databricks

Databricks Exam Guide

Databricks Certified Data Engineer Professional

[Provide Exam Guide Feedback]

Update Information

Updates	Date
New exam is live in Webassessor (cutover date)	Oct 9, 2026
The current exam is available through	Oct 8, 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.

Which exam will you take? This guide covers two versions of this exam. The exam changes Webassessor on **Oct 9, 2026**. Use the version that matches *your* scheduled exam date.

- If your exam is on or before Oct 8, 2026, prepare with the **Current Exam** content — the “Current exam” column in About the Exam, and the Current Exam outline.
- If your exam is on or after Oct 9, 2026, prepare with the **New Exam** content — the “New exam” column in About the Exam, and the New Exam outline.

Everything not split into two versions below — registration, recertification, recommended preparation, and documentation — applies to both.

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. Because this exam is changing, this guide covers both the exam currently live and the exam that goes live on the cutover date listed above. Identify your exam date, then prepare against the matching version. Please check back two weeks before your exam to make sure you have the most current version.

Audience Description

The Databricks Certified Data Engineering Professional exam validates a candidate's advanced skills in building, optimizing, and maintaining production-grade data engineering solutions on the Databricks Platform. Successful candidates demonstrate expertise across core platform features such as Delta

Lake, Unity Catalog, Auto Loader, Lakeflow Declarative Pipelines, Databricks Compute (including serverless), Lakeflow Jobs and the Medallion Architecture. This Certification assesses the ability to design secure, reliable, and cost-effective ETL Pipelines, process complex data from diverse sources using Python and SQL, and apply best practices in schema management, observability, governance, and performance optimization. Candidates are also tested on implementing streaming workloads, orchestrating workflows, leveraging DevOps & CI/CD, and deploying using tools such as the Databricks CLI, REST API, and Asset Bundles. Individuals who pass this certification exam can be expected to complete advanced data engineering tasks using Databricks and its associated tools.

About the Exam

Item	Current exam (through Oct 8, 2026))	New exam (from Oct 9, 2026)
Number of scored questions	59 scored multiple-choice questions	60 scored multiple-choice questions
Possible unscored questions	Up to 10 Unscored questions may be included (see "Unscored content" below)	Up to 10 Unscored questions may be included (see "Unscored content" below)
Total number of questions you may see	59 scored questions plus unscored questions	60 scored questions plus unscored questions
Time limit	120 minutes (additional time is already factored in to account for unscored questions)	120 minutes (additional time is already factored in to account for unscored questions)
Registration fee	USD 200, plus applicable taxes as required per local law	USD 200, plus applicable taxes as required per local law
Delivery method	Online proctored or test center	Online proctored or test center
Languages offered	English; Japanese; Brazilian Portuguese (pt-BR); Korean	English;
Test aides	None allowed	None allowed
Prerequisite	None required; but related training highly recommended	None required; but related training highly recommended
Recommended experience	One year of hands-on experience in the data engineering tasks with the listed exam outline is highly recommended	One year of hands-on experience in the data engineering tasks with the listed exam outline is highly recommended
Validity	2 years	2 years

Recertification: Recertification is required every two years to maintain your certified status. To recertify, you must take the current live exam.

Unscored content: Exams may include unscored questions to gather statistical information for future use. These questions 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: Advanced Data Engineering with Databricks
- Self-paced (available in Databricks Academy):
    - Advanced Techniques with Apache Spark™ Declarative Pipeline
    - Databricks Data Privacy
    - Databricks Performance Optimization
    - Automated Deployment with Declarative Automation Bundles

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 — Current Exam (Available until Oct 8, 2026)

Section 1: Developing Code for Data Processing using Python & SQL(22%)

- Using Python and Tools for development
    - Design and implement a scalable Python project structure optimized for Declarative Automation Bundles (formerly Databricks Asset Bundles / DABs), enabling modular development, deployment automation, and CI/CD integration.

