

# **DATABRICKS-CERTIFIED- PR OFSSIONAL-DATA-ENGINEER<sup>Q&As</sup>**

Databricks Certified Professional Data Engineer Exam

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

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

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

**100% Passing Guarantee**  
**100% Money Back Assurance**

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

- ⚙️ **Instant Download** After Purchase
- ⚙️ **100% Money Back** Guarantee
- ⚙️ **365 Days** Free Update
- ⚙️ **800,000+** Satisfied Customers



**QUESTION 1**

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.

Correct 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.

**QUESTION 2**

The data engineering team has configured a job to process customer requests to be forgotten (have their data deleted). All user data that needs to be deleted is stored in Delta Lake tables using default table settings.

The team has decided to process all deletions from the previous week as a batch job at 1am each Sunday. The total duration of this job is less than one hour. Every Monday at 3am, a batch job executes a series of VACUUM commands on all Delta Lake tables throughout the organization.

The compliance officer has recently learned about Delta Lake's time travel functionality. They are concerned that this might allow continued access to deleted data.

Assuming all delete logic is correctly implemented, which statement correctly addresses this concern?

- A. Because the vacuum command permanently deletes all files containing deleted records, deleted records may be accessible with time travel for around 24 hours.
- B. Because the default data retention threshold is 24 hours, data files containing deleted records will be retained until the vacuum job is run the following day.

- C. Because Delta Lake time travel provides full access to the entire history of a table, deleted records can always be recreated by users with full admin privileges.
- D. Because Delta Lake's delete statements have ACID guarantees, deleted records will be permanently purged from all storage systems as soon as a delete job completes.
- E. Because the default data retention threshold is 7 days, data files containing deleted records will be retained until the vacuum job is run 8 days later.

Correct Answer: A

Explanation: This is the correct answer because Delta Lake's delete statements do not physically remove the data files that contain the deleted records, but only mark them as logically deleted in the transaction log. These files are still

accessible with time travel until they are permanently deleted by the vacuum command. The default data retention threshold for vacuum is 7 days, but in this case it is overridden by setting it to 24 hours in each vacuum command. Therefore,

deleted records may be accessible with time travel for around 24 hours after they are deleted, until they are vacuumed. Verified References:

[Databricks Certified Data Engineer Professional], under "Delta Lake" section; [Databricks Documentation], under "Optimizations - Vacuum" section.

---

### QUESTION 3

Review the following error traceback:

```
-----  
AnalysisException                                Traceback (most recent call last)  
<command-3293767849433948> in <module>  
----> 1 display(df.select(3*"heartrate"))  
  
/databricks/spark/python/pyspark/sql/dataframe.py in select(self, *cols)  
    1690         [Row(name='Alice', age=12), Row(name='Bob', age=15)]  
    1691         """)  
-> 1692         jdf = self._jdf.select(self._jcols(*cols))  
    1693         return DataFrame(jdf, self.sql_ctx)  
    1694  
  
/databricks/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py in __call__(self, *args)  
    1302  
    1303         answer = self.gateway_client.send_command(command)  
-> 1304         return_value = get_return_value(  
    1305             answer, self.gateway_client, self.target_id, self.name)  
    1306  
  
/databricks/spark/python/pyspark/sql/utils.py in deco(*a, **kw)  
    121         # Hide where the exception came from that shows a non-Pythonic  
    122         # JVM exception message.  
--> 123         raise converted from None  
    124     else:  
    125         raise  
  
AnalysisException: cannot resolve 'heartrateheartrateheartrate' given input columns:  
[spark_catalog.database.table.device_id, spark_catalog.database.table.heartrate,  
spark_catalog.database.table.mrn, spark_catalog.database.table.time];  
'Project ['heartrateheartrateheartrate]  
+- SubqueryAlias spark_catalog.database.table  
   +- Relation[device_id#75L,heartrate#76,mrn#77L,time#78] parquet
```

Which statement describes the error being raised?

