

# Databricks

## Exam Questions Databricks-Certified-Professional-Data-Engineer

Databricks Certified Data Engineer Professional Exam



**NEW QUESTION 1**

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

The silver\_device\_recordings table 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 encoding used by Databricks is optimized for storing string data; newly- added native support for querying JSON strings means that string types 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 writing code is the largest cost associated with data engineering workloads; as such, automating table 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.

**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.

**NEW QUESTION 2**

Which of the following technologies can be used to identify key areas of text when parsing Spark Driver log4j output?

- A. Regex
- B. Julia
- C. pyspark.ml.feature
- D. Scala Datasets
- E. C++

**Answer:** A

**Explanation:**

Regex, or regular expressions, are a powerful way of matching patterns in text. They can be used to identify key areas of text when parsing Spark Driver log4j output, such as the log level, the timestamp, the thread name, the class name, the method name, and the message. Regex can be applied in various languages and frameworks, such as Scala, Python, Java, Spark SQL, and Databricks notebooks. References:

? <https://docs.databricks.com/notebooks/notebooks-use.html#use-regular-expressions>

? <https://docs.databricks.com/spark/latest/spark-sql/udf-scala.html#using-regular-expressions-in-udfs>

? [https://docs.databricks.com/spark/latest/sparkr/functions/regexp\\_extract.html](https://docs.databricks.com/spark/latest/sparkr/functions/regexp_extract.html)

? [https://docs.databricks.com/spark/latest/sparkr/functions/regexp\\_replace.html](https://docs.databricks.com/spark/latest/sparkr/functions/regexp_replace.html)

**NEW QUESTION 3**

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?

- 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.

References:

? <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>

**NEW QUESTION 4**

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. The user\_id field 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 named account\_history which maintains a full record of all data in the same schema as the source. The next table in the system is named account\_current and is implemented as a Type 1 table representing the most recent value for each unique user\_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 described account\_current table as part of each hourly batch job?

- A. Use Auto Loader to subscribe to new files in the account history directory; configure a Structured Streaming 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 login 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.

**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.

**NEW QUESTION 5**

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

The silver\_device\_recordings table will be used downstream for highly selective joins on a number of fields, and will also be leveraged by the machine learning team to filter on a handful of relevant fields, in total, 15 fields have been identified that will often be used for filter and join logic.

The data engineer is trying to determine the best approach for dealing with these nested fields before declaring the table schema.

Which of the following accurately presents information about Delta Lake and Databricks that may impact their decision-making process?

- A. Because Delta Lake uses Parquet for data storage, Dremel encoding information for nesting can be directly referenced by the Delta transaction log.
- B. Tungsten encoding used by Databricks is optimized for storing string data: newly-added native support for querying JSON strings means that string types are always most efficient.
- C. Schema inference and evolution on Databricks ensure that inferred types will always accurately match the data types used by downstream systems.
- D. By default Delta Lake collects statistics on the first 32 columns in a table; these statistics are leveraged for data skipping when executing selective queries.

**Answer: D**

**Explanation:**

Delta Lake, built on top of Parquet, enhances query performance through data skipping, which is based on the statistics collected for each file in a table. For tables with a large number of columns, Delta Lake by default collects and stores statistics only for the first 32 columns. These statistics include min/max values and null counts, which are used to optimize query execution by skipping irrelevant data files. When dealing with highly nested JSON structures, understanding this behavior is crucial for schema design, especially when determining which fields should be flattened or prioritized in the table structure to leverage data skipping efficiently for performance optimization. References: Databricks documentation on Delta Lake optimization techniques, including data skipping and statistics collection (<https://docs.databricks.com/delta/optimizations/index.html>).

**NEW QUESTION 6**

To reduce storage and compute costs, the data engineering team has been tasked with curating a series of aggregate tables leveraged by business intelligence dashboards, customer-facing applications, production machine learning models, and ad hoc analytical queries.

The data engineering team has been made aware of new requirements from a customer-facing application, which is the only downstream workload they manage entirely. As a result, an aggregate table used by numerous teams across the organization will need to have a number of fields renamed, and additional fields will also be added.

Which of the solutions addresses the situation while minimally interrupting other teams in the organization without increasing the number of tables that need to be managed?

