

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 a type error because a column object cannot be multiplied.
- **D. There is no column in the table named heartrateheartrateheartrate**
- E. The code executed was PvSoark but was executed in a Scala notebook.

Answer: D

Explanation:

The error being raised is an AnalysisException, which is a type of exception that occurs when Spark SQL cannot analyze or execute a query due to some logical or semantic error. In this case, the error message indicates that the query cannot resolve the column name 'heartrateheartrateheartrate' given the input columns 'heartrate' and 'age'. This means that there is no column in the table named 'heartrateheartrateheartrate', and the query is invalid. A possible cause of this error is a typo or a copy-paste mistake in the query. To fix this error, the query should use a valid column name that exists in the table, such as 'heartrate'. References: AnalysisException

NEW QUESTION # 18

A data engineer deploys a multi-task Databricks job that orchestrates three notebooks. One task intermittently fails with Exit Code 1 but succeeds on retry. The engineer needs to collect detailed logs for the failing attempts, including stdout/stderr and cluster lifecycle context, and share them with the platform team.

What steps the data engineer needs to follow using built-in tools?

- A. Download worker logs directly from the Spark UI and ignore driver logs, as worker logs contain stdout/stderr for all tasks and cluster events.
- B. Export the notebook run results to HTML; this bundle includes complete stdout, stderr, and cluster event history across all tasks.
- **C. From the job run details page, export the job's logs or configure log delivery; then retrieve the compute driver logs and event logs from the compute details page to correlate stdout/stderr with cluster events.**
- D. Use the notebook interactive debugger to re-run the entire multi-task job, and capture step-through traces for the failing task.

Answer: C

Explanation:

The recommended way to troubleshoot and collect detailed job logs is through the Job Run Details page in Databricks. From there, engineers can export run logs or configure automatic log delivery to a storage destination. The driver and event logs available under compute details provide stdout, stderr, and cluster lifecycle context required for root-cause analysis.

Reference Source: Databricks Jobs Monitoring and Logging Documentation - "Access driver logs and configure log delivery."

NEW QUESTION # 19

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 Read
- **B. No permissions**
- C. Can Manage
- D. Can Run
- E. Can Edit

Answer: B

Explanation:

This is the correct answer because it is the maximum notebook permissions that can be granted to the user without allowing

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

NEW QUESTION # 20

Which of the following techniques structured streaming uses to ensure recovery of failures during stream processing?

- A. Checkpointing and Watermarking
- B. Checkpointing and Idempotent sinks
- C. Write ahead logging and watermarking
- D. Delta time travel
- E. Checkpointing and write-ahead logging
- F. The stream will failover to available nodes in the cluster

Answer: E

Explanation:

Explanation

The answer is Checkpointing and write-ahead logging.

Structured Streaming uses checkpointing and write-ahead logs to record the offset range of data being processed during each trigger interval.

NEW QUESTION # 21

A data engineer is tasked with ensuring that a Delta table in Databricks continuously retains deleted files for 15 days (instead of the default 7 days), in order to permanently comply with the organization's data retention policy.

Which code snippet correctly sets this retention period for deleted files?

- A. `spark.sql("VACUUM my_table RETAIN 15 HOURS")`
- B. `from delta.tables import *`
`deltaTable = DeltaTable.forPath(spark, "/mnt/data/my_table")`
`deltaTable.deletedFileRetentionDuration = "interval 15 days"`
- C. `spark.conf.set("spark.databricks.delta.deletedFileRetentionDuration", "15 days")`
- D. `spark.sql("ALTER TABLE my_table SET TBLPROPERTIES ('delta.deletedFileRetentionDuration' = 'interval 15 days')")`

Answer: D

Explanation:

In Delta Lake, the property `delta.deletedFileRetentionDuration` controls how long deleted data files are retained before being permanently removed during a VACUUM operation.

By default, this retention duration is set to 7 days.

To comply with stricter retention requirements, organizations can explicitly update the table property using an ALTER TABLE statement.

Option A uses the correct SQL command:

`ALTER TABLE my_table SET TBLPROPERTIES ('delta.deletedFileRetentionDuration' = 'interval 15 days')` This updates the Delta table metadata so that all future operations respect the 15-day retention policy for deleted files.

Why not the others?

B: This code incorrectly tries to set the property via the DeltaTable API. Delta's Python API does not expose direct attributes like `deletedFileRetentionDuration`; instead, properties must be set through ALTER TABLE or DataFrameWriter options.

C: VACUUM ... RETAIN specifies a one-time file cleanup action (e.g., retaining 15 hours of history), not a persistent retention policy. It cannot be used to set a continuous retention duration.

D: Setting `spark.conf` applies a session-level configuration and does not permanently update the table's retention metadata. Once the session ends, this configuration is lost.

Therefore, Option A is the correct and documented approach for persistently enforcing a 15-day deleted file retention period in

myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt,
myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt,
myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, Disposable vapes