# Free Questions for Databricks-Certified-Professional-Data-Engineer by vceexamstest

## Shared by Huffman on 15-04-2024

**For More Free Questions and Preparation Resources**

**Check the Links on Last Page**

# Question 1

**Question Type:** MultipleChoice

The data engineer team is configuring environment for development testing, and production before beginning migration on a new data pipeline. The team requires extensive testing on both the code and data resulting from code execution, and the team want to develop and test against similar production data as possible.

A junior data engineer suggests that production data can be mounted to the development testing environments, allowing pre production code to execute against production dat

a. Because all users have

Admin privileges in the development environment, the junior data engineer has offered to configure permissions and mount this data for the team.

Which statement captures best practices for this situation?

## Options:

**A-** Because access to production data will always be verified using passthrough credentials it is safe to mount data to any Databricks development environment.

**B-** All developer, testing and production code and data should exist in a single unified workspace; creating separate environments for testing and development further reduces risks.

**C-** In environments where interactive code will be executed, production data should only be accessible with read permissions; creating isolated databases for each environment further reduces risks.

**D-** Because delta Lake versions all data and supports time travel, it is not possible for user error or malicious actors to permanently delete production data, as such it is generally safe to mount production data anywhere.

### Answer:

C

### Explanation:

The best practice in such scenarios is to ensure that production data is handled securely and with proper access controls. By granting only read access to production data in development and testing environments, it mitigates the risk of unintended data modification. Additionally, maintaining isolated databases for different environments helps to avoid accidental impacts on production data and systems.

Databricks best practices for securing data: https://docs.databricks.com/security/index.html

## Question 2

**Question Type:** **MultipleChoice**

A data pipeline uses Structured Streaming to ingest data from kafka to Delta Lake. Data is being stored in a bronze table, and includes the Kafka_generated timesamp, key, and value. Three months after the pipeline is deployed the data engineering team has noticed some latency issued during certain times of the day.

A senior data engineer updates the Delta Table's schema and ingestion logic to include the current timestamp (as recoded by Apache Spark) as well the Kafka topic and partition. The team plans to use the additional metadata fields to diagnose the transient processing delays:

Which limitation will the team face while diagnosing this problem?

## Options:

**A-** New fields not be computed for historic records.

**B-** Updating the table schema will invalidate the Delta transaction log metadata.

**C-** Updating the table schema requires a default value provided for each file added.

**D-** Spark cannot capture the topic partition fields from the kafka source.

## Answer:

A

## Explanation:

When adding new fields to a Delta table's schema, these fields will not be retrospectively applied to historical records that were ingested before the schema change. Consequently, while the team can use the new metadata fields to investigate transient processing delays moving forward, they will be unable to apply this diagnostic approach to past data that lacks these fields.

Databricks documentation on Delta Lake schema management: https://docs.databricks.com/delta/delta-batch.html#schema-management

# Question 3

**Question Type:** **MultipleChoice**

A DLT pipeline includes the following streaming tables:

Raw_lot ingest raw device measurement data from a heart rate tracking device.

Bgm_stats incrementally computes user statistics based on BPM measurements from raw_lot.

How can the data engineer configure this pipeline to be able to retain manually deleted or updated records in the raw_iot table while recomputing the downstream table when a pipeline update is run?

## Options:

**A-** Set the skipChangeCommits flag to true on bpm_stats

**B-** Set the SkipChangeCommits flag to true raw_lot

**C-** Set the pipelines, reset, allowed property to false on bpm_stats

**D-** Set the pipelines, reset, allowed property to false on raw_iot

### Answer:

D

### Explanation:

In Databricks Lakehouse, to retain manually deleted or updated records in the raw_iot table while recomputing downstream tables when a pipeline update is run, the property pipelines.reset.allowed should be set to false. This property prevents the system from resetting the state of the table, which includes the removal of the history of changes, during a pipeline update. By keeping this property as false, any changes to the raw_iot table, including manual deletes or updates, are retained, and recomputation of downstream tables, such as bpm_stats, can occur with the full history of data changes intact.

Databricks documentation on DLT pipelines: https://docs.databricks.com/data-engineering/delta-live-tables/delta-live-tables-overview.html

## Question 4

**Question Type:** MultipleChoice

Which statement describes Delta Lake optimized writes?