- A. The code executed was PySoark but was executed in a Scala notebook.
- B. There is no column in the table named heartrateheartrateheartrate
- C. There is a type error because a column object cannot be multiplied.
- D. There is a type error because a DataFrame object cannot be multiplied.
- E. There is a syntax error because the heartrate column is not correctly identified as a column.

Correct Answer: E

Explanation: The error is a Py4JJavaError, which means that an exception was thrown in Java code called by Python code using Py4J. Py4J is a library that enables Python programs to dynamically access Java objects in a Java Virtual

Machine (JVM). PySpark uses Py4J to communicate with Spark's JVM-based engine. The error message shows that the exception was thrown by org.apache.spark.sql.AnalysisException, which means that an error occurred during the

analysis phase of Spark SQL query processing. The error message also shows that the cause of the exception was "cannot resolve 'heartrateheartrateheartrate' given input columns". This means that Spark could not find a column named

heartrateheartrateheartrate in the input DataFrame or Dataset. The reason for this error is that there is a syntax error in the code that caused this exception. The code

is:

```
df.withColumn("heartrate", heartrate * 3)
```

The code tries to create a new column called heartrate by multiplying an existing column called heartrate by 3. However, the code does not correctly identify the heartrate column as a column object, but rather as a plain Python variable. This

causes PySpark to concatenate the variable name with itself three times, resulting in heartrateheartrateheartrate, which is not a valid column name. To fix this error, the code should use one of the following ways to identify the heartrate

column as a column object:

```
df.withColumn("heartrate", df["heartrate"] * 3) df.withColumn("heartrate", df.heartrate * 3) df.withColumn("heartrate", col("heartrate") * 3)
```

Verified References: [Databricks Certified Data Engineer Professional], under "Spark Core" section; Py4J Documentation, under "What is Py4J?"; Databricks Documentation, under "Query plans - Analysis phase"; Databricks Documentation, under "Accessing columns".

#### QUESTION 4

The DevOps team has configured a production workload as a collection of notebooks scheduled to run daily using the Jobs UI. A new data engineering hire is onboarding to the team and has requested access to one of these notebooks to review the production logic.

What are the maximum notebook permissions that can be granted to the user without allowing accidental changes to production code or data?

- A. Can Manage
- B. Can Edit
- C. No permissions
- D. Can Read
- E. Can Run

Correct Answer: D

Explanation: This is the correct answer because it is the maximum notebook permissions that can be granted to the user without allowing accidental changes to production code or data. Notebook permissions are used to control access to notebooks in Databricks workspaces. There are four types of notebook permissions: Can Manage, Can Edit, Can Run, and Can Read. Can Manage allows full control over the notebook, including editing, running, deleting, exporting, and changing permissions. Can Edit allows modifying and running the notebook, but not changing permissions or deleting it. Can Run allows executing commands in an existing cluster attached to the notebook, but not modifying or exporting it. Can Read allows viewing the notebook content, but not running or modifying it. In this case, granting Can Read permission to the user will allow them to review the production logic in the notebook without allowing them to make any changes to it or run any commands that may affect production data. Verified References: [Databricks Certified Data Engineer Professional], under "Databricks Workspace" section; Databricks Documentation, under "Notebook

permissions" section.

### QUESTION 5

A junior developer complains that the code in their notebook isn't producing the correct results in the development environment. A shared screenshot reveals that while they're using a notebook versioned with Databricks Repos, they're using a personal branch that contains old logic. The desired branch named dev-2.3.9 is not available from the branch selection dropdown.

Which approach will allow this developer to review the current logic for this notebook?

- A. Use Repos to make a pull request use the Databricks REST API to update the current branch to dev-2.3.9
- B. Use Repos to pull changes from the remote Git repository and select the dev-2.3.9 branch.
- C. Use Repos to checkout the dev-2.3.9 branch and auto-resolve conflicts with the current branch
- D. Merge all changes back to the main branch in the remote Git repository and clone the repo again
- E. Use Repos to merge the current branch and the dev-2.3.9 branch, then make a pull request to sync with the remote repository

Correct Answer: B

Explanation: This is the correct answer because it will allow the developer to update their local repository with the latest changes from the remote repository and switch to the desired branch. Pulling changes will not affect the current branch or create any conflicts, as it will only fetch the changes and not merge them. Selecting the dev-2.3.9 branch from the dropdown will checkout that branch and display its contents in the notebook. Verified References: [Databricks Certified Data Engineer Professional], under "Databricks Tooling" section; Databricks Documentation, under "Pull changes from a remote repository" section.

