DUMPSARENA

Databricks Certified Data Analyst Associate Exam

Databricks Databricks-Certified-Data-Analyst-Associate

Version Demo

Total Demo Questions: 16

Total Premium Questions: 167

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Data Management	79
Topic 2, Data Analysis	55
Topic 3, Data Visualization and Reporting	33
Total	167

QUESTION NO: 1

Which of the following is a benefit of Databricks SQL using ANSI SQL as its standard SQL dialect?

A. It has increased customization capabilities

B. It is easy to migrate existing SQL queries to Databricks SQL

C. It allows for the use of Photon's computation optimizations

D. It is more performant than other SQL dialects

E. It is more compatible with Spark's interpreters

ANSWER: B

Explanation:

It is easy to migrate existing SQL queries to Databricks SQL is correct because ANSI SQL provides a broadly recognized, standards-based foundation for SQL syntax and behavior. Organizations commonly move workloads from established data warehouse platforms to Databricks, and alignment with ANSI SQL reduces the amount of query rewriting and retraining required during that transition. Analysts can apply familiar constructs, expressions, joins, aggregations, and other standard SQL patterns with greater confidence that their intent will transfer cleanly to Databricks SQL.

Databricks uses ANSI SQL compliance to improve consistency with the SQL standard, including more rigorous type handling and predictable SQL semantics. This standards-oriented approach helps teams modernize analytics workloads while retaining existing SQL skills and much of their existing query logic. Databricks also supports a range of commonly used SQL capabilities that facilitate migrations from data warehouses, enabling users to adopt the platform without having to learn an entirely proprietary query language first. The result is a lower-friction path for bringing established reporting, dashboarding, and analytical workloads into the lakehouse. See the Databricks documentation on ANSI compliance and the Databricks article on ANSI SQL and data warehouse migrations.

QUESTION NO: 2

A data engineering team has created a Structured Streaming pipeline that processes data in micro-batches and populates gold-level tables. The microbatches are triggered every minute.

A data analyst has created a dashboard based on this gold-level data. The project stakeholders want to see the results in the dashboard updated within one minute or less of new data becoming available within the gold-level tables.

Which of the following cautions should the data analyst share prior to setting up the dashboard to complete this task?

A. The required compute resources could be costly

B. The gold-level tables are not appropriately clean for business reporting

C. The streaming data is not an appropriate data source for a dashboard

D. The streaming cluster is not fault tolerant

E. The dashboard cannot be refreshed that quickly

ANSWER: A

Explanation:

The required compute resources could be costly. Meeting a near-real-time dashboard expectation requires more than the streaming pipeline’s one-minute micro-batch trigger: the dashboard’s underlying query must also be executed frequently enough to detect and display each newly committed update to the gold tables. Those recurring query executions require an available SQL warehouse or other supported compute resource. Frequent refreshes can prevent a warehouse from auto-stopping or cause it to start repeatedly, and the resulting DBU and cloud-resource consumption can become material, particularly when dashboard queries scan large gold tables or serve many viewers.

The analyst should therefore set expectations about the cost-versus-latency tradeoff before implementing the refresh pattern. They should estimate the query runtime and refresh frequency, select an appropriately sized warehouse, and consider query optimization or pre-aggregated gold tables so that refreshes complete reliably within the requested latency target. Databricks documents that SQL warehouses consume DBUs while running and that dashboard refresh schedules execute the associated dataset queries. These operational characteristics make compute cost the relevant caution when stakeholders request minute-level visibility. See Databricks SQL warehouse types and billing behavior and Databricks dashboard refresh scheduling.

QUESTION NO: 3

Consider the following two statements:

Statement 1:

Statement 2:

Which of the following describes how the result sets will differ for each statement when they are run in Databricks SQL?

A. The first statement will return all data from the customers table and matching data from the orders table. The second statement will return all data from the orders table and matching data from the customers table. Any missing data will be filled in with NULL.

B. When the first statement is run, only rows from the customers table that have at least one match with the orders table on customer_id will be returned. When the second statement is run, only those rows in the customers table that do not have at least one match with the orders table on customer_id will be returned.