## Options:

**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.

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

# Question 5

**Question Type:** **MultipleChoice**

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?

## Options:

**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.

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

## Question 6

**Question Type:** MultipleChoice

The following table consists of items found in user carts within an e-commerce website.

```
Carts (id LONG, items ARRAY<STRUCT<id: LONG, count: INT>>)
id  items                                                email
1001[{id: "DESK65", count: 1}]                           "u1@domain.com"
1002[{id: "KYBD45", count: 1}, {id: "M27", count: 2}]"u2@domain.com"
1003[{id: "M27", count: 1}]                              "u3@domain.com"
```

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

```
MERGE INTO carts c
USING updates u
ON c.id = u.id
WHEN MATCHED
  THEN UPDATE SET *
```

How would the following update be handled?

```
(new nested field, missing existing column)
id  items
1001[{id: "DESK65", count: 2, coupon: "BOGO50"}]
```

How would the following update be handled?

## Options:

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

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

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

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

## Answer:

D

### Explanation:

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

Databricks documentation on schema evolution in Delta Lake: https://docs.databricks.com/delta/delta-batch.html#schema-evolution

## Question 7

**Question Type:** **MultipleChoice**

The data science team has created and logged a production using MLFlow. The model accepts a list of column names and returns a new column of type DOUBLE.

The following code correctly imports the production model, load the customer table containing the customer_id key column into a Dataframe, and defines the feature columns needed for the model.

```
model = mlflow.pyfunc.spark_udf (spark,
model_uri="models:/churn/prod")

df = spark.table("customers")

columns = ["account_age", "time_since_last_seen", "app_rating"]
```

Which code block will output DataFrame with the schema'' customer_id LONG, predictions DOUBLE''?

## Options:

**A-** Model, predict (df, columns)

**B-** Df, map (lambda k:midel (x [columns]) ,select (''customer_id predictions'')

**C-** Df. Select (''customer_id''.
Model (''columns) alias (''predictions'')

**D-** Df.apply(model, columns). Select (''customer_id, prediction''

## Answer:

A

## Explanation:

Given the information that the model is registered with MLflow and assuming predict is the method used to apply the model to a set of columns, we use the model.predict() function to apply the model to the DataFrame df using the specified columns. The model.predict()

function is designed to take in a DataFrame and a list of column names as arguments, applying the trained model to these features to produce a predictions column. When working with PySpark, this predictions column needs to be selected alongside the customer_id to create a new DataFrame with the schema customer_id LONG, predictions DOUBLE.

MLflow documentation on using Python function models: https://www.mlflow.org/docs/latest/models.html#python-function-python

PySpark MLlib documentation on model prediction: https://spark.apache.org/docs/latest/ml-pipeline.html#pipeline

# Question 8

**Question Type:** MultipleChoice

A developer has successfully configured credential for Databricks Repos and cloned a remote Git repository. Hey don not have privileges to make changes to the main branch, which is the only branch currently visible in their workspace.

Use Response to pull changes from the remote Git repository commit and push changes to a branch that appeared as a changes were pulled.

## Options:

**A-** Use Repos to merge all differences and make a pull request back to the remote repository.

**B-** Use repos to merge all difference and make a pull request back to the remote repository.

**C-** Use Repos to create a new branch commit all changes and push changes to the remote Git repertory.

**D-** Use repos to create a fork of the remote repository commit all changes and make a pull request on the source repository

### Answer:

C

### Explanation:

In Databricks Repos, when a user does not have privileges to make changes directly to the main branch of a cloned remote Git repository, the recommended approach is to create a new branch within the Databricks workspace. The developer can then make changes in this new branch, commit those changes, and push the new branch to the remote Git repository. This workflow allows for isolated development without affecting the main branch, enabling the developer to propose changes via a pull request from the new branch to the main branch in the remote repository. This method adheres to common Git collaboration workflows, fostering code review and collaboration while ensuring the integrity of the main branch.

Databricks documentation on using Repos with Git: https://docs.databricks.com/repos.html

## Question 9

**Question Type:** MultipleChoice

The business reporting tem requires that data for their dashboards be updated every hour. The total processing time for the pipeline that extracts transforms and load the data for their pipeline runs in 10 minutes.

