Question 1

Question Type: MultipleChoice

The data science team has requested assistance in accelerating queries on free form text from user reviews. The data is currently stored in Parquet with the below schema:

item_id INT, user_id INT, review_id INT, rating FLOAT, review STRING

The review column contains the full text of the review left by the user. Specifically, the data science team is looking to identify if any of 30 key words exist in this field.

A junior data engineer suggests converting this data to Delta Lake will improve query performance.

Which response to the junior data engineer s suggestion is correct?

Options:

A- Delta Lake statistics are not optimized for free text fields with high cardinality.
B- Text data cannot be stored with Delta Lake.
C- ZORDER ON review will need to be run to see performance gains.
D- The Delta log creates a term matrix for free text fields to support selective filtering.
E- Delta Lake statistics are only collected on the first 4 columns in a table.

Answer:

A

Explanation:

Converting the data to Delta Lake may not improve query performance on free text fields with high cardinality, such as the review column. This is because Delta Lake collects statistics on the minimum and maximum values of each column, which are not very useful for filtering or skipping data on free text fields. Moreover, Delta Lake collects statistics on the first 32 columns by default, which may not include the review column if the table has more columns. Therefore, the junior data engineer's suggestion is not correct. A better approach would be to use a full-text search engine, such as Elasticsearch, to index and query the review column. Alternatively, you can use natural language processing techniques, such as tokenization, stemming, and lemmatization, to preprocess the review column and create a new column with normalized terms that can be used for filtering or skipping data.Reference:

Optimizations: https://docs.delta.io/latest/optimizations-oss.html

Full-text search with Elasticsearch: https://docs.databricks.com/data/data-sources/elasticsearch.html

Natural language processing: https://docs.databricks.com/applications/nlp/index.html

Question 2

Question Type: MultipleChoice

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?

Options:

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

Answer:

D

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-id 1234-567890-abcde123 --whl dbfs:/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.

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

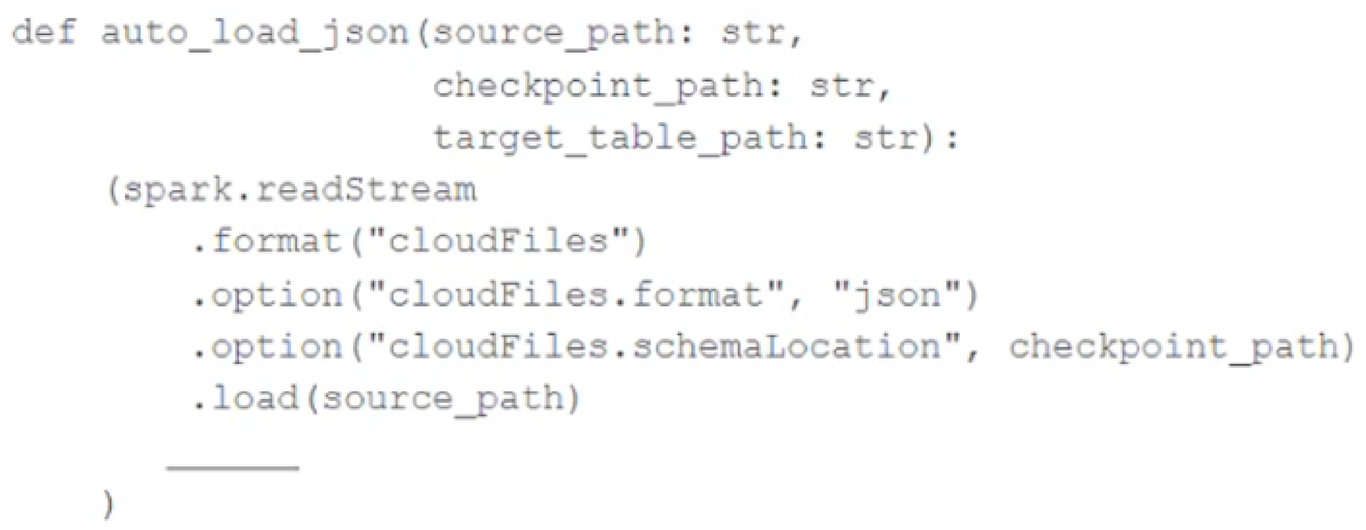
Install or update the Databricks CLI: https://docs.databricks.com/en/dev-tools/cli/install.html

Question 3

Question Type: MultipleChoice

In order to facilitate near real-time workloads, a data engineer is creating a helper function to leverage the schema detection and evolution functionality of Databricks Auto Loader. The desired function will automatically detect the schema of the source directly, incrementally process JSON files as they arrive in a source directory, and automatically evolve the schema of the table when new fields are detected.

The function is displayed below with a blank:



```
def auto_load_json(source_path: str,
                   checkpoint_path: str,
                   target_table_path: str):
    (spark.readStream
        .format("cloudFiles")
        .option("cloudFiles.format", "json")
        .option("cloudFiles.schemaLocation", checkpoint_path)
        .load(source_path)

       ______
    )
```

Which response correctly fills in the blank to meet the specified requirements?