C. There is no difference between the result sets for both statements.

D. Both statements will fail because Databricks SQL does not support those join types.

E. When the first statement is run, all rows from the customers table will be returned and only the customer_id from the orders table will be returned. When the second statement is run, only those rows in the customers table that do not have at least one match with the orders table on customer_id will be returned.

ANSWER: B

Explanation:

“When the first statement is run, only rows from the customers table that have at least one match with the orders table on customer_id will be returned. When the second statement is run, only those rows in the customers table that do not have at least one match with the orders table on customer_id will be returned.” is correct because a LEFT SEMI JOIN and a LEFT ANTI JOIN both evaluate whether each row in the left relation has a matching row in the right relation using the join condition. A LEFT SEMI JOIN returns the left-side rows for which at least one match exists. In this case, it identifies customers that have placed one or more orders. The result contains columns from the customers relation only; the right-side relation is used to determine match existence rather than to contribute output columns. A LEFT ANTI JOIN performs the complementary existence test: it returns left-side rows for which no matching right-side row exists. Here, it identifies customers with no orders matching their customer_id. These join forms are particularly useful for existence and nonexistence analysis without needing to return order details or create duplicate customer rows when multiple matching orders exist. Databricks SQL explicitly supports both LEFT SEMI and LEFT ANTI join types in its join syntax and semantics. See the Databricks SQL JOIN reference and the Apache Spark SQL JOIN documentation.

QUESTION NO: 4

A data team has been given a series of projects by a consultant that need to be implemented in the Databricks Lakehouse Platform.

Which of the following projects should be completed in Databricks SQL?

A. Testing the quality of data as it is imported from a source

B. Tracking usage of feature variables for machine learning projects

C. Combining two data sources into a single, comprehensive dataset

D. Segmenting customers into like groups using a clustering algorithm

E. Automating complex notebook-based workflows with multiple tasks

ANSWER: C

Explanation:

Combining two data sources into a single, comprehensive dataset is an appropriate Databricks SQL workload because Databricks SQL is designed for analysts and other SQL users to query, transform, and analyze data in the lakehouse. A team can use standard SQL statements such as `JOIN`, `UNION`, common table expressions, and views to integrate records from different tables or supported data sources into a curated analytical dataset. The resulting dataset can be persisted as a table, exposed through a view, queried by downstream users, and used as the source for Databricks SQL visualizations and dashboards.

Databricks SQL works with data managed in Unity Catalog and supports querying many data formats through the lakehouse architecture. This makes SQL a practical interface for producing a unified business-facing dataset—for example, enriching transaction records with customer attributes or consolidating related operational datasets for reporting. Databricks documentation describes Databricks SQL as a service for running SQL commands on data objects and creating visualizations and dashboards, while its SQL language documentation covers joins and other relational operations used to combine datasets. See the Databricks SQL documentation and the Databricks SQL JOIN reference.

QUESTION NO: 5

Delta Lake stores table data as a series of data files, but it also stores a lot of other information. Which of the following is stored alongside data files when using Delta Lake?

A. None of these

B. Table metadata, data summary visualizations, and owner account information

C. Table metadata

D. Data summary visualizations

E. Owner account information

ANSWER: C

Explanation:

Table metadata is stored alongside Delta Lake data files in the transaction log, located in the table's `_delta_log` directory. Delta Lake stores the table's actual records in Parquet data files, while the transaction log records the metadata and transactional state needed to interpret those files as a reliable table. This log includes information such as the table schema, partition columns, table configuration, protocol versions, and references to the data files that make up each committed version of the table. It also records transactional changes, including which files were added or removed by operations such as writes, updates, deletes, merges, and optimizations.

This metadata is fundamental to Delta Lake's ACID transaction guarantees and enables capabilities such as schema enforcement, consistent reads, auditing of table history, and time travel to prior table versions. When a query reads a Delta table, the Delta transaction log determines the valid set of data files for the requested table version rather than treating every file in the storage location as current table data. Databricks documents the Delta transaction log as the source of truth for a Delta table's state and metadata. See Work with Delta Lake table history and What is Delta Lake?.

