

Databricks Certified Data Engineer Associate Renewal Exam

[Provide Exam Guide Feedback](#)

Update Information

Updates	Date
Exam Published	Jun 15, 2026
Exam Guide Updated	Aug 6, 2026

Please check the exam guide on our webpages two weeks before your exam date to ensure you are preparing for the current version of the exam.

Purpose of this Exam Guide

The purpose of this exam guide is to give you an overview of the exam and what it covers, to help you determine your exam readiness. This document is updated whenever there are changes to the exam (and indicates when those changes take effect) so that you can prepare with confidence. This guide covers the exam version that is live as of the **effective date** listed above.

Audience Description

This Databricks Certified Data Engineer Associate Renewal exam assesses an individual's ability to utilize the Databricks Platform to execute foundational data engineering tasks. The exam assesses a Databricks Certified Data Engineer Associate's knowledge of the tasks related to Data Ingestion, Data Loading, Data Transformation, and Modeling– such as the ability to perform Extract, Transform, Load (ETL) tasks using PySpark / SQL, working with Lakeflow Jobs, and CI/CD. Finally, the exam assesses test takers' understanding of troubleshooting, monitoring, and optimization techniques, as well as their knowledge of how to achieve Governance and Security within the Databricks Platform.

About the Exam

Item	Detail
Number of scored items	25 scored multiple-choice questions
Possible unscored items	None
Total number of questions you may see	25 scored questions
Time limit	60 minutes
Registration fee	USD 200, plus applicable taxes as required per local law
Delivery method	Online Proctored Only
Languages offered	English, 日本語 (Japanese), Português BR (Brazilian Portuguese), 한국어 (Korean)
Test aides	None allowed
Prerequisite	This renewal exam is available to individuals who hold an existing Databricks Certified Data Engineer Associate credential that has either expired or is within 2 months of expiration.
Recommended experience	At least 12 months of hands-on experience with the Databricks Platform, aligned to the listed exam outline, is highly recommended.
Validity	2 years

Recertification: Recertification is required every two years to maintain your certified status. To recertify, you can take the Data Engineer Associate Renewal Exam or the current full exam.

Unscored content: Exams may include unscored items to gather statistical information for future use. These items are not identified on the exam and do not affect your score, and additional time is factored into the time limit to account for them.

Recommended Preparation

Training

- **Instructor-led:** Data Engineering with Databricks
- **Self-paced (available in Databricks Academy):**
 - Data Ingestion with Lakeflow Connect
 - Deploy Workloads with Lakeflow Jobs

- DevOps Essentials for Data Engineering
- Build Data Pipelines with Apache Spark™ Declarative pipeline
- Get Started with Data Governance on Databricks

AI Prep Guide – Any Databricks Exam

The [Databricks AI Prep Guide](#) shows you how to turn the AI chatbot you already use into a personalized tutor for any Databricks certification: prime it with the official exam guide, catch the outdated answers AI tends to give, find your weak spots, and pair it all with the hands-on practice every Databricks exam demands.

Hands-On Practice

Not sure what to practice? The AI Prep Guide (see the AI Prep Guide section above) includes a prompt under “Step 5 — Hands-on minimum” that has an AI chatbot generate a personalized checklist of 6–10 hands-on tasks mapped to this exam’s objectives, each small enough to complete in Databricks [Free Edition](#).

Databricks Documentation

As you prepare, study the official Databricks documentation for the platform areas covered on this exam. It is the most current, authoritative source. Use it to judge how deeply to study each topic and to ensure you are learning up-to-date, non-deprecated features rather than out-of-date material.

Official documentation: [Databricks documentation](#)

Exam Outline

Section 1: Data Ingestion and Loading (32%)

- Use COPY INTO to perform incremental file ingestion from cloud object storage into Unity Catalog-governed Delta tables.
- Use Auto Loader to incrementally ingest files into Unity Catalog-governed Delta tables
- Configure Lakeflow Connect to reliably ingest data from diverse enterprise sources into Unity Catalog-governed tables.
- Use JDBC/ODBC or REST clients in notebooks to land data into cloud storage or directly into Unity Catalog-governed tables.
- Prioritize between Auto Loader, Lakeflow Connect partner connectors, and other ingestion methods based on volume, frequency, data type, and governance needs.
- Ingest semi-structured and unstructured data (e.g., JSON and nested data) via Lakeflow Connect and managed connectors into Delta tables.

