer_lifetime_sales_summary table will be overwritten by aggregated values calculated from all records in the silver_customer_sales table as a batch job.
- D. An incremental job will leverage running information in the state store to update aggregate values in the gold_customer_lifetime_sales_summary table.
- E. An incremental job will detect if new rows have been written to the silver_customer_sales table; if new rows are detected, all aggregates will be recalculated and used to overwrite the gold_customer_lifetime_sales_summary table.

Answer: C

Explanation:

This code is using the pyspark.sql.functions library to group the silver_customer_sales table by customer_id and then aggregate the data using the minimum sale date, maximum sale total, and sum of distinct order ids. The resulting aggregated data is then written to the gold_customer_lifetime_sales_summary table, overwriting any existing data in that table. This is a batch job that does not use any incremental or streaming logic, and does not perform any merge or update operations. Therefore, the code will overwrite the gold table with the aggregated values from the silver table every time it is executed. References:

- ? <https://docs.databricks.com/spark/latest/dataframes-datasets/introduction-to-dataframes-python.html>
- ? <https://docs.databricks.com/spark/latest/dataframes-datasets/transforming-data-with-dataframes.html>
- ? <https://docs.databricks.com/spark/latest/dataframes-datasets/aggregating-data-with-dataframes.html>

NEW QUESTION 33

A team of data engineer are adding tables to a DLT pipeline that contain repetitive expectations for many of the same data quality checks. One member of the team suggests reusing these data quality rules across all tables defined for this pipeline. What approach would allow them to do this?

- A. Maintain data quality rules in a Delta table outside of this pipeline's target schema, providing the schema name as a pipeline parameter.
- B. Use global Python variables to make expectations visible across DLT notebooks included in the same pipeline.
- C. Add data quality constraints to tables in this pipeline using an external job with access to pipeline configuration files.
- D. Maintain data quality rules in a separate Databricks notebook that each DLT notebook of file.

Answer: A

Explanation:

Maintaining data quality rules in a centralized Delta table allows for the reuse of these rules across multiple DLT (Delta Live Tables) pipelines. By storing these rules outside the pipeline's target schema and referencing the schema name as a pipeline parameter, the team can apply the same set of data quality checks to different tables within the pipeline. This approach ensures consistency in data quality validations and reduces redundancy in code by not having to replicate the same rules in each DLT notebook or file. References:

- ? [Databricks Documentation on Delta Live Tables: Delta Live Tables Guide](#)

NEW QUESTION 35

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

```

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

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

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

...

```

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

```

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

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

```

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

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

Answer: A

Explanation:

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

NEW QUESTION 40

The data engineer is using Spark's MEMORY_ONLY storage level. Which indicators should the data engineer look for in the spark UI's Storage tab to signal that a cached table is not performing optimally?

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

Answer: C

Explanation:

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

NEW QUESTION 42

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

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

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

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

Answer: C

Explanation:

In Databricks, using the Secrets module allows for secure management of sensitive information such as database credentials. Granting 'Read' permissions on a secret key that maps to database credentials for a specific team ensures that only members of that team can access these credentials. This approach aligns with

the principle of least privilege, granting users the minimum level of access required to perform their jobs, thus enhancing security.

References:

? Databricks Documentation on Secret Management: Secrets

NEW QUESTION 46

Assuming that the Databricks CLI has been installed and configured correctly, which Databricks CLI command can be used to upload a custom Python Wheel to object storage mounted with the DBFS for use with a production job?

- A. configure
- B. fs
- C. jobs
- D. libraries
- E. workspace

Answer: B

Explanation:

The libraries command group allows you to install, uninstall, and list libraries on Databricks clusters. You can use the libraries install command to install a custom Python Wheel on a cluster by specifying the --whl option and the path to the wheel file. For example, you can use the following command to install a custom Python Wheel named mylib-0.1-py3-none-any.whl on a cluster with the id 1234-567890-abcde123:

```
databricks libraries install --cluster-id1234-567890-abcde123--whldbfs:/mnt/mylib/mylib-0.1-py3-none-any.whl
```

This will upload the custom Python Wheel to the cluster and make it available for use with a production job. You can also use the libraries uninstall command to uninstall a library from a cluster, and the libraries list command to list the libraries installed on a cluster. References:

? Libraries CLI (legacy): <https://docs.databricks.com/en/archive/dev-tools/cli/libraries-cli.html>

? Library operations: <https://docs.databricks.com/en/dev-tools/cli/commands.html#library-operations>

? Install or update the Databricks CLI: <https://docs.databricks.com/en/dev-tools/cli/install.html>

NEW QUESTION 49

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

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

Cmd 2
rawDF.printSchema()

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

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

Cmd 5
display(finalDF)