QUESTION NO: 6

A data analyst is working with the following table my_table:

customer_name dollars_spent

Hex Sprockets [125.34, 100.15, 9003.99]

Dented Fenders [16.99, 200.85, 33.49, 88.17]

The analyst wants to divide each value in the dollars_spent array by 100 to get the spend in terms of hundreds of dollars using the following code block:

SELECT

customer_name,

_____

FROM my_table;

Which line of code can be used to fill in the blank so that the above code block successfully completes the task?

A. transform(dollars_spent, value -> value / 100) AS hundreds_spent

ANSWER: A

Explanation:

`transform(dollars_spent, value -> value / 100) AS hundreds_spent` is the appropriate expression because `transform` is a Databricks SQL higher-order function for processing every element of an array while preserving an array result. Its first argument is the input array, `dollars_spent`. Its second argument is a lambda expression, in which `value` represents one array element at a time and `value / 100` defines the replacement value to emit for that element. Thus, a value such as `125.34` becomes `1.2534`, and the complete result remains an array containing the converted spend amounts for each customer. The alias `hundreds_spent` gives the derived array a clear output-column name in the query result. Databricks documents `transform` as applying a function to each array element and returning an array, and documents lambda functions as the syntax used by higher-order functions to express per-element logic. See the Databricks transform function reference and the Databricks lambda functions reference.

QUESTION NO: 7

A distributed team of data analysts share computing resources on an interactive cluster with autoscaling configured. In order to better manage costs and query throughput, the workspace administrator is hoping to evaluate whether cluster upscaling is caused by many concurrent users or resource-intensive queries.

In which location can one review the timeline for cluster resizing events?

A. Workspace audit logs

B. Driver's log file

C. Ganglia

D. Cluster Event Log

E. Executor's log file

ANSWER: D

Explanation:

Cluster Event Log is the correct location because Databricks records operational changes to a cluster as cluster events, providing a chronological activity history. For an autoscaling interactive cluster, this timeline includes resize-related activity, such as requests to add or remove workers and the corresponding completion or failure events. Reviewing these events lets an administrator establish exactly when scaling occurred and correlate those times with workload behavior, including periods of heavy concurrent usage or queries that require substantial compute resources. The event history is accessed from the cluster's details page in the Databricks UI, where the Events tab displays cluster lifecycle and configuration activity. This

makes it the appropriate source for investigating the timing of autoscaling decisions and supporting cost and throughput analysis. Databricks documents that cluster events can be retrieved through the cluster events API and include activities associated with a cluster's operation. Its autoscaling documentation also identifies resize events, including events such as *Cluster resize request started*, as part of autoscaling behavior. See the Databricks cluster events API documentation and the Databricks documentation for managing clusters.

QUESTION NO: 8

A data analyst has opened the SQL Editor page and written a new SQL statement. The data analyst now wants to save that statement to easily refer back to it later and add it to a dashboard. The results of the SQL statement must be able to be displayed as a counter, table, or data visualization.

Which approach should the data analyst use to accomplish this task?

A. Save the SQL statement as a Dashboard

B. Save the SQL statement within a Notebook

C. Save the SQL statement as a Query

D. Save the SQL statement in the Query History page

ANSWER: C

Explanation:

Save the SQL statement as a Query. In Databricks SQL, a query is a saved SQL statement that can be named, organized, reopened, edited, and rerun from the workspace. Saving it as a query provides the reusable object needed for both future reference and dashboard development. After running the query, the analyst can configure an appropriate visualization of its result set, including a table, chart, or a single-value/counter-style visualization when the query returns a suitable aggregate value. The saved query can then be added to a Databricks SQL dashboard as a dataset or visualization, allowing the dashboard to present refreshed query results to its viewers. This workflow separates the reusable SQL logic from the dashboard presentation layer: the query defines and retrieves the data, while the dashboard arranges the resulting tables, counters, and charts. Databricks documents that queries created in the SQL editor can be saved and that saved queries can be used to create visualizations and dashboard content. See the Databricks SQL queries documentation and the Databricks dashboards documentation for the supported query and visualization workflow.

QUESTION NO: 9

A data engineering team has created a Structured Streaming pipeline that processes data in micro-batches and populates gold-level tables. The microbatches are triggered every 10 minutes.