Assuming normal operating conditions, which configuration will meet their service-level agreement requirements with the lowest cost?

## Options:

**A-** Schedule a jo to execute the pipeline once and hour on a dedicated interactive cluster.

**B-** Schedule a Structured Streaming job with a trigger interval of 60 minutes.

**C-** Schedule a job to execute the pipeline once hour on a new job cluster.

**D-** Configure a job that executes every time new data lands in a given directory.

## Answer:

C

## Explanation:

Scheduling a job to execute the data processing pipeline once an hour on a new job cluster is the most cost-effective solution given the scenario. Job clusters are ephemeral in nature; they are spun up just before the job execution and terminated upon completion, which means you only incur costs for the time the cluster is active. Since the total processing time is only 10 minutes, a new job cluster created for each hourly execution minimizes the running time and thus the cost, while also fulfilling the requirement for hourly data updates for

the business reporting team's dashboards.

Databricks documentation on jobs and job clusters: https://docs.databricks.com/jobs.html

# Question 10

**Question Type:** **MultipleChoice**

Which statement regarding spark configuration on the Databricks platform is true?

## Options:

**A-** Spark configuration properties set for an interactive cluster with the Clusters UI will impact all notebooks attached to that cluster.

**B-** When the same spar configuration property is set for an interactive to the same interactive cluster.

**C-** Spark configuration set within an notebook will affect all SparkSession attached to the same interactive cluster

**D-** The Databricks REST API can be used to modify the Spark configuration properties for an interactive cluster without interrupting jobs.

## Answer:

A

## Explanation:

When Spark configuration properties are set for an interactive cluster using the Clusters UI in Databricks, those configurations are applied at the cluster level. This means that all notebooks attached to that cluster will inherit and be affected by these configurations. This approach ensures consistency across all executions within that cluster, as the Spark configuration properties dictate aspects such as memory allocation, number of executors, and other vital execution parameters. This centralized configuration management helps maintain standardized execution environments across different notebooks, aiding in debugging and performance optimization.

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

# Question 11

**Question Type:** MultipleChoice

The view updates represents an incremental batch of all newly ingested data to be inserted or updated in the customers table.

The following logic is used to process these records.

MERGE INTO customers

USING (

SELECT updates.customer_id as merge_ey, updates .*

FROM updates

UNION ALL

SELECT NULL as merge_key, updates .*

FROM updates JOIN customers

ON updates.customer_id = customers.customer_id

WHERE customers.current = true AND updates.address customers.address

) staged_updates

ON customers.customer_id = mergekey

WHEN MATCHED AND customers. current = true AND customers.address staged_updates.address THEN

UPDATE SET current = false, end_date = staged_updates.effective_date

WHEN NOT MATCHED THEN

INSERT (customer_id, address, current, effective_date, end_date)

VALUES (staged_updates.customer_id, staged_updates.address, true, staged_updates.effective_date, null)

Which statement describes this implementation?

## Options:

**A-** The customers table is implemented as a Type 2 table; old values are overwritten and new customers are appended.

**B-** The customers table is implemented as a Type 1 table; old values are overwritten by new values and no history is maintained.

**C-** The customers table is implemented as a Type 2 table; old values are maintained but marked as no longer current and new values are inserted.

**D-** The customers table is implemented as a Type 0 table; all writes are append only with no changes to existing values.

## Answer:

C

## Explanation:

The provided MERGE statement is a classic implementation of a Type 2 SCD in a data warehousing context. In this approach, historical data is preserved by keeping old records (marking them as not current) and adding new records for changes. Specifically, when a match is found and there's a change in the address, the existing record in the customers table is updated to mark it as no longer current (current = false), and an end date is assigned (end_date = staged_updates.effective_date). A new record for the customer is then inserted with the updated information, marked as current. This method ensures that the full history of changes to customer information is maintained in the table, allowing for time-based analysis of customer data. Reference: Databricks documentation on implementing SCDs using Delta Lake and the MERGE statement (https://docs.databricks.com/delta/delta-update.html#upsert-into-a-table-using-merge).