- A. Send all users notice that the schema for the table will be changing; include in the communication the logic necessary to revert the new table schema to match historic queries.
- B. Configure a new table with all the requisite fields and new names and use this as the source for the customer-facing application; create a view that maintains the original data schema and table name by aliasing select fields from the new table.
- C. Create a new table with the required schema and new fields and use Delta Lake's deep clone functionality to sync up changes committed to one table to the corresponding table.
- D. Replace the current table definition with a logical view defined with the query logic currently writing the aggregate table; create a new table to power the customer-facing application.
- E. Add a table comment warning all users that the table schema and field names will be changing on a given date; overwrite the table in place to the specifications of the customer-facing application.

**Answer: B**

**Explanation:**

This is the correct answer because it addresses the situation while minimally interrupting other teams in the organization without increasing the number of tables that need to be managed. The situation is that an aggregate table used by numerous teams across the organization will need to have a number of fields renamed, and additional fields will also be added, due to new requirements from a customer-facing application. By configuring a new table with all the requisite fields and new names and using this as the source for the customer-facing application, the data engineering team can meet the new requirements without affecting other teams that rely on the existing table schema and name. By creating a view that maintains the original data schema and table name by aliasing select fields from the new table, the data engineering team can also avoid duplicating data or creating additional tables that need to be managed. Verified References: [Databricks Certified Data Engineer Professional], under “Lakehouse” section; Databricks Documentation, under “CREATE VIEW” section.

**NEW QUESTION 7**

A developer has successfully configured credential for Databricks Repos and cloned a remote Git repository. They do 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 change were pulled.

- 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 repository.
- 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.

References:

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

**NEW QUESTION 8**

What is a method of installing a Python package scoped at the notebook level to all nodes in the currently active cluster?

- A. Use %pip install in a notebook cell
- B. Run source env/bin/activate in a notebook setup script
- C. Install libraries from PyPi using the cluster UI
- D. Use %sh install in a notebook cell

**Answer: C**

**Explanation:**

Installing a Python package scoped at the notebook level to all nodes in the currently active cluster in Databricks can be achieved by using the Libraries tab in the cluster UI. This interface allows you to install libraries across all nodes in the cluster. While the %pip command in a notebook cell would only affect the driver node, using the cluster UI ensures that the package is installed on all nodes.

References:

? Databricks Documentation on Libraries: Libraries

**NEW QUESTION 9**

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 named preds with 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)

```
(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 A
- B. Option B
- C. Option C
- D. Option D
- E. Option E

**Answer:** A

#### NEW QUESTION 10

A data architect has designed a system in which two Structured Streaming jobs will concurrently write to a single bronze Delta table. Each job is subscribing to a different topic from an Apache Kafka source, but they will write data with the same schema. To keep the directory structure simple, a data engineer has decided to nest a checkpoint directory to be shared by both streams.

The proposed directory structure is displayed below:

Which statement describes whether this checkpoint directory structure is valid for the given scenario and why?

- A. No; Delta Lake manages streaming checkpoints in the transaction log.
- B. Yes; both of the streams can share a single checkpoint directory.
- C. No; only one stream can write to a Delta Lake table.
- D. Yes; Delta Lake supports infinite concurrent writers.
- E. No; each of the streams needs to have its own checkpoint directory.

**Answer:** E

#### Explanation:

This is the correct answer because checkpointing is a critical feature of Structured Streaming that provides fault tolerance and recovery in case of failures. Checkpointing stores the current state and progress of a streaming query in a reliable storage system, such as DBFS or S3. Each streaming query must have its own checkpoint directory that is unique and exclusive to that query. If two streaming queries share the same checkpoint directory, they will interfere with each other and cause unexpected errors or data loss. Verified References: [Databricks Certified Data Engineer Professional], under "Structured Streaming" section; Databricks Documentation, under "Checkpointing" section.

#### NEW QUESTION 10

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"?

- A. Model, predict (df, columns)
- B. Df, map (lambda k: model (x [columns]) ,select ("customer\_id predictions"))
- C. D
- D. Select ("customer\_id". Model ("columns) alias ("predictions"))
- E. 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.

References:

? 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>

#### NEW QUESTION 13

A CHECK constraint has been successfully added to the Delta table named activity\_details using the following logic:

A batch job is attempting to insert new records to the table, including a record where latitude = 45.50 and longitude = 212.67.

Which statement describes the outcome of this batch insert?

- A. The write will fail when the violating record is reached; any records previously processed will be recorded to the target table.
- B. The write will fail completely because of the constraint violation and no records will be inserted into the target table.
- C. The write will insert all records except those that violate the table constraints; the violating records will be recorded to a quarantine table.
- D. The write will include all records in the target table; any violations will be indicated in the boolean column named valid\_coordinates.
- E. The write will insert all records except those that violate the table constraints; the violating records will be reported in a warning log.

**Answer:** B

**Explanation:**

The CHECK constraint is used to ensure that the data inserted into the table meets the specified conditions. In this case, the CHECK constraint is used to ensure that the latitude and longitude values are within the specified range. If the data does not meet the specified conditions, the write operation will fail completely and no records will be inserted into the target table. This is because Delta Lake supports ACID transactions, which means that either all the data is written or none of it is written. Therefore, the batch insert will fail when it encounters a record that violates the constraint, and the target table will not be updated. References:

? Constraints: <https://docs.delta.io/latest/delta-constraints.html>

? ACID Transactions: <https://docs.delta.io/latest/delta-intro.html#acid-transactions>

**NEW QUESTION 15**

A table named user\_ltv is being used to create a view that will be used by data analysis on various teams. Users in the workspace are configured into groups, which are used for setting up data access using ACLs.

The user\_ltv table has the following schema:

```
email STRING, age INT, ltv INT
```

The following view definition is executed:

```
CREATE VIEW user_ltv_no_minors AS
SELECT email, age, ltv
FROM user_ltv
WHERE
  CASE
    WHEN is_member("auditing") THEN TRUE
    ELSE age >= 18
  END
```

An analyze who is not a member of the auditing group executing the following query:

```
SELECT * FROM user_ltv_no_minors
```

Which result will be returned by this query?

- A. All columns will be displayed normally for those records that have an age greater than 18; records not meeting this condition will be omitted.
- B. All columns will be displayed normally for those records that have an age greater than 17; records not meeting this condition will be omitted.
- C. All age values less than 18 will be returned as null values all other columns will be returned with the values in user\_ltv.
- D. All records from all columns will be displayed with the values in user\_ltv.

**Answer:** A

**Explanation:**

Given the CASE statement in the view definition, the result set for a user not in the auditing group would be constrained by the ELSE condition, which filters out records based on age. Therefore, the view will return all columns normally for records with an age greater than 18, as users who are not in the auditing group will not satisfy the is\_member('auditing') condition. Records not meeting the age > 18 condition will not be displayed.

**NEW QUESTION 16**

The downstream consumers of a Delta Lake table have been complaining about data quality issues impacting performance in their applications. Specifically, they have complained that invalid latitude and longitude values in the activity\_details table have been breaking their ability to use other geolocation processes.

A junior engineer has written the following code to add CHECK constraints to the Delta Lake table:

```
ALTER TABLE activity_details
ADD CONSTRAINT valid_coordinates
CHECK (
  latitude >= -90 AND
  latitude <= 90 AND
  longitude >= -180 AND
  longitude <= 180);
```

A senior engineer has confirmed the above logic is correct and the valid ranges for latitude and longitude are provided, but the code fails when executed. Which statement explains the cause of this failure?

- A. Because another team uses this table to support a frequently running application, two- phase locking is preventing the operation from committing.
- B. The activity details table already exists; CHECK constraints can only be added during initial table creation.
- C. The activity details table already contains records that violate the constraints; all existing data must pass CHECK constraints in order to add them to an existing table.
- D. The activity details table already contains records; CHECK constraints can only be added prior to inserting values into a table.
- E. The current table schema does not contain the field valid coordinates; schema evolution will need to be enabled before altering the table to add a constraint.

**Answer:** C

**Explanation:**

The failure is that the code to add CHECK constraints to the Delta Lake table fails when executed. The code uses ALTER TABLE ADD CONSTRAINT commands to add two CHECK constraints to a table named activity\_details. The first constraint checks if the latitude value is between -90 and 90, and the second constraint

checks if the longitude value is between -180 and 180. The cause of this failure is that the activity\_details table already contains records that violate these constraints, meaning that they have invalid latitude or longitude values outside of these ranges. When adding CHECK constraints to an existing table, Delta Lake verifies that all existing data satisfies the constraints before adding them to the table. If any record violates the constraints, Delta Lake throws an exception and aborts the operation. Verified References: [Databricks Certified Data Engineer Professional], under “Delta Lake” section; Databricks Documentation, under “Add a CHECK constraint to an existing table” section. <https://docs.databricks.com/en/sql/language-manual/sql-ref-syntax-ddl-alter-table.html#add-constraint>

#### NEW QUESTION 20

Which statement regarding stream-static joins and static Delta tables is correct?

- A. Each microbatch of a stream-static join will use the most recent version of the static Delta table as of each microbatch.
- B. Each microbatch of a stream-static join will use the most recent version of the static Delta table as of the job's initialization.
- C. The checkpoint directory will be used to track state information for the unique keys present in the join.
- D. Stream-static joins cannot use static Delta tables because of consistency issues.
- E. The checkpoint directory will be used to track updates to the static Delta table.

**Answer:** A

#### Explanation:

This is the correct answer because stream-static joins are supported by Structured Streaming when one of the tables is a static Delta table. A static Delta table is a Delta table that is not updated by any concurrent writes, such as appends or merges, during the execution of a streaming query. In this case, each microbatch of a stream-static join will use the most recent version of the static Delta table as of each microbatch, which means it will reflect any changes made to the static Delta table before the start of each microbatch. Verified References: [Databricks Certified Data Engineer Professional], under “Structured Streaming” section; Databricks Documentation, under “Stream and static joins” section.

#### NEW QUESTION 25

A data engineer, User A, has promoted a new pipeline to production by using the REST API to programmatically create several jobs. A DevOps engineer, User B, has configured an external orchestration tool to trigger job runs through the REST API. Both users authorized the REST API calls using their personal access tokens.

Which statement describes the contents of the workspace audit logs concerning these events?

- A. Because the REST API was used for job creation and triggering runs, a Service Principal will be automatically used to identify these events.
- B. Because User B last configured the jobs, their identity will be associated with both the job creation events and the job run events.
- C. Because these events are managed separately, User A will have their identity associated with the job creation events and User B will have their identity associated with the job run events.
- D. Because the REST API was used for job creation and triggering runs, user identity will not be captured in the audit logs.
- E. Because User A created the jobs, their identity will be associated with both the job creation events and the job run events.