A data analyst has created a dashboard based on this gold level data. The project stakeholders want to see the results in the dashboard updated within 10 minutes or less of new data becoming available within the gold-level tables.

What is the ability to ensure the streamed data is included in the dashboard at the standard requested by the project stakeholders?

A. A refresh schedule with an interval of 10 minutes or less

B. A refresh schedule with an always-on SQL Warehouse (formerly known as SQL Endpoint

C. A refresh schedule with stakeholders included as subscribers

D. A refresh schedule with a Structured Streaming cluster

ANSWER: A

Explanation:

A refresh schedule with an interval of 10 minutes or less is the appropriate capability because it causes the dashboard's underlying queries to run frequently enough to incorporate newly committed data from the gold-level tables. The streaming pipeline makes new data available on its micro-batch cadence, and the dashboard refresh schedule controls when the dashboard re-executes its queries and updates the results presented to users. By selecting a cadence no longer than 10 minutes, the analyst aligns dashboard refreshes with the stakeholder freshness requirement and ensures newly available gold-table data is picked up on the next scheduled refresh.

In practice, the selected SQL warehouse must be available and have sufficient capacity for the dashboard queries to complete promptly. The refresh interval establishes the maximum planned delay before the dashboard checks the tables again; query execution time should also be considered when validating the end-to-end service level. Databricks documents how dashboard schedules refresh query results automatically and identifies the warehouse used to run those refreshes. See Schedule a dashboard refresh and Configure SQL warehouses.

QUESTION NO: 10

A data organization has a team of engineers developing data pipelines following the medallion architecture using Delta Live Tables. While the data analysis team working on a project is using goldlayer tables from these pipelines, they need to perform some additional processing of these tables prior to performing their analysis.

Which of the following terms is used to describe this type of work?

A. Data blending

B. Last-mile

C. Data testing

D. Last-mile ETL

E. Data enhancement

ANSWER: D

Explanation:

Last-mile ETL is the term for additional, use-case-specific data preparation performed after a centrally managed pipeline has produced curated data. In this scenario, the engineering team has already made gold-layer tables available through its medallion-architecture pipeline. The analysis team's further processing is therefore not the core ingestion and transformation work of the shared pipeline; it is the final preparation needed to make trusted gold data suitable for a particular analytical question, dashboard, report, or model.

Last-mile ETL commonly includes operations such as selecting a relevant subset of records, joining gold tables with project-specific sources, applying business-specific filters, deriving metrics, aggregating results, or reshaping data for analysis. Analysts can perform this work in Databricks SQL or notebooks while retaining the governed, reusable gold tables as their upstream source. This separation enables data engineering teams to maintain reliable shared datasets while allowing analytics teams to address the specific requirements of their own projects without altering the central pipeline.

Databricks describes the medallion architecture and the purpose of gold-layer data in its Medallion architecture documentation. Additional guidance on preparing and querying data for analytics is available in the Databricks SQL documentation.

QUESTION NO: 11

A data engineer wants to schedule their Databricks SQL dashboard to refresh once per day, but they only want the associated SQL endpoint to be running when it is necessary.

Which of the following approaches can the data engineer use to minimize the total running time of the SQL endpoint used in the refresh schedule of their dashboard?

A. They can ensure the dashboardâ€™s SQL endpoint matches each of the queriesâ€™ SQL endpoints.

B. They can set up the dashboardâ€™s SQL endpoint to be serverless.

C. They can turn on the Auto Stop feature for the SQL endpoint.

D. They can reduce the cluster size of the SQL endpoint.

E. They can ensure the dashboardâ€™s SQL endpoint is not one of the included queryâ€™s SQL endpoint.

ANSWER: C

Explanation:

They can turn on the Auto Stop feature for the SQL endpoint. Auto Stop is the SQL warehouse setting designed to limit how long compute remains active after work has finished. The data engineer configures an idle timeout, and the warehouse automatically stops after it has had no activity for that configured number of minutes. When the daily dashboard refresh runs, the warehouse can start to execute the dashboard's queries and then stop automatically once the refresh has completed and the idle timeout elapses. This directly supports the requirement that the endpoint run only when needed, reducing the time during which warehouse compute is running.