    - Manage and troubleshoot external third-party library installations and dependencies in Databricks, including PyPI packages, local wheels, and source archives.
    - Develop User-Defined Functions (UDFs) using Pandas/Python UDF.
- Building and Testing an ETL pipeline with Lakeflow Declarative Pipelines, SQL, and Apache Spark on the Databricks Platform
    - Build and manage reliable, production-ready data pipelines for batch and streaming data using Lakeflow Declarative Pipelines and Autoloader.
    - Create and Automate ETL workloads using Jobs via UI/APIs/CLI.
    - Explain the advantages and disadvantages of streaming tables compared to materialized views.
    - Use AUTO CDC APIs (formerly APPLY CHANGES) to simplify CDC in Lakeflow Declarative Pipelines.
    - Compare Spark Structured Streaming and Lakeflow Declarative Pipelines to determine the optimal approach for building scalable ETL pipelines.
    - Create a pipeline component that uses control flow operators (e.g., if/else, for/each, etc.).
    - Choose the appropriate configs for environments and dependencies, high memory for notebook tasks, and auto-optimization to disallow retries.
    - Develop unit and integration tests using assertDataFrameEqual, assertSchemaEqual, DataFrame.transform, and testing frameworks, to ensure code correctness, including a built-in debugger.

Section 2: Data Ingestion & Acquisition (7%)

- Design and implement data ingestion pipelines to efficiently ingest a variety of data formats including Delta Lake, Parquet, ORC, AVRO, JSON, CSV, XML, Text and Binary from diverse sources such as message buses and cloud storage.
- Create an append-only data pipeline capable of handling both batch and streaming data using Delta.

Section 3: Data Transformation, Cleansing, and Quality (10%)

- Write efficient Spark SQL and PySpark code to apply advanced data transformations, including window functions, joins, and aggregations, to manipulate and analyze large Datasets.
- Develop a quarantining process for bad data with Lakeflow Declarative Pipelines, or autoloader in classic jobs.

Section 4: Data Sharing and Federation (10%)

- Demonstrate delta sharing securely between Databricks deployments using Databricks to Databricks Sharing (D2D) or to external platforms using the open sharing protocol (D2O).
- Configure Lakehouse Federation with proper governance across the supported source Systems.
- Use Delta Share to share live data from Lakehouse to any computing platform.

Section 5: Monitoring and Alerting (10%)

- Monitoring
    - Use system tables for observability over resource utilization, cost, auditing and workload monitoring.
    - Use Query Profiler UI and Spark UI to monitor workloads.
    - Use the Databricks REST APIs/Databricks CLI for monitoring jobs and pipelines.

    - Use Lakeflow Declarative Pipelines Event Logs to monitor pipelines.
- Alerting
    - Use SQL Alerts to monitor data quality.
    - Use the Lakeflow Jobs UI and Jobs API to set up notifications for job status and performance issues.

Section 6: Cost & Performance Optimization (13%)

- Understand how / why using Unity Catalog managed tables reduces operations
- Overhead and maintenance burden.
- Understand delta optimization techniques, such as deletion vectors and liquid clustering.
- Understand the optimization techniques used by Databricks to ensure the performance of queries on large datasets (data skipping, file pruning, etc.).
- Apply Change Data Feed (CDF) to address specific limitations of streaming tables and enhance latency.
- Use the query profile to analyze the query and identify bottlenecks, such as bad data skipping, inefficient types of joins, and data shuffling.

Section 7: Ensuring Data Security and Compliance(10%)

- Applying Data Security mechanisms.
- Use ACLs to secure Workspace Objects, enforcing the principle of least privilege, including enforcing principles like least privilege, policy enforcement.
- Use row filters and column masks to filter and mask sensitive table data.
- Apply anonymization and pseudonymization methods, such as Hashing, Tokenization, Suppression, and generalization, to confidential data.
- Ensuring Compliance
- Implement a compliant batch & streaming pipeline that detects and masks PII to ensure data privacy.
- Develop a data-purging solution that ensures compliance with data retention policies.

Section 8: Data Governance (7%)

- Create and add descriptions/metadata about enterprise data to make it more discoverable.
- Demonstrate understanding of Unity Catalog permission inheritance model

Section 9: Debugging and Deploying (10%)

- Debugging and Troubleshooting
    - Identify pertinent diagnostic information using Spark UI, cluster logs, system tables, and query profiles to troubleshoot errors.
    - Analyze the errors and remediate failed job runs using job repairs and parameter overrides.
    - Use Lakeflow Declarative Pipelines event logs and the Spark UI to debug Lakeflow Declarative Pipelines and Spark pipelines.
- Deploying CI/CD
    - Build and deploy Databricks resources using Declarative Automation Bundles (formerly Databricks Asset Bundles)
    - Configure and integrate with Git-based CI/CD workflows, Databricks Git folders (formerly Repos) for notebook and code deployment.

Section 10: Data Modeling (7%)

- Design and implement scalable data models using Delta Lake to manage large datasets.
- Simplify data layout decisions and optimize query performance using Liquid Clustering.
- Identify the benefits of using liquid Clustering over Partitioning and ZOrder.
- Design Dimensional Models for analytical workloads to ensure efficient querying and aggregation.

Exam Outline — New Exam (Starting October 9, 2026)

Section 1: Developing Code for Data Processing using Python and SQL (23%)

- Implement scalable Python project structures for Declarative Automation Bundles (formerly Databricks Asset Bundles / DABs) to support modular development and CI/CD integration.
- Apply troubleshooting techniques to dependency conflicts and installation failures for external libraries (PyPI, local wheels, source archives) across serverless, pipeline, and bundle-deployed environments.
- Develop User-Defined Functions (UDFs) using Pandas, Python, and SQL, including Unity Catalog functions for the given constraints.
- Create production-ready data pipelines for streaming data using Lakeflow Declarative Pipelines and Auto Loader.
- Create and automate ETL workloads using Lakeflow Jobs via the UI, API, and CLI.
- Choose between a streaming table and a materialized view for a given latency, cost, and refresh requirement.
- Implement CDC pipelines using AUTO CDC APIs (formerly APPLY CHANGES) in Lakeflow Declarative Pipelines, including SCD Type 1 and Type 2 via stored_as_scd_type.
- Choose between Spark Structured Streaming and Apache Spark™ Declarative Pipelines for scalable ETL given operational constraints.
- Create Lakeflow Jobs that use control-flow operators such as If/Else conditions and For Each loops.
- Choose appropriate compute and configuration for environments and dependencies — including serverless compute (serverless environments, dependency management, performance mode), high-memory notebook tasks, and auto-optimization settings (e.g., disallowing retries).
- Develop unit and integration tests using assertDataFrameEqual, assertSchemaEqual, DataFrame.transform, and testing frameworks to ensure code correctness.
- Apply Structured Streaming stateful-processing semantics, including watermarks, output modes, foreachBatch, and checkpoints for fault-tolerant exactly-once state recovery.

Section 2: Data Ingestion & Acquisition (12%)

- Develop data ingestion pipelines to ingest a variety of data formats, including Delta Lake, Parquet, JSON, Iceberg, CSV, and Binary from diverse sources such as message buses (Kafka, Kinesis, Pub/Sub) and cloud storage.
- Configure incremental CDC pipelines using Lakeflow Pipelines with Delta or Iceberg as the target table format.

- Configure CDC ingestion pipelines from relational database sources, including SQL Server, MySQL, and PostgreSQL, using Lakeflow Connect.
- Implement OpenSharing (D2D and Databricks-to-Open) and Clean Rooms for privacy-preserving collaboration.
- Configure Lakehouse Federation with governance across supported source systems, applying UC permissions and connection-level credentials.

Section 3: Data Manipulation (12%)

- Apply advanced data transformations, including window functions, joins, and aggregations, using Spark SQL and PySpark to process large datasets.
- Develop a model and query semi-structured data using the VARIANT data type and related functions (e.g., parse_json, variant_get).
- Apply AI functions, including ai_query, to perform model inference within data pipelines for enrichment and classification tasks.
- Implement data quality expectations in Lakeflow Declarative Pipelines to quarantine, drop, or fail on bad records.

Section 4: Monitoring and Alerting (10%)

- Use Databricks system tables (billing, compute, access, lakeflow) for cost analysis, auditing, and workload monitoring.
- Use Databricks REST APIs / CLI and SDK for monitoring and analyzing jobs and pipelines.
- Use Apache Spark Declarative Pipelines event logs to monitor pipeline health and data quality.
- Use Databricks Lakehouse alerts to monitor governed business metrics, data quality and anomaly signals, usage and cost, SQL warehouse/query health, audit and security events, AI agent quality, and Lakeflow Job branching.
- Use Lakeflow Jobs UI and Jobs API to monitor job status and performance metrics.

Section 5: Cost & Performance Optimization(15%)

- Explain how Unity Catalog managed tables, Predictive Optimization, and Liquid Clustering reduce operational and maintenance overhead for a given workload.
- Choose the appropriate Delta optimization technique (deletion vectors, Liquid Clustering, CLUSTER BY AUTO) for a given table access pattern.
- Understand how caching (Delta cache) improves query performance for repeated reads of the same data.
- Apply Change Data Feed (CDF) to expose row-level changes (updates/deletes) for efficient incremental downstream processing.
- Use the query profile to identify performance bottlenecks, such as data-skipping inefficiencies, join strategies, and shuffle operations.
- Compare Liquid Clustering vs partitioning/ZORDER for a given table size and query pattern.

Section 6: Ensuring Data Security and Compliance (8%)

- Apply least-privilege access control lists (ACLs) to secure Unity Catalog securable objects and workspace resources.
- Apply attribute-based access control (ABAC) policies with governed tags to enforce row filters and column masks at scale.

- Apply anonymization and pseudonymization methods — such as hashing, tokenization, suppression, and generalization — to confidential data (e.g., using column masks and related Unity Catalog features).
- Implement a compliant data pipeline that enforces PII detection and masking controls across batch and streaming workloads using Unity Catalog features.
- Implement a data purging strategy that satisfies data retention and deletion policies (e.g., GDPR right-to-erasure, GDPR right-to-be-forgotten deletion) using Delta Lake and Unity Catalog features.

Section 7: Data Governance (5%)

- Demonstrate understanding of Unity Catalog tags and comments as mechanisms for adding metadata to securable objects to improve data discoverability.
- Demonstrate understanding of the Unity Catalog permission inheritance model as a mechanism for managing access control across catalogs, schemas, and objects.

Section 8: Debugging and Deploying (10%)

- Identify pertinent diagnostic information using Spark UI, cluster logs, system tables, and query profiles to troubleshoot errors.
- Analyze the errors and remediate the failed job runs with job repairs and parameter overrides.
- Use event logs and the Spark UI to debug pipelines and Spark workloads.
- Deploy Databricks resources using Declarative Automation Bundles (formerly Databricks Asset Bundles).
- Configure Git-based CI/CD workflows using Databricks Git folders (formerly Repos) to deploy notebooks and code.

Section 9: Data Modeling (5%)

- Design scalable Delta/Iceberg table layouts for large data assets by mapping partitioning to data grain, aligning clustering to relationship access patterns, and maintaining balanced file sizes through compaction.
- Design dimensional models for analytical workloads, leveraging Materialized Views for pre-computed aggregation and Unity Catalog Metric Views for governed, reusable metric definitions, to ensure efficient querying and aggregation.

Sample Questions

The purpose 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.

The sample questions below align with the New Exam outline. If your exam is on or before **Oct 9, 2026**, these questions are still useful practice, but a few may cover objectives that are not on your exam. Your exam outline is the authoritative list of what you will be tested on.

Question 1

Objective: *Choose between a streaming table and a materialized view for a given latency, cost, and refresh requirement.*

A gold-layer object in a Lakeflow Declarative Pipeline aggregates daily order totals from a bronze source. Downstream BI dashboards read it on a schedule, and the aggregate must always reflect the full history, including late-arriving updates to prior days, not just newly appended rows.

Which object should the engineer define?

A. A streaming table, because it processes each source row exactly once and is the lowest-cost option for any aggregation.
B. A temporary view, because it always reflects the latest data at query time at no storage cost.
C. A materialized view, because it incrementally maintains the full aggregate result, can be refreshed on a schedule, and reflects late-arriving changes.
D. A streaming table with a foreachBatch merge, because materialized views cannot perform aggregations.

Question 2

Objective: *Apply Structured Streaming stateful-processing semantics, including watermarks, output modes, foreachBatch, and checkpoints for fault-tolerant exactly-once state recovery.*

A Structured Streaming query performs an event-time windowed aggregation. It must bound state growth caused by very late events, and it must resume after a driver failure without double-counting results.

Which configuration meets both requirements?

A. Set a watermark on the event-time column and rely on the query's checkpoint to restore offsets and state on restart.
B. Use complete output mode with no watermark to retain all state for correctness.
C. Increase spark.sql.shuffle.partitions and disable checkpointing to speed up recovery.
D. Wrap the aggregation in a foreachBatch and manually commit source offsets to a Delta table.

Question 3

Objective: *Configure CDC ingestion pipelines from relational database sources, including SQL Server, MySQL, and PostgreSQL, using Lakeflow Connect.*

A data engineer must continuously replicate insert, update, and delete change events from an operational PostgreSQL database into the lakehouse with minimal custom code.
Which approach fits the requirement?

A. Schedule a nightly JDBC full-table read and overwrite the target table each night.
B. Configure a Lakeflow Connect managed connector for PostgreSQL to ingest change data capture into the lakehouse.
C. Export the database to CSV in cloud storage and ingest the files with Auto Loader.
D. Use OpenSharing to share the PostgreSQL tables directly with the lakehouse.

Question 4

Objective: *Develop a model and query semi-structured data using the VARIANT data type and related functions (e.g., parse_json, variant_get).*

A bronze table stores raw JSON payloads of varying shape in a single string column. The engineer wants efficient storage and fast field extraction without pre-declaring a fixed schema.

What should they do?

A. Parse the payloads into a VARIANT column with parse_json and extract fields using variant_get and colon-path access.
B. Explode the JSON into one row per key using a fixed STRUCT schema.
C. Keep the payloads as STRING and use regexp_extract for each field at query time.
D. Convert the JSON to Parquet with a strict, fixed schema before ingestion.

Question 5

Objective: *Use Databricks system tables (billing, compute, access, lakeflow) for cost analysis, auditing, and workload monitoring.*

A data engineer's team must attribute last month's DBU spend to specific jobs and SQL warehouses and identify the most expensive workloads, using governed data already available in Unity Catalog.

Which source should they query?

A. The Spark UI event timeline for each cluster.
B. Per-cluster driver logs downloaded from the workspace.
C. The system.billing.usage table joined with the compute and pricing system tables in Unity Catalog.
D. The account console billing PDF export.

Question 6

Objective: *Choose the appropriate Delta optimization technique (deletion vectors, Liquid Clustering, CLUSTER BY AUTO) for a given table access pattern.*

A data engineer finds that a large Delta table receives frequent small MERGE operations that update and delete individual rows. The team wants to avoid rewriting entire data files on every change while keeping reads fast.

Which technique addresses the write amplification?

A. Partition the table by its primary key.
B. Enable deletion vectors so updates and deletes are marked at the row level without immediately rewriting whole files.
C. Run VACUUM with zero-hour retention after every MERGE.
D. Disable predictive optimization to prevent background file rewrites.

Question 7

Objective: *Apply attribute-based access control (ABAC) policies with governed tags to enforce row filters and column masks at scale.*

A data engineer is working for an organisation with hundreds of tables containing PII and wants a single, centrally managed rule to mask any column tagged 'pii' across all current and future tables, without altering each table individually.

What should they implement?

A. Run a per-table ALTER COLUMN SET MASK statement on every existing table.
B. Attach a row filter function to each schema.
C. Create dynamic views that duplicate every base table with masking logic.
D. Define an ABAC policy that applies a column mask to any column carrying the governed 'pii' tag.

Question 8

Objective: *Demonstrate understanding of the Unity Catalog permission inheritance model as a mechanism for managing access control across catalogs, schemas, and objects.*

A data engineer grants SELECT on a catalog to the analysts group. New schemas and tables are later created in that catalog.

Assuming no conflicting grants, what access do the analysts have to the new tables?

A. They can SELECT the new tables, because a privilege granted at the catalog level is inherited by the schemas and objects within it, including ones created later.
B. They have no access until SELECT is granted again on each new table.
C. They can SELECT only the tables that existed at the time of the grant.
D. They automatically gain ownership of the new tables.

Question 9

Objective: *Deploy Databricks resources using Declarative Automation Bundles (formerly Databricks Asset Bundles).*

A data engineer wants the same job and pipeline definitions deployed identically to dev, staging, and prod from source control, with environment-specific overrides and CI/CD.

Which approach meets the requirement?

A. Manually recreate the jobs in each workspace UI, then export and import JSON.
B. Define the resources in a databricks.yml bundle with per-target overrides and deploy with `databricks bundle deploy -t <target>`.
C. Copy notebooks between workspaces using only the workspace import CLI.
D. Use a single shared workspace for all environments to avoid configuration drift.

Question 10

Objective: *Design dimensional models for analytical workloads, leveraging Materialized Views for pre-computed aggregation and Unity Catalog Metric Views for governed, reusable metric definitions, to ensure efficient querying and aggregation.*

A data engineering team computes "net revenue" with slightly different SQL in each of its dashboards, producing inconsistent numbers across reports. The same dashboards are also slow, because each one recomputes large joins and aggregations at query time. The team needs one authoritative definition of the metric and pre-aggregated results that the dashboards can read directly.

Which implementation should the engineer choose?

A. Add a comment on each dashboard describing how to compute net revenue.
B. Grant every analyst edit access to a shared SQL