DATABRICKS-CERTIFIED-PROFESSIONAL-DATA-ENGINEER Q&As

Databricks Certified Professional Data Engineer Exam

Pass Databricks DATABRICKS-CERTIFIED-PROFESSIONAL-DATA-ENGINEER Exam with 100% Guarantee

Free Download Real Questions & Answers **PDF** and **VCE** file from:

https://www.leads4pass.com/databricks-certified-professional-data-engineer.html

100% Passing Guarantee
100% Money Back Assurance

Following Questions and Answers are all new published by Databricks Official Exam Center

QUESTION 1

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

Thesilver_device_recordingstable 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 encodingused by Databricksis optimized for storing stringdata; newly- added nativesupport for querying JSON strings means that stringtypes 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 writingcode is the largest cost associated with data engineering workloads; as such, automatingtable 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.

Correct 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.

QUESTION 2

An hourly batch job is configured to ingest data files from a cloud object storage container where each batch represent all records produced by the source system in a given hour. The batch job to process these records into the Lakehouse is sufficiently delayed to ensure no late-arriving data is missed. Theuser_idfield represents a unique key for the data, which has the following schema:

user_id BIGINT, username STRING, user_utc STRING, user_region STRING, last_login BIGINT, auto_pay BOOLEAN, last_updated BIGINT

New records are all ingested into a table namedaccount_historywhich maintains a full record of all data in the same schema as the source. The next table in the system is namedaccount_currentand is implemented as a Type 1 table representing the most recent value for each uniqueuser_id.

Assuming there are millions of user accounts and tens of thousands of records processed hourly, which implementation can be used to efficiently update the describedaccount_currenttable as part of each hourly batch job?

A. Use Auto Loader to subscribe to new files in the account history directory; configure a Structured Streaminq trigger once job to batch update newly detected files into the account current table.

B. Overwrite the account current table with each batch using the results of a query against the account history table grouping by user id and filtering for the max value of last updated.

C. Filter records in account history using the last updated field and the most recent hour processed, as well as the max last iogin by user id write a merge statement to update or insert the most recent value for each user id.

D. Use Delta Lake version history to get the difference between the latest version of account history and one version prior, then write these records to account current.

E. Filter records in account history using the last updated field and the most recent hour processed, making sure to deduplicate on username; write a merge statement to update or insert the most recent value for each username.

Correct Answer: C

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

QUESTION 3

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 namedpredswith 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. preds.write.mode("append").saveAsTable("churn_preds")
B. preds.write.format("delta").save("/preds/churn_preds") C)
C.

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

D.

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

E.

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

A. Option

B. Option

C. Option

D. Option

E. Option

Correct Answer: C

Explanation: This is the correct answer because it will save the predictions to a Delta Lake table with the ability to compare all predictions across time. The code uses the mergeInto method to perform an upsert operation, which means it will insert new records or update existing records based on the customer_id and date columns. This way, the table will always contain the latest predictions for each customer and date, and also keep the history of previous predictions. The code also uses a new job cluster to run the job, which will minimize the compute costs as it will be created and terminated for each run. Verified References: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Upsert into a table using merge" section.

QUESTION 4

Incorporating unit tests into a PySpark application requires upfront attention to the design of your jobs, or a potentially significant refactoring of existing code.

Which statement describes a main benefit that offset this additional effort?

A. Improves the quality of your data

B. Validates a complete use case of your application

C. Troubleshooting is easier since all steps are isolated and tested individually

D. Yields faster deployment and execution times

E. Ensures that all steps interact correctly to achieve the desired end result

Correct Answer: C

QUESTION 5

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

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

A. Scala is the only language that can be accurately tested using interactive notebooks; because the best performance is achieved by using Scala code compiled to JARs. all PySpark and Spark SQL logic should be refactored.

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

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

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

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

Correct Answer: D

Explanation: This is the correct answer because it explains which of the following adjustments will get a more accurate measure of how code is likely to perform in production. The adjustment is that calling display() forces a job to trigger, while many transformations will only add to the logical query plan; because of caching, repeated execution of the same logic does not provide meaningful results. When developing code in Databricks notebooks, one should be aware of how Spark handles transformations and actions. Transformations are operations that create a new DataFrame or Dataset from an existing one, such as filter, select, or join. Actions are operations that trigger a computation on a DataFrame or Dataset and return a result to the driver program or write it to storage, such as count, show, or save. Calling display() on a DataFrame or Dataset is also an action that triggers a computation and displays the result in a notebook cell. Spark

uses lazy evaluation for transformations, which means that they are not executed until an action is called. Spark also uses caching to store intermediate results in memory or disk for faster access in subsequent actions. Therefore, calling display() forces a job to trigger, while many transformations will only add to the logical query plan; because of caching, repeated execution of the same logic does not provide meaningful results. To get a more accurate measure of how code is likely to perform in production, one should avoid calling display() too often or clear the cache before running each cell. Verified References: [Databricks Certified Data Engineer Professional], under "Spark Core" section; Databricks Documentation, under "Lazy evaluation" section; Databricks Documentation, under "Caching" section.