The timeout should be selected to accommodate any expected follow-up queries or refresh activity while avoiding an unnecessarily long idle period. Databricks documents that the auto-stop setting controls whether a SQL warehouse stops after being idle for a specified duration, and that warehouses can start automatically when a query requires them. This makes Auto Stop appropriate for scheduled, periodic dashboard workloads where keeping compute continuously available is unnecessary. See Configure SQL warehouses and Databricks SQL warehouses documentation.

QUESTION NO: 12

A data analyst is working with gold-layer tables to complete an ad-hoc project. A stakeholder has provided the analyst with an additional dataset that can be used to augment the gold-layer tables already in use.

Which of the following terms is used to describe this data augmentation?

A. Data testing

B. Ad-hoc improvements

C. Last-mile

D. Last-mile ETL

E. Data enhancement

ANSWER: D

Explanation:

Last-mile ETL describes the project-specific data preparation that an analyst performs after receiving curated, business-ready gold-layer data. Gold tables are designed to provide refined, high-quality data for reporting, analytics, and other consumption workloads. However, an ad-hoc analysis can require information that is not present in the centrally managed gold tables, such as a stakeholder-provided file, a department-specific reference dataset, or a project-specific metric.

In this situation, the analyst combines the additional dataset with the existing gold-layer tables and applies any necessary shaping, joins, calculations, or quality handling needed for that specific analysis. This work is termed “last-mile” because it occurs close to the final business consumption use case rather than being a broadly reusable transformation in the shared data pipeline. It enables analysts to meet a particular stakeholder need while preserving the centrally curated gold layer as the trusted foundation.

Databricks describes the gold layer as containing data that has been refined and aggregated for analytics and reporting workloads. See the Databricks medallion architecture documentation and the Databricks medallion architecture overview.

QUESTION NO: 13

A data analyst wants to return all rows from the `transactions` table where the `transaction_date` occurred during January 2025.

Which predicate can the analyst use to include dates from January 1, 2025 through January 31, 2025?

A. transaction_date BETWEEN DATE '2025-01-01' AND DATE '2025-01-31'

B. transaction_date IN DATE '2025-01-01', DATE '2025-01-31'

C. transaction_date = DATE '2025-01-01' OR DATE '2025-01-31'

D. transaction_date AFTER DATE '2025-01-01' BEFORE DATE '2025-01-31'

ANSWER: A

Explanation:

`transaction_date BETWEEN DATE '2025-01-01' AND DATE '2025-01-31'` is correct because `BETWEEN` includes both boundary values. When `transaction_date` is a DATE column, this predicate returns dates on or after January 1 and on or before January 31.

Using typed date literals makes the intended comparison explicit and avoids relying on implicit string conversion. If the column were a timestamp rather than a date, an analyst would commonly use an inclusive lower boundary and exclusive next-month boundary to account for times during the final day. See the Databricks SQL documentation for BETWEEN and DATE type.

QUESTION NO: 14

A data analysis team is working with the table_bronze SQL table as a source for one of its most complex projects. A stakeholder of the project notices that some of the downstream data is duplicative. The analysis team identifies table_bronze as the source of the duplication.

Which of the following queries can be used to deduplicate the data from table_bronze and write it to a new table table_silver?

A)

CREATE TABLE table_silver AS

SELECT DISTINCT *

FROM table_bronze;

B)

CREATE TABLE table_silver AS

INSERT *

FROM table_bronze;

C)

CREATE TABLE table_silver AS

MERGE DEDUPLICATE *

FROM table_bronze;

D)

INSERT INTO TABLE table_silver

SELECT * FROM table_bronze;

E)

INSERT OVERWRITE TABLE table_silver

SELECT * FROM table_bronze;

A. CREATE TABLE table_silver ASSELECT DISTINCT *FROM table_bronze;

B. CREATE TABLE table_silver ASINSERT *FROM table_bronze;

C. CREATE TABLE table_silver ASMERGE DEDUPLICATE *FROM table_bronze;

D. INSERT INTO TABLE table_silverSELECT * FROM table_bronze;

E. INSERT OVERWRITE TABLE table_silverSELECT * FROM table_bronze;

ANSWER: A

Explanation:

`CREATE TABLE table_silver AS SELECT DISTINCT * FROM table_bronze;` is the appropriate query because it combines a create-table-as-select operation with row-level deduplication. In Databricks SQL, `CREATE TABLE ... AS query` creates a new table and populates it with the result set returned by the query. This makes it suitable when the target, `table_silver`, is intended to be newly materialized from the bronze source.