**Answer:** C

#### Explanation:

The events are that a data engineer, User A, has promoted a new pipeline to production by using the REST API to programmatically create several jobs, and a DevOps engineer, User B, has configured an external orchestration tool to trigger job runs through the REST API. Both users authorized the REST API calls using their personal access tokens. The workspace audit logs are logs that record user activities in a Databricks workspace, such as creating, updating, or deleting objects like clusters, jobs, notebooks, or tables. The workspace audit logs also capture the identity of the user who performed each activity, as well as the time and details of the activity. Because these events are managed separately, User A will have their identity associated with the job creation events and User B will have their identity associated with the job run events in the workspace audit logs. Verified References: [Databricks Certified Data Engineer Professional], under “Databricks Workspace” section; Databricks Documentation, under “Workspace audit logs” section.

#### NEW QUESTION 27

A team of data engineer are adding tables to a DLT pipeline that contain repetitive expectations for many of the same data quality checks. One member of the team suggests reusing these data quality rules across all tables defined for this pipeline.

What approach would allow them to do this?

- A. Maintain data quality rules in a Delta table outside of this pipeline's target schema, providing the schema name as a pipeline parameter.
- B. Use global Python variables to make expectations visible across DLT notebooks included in the same pipeline.
- C. Add data quality constraints to tables in this pipeline using an external job with access to pipeline configuration files.
- D. Maintain data quality rules in a separate Databricks notebook that each DLT notebook of file.

**Answer:** A

#### Explanation:

Maintaining data quality rules in a centralized Delta table allows for the reuse of these rules across multiple DLT (Delta Live Tables) pipelines. By storing these rules outside the pipeline's target schema and referencing the schema name as a pipeline parameter, the team can apply the same set of data quality checks to different tables within the pipeline. This approach ensures consistency in data quality validations and reduces redundancy in code by not having to replicate the same rules in each DLT notebook or file. References:

? Databricks Documentation on Delta Live Tables: Delta Live Tables Guide

#### NEW QUESTION 31

A Spark job is taking longer than expected. Using the Spark UI, a data engineer notes that the Min, Median, and Max Durations for tasks in a particular stage show the minimum and median time to complete a task as roughly the same, but the max duration for a task to be roughly 100 times as long as the minimum.

Which situation is causing increased duration of the overall job?

- A. Task queueing resulting from improper thread pool assignment.
- B. Spill resulting from attached volume storage being too small.
- C. Network latency due to some cluster nodes being in different regions from the source data
- D. Skew caused by more data being assigned to a subset of spark-partitions.
- E. Credential validation errors while pulling data from an external system.



**Answer:** D

**Explanation:**

This is the correct answer because skew is a common situation that causes increased duration of the overall job. Skew occurs when some partitions have more data than others, resulting in uneven distribution of work among tasks and executors. Skew can be caused by various factors, such as skewed data distribution, improper partitioning strategy, or join operations with skewed keys. Skew can lead to performance issues such as long-running tasks, wasted resources, or even task failures due to memory or disk spills. Verified References: [Databricks Certified Data Engineer Professional], under “Performance Tuning” section; Databricks Documentation, under “Skew” section.

**NEW QUESTION 35**

The data engineer team has been tasked with configured connections to an external database that does not have a supported native connector with Databricks. The external database already has data security configured by group membership. These groups map directly to user group already created in Databricks that represent various teams within the company.

A new login credential has been created for each group in the external database. The Databricks Utilities Secrets module will be used to make these credentials available to Databricks users.

Assuming that all the credentials are configured correctly on the external database and group membership is properly configured on Databricks, which statement describes how teams can be granted the minimum necessary access to using these credentials?

- A. “Read” permissions should be set on a secret key mapped to those credentials that will be used by a given team.
- B. No additional configuration is necessary as long as all users are configured as administrators in the workspace where secrets have been added.
- C. “Read” permissions should be set on a secret scope containing only those credentials that will be used by a given team.
- D. “Manage” permission should be set on a secret scope containing only those credentials that will be used by a given team.