Section 2: Data Transformation and Modeling (24%)

- Apply Data Cleaning, reading bronze tables with PySpark/SQL (handle nulls, standardize data types), and write the results to silver tables.
- Identify common Spark tuning parameters and understand their general effect on performance.
- Differentiate Gold-layer objects in Unity Catalog (materialized views, views, streaming tables, regular tables) and their use for BI/analytics.
- Apply Data quality checks and validation rules to ensure reliable Silver and Gold datasets

Section 3: Working with Lakeflow Jobs (12%)

- Implement control flows (task retries, branching, looping) using Lakeflow Jobs to orchestrate multi-step data workflows
- Configure common task types and dependencies in Lakeflow Jobs using a DAG-based workflow
- Configure Lakeflow Jobs schedules and triggers, including scheduled, file-arrival, table-update, and continuous triggers.

Section 4: Implementing CI/CD (12%)

- Manage code in Databricks workspace UI using Git Folders: create/switch branches, commit, push, and open pull requests.
- Use Declarative Automation Bundle environment targets (dev, test, prod) to promote a codebase across environments.
- Use the Databricks CLI to validate and deploy Declarative Automation Bundles (DABs)

Section 5: Troubleshooting, Monitoring, and Optimization (12%)

- Identify trends in job performance using the Lakeflow Jobs run history view.
- Use the Lakeflow Jobs UI to monitor pipeline health, job statuses, DAG task graphs, run times, and failure rates.
- Identify the purpose and benefits of Liquid Clustering and predictive optimization as managed Delta Lake features.

Section 6: Governance and Security (8%)

- Understand the purpose of column-level masking and row-level security and identify when they should be applied.
- Understand the features of Unity Catalog ABAC policies and how they support centralized data access control.

Sample Questions

The purpose of sample questions is to show you the objectives as they are stated in the exam guide and to give you a sample question that aligns with each objective. The exam guide lists the objectives that could be covered on an exam; the best way to prepare is to review the full exam outline for your exam date above.

Question 1:

Objective: *Use COPY INTO to perform incremental file ingestion from cloud object storage into Unity Catalog-governed Delta tables.*

A data engineer needs to load only new CSV files that have arrived in an external cloud object storage location into a Unity Catalog-governed Delta table, without reprocessing files that were already ingested in previous runs. The engineer decides to use COPY INTO for this task.

Which configuration correctly implements incremental file ingestion using COPY INTO?

- A. `COPY INTO my_catalog.my_schema.my_table FROM 'abfss://container@storage.dfs.core.windows.net/data/' FILEFORMAT = CSV FORMAT_OPTIONS ('header' = 'true', 'inferSchema' = 'true') COPY_OPTIONS ('force' = 'false')`
- B. `COPY INTO my_catalog.my_schema.my_table FROM 'abfss://container@storage.dfs.core.windows.net/data/' FILEFORMAT = CSV FORMAT_OPTIONS ('header' = 'true', 'inferSchema' = 'true') COPY_OPTIONS ('force' = 'true')`
- C. `INSERT INTO my_catalog.my_schema.my_table SELECT * FROM read_files('abfss://container@storage.dfs.core.windows.net/data/', format => 'csv', header => 'true') WHERE _metadata.file_modification_time > (SELECT MAX(ingestion_time) FROM my_catalog.my_schema.my_table)`
- D. `COPY INTO my_catalog.my_schema.my_table FROM 'abfss://container@storage.dfs.core.windows.net/data/' FILEFORMAT = PARQUET COPY_OPTIONS ('force' = 'false')`

Question 2:

Objective: *Prioritize between Auto Loader, Lakeflow Connect partner connectors, and other ingestion methods based on volume, frequency, data type, and governance needs.*

A data engineer is designing an ingestion pipeline for a SaaS application that delivers new JSON files to a cloud object storage bucket at irregular intervals throughout the day, with file counts ranging from a few per hour during off-peak periods to thousands per hour during peak periods. The engineer needs the pipeline to automatically detect and process only new files, without reprocessing files already ingested, and to scale gracefully across both low- and high-volume periods.

