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.geekcert.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

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

A. Because Parquet compresses data row by row. strings will only be compressed when a character is repeated multiple times.

B. Delta Lake automatically collects statistics on the first 32 columns of each table which are leveraged in data skipping based on query filters.

C. Views in the Lakehouse maintain a valid cache of the most recent versions of source tables at all times.

D. Primary and foreign key constraints can be leveraged to ensure duplicate values are never entered into a dimension table.

E. Z-order can only be applied to numeric values stored in Delta Lake tables

Correct Answer: A

Explanation: This is the correct answer because it is true of Delta Lake and the Lakehouse. Delta Lake uses Parquet as the underlying storage format for data files. Parquet is a columnar format that compresses data by column rather than by row. This means that Parquet can achieve high compression ratios for columns that have low cardinality or high repetition of values, such as integers, booleans, or dates. However, for columns that have high cardinality or low repetition of values, such as strings, Parquet cannot compress data very well. Therefore, strings will only be compressed when a character is repeated multiple times within a row. Verified References:[Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Delta Lake core features - Schema enforcement and evolution" section.

QUESTION 2

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 3

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

A. importlib.resource path

B. ,sys.path

C. os-path

D. pypi.path

E. pylib.source

Correct Answer: B

QUESTION 4

When scheduling Structured Streaming jobs for production, which configuration automatically recovers from query failures and keeps costs low?

A. Cluster: New Job Cluster; Retries: Unlimited; Maximum Concurrent Runs: Unlimited

B. Cluster: New Job Cluster; Retries: None; Maximum Concurrent Runs: 1

C. Cluster: Existing All-Purpose Cluster; Retries: Unlimited; Maximum Concurrent Runs: 1

D. Cluster: Existing All-Purpose Cluster; Retries: Unlimited; Maximum Concurrent Runs: 1

E. Cluster: Existing All-Purpose Cluster; Retries: None; Maximum Concurrent Runs: 1

Correct Answer: B

Explanation: This is the best configuration for scheduling Structured Streaming jobs for production, as it automatically recovers from query failures and keeps costs low. A new job cluster is created for each run of the job and terminated when the job completes, which saves costs and avoids resource contention. Retries are not needed for Structured Streaming jobs, as they can automatically recover from failures using checkpointing and write-ahead logs. Maximum concurrent runs should be set to 1 to avoid duplicate output or data loss. Verified References: Databricks Certified Data Engineer Professional, under "Monitoring and Logging" section; Databricks Documentation, under "Schedule streaming jobs" section.

QUESTION 5

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

Correct 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.