The `DISTINCT` keyword removes duplicate rows from the query result. Because the query uses `*`, rows are considered duplicates only when every selected column value is the same. The resulting `table_silver` therefore contains one copy of each distinct full row found in `table_bronze`. This is a straightforward deduplication pattern when complete-row duplicates are the issue and no business key, recency rule, or record-precedence logic has been specified. If deduplication needed to retain one record per key according to a timestamp or other ordering rule, a window function such as `ROW_NUMBER` would generally be required instead. Databricks documents both the `CREATE TABLE AS SELECT` syntax and the behavior of `SELECT DISTINCT` in its SQL language reference: CREATE TABLE and SELECT.

QUESTION NO: 15

A data analyst has written and saved a series of queries that reveal trends that need to be monitored by several stakeholders.

Which tool should the data analyst use to share the results of all of the queries to be viewed at once?

A. A SQL warehouse

B. A Query History page

C. A dashboard

D. A data visualization tab on a Query page

ANSWER: C

Explanation:

A dashboard is the appropriate tool because it brings together the results of multiple saved queries in one shareable, organized view. The analyst can add visualizations and result tables from the saved queries to a single dashboard, allowing stakeholders to monitor related trends without opening each query individually. Dashboards are designed for communicating insights to broader audiences: they can be shared with users or groups, configured with permissions, and refreshed so viewers see current query results. This makes a dashboard especially suitable when several stakeholders need a consolidated, at-a-glance view of ongoing metrics and trends. Databricks SQL dashboards also support presentation-oriented layout and visualization choices, enabling the analyst to provide context and make the combined results easier to interpret. For details on creating and sharing dashboards, see the Databricks dashboards documentation. For information about the underlying saved queries used as dashboard datasets, see Databricks SQL queries documentation.

QUESTION NO: 16

A data analyst is processing a complex aggregation on a table with zero null values and the query returns the following result:

group_1	group_2	sum
null	null	100
A	null	50
A	Y	30
A	Z	20
B	null	50
B	Y	40
B	Z	1

Which query did the analyst execute in order to get this result?

A)

```
SELECT
    group_1,
    group_2,
    sum(values) AS sum
FROM my_table
GROUP BY group_1, group_2;
```

B)

```
SELECT
    group_1,
    group_2,
    sum(values) AS sum
FROM my_table
GROUP BY group_1, group_2 INCLUDING NULL;
```

C)

```
SELECT
    group_1,
    group_2,
    sum(values) AS sum
FROM my_table
GROUP BY group_1, group_2 INCLUDING NULL;
```

D)

```
SELECT
    group_1,
    group_2,
    sum(values) AS sum
FROM my_table
GROUP BY group_1, group_2 WITH CUBE;
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: D

Explanation:

The query represented by **Option D** is correct because `WITH CUBE` produces multiple aggregation levels from the specified grouping columns. For two grouping columns, a cube calculates the detailed aggregation for both columns together, a subtotal for each value of the first column, a subtotal for each value of the second column, and a grand total. In the result, the `NULL` values are therefore not source-table nulls. Instead, they are placeholders in the output that indicate a grouping column has been rolled up for a subtotal or grand-total row.

Databricks SQL supports `CUBE` as an advanced grouping expression in the `GROUP BY` clause. Conceptually, `CUBE` expands to all grouping sets (all combinations) for its listed grouping expressions. Thus, when the input table has no null values but the result includes rows with nulls alongside aggregated measures, a cube operation directly accounts for that shape of output. This is particularly useful for analytical reports that require detailed values, subtotals across either dimension, and an overall total in a single query result. See the Databricks GROUP BY clause reference and GROUPING SETS, ROLLUP, and CUBE reference.