Which ingestion method should the data engineer use?

- A. Write a scheduled batch job that runs every 15 minutes and uses `spark.read.json` to load all files from the storage path, filtering out previously seen files by comparing against a manually maintained log table.
- B. Use a Lakeflow Connect partner connector configured to pull data directly from the SaaS application's API, which provides built-in schema mapping and governance integration for the source's structured data.
- C. Configure Auto Loader with `cloudFiles` format to monitor the cloud storage path, using its file notification or directory listing mode to track ingested files via an internal checkpoint.
- D. Use `COPY INTO` with a Delta Lake target table to load files from the cloud storage path on a fixed schedule, relying on its idempotent load tracking to skip already-ingested files.

Question 3:

Objective: *Identify common Spark tuning parameters and understand their general effect on performance.*

A data engineer notices that a Spark job processing a large Delta Lake table is running slowly. After reviewing the Spark UI, the engineer observes that most tasks complete in under two seconds, but a small number of tasks take over three minutes, causing the entire stage to stall. The table has a skewed distribution on the `order_region` column, which is used as the join key. The engineer confirms that adaptive query execution's skew-join handling has been explicitly disabled in the cluster's Spark configuration.

Which configuration parameter should the data engineer set to address this performance problem?

- A. Set `spark.sql.shuffle.partitions` to a value equal to the number of executor cores available in the cluster.
- B. Set `spark.sql.adaptive.skewJoin.enabled` to `true` to allow the adaptive query engine to split skewed partitions automatically.
- C. Set `spark.sql.files.maxPartitionBytes` to a smaller value so that each task reads fewer bytes from the source files.
- D. Set `spark.databricks.delta.optimizeWrite.enabled` to `true` so that Delta Lake merges small files before the join is executed.

Question 4:

Objective: *Apply Data quality checks and validation rules to ensure reliable Silver and Gold datasets*

A data engineer is building a Silver layer table in a Lakeflow pipeline that ingests customer transaction records from a Bronze source. The engineer needs to enforce two data quality rules: transactions with a null `transaction_id` must be quarantined rather than dropped, and transactions with an amount less than or equal to zero must cause the pipeline to fail immediately.

Which two constraint declarations satisfy these requirements?

- A. Create two streaming tables from the same Bronze source with inverted filters — valid rows (transaction_id IS NOT NULL) go to the main table, null-transaction_id rows go to a quarantine table.
- B. CONSTRAINT valid_transaction_id EXPECT (transaction_id IS NOT NULL) ON VIOLATION DROP ROW
- C. CONSTRAINT positive_amount EXPECT (amount > 0) ON VIOLATION FAIL UPDATE
- D. Create two materialized views from the same Bronze source with inverted filters — valid rows (transaction_id IS NOT NULL) go to the main view, null-transaction_id rows go to a quarantine view.
- E. CONSTRAINT valid_transaction_id EXPECT (transaction_id IS NOT NULL) ON VIOLATION FAIL UPDATE

Question 5:

Objective: *Configure Lakeflow Jobs schedules and triggers, including scheduled, file-arrival, table-update, and continuous triggers.*

A data engineer needs to trigger a Lakeflow Jobs job automatically whenever new files land in a cloud object storage location, without polling on a fixed time interval.

Which approach should the data engineer use for selecting the trigger type to configure for the job?

- A. Use File-arrival trigger, which monitors a specified cloud storage path and starts a job run when new files are detected at that location.
- B. Use Continuous trigger, which keeps the job running perpetually and processes new files as an ongoing stream without discrete runs.
- C. Use Scheduled trigger with a one-minute cron interval, which approximates real-time file detection by polling the storage path every minute.
- D. Use Table-update trigger, which fires a job run whenever a Delta table registered in Unity Catalog receives new data written to it.