### QUESTION 6

A junior data engineer has configured a workload that posts the following JSON to the Databricks REST API endpoint `2.0/jobs/create`.

```
{
  "name": "Ingest new data",
  "existing_cluster_id": "6015-954420-peace720",
  "notebook_task": {
    "notebook_path": "/Prod/ingest.py"
  }
}
```

Assuming that all configurations and referenced resources are available, which statement describes the result of executing this workload three times?

- A. Three new jobs named "Ingest new data" will be defined in the workspace, and they will each run once daily.
- B. The logic defined in the referenced notebook will be executed three times on new clusters with the configurations of

the provided cluster ID.

- C. Three new jobs named "Ingest new data" will be defined in the workspace, but no jobs will be executed.
- D. One new job named "Ingest new data" will be defined in the workspace, but it will not be executed.
- E. The logic defined in the referenced notebook will be executed three times on the referenced existing all purpose cluster.

Correct Answer: C

Explanation: This is the correct answer because the JSON posted to the Databricks REST API endpoint `2.0/jobs/create` defines a new job with a name, an existing cluster id, and a notebook task. However, it does not specify any schedule or trigger for the job execution. Therefore, three new jobs with the same name and configuration will be created in the workspace, but none of them will be executed until they are manually triggered or scheduled. Verified References: [Databricks Certified Data Engineer Professional], under "Monitoring and Logging" section; [Databricks Documentation], under "Jobs API - Create" section.

---

## QUESTION 7

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 8

The data engineering team maintains the following code:



```
accountDF = spark.table("accounts")
orderDF = spark.table("orders")
itemDF = spark.table("items")

orderWithItemDF = (orderDF.join(
    itemDF,
    orderDF.itemID == itemDF.itemID)
    .select(
        orderDF.accountID,
        orderDF.itemID,

        itemDF.itemName))

finalDF = (accountDF.join(
    orderWithItemDF,
    accountDF.accountID == orderWithItemDF.accountID)
    .select(
        orderWithItemDF["*"],

        accountDF.city))

(finalDF.write
    .mode("overwrite")
    .table("enriched_itemized_orders_by_account"))
```

Assuming that this code produces logically correct results and the data in the source tables has been de-duplicated and validated, which statement describes what will occur when this code is executed?

- A. A batch job will update the enriched\_itemized\_orders\_by\_account table, replacing only those rows that have different values than the current version of the table, using accountID as the primary key.
- B. The enriched\_itemized\_orders\_by\_account table will be overwritten using the current valid version of data in each of the three tables referenced in the join logic.
- C. An incremental job will leverage information in the state store to identify unjoined rows in the source tables and write these rows to the enriched\_itemized\_orders\_by\_account table.
- D. An incremental job will detect if new rows have been written to any of the source tables; if new rows are detected, all results will be recalculated and used to overwrite the enriched\_itemized\_orders\_by\_account table.

E. No computation will occur until `enriched_itemized_orders_by_account` is queried; upon query materialization, results will be calculated using the current valid version of data in each of the three tables referenced in the join logic.

Correct Answer: B

Explanation: This is the correct answer because it describes what will occur when this code is executed. The code uses three Delta Lake tables as input sources: `accounts`, `orders`, and `order_items`. These tables are joined together using SQL queries to create a view called `new_enriched_itemized_orders_by_account`, which contains information about each order item and its associated account details. Then, the code uses `write.format("delta").mode("overwrite")` to overwrite a target table called `enriched_itemized_orders_by_account` using the data from the view. This means that every time this code is executed, it will replace all existing data in the target table with new data based on the current valid version of data in each of the three input tables. Verified References: [Databricks Certified Data Engineer Professional], under "Delta Lake" section; Databricks Documentation, under "Write to Delta tables" section.