**Answer:** C

**Explanation:**

In Databricks, using the Secrets module allows for secure management of sensitive information such as database credentials. Granting 'Read' permissions on a secret key that maps to database credentials for a specific team ensures that only members of that team can access these credentials. This approach aligns with the principle of least privilege, granting users the minimum level of access required to perform their jobs, thus enhancing security.

References:

? Databricks Documentation on Secret Management: Secrets

**NEW QUESTION 37**

A table in the Lakehouse named customer\_churn\_params is used in churn prediction by the machine learning team. The table contains information about customers derived from a number of upstream sources. Currently, the data engineering team populates this table nightly by overwriting the table with the current valid values derived from upstream data sources.

The churn prediction model used by the ML team is fairly stable in production. The team is only interested in making predictions on records that have changed in the past 24 hours.

Which approach would simplify the identification of these changed records?

- A. Apply the churn model to all rows in the customer\_churn\_params table, but implement logic to perform an upsert into the predictions table that ignores rows where predictions have not changed.
- B. Convert the batch job to a Structured Streaming job using the complete output mode; configure a Structured Streaming job to read from the customer\_churn\_params table and incrementally predict against the churn model.
- C. Calculate the difference between the previous model predictions and the current customer\_churn\_params on a key identifying unique customers before making new predictions; only make predictions on those customers not in the previous predictions.
- D. Modify the overwrite logic to include a field populated by calling spark.sql.functions.current\_timestamp() as data are being written; use this field to identify records written on a particular date.
- E. Replace the current overwrite logic with a merge statement to modify only those records that have changed; write logic to make predictions on the changed records identified by the change data feed.

**Answer:** E

**Explanation:**

The approach that would simplify the identification of the changed records is to replace the current overwrite logic with a merge statement to modify only those records that have changed, and write logic to make predictions on the changed records identified by the change data feed. This approach leverages the Delta Lake features of merge and change data feed, which are designed to handle upserts and track row-level changes in a Delta table<sup>12</sup>. By using merge, the data engineering team can avoid overwriting the entire table every night, and only update or insert the records that have changed in the source data. By using change data feed, the ML team can easily access the change events that have occurred in the customer\_churn\_params table, and filter them by operation type (update or insert) and timestamp. This way, they can only make predictions on the records that have changed in the past 24 hours, and avoid re-processing the unchanged records. The other options are not as simple or efficient as the proposed approach, because:

? Option A would require applying the churn model to all rows in the customer\_churn\_params table, which would be wasteful and redundant. It would also require implementing logic to perform an upsert into the predictions table, which would be more complex than using the merge statement.

? Option B would require converting the batch job to a Structured Streaming job, which would involve changing the data ingestion and processing logic. It would also require using the complete output mode, which would output the entire result table every time there is a change in the source data, which would be inefficient and costly.

? Option C would require calculating the difference between the previous model predictions and the current customer\_churn\_params on a key identifying unique customers, which would be computationally expensive and prone to errors. It would also require storing and accessing the previous predictions, which would add extra storage and I/O costs.

? Option D would require modifying the overwrite logic to include a field populated by calling spark.sql.functions.current\_timestamp() as data are being written, which would add extra complexity and overhead to the data engineering job. It would also require using this field to identify records written on a particular date, which would be less accurate and reliable than using the change data feed.

References: Merge, Change data feed

**NEW QUESTION 39**

The security team is exploring whether or not the Databricks secrets module can be leveraged for connecting to an external database.

After testing the code with all Python variables being defined with strings, they upload the password to the secrets module and configure the correct permissions for the currently active user. They then modify their code to the following (leaving all other variables unchanged).