Question 6

Objective: *Identify common performance bottlenecks such as data skew, shuffling, and disk spilling by interpreting stage-level metrics in the Spark UI.*

A data engineer notices that a batch job's duration has doubled after a new data source was onboarded. In the Spark UI, the longest stage shows that most tasks finish in under 30 seconds, but one task takes over 10 minutes. The stage's task summary shows Min/Median shuffle read near 400 MB, while Max shuffle read exceeds 5 GB. The engineer confirms that adaptive query execution's skew-join handling has been explicitly disabled in the cluster's Spark configuration

Which solution reduces the job runtime?

- A. Increase cluster size to add more executors so the slow task finishes faster
- B. Confirm adaptive query execution with skew join handling is active to automatically split the oversized partition at runtime
- C. Reduce `spark.sql.shuffle.partitions` to coalesce more work into fewer tasks
- D. Manually repartition the dataset using a salt key before the join to distribute skewed keys evenly

Question 7

Objective: *Ingest semi-structured and unstructured data (e.g., JSON and nested data) via Lakeflow Connect and managed connectors into Delta tables.*

A data engineer needs to configure a Lakeflow Connect managed connector to ingest JSON records from a SaaS source into a Delta table in Unity Catalog. The source payload contains a nested object field called `order_details` with variable sub-keys that change between ingestion runs. The engineer wants the connector to automatically incorporate new sub-keys into the target Delta table without requiring manual schema updates.

Which configuration meets the requirement?

- A. Set the schema evolution mode to 'rescue' in the connector's ingestion settings so that records containing unrecognized sub-keys are written to a separate rescued-data column rather than expanding the target schema.
- B. Enable the 'mergeSchema' option on the target Delta table's `TBLPROPERTIES` so that the connector merges incoming schema changes at write time without any connector-level configuration.
- C. Set the schema evolution mode to 'failOnNewColumns' in the connector's ingestion settings and manually run `ALTER TABLE ADD COLUMN` after each ingestion run to incorporate new sub-keys.
- D. Rely on the managed connector's automatic schema evolution so that newly discovered sub-keys are merged into the target table's schema on the next pipeline run, without manual intervention.

Question 8

Objective: *Prioritize between Auto Loader, Lakeflow Connect partner connectors, and other ingestion methods based on volume, frequency, data type, and governance needs.*

A data engineer is building downstream pipelines to consume Databricks audit logs from a customer-owned S3 bucket. Before implementing schema inference and checkpointing, they want to understand the delivery format, typical ingestion latency, and whether files may be overwritten.

What is Databricks audit log storage behavior?

- A. Files are delivered as JSON with typical event logging under 15 minutes after delivery begins, and new deliveries can overwrite existing files
- B. Files are delivered as CSV with sub-minute latency guarantees, and overwrites never occur once a file is written to preserve immutability
- C. Files are delivered as Parquet with eventual consistency beyond 24 hours, and overwrites are disabled to simplify streaming ingestion
- D. Files are delivered as JSON with delivery on a weekly batch cadence, and overwrites replace prior content completely without appending

Question 9

Objective: *Identify trends in job performance using the Lakeflow Jobs run history view.*

A data engineer wants to visually spot whether a nightly job's run times have been creeping up over the past several weeks, and be alerted when any single run takes unusually long. The engineer opens the job's Runs tab and switches to the matrix view.

Which statement correctly describes how the matrix view helps identify duration trends?

- A. The height of each run and task bar visually represents its duration, and a warning indicator appears for any run that exceeds the job's configured expected completion time.
- B. The matrix view automatically calculates a rolling-average duration and flags any run that is more than two standard deviations above it.
- C. The matrix view generates a projected cost estimate for future runs based on historical duration and the cluster's DBU rate.
- D. The matrix view groups runs by day of the week and displays the median duration for each group in a bar chart.

Question 10

Objective: *Manage code in Databricks workspace UI using Git Folders: create/switch branches, commit, push, and open pull requests.*

A team wants a modular way to deploy, version, and orchestrate ETL pipelines in Databricks—enabling CI/CD and repeatability.

Which feature supports this requirement?

- A. Use models in Unity Catalog to represent ETL jobs, where each model stores the pipeline code artefact, and CI/CD promotes versions by updating model aliases tied to Job tasks.
- B. Package transformation logic as wheel libraries stored in Unity Catalog Volumes and bind them to Jobs tasks to ensure deterministic deployment across environments.
- C. Package API logic inside a Volume-mounted notebook, and use Jobs API v2 to trigger the notebook, depending on notebook revision history to act as a versioning system.

- D. Use DABs to define resources and code assets, version them in Git, and promote deployments across environments through automated CI/CD actions.

Answer Key and Explanations

#	Answer	Explanation
1	A	COPY INTO tracks which files have already been loaded by maintaining an internal load history in the Delta table's transaction log. When 'force' is set to 'false' (the default), COPY INTO skips any file that has been previously ingested, making each subsequent run process only newly arrived files.
2	C	Auto Loader is purpose-built for exactly this scenario: it monitors a cloud object storage path for new files, uses an internal checkpoint to track which files have already been processed so they are never re-ingested, and automatically scales from low to high file volumes using either directory listing mode (for lower volumes) or file notification mode (for high-volume, event-driven arrival).
3	B	When a join key has a highly skewed value distribution, a small number of shuffle partitions receive far more rows than others, causing those tasks to run disproportionately longer and stall the stage. Spark's Adaptive Query Execution (AQE) includes a skew-join optimization that detects partitions exceeding a configurable size threshold and automatically splits them into smaller sub-partitions, distributing the work more evenly across executors. Setting <code>spark.sql.adaptive.skewJoin.enabled</code> to true activates this mechanism at runtime without requiring the engineer to manually repartition the data or change the query.
4	A, C	A — correct pattern: inverted filters split valid/invalid rows into separate tables. C — FAIL UPDATE genuinely halts the pipeline on violation, meeting the "fail immediately" requirement.
5	A	The file-arrival trigger in Lakeflow Jobs is designed specifically to start a job run when new files appear at a monitored cloud storage path, making it the appropriate choice when the engineer wants event-driven execution tied to file landing rather than a fixed schedule.

6	B	One task at 10+ minutes vs. others under 30 seconds, with max shuffle read (5+ GB) far above the median (~400 MB), is classic partition skew. AQE flags a partition as skewed if it's both 5× the median size and over 256 MB — this one meets neither. Re-enabling skew-join handling lets AQE automatically split the oversized partition. Option 1 doesn't help — the entire skewed partition still lands on a single task regardless of the executor count. Option 3 makes it worse by packing more data into fewer tasks. Option 4 (salting) would also work, but it's a manual code change AQE makes unnecessary once re-enabled.
7	D	Managed connectors auto-handle new columns — new fields are picked up on the next run with no manual step. Option 4 states this accurately without inventing a setting name or claiming nested fields become "top-level" columns. Option 1 (rescue) deliberately avoids schema evolution rather than performing it. Option 2 misapplies mergeSchema, a write-time table option that the connector doesn't expose for configuration. Option 3 requires manual ALTER TABLE, which the requirement rules out.
8	A	Databricks audit logs are delivered as JSON, typically within about 15 minutes of delivery starting, and Databricks can overwrite existing files (appending new events rather than replacing them). Only option 1 matches all three facts — the others get the format, latency, or overwrite behavior wrong. Practical implication: since files can be appended to after first arrival, downstream ingestion and checkpointing need to handle re-reading a file, not just processing it once.
9	A	Bar height in the matrix view visually represents run/task duration, and Databricks does show a warning indicator when a run exceeds the job's configured expected completion time — both parts are real, documented behavior.
10	D	Declarative Automation Bundles (DABs), now also referred to as Declarative Automation Bundles, are Databricks' documented recommended approach for exactly this: defining jobs, pipelines, and other resources as source files (YAML) that you version in Git and deploy across dev/staging/production through CI/CD tooling (GitHub Actions, Azure DevOps, Jenkins, etc.).