A.
```
.writeStream
.option("mergeSchema", True)
.start(target_table_path)
```

B.
```
.writeStream
.option("checkpointLocation", checkpoint_path)
.option("mergeSchema", True)
.trigger(once=True)
.start(target_table_path)
```

C.
```
.write
.option("checkpointLocation", checkpoint_path)
.option("mergeSchema", True)
.outputMode("append")
.save(target_table_path)
```

D.
```
.write
.option("mergeSchema", True)
.mode("append")
.save(target_table_path)
```

E.
```
.writeStream
.option("checkpointLocation", checkpoint_path)
.option("mergeSchema", True)
.start(target_table_path)
```

Options:

A- Option A
B- Option B
C- Option C
D- Option D
E- Option E

Answer:

B

Explanation:

Option B correctly fills in the blank to meet the specified requirements. Option B uses the ''cloudFiles.schemaLocation'' option, which is required for the schema detection and evolution functionality of Databricks Auto Loader. Additionally, option B uses the ''mergeSchema'' option, which is required for the schema evolution functionality of Databricks Auto Loader. Finally, option B uses the ''writeStream'' method, which is required for the incremental processing of JSON files as they arrive in a source directory. The other options are incorrect because they either omit the required options, use the wrong method, or use the wrong format.Reference:

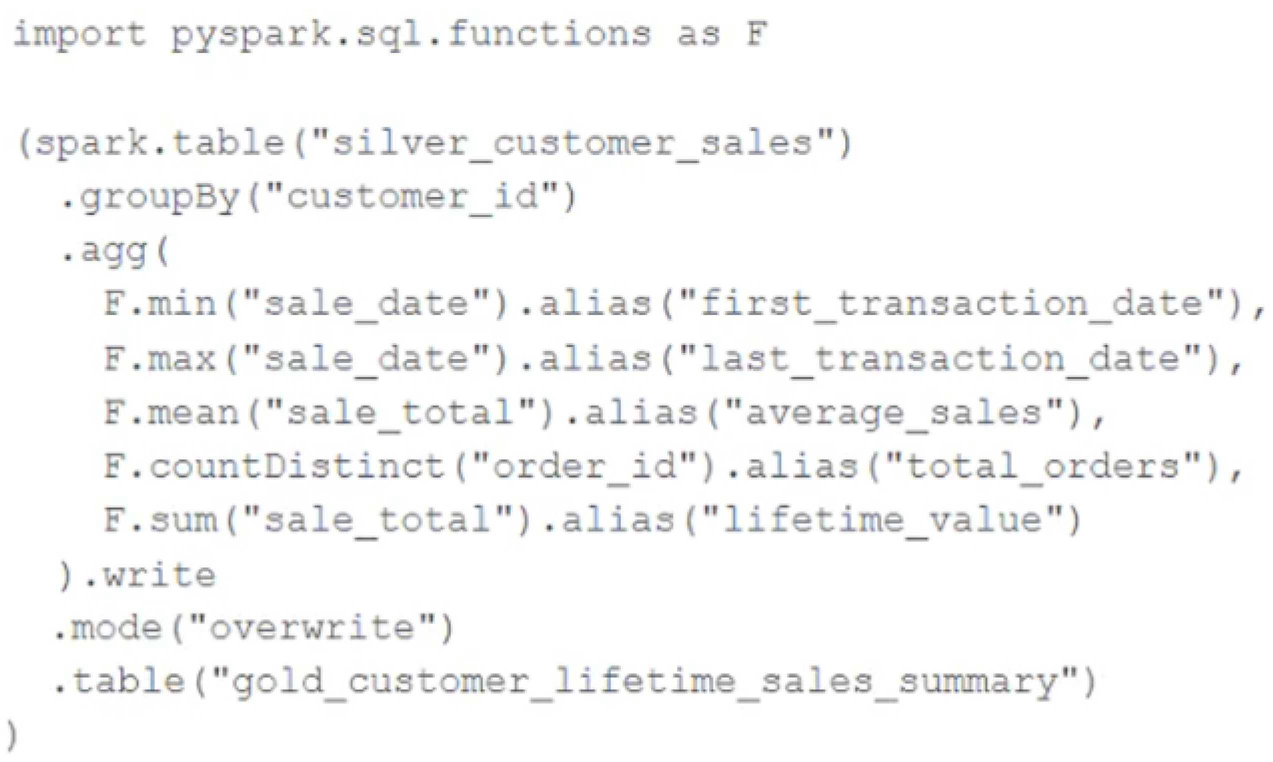
Configure schema inference and evolution in Auto Loader:
https://docs.databricks.com/en/ingestion/auto-loader/schema.html

Write streaming data:
https://docs.databricks.com/spark/latest/structured-streaming/writing-streaming-data.html

Question 4

Question Type: MultipleChoice

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?

Options:

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.Reference:

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

Question 5

Question Type: MultipleChoice

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 task A fails during a scheduled run, which statement describes the results of this run?

Options:

A- Because all tasks are managed as a dependency graph, no changes will be committed to the

Lakehouse until all tasks have successfully been completed.
B- Tasks B and C will attempt to run as configured; any changes made in task A will be rolled back due to task failure.
C- Unless all tasks complete successfully, no changes will be committed to the Lakehouse; because task A failed, all commits will be rolled back automatically.
D- Tasks B and C will be skipped; some logic expressed in task A may have been committed before task failure.
E- Tasks B and C will be skipped; task A will not commit any changes because of stage failure.

Answer:

D

Explanation:

When a Databricks job runs multiple tasks with dependencies, the tasks are executed in a dependency graph. If a task fails, the downstream tasks that depend on it are skipped and marked as Upstream failed. However, the failed task may have already committed some changes to the Lakehouse before the failure occurred, and those changes are not rolled back automatically. Therefore, the job run may result in a partial update of the Lakehouse. To avoid this, you can use thetransactional writesfeature of Delta Lake to ensure that the changes are only committed when the entire job run succeeds. Alternatively, you can use theRun ifcondition to configure tasks to run even when some or all of their dependencies have failed, allowing your job to recover from failures and continue running.Reference:

transactional writes: https://docs.databricks.com/delta/delta-intro.html#transactional-writes

Run if: https://docs.databricks.com/en/workflows/jobs/conditional-tasks.html

Question 6

Question Type: MultipleChoice

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?

Options:

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.

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