```
password = dbutils.secrets.get(scope="db_creds", key="jdbc_password")

print(password)

df = (spark
      .read
      .format("jdbc")
      .option("url", connection)
      .option("dbtable", tablename)
      .option("user", username)
      .option("password", password)
      )
```

Which statement describes what will happen when the above code is executed?

- A. The connection to the external table will fail; the string "redacted" will be printed.
- B. An interactive input box will appear in the notebook; if the right password is provided, the connection will succeed and the encoded password will be saved to DBFS.
- C. An interactive input box will appear in the notebook; if the right password is provided, the connection will succeed and the password will be printed in plain text.
- D. The connection to the external table will succeed; the string value of password will be printed in plain text.
- E. The connection to the external table will succeed; the string "redacted" will be printed.

**Answer:** E

**Explanation:**

This is the correct answer because the code is using the `dbutils.secrets.get` method to retrieve the password from the secrets module and store it in a variable. The secrets module allows users to securely store and access sensitive information such as passwords, tokens, or API keys. The connection to the external table will succeed because the password variable will contain the actual password value. However, when printing the password variable, the string "redacted" will be displayed instead of the plain text password, as a security measure to prevent exposing sensitive information in notebooks. Verified References: [Databricks Certified Data Engineer Professional], under "Security & Governance" section; Databricks Documentation, under "Secrets" section.

**NEW QUESTION 44**

Two of the most common data locations on Databricks are the DBFS root storage and external object storage mounted with `dbutils.fs.mount()`. Which of the following statements is correct?

- A. DBFS is a file system protocol that allows users to interact with files stored in object storage using syntax and guarantees similar to Unix file systems.
- B. By default, both the DBFS root and mounted data sources are only accessible to workspace administrators.
- C. The DBFS root is the most secure location to store data, because mounted storage volumes must have full public read and write permissions.
- D. Neither the DBFS root nor mounted storage can be accessed when using `%sh` in a Databricks notebook.
- E. The DBFS root stores files in ephemeral block volumes attached to the driver, while mounted directories will always persist saved data to external storage between sessions.

**Answer:** A

**Explanation:**

DBFS is a file system protocol that allows users to interact with files stored in object storage using syntax and guarantees similar to Unix file systems<sup>1</sup>. DBFS is not a physical file system, but a layer over the object storage that provides a unified view of data across different data sources<sup>1</sup>. By default, the DBFS root is accessible to all users in the workspace, and the access to mounted data sources depends on the permissions of the storage account or container<sup>2</sup>. Mounted storage volumes do not need to have full public read and write permissions, but they do require a valid connection string or access key to be provided when mounting<sup>3</sup>. Both the DBFS root and mounted storage can be accessed when using `%sh` in a Databricks notebook, as long as the cluster has FUSE enabled<sup>4</sup>. The DBFS root does not store files in ephemeral block volumes attached to the driver, but in the object storage associated with the workspace<sup>1</sup>. Mounted directories will persist saved data to external storage between sessions, unless they are unmounted or deleted<sup>3</sup>. References: DBFS, Work with files on Azure Databricks, Mounting cloud object storage on Azure Databricks, Access DBFS with FUSE

**NEW QUESTION 48**

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`

**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.

**NEW QUESTION 51**

A data architect has heard about lake's built-in versioning and time travel capabilities. For auditing purposes they have a requirement to maintain a full of all valid street addresses as they appear in the customers table.

The architect is interested in implementing a Type 1 table, overwriting existing records with new values and relying on Delta Lake time travel to support long-term auditing. A data engineer on the project feels that a Type 2 table will provide better performance and scalability.

Which piece of information is critical to this decision?

- A. Delta Lake time travel does not scale well in cost or latency to provide a long-term versioning solution.
- B. Delta Lake time travel cannot be used to query previous versions of these tables because Type 1 changes modify data files in place.
- C. Shallow clones can be combined with Type 1 tables to accelerate historic queries for long-term versioning.
- D. Data corruption can occur if a query fails in a partially completed state because Type 2 tables requiresSetting multiple fields in a single update.

**Answer:** A

**Explanation:**

Delta Lake's time travel feature allows users to access previous versions of a table, providing a powerful tool for auditing and versioning. However, using time travel as a long-term versioning solution for auditing purposes can be less optimal in terms of cost and performance, especially as the volume of data and the number of versions grow. For maintaining a full history of valid street addresses as they appear in a customers table, using a Type 2 table (where each update creates a new record with versioning) might provide better scalability and performance by avoiding the overhead associated with accessing older versions of a large table. While Type 1 tables, where existing records are overwritten with new values, seem simpler and can leverage time travel for auditing, the critical piece of information is that time travel might not scale well in cost or latency for long- term versioning needs, making a Type 2 approach more viable for performance and scalability. References:

? Databricks Documentation on Delta Lake's Time Travel: Delta Lake Time Travel

? Databricks Blog on Managing Slowly Changing Dimensions in Delta Lake: Managing SCDs in Delta Lake

**NEW QUESTION 54**

A nightly job ingests data into a Delta Lake table using the following code:

```
from pyspark.sql.functions import current_timestamp, input_file_name, col
from pyspark.sql.column import Column