### QUESTION 9

A table named `user_ltv` is being used to create a view that will be used by data analysts 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 email_ltv AS
SELECT
CASE WHEN
    is_member('marketing') THEN email
    ELSE 'REDACTED'
END AS email,
ltv
FROM user_ltv
```

An analyst who is not a member of the marketing group executes the following query:

```
SELECT * FROM email_ltv
```

Which statement describes the results returned by this query?

- A. Three columns will be returned, but one column will be named "redacted" and contain only null values.
- B. Only the email and ltv columns will be returned; the email column will contain all null values.
- C. The email and ltv columns will be returned with the values in user ltv.
- D. The email, age, and ltv columns will be returned with the values in user ltv.

E. Only the email and ltv columns will be returned; the email column will contain the string "REDACTED" in each row.

Correct Answer: E

Explanation: The code creates a view called email\_ltv that selects the email and ltv columns from a table called user\_ltv, which has the following schema: email STRING, age INT, ltv INT. The code also uses the CASE WHEN expression to replace the email values with the string "REDACTED" if the user is not a member of the marketing group. The user who executes the query is not a member of the marketing group, so they will only see the email and ltv columns, and the email column will contain the string "REDACTED" in each row. Verified References: [Databricks Certified Data Engineer Professional], under "Lakehouse" section; Databricks Documentation, under "CASE expression" section.

## QUESTION 10

The data engineering team is migrating an enterprise system with thousands of tables and views into the Lakehouse. They plan to implement the target architecture using a series of bronze, silver, and gold tables. Bronze tables will almost exclusively be used by production data engineering workloads, while silver tables will be used to support both data engineering and machine learning workloads. Gold tables will largely serve business intelligence and reporting purposes. While personal identifying information (PII) exists in all tiers of data, pseudonymization and anonymization rules are in place for all data at the silver and gold levels.

The organization is interested in reducing security concerns while maximizing the ability to collaborate across diverse teams.

Which statement exemplifies best practices for implementing this system?

- A. Isolating tables in separate databases based on data quality tiers allows for easy permissions management through database ACLs and allows physical separation of default storage locations for managed tables.
- B. Because databases on Databricks are merely a logical construct, choices around database organization do not impact security or discoverability in the Lakehouse.
- C. Storing all production tables in a single database provides a unified view of all data assets available throughout the Lakehouse, simplifying discoverability by granting all users view privileges on this database.
- D. Working in the default Databricks database provides the greatest security when working with managed tables, as these will be created in the DBFS root.
- E. Because all tables must live in the same storage containers used for the database they're created in, organizations should be prepared to create between dozens and thousands of databases depending on their data isolation requirements.

Correct Answer: A

Explanation: This is the correct answer because it exemplifies best practices for implementing this system. By isolating tables in separate databases based on data quality tiers, such as bronze, silver, and gold, the data engineering team can achieve several benefits. First, they can easily manage permissions for different users and groups through database ACLs, which allow granting or revoking access to databases, tables, or views. Second, they can physically separate the default storage locations for managed tables in each database, which can improve performance and reduce costs. Third, they can provide a clear and consistent naming convention for the tables in each database, which can improve discoverability and usability. Verified References: [Databricks Certified Data Engineer Professional], under "Lakehouse" section; Databricks Documentation, under "Database object privileges" section.

[Latest DATABRICKS-CERT](#) [DATABRICKS-CERTIFIED-](#) [DATABRICKS-CERTIFIED-](#)

[DATABRICKS-CERTIFIED-PROFESSIONAL-DATA-ENGINEER Dumps](#)

[DATABRICKS-CERTIFIED-PROFESSIONAL-DATA-ENGINEER VCE Dumps](#)

[DATABRICKS-CERTIFIED-PROFESSIONAL-DATA-ENGINEER Practice Test](#)