Cmd 6

Cmd 7
finalDF.write.mode("append").saveAsTable("flat_data")
```

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

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

Answer: E

Explanation:

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

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

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

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

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

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

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

NEW QUESTION 50

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

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

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

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

Answer: E

Explanation:

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

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

NEW QUESTION 55

A table is registered with the following code:

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

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

Answer: B

NEW QUESTION 58

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

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

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

- A. Yes; Delta Lake ACID guarantees provide assurance that the delete command succeeded fully and permanently purged these records.
- B. No; the Delta cache may return records from previous versions of the table until the cluster is restarted.
- C. Yes; the Delta cache immediately updates to reflect the latest data files recorded to disk.
- D. No; the Delta Lake delete command only provides ACID guarantees when combined with the merge into command.
- E. No; files containing deleted records may still be accessible with time travel until a vacuum command is used to remove invalidated data files.

Answer: E

Explanation:

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

NEW QUESTION 61

Which statement describes integration testing?

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

Answer: D

Explanation:

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

NEW QUESTION 62

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

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

print(password)

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

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

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

Answer: E

Explanation:

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

NEW QUESTION 66

A Delta table of weather records is partitioned by date and has the below schema: date DATE, device_id INT, temp FLOAT, latitude FLOAT, longitude FLOAT. To find all the records from within the Arctic Circle, you execute a query with the below filter:

```
latitude > 66.3
```

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

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

Answer: D

Explanation:

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

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

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

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

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

NEW QUESTION 68

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

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

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

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

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

- A. Because the vacuum command permanently deletes all files containing deleted records, deleted records may be accessible with time travel for around 24 hours.
- B. Because the default data retention threshold is 24 hours, data files containing deleted records will be retained until the vacuum job is run the following day.
- C. Because Delta Lake time travel provides full access to the entire history of a table, deleted records can always be recreated by users with full admin privileges.
- D. Because Delta Lake's delete statements have ACID guarantees, deleted records will be permanently purged from all storage systems as soon as a delete job completes.
- E. Because the default data retention threshold is 7 days, data files containing deleted records will be retained until the vacuum job is run 8 days later.

Answer: E

Explanation:

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

NEW QUESTION 72

Two of the most common data locations on Databricks are the DBFS root storage and external object storage mounted with `dbutils.fs.mount()`. Which of the following statements is correct?

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

Answer: A

Explanation:

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

NEW QUESTION 76

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

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

Answer: A

Explanation:

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

NEW QUESTION 78

The data engineering team has configured a Databricks SQL query and alert to monitor the values in a Delta Lake table. The `recent_sensor_recordings` table contains an identifying `sensor_id` alongside the timestamp and temperature for the most recent 5 minutes of recordings.

The below query is used to create the alert:

```
SELECT MEAN(temperature), MAX(temperature), MIN(temperature)
FROM recent_sensor_recordings
GROUP BY sensor_id
```

The query is set to refresh each minute and always completes in less than 10 seconds. The alert is set to trigger when `mean (temperature) > 120`. Notifications are triggered to be sent at most every 1 minute.

If this alert raises notifications for 3 consecutive minutes and then stops, which statement must be true?

- A. The total average temperature across all sensors exceeded 120 on three consecutive executions of the query
- B. The `recent_sensor_recordingstable` was unresponsive for three consecutive runs of the query
- C. The source query failed to update properly for three consecutive minutes and then restarted
- D. The maximum temperature recording for at least one sensor exceeded 120 on three consecutive executions of the query
- E. The average temperature recordings for at least one sensor exceeded 120 on three consecutive executions of the query

Answer: E

Explanation:

This is the correct answer because the query is using a `GROUP BY` clause on the `sensor_id` column, which means it will calculate the mean temperature for each sensor separately. The alert will trigger when the mean temperature for any sensor is greater than 120, which means at least one sensor had an average temperature above 120 for three consecutive minutes. The alert will stop when the mean temperature for all sensors drops below 120. Verified References: [Databricks Certified Data Engineer Professional], under "SQL Analytics" section; Databricks Documentation, under "Alerts" section.

NEW QUESTION 83

Which statement describes Delta Lake optimized writes?

- A. A shuffle occurs prior to writing to try to group data together resulting in fewer files instead of each executor writing multiple files based on directory partitions.
- B. Optimized writes logical partitions instead of directory partitions partition boundaries are only represented in metadata fewer small files are written.
- C. An asynchronous job runs after the write completes to detect if files could be further compacted; yes, an OPTIMIZE job is executed toward a default of 1 GB.
- D. Before a job cluster terminates, OPTIMIZE is executed on all tables modified during the most recent job.

Answer: A

Explanation:

Delta Lake optimized writes involve a shuffle operation before writing out data to the Delta table. The shuffle operation groups data by partition keys, which can lead to a reduction in the number of output files and potentially larger files, instead of multiple smaller files. This approach can significantly reduce the total number of files in the table, improve read performance by reducing the metadata overhead, and optimize the table storage layout, especially for workloads with many small files.

References:

? Databricks documentation on Delta Lake performance tuning: <https://docs.databricks.com/delta/optimizations/auto-optimize.html>

NEW QUESTION 84

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. return spark.readStream.load("bronze")
C. return (spark.read
 .table("bronze")
 .filter(col("ingest_time") == current_timestamp())
)

D.return

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

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

Answer: E

Explanation:

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

NEW QUESTION 89

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

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

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

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

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

Answer: A

Explanation:

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

NEW QUESTION 94

Which of the following is true of Delta Lake and the Lakehouse?

- A. Because Parquet compresses data row by row
- B. strings will only be compressed when a character is repeated multiple times.
- C. Delta Lake automatically collects statistics on the first 32 columns of each table which are leveraged in data skipping based on query filters.
- D. Views in the Lakehouse maintain a valid cache of the most recent versions of source tables at all times.
- E. Primary and foreign key constraints can be leveraged to ensure duplicate values are never entered into a dimension table.
- F. Z-order can only be applied to numeric values stored in Delta Lake tables

Answer: B

Explanation:

<https://docs.delta.io/2.0.0/table-properties.html>

Delta Lake automatically collects statistics on the first 32 columns of each table, which are leveraged in data skipping based on query filters¹. Data skipping is a performance optimization technique that aims to avoid reading irrelevant data from the storage layer¹. By collecting statistics such as min/max values, null counts, and bloom filters, Delta Lake can efficiently prune unnecessary files or partitions from the query plan¹. This can significantly improve the query performance and reduce the I/O cost.

The other options are false because:

? Parquet compresses data column by column, not row by row². This allows for better compression ratios, especially for repeated or similar values within a column².

? Views in the Lakehouse do not maintain a valid cache of the most recent versions of source tables at all times³. Views are logical constructs that are defined by a SQL query on one or more base tables³. Views are not materialized by default, which means they do not store any data, but only the query definition³.

Therefore, views always reflect the latest state of the source tables when queried³. However, views can be cached manually using the CACHE TABLE or CREATE TABLE AS SELECT commands.

? Primary and foreign key constraints can not be leveraged to ensure duplicate values are never entered into a dimension table. Delta Lake does not support enforcing primary and foreign key constraints on tables. Constraints are logical rules that define the integrity and validity of the data in a table. Delta Lake relies on the application logic or the user to ensure the data quality and consistency.

? Z-order can be applied to any values stored in Delta Lake tables, not only numeric values. Z-order is a technique to optimize the layout of the data files by sorting them on one or more columns. Z-order can improve the query performance by clustering related values together and enabling more efficient data skipping. Z-order can be applied to any column that has a defined ordering, such as numeric, string, date, or boolean values.

References: Data Skipping, Parquet Format, Views, [Caching], [Constraints], [Z-Ordering]

NEW QUESTION 96

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

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

F. In the Query Detail screen, by interpreting the Physical Plan

Answer: E

Explanation:

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

NEW QUESTION 101

A new data engineer notices that a critical field was omitted from an application that writes its Kafka source to Delta Lake. This happened even though the critical field was in the Kafka source. That field was further missing from data written to dependent, long-term storage. The retention threshold on the Kafka service is seven days. The pipeline has been in production for three months.

Which describes how Delta Lake can help to avoid data loss of this nature in the future?

- A. The Delta log and Structured Streaming checkpoints record the full history of the Kafka producer.
- B. Delta Lake schema evolution can retroactively calculate the correct value for newly added fields, as long as the data was in the original source.
- C. Delta Lake automatically checks that all fields present in the source data are included in the ingestion layer.
- D. Data can never be permanently dropped or deleted from Delta Lake, so data loss is not possible under any circumstance.
- E. Ingesting all raw data and metadata from Kafka to a bronze Delta table creates a permanent, replayable history of the data state.

Answer: E

Explanation:

This is the correct answer because it describes how Delta Lake can help to avoid data loss of this nature in the future. By ingesting all raw data and metadata from Kafka to a bronze Delta table, Delta Lake creates a permanent, replayable history of the data state that can be used for recovery or reprocessing in case of errors or omissions in downstream applications or pipelines. Delta Lake also supports schema evolution, which allows adding new columns to existing tables without affecting existing queries or pipelines. Therefore, if a critical field was omitted from an application that writes its Kafka source to Delta Lake, it can be easily added later and the data can be reprocessed from the bronze table without losing any information. Verified References: [Databricks Certified Data Engineer Professional], under “Delta Lake” section; Databricks Documentation, under “Delta Lake core features” section.

NEW QUESTION 105

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

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

Answer: B

Explanation:

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

NEW QUESTION 107

.....

Thank You for Trying Our Product

* 100% Pass or Money Back

All our products come with a 90-day Money Back Guarantee.

* One year free update

You can enjoy free update one year. 24x7 online support.

* Trusted by Millions

We currently serve more than 30,000,000 customers.

* Shop Securely

All transactions are protected by VeriSign!

100% Pass Your Databricks-Certified-Professional-Data-Engineer Exam with Our Prep Materials Via below:

<https://www.certleader.com/Databricks-Certified-Professional-Data-Engineer-dumps.html>