def ingest_daily_batch(time_col: Column, year:int, month:int, day:int):
    (spark.read
     .format("parquet")
     .load(f"/mnt/daily_batch/{year}/{month}/{day}")
     .select("time_col.alias('ingest_time'),
            input_file_name().alias('source_file')
            )
     .write
     .mode("append")
     .saveAsTable("bronze"))
```

The next step in the pipeline requires a function that returns an object that can be used to manipulate new records that have not yet been processed to the next table in the pipeline.

Which code snippet completes this function definition? def new\_records():

- A. return spark.readStream.table("bronze")
- B. return spark.readStream.load("bronze")
- C. return (spark.read
 .table("bronze")
 .filter(col("ingest\_time") == current\_timestamp())
 )
- D. return spark.read.option("readChangeFeed", "true").table ("bronze")
- E. return (spark.read
 .table("bronze")
 .filter(col("source\_file") == f"/mnt/daily\_batch/{year}/{month}/{day}")
 )

**Answer:** E

**Explanation:**

<https://docs.databricks.com/en/delta/delta-change-data-feed.html>

**NEW QUESTION 59**

Which is a key benefit of an end-to-end test?

- A. It closely simulates real world usage of your application.
- B. It pinpoint errors in the building blocks of your application.
- C. It provides testing coverage for all code paths and branches.
- D. It makes it easier to automate your test suite

**Answer:** A

**Explanation:**

End-to-end testing is a methodology used to test whether the flow of an application, from start to finish, behaves as expected. The key benefit of an end-to-end test is that it closely simulates real-world, user behavior, ensuring that the system as a whole operates correctly.

References:

? Software Testing: End-to-End Testing

**NEW QUESTION 63**

A new data engineer notices that a critical field was omitted from an application that writes its Kafka source to Delta Lake. This happened even though the critical field was in the Kafka source. That field was further missing from data written to dependent, long-term storage. The retention threshold on the Kafka service is seven days. The pipeline has been in production for three months.

Which describes how Delta Lake can help to avoid data loss of this nature in the future?

- A. The Delta log and Structured Streaming checkpoints record the full history of the Kafka producer.
- B. Delta Lake schema evolution can retroactively calculate the correct value for newly added fields, as long as the data was in the original source.
- C. Delta Lake automatically checks that all fields present in the source data are included in the ingestion layer.
- D. Data can never be permanently dropped or deleted from Delta Lake, so data loss is not possible under any circumstance.
- E. Ingesting all raw data and metadata from Kafka to a bronze Delta table creates a permanent, replayable history of the data state.

**Answer:** E

**Explanation:**

This is the correct answer because it describes how Delta Lake can help to avoid data loss of this nature in the future. By ingesting all raw data and metadata from Kafka to a bronze Delta table, Delta Lake creates a permanent, replayable history of the data state that can be used for recovery or reprocessing in case of errors or omissions in downstream applications or pipelines. Delta Lake also supports schema evolution, which allows adding new columns to existing tables without affecting existing queries or pipelines. Therefore, if a critical field was omitted from an application that writes its Kafka source to Delta Lake, it can be easily added later and the data can be reprocessed from the bronze table without losing any information. Verified References: [Databricks Certified Data Engineer Professional], under “Delta Lake” section; Databricks Documentation, under “Delta Lake core features” section.

**NEW QUESTION 65**

Each configuration below is identical to the extent that each cluster has 400 GB total of RAM, 160 total cores and only one Executor per VM. Given a job with at least one wide transformation, which of the following cluster configurations will result in maximum performance?

- A. • Total VMs: 1 • 400 GB per Executor • 160 Cores / Executor
- B. • Total VMs: 8 • 50 GB per Executor • 20 Cores / Executor
- C. • Total VMs: 4 • 100 GB per Executor • 40 Cores/Executor
- D. • Total VMs: 2 • 200 GB per Executor • 80 Cores / Executor

**Answer:** B

**Explanation:**

This is the correct answer because it is the cluster configuration that will result in maximum performance for a job with at least one wide transformation. A wide transformation is a type of transformation that requires shuffling data across partitions, such as join, groupBy, or orderBy. Shuffling can be expensive and time-consuming, especially if there are too many or too few partitions. Therefore, it is important to choose a cluster configuration that can balance the trade-off between parallelism and network overhead. In this case, having 8 VMs with 50 GB per executor and 20 cores per executor will create 8 partitions, each with enough memory and CPU resources to handle the shuffling efficiently. Having fewer VMs with more memory and cores per executor will create fewer partitions, which will reduce parallelism and increase the size of each shuffle block. Having more VMs with less memory and cores per executor will create more partitions, which will increase parallelism but also increase the network overhead and the number of shuffle files. Verified References: [Databricks Certified Data Engineer Professional], under “Performance Tuning” section; Databricks Documentation, under “Cluster configurations” section.

**NEW QUESTION 69**

.....

## Thank You for Trying Our Product

### We offer two products:

1st - We have Practice Tests Software with Actual Exam Questions

2nd - Questions and Answers in PDF Format

### Databricks-Certified-Professional-Data-Engineer Practice Exam Features:

- \* Databricks-Certified-Professional-Data-Engineer Questions and Answers Updated Frequently
- \* Databricks-Certified-Professional-Data-Engineer Practice Questions Verified by Expert Senior Certified Staff
- \* Databricks-Certified-Professional-Data-Engineer Most Realistic Questions that Guarantee you a Pass on Your FirstTry
- \* Databricks-Certified-Professional-Data-Engineer Practice Test Questions in Multiple Choice Formats and Updatesfor 1 Year

**100% Actual & Verified — Instant Download, Please Click**  
**[Order The Databricks-Certified-Professional-Data-Engineer Practice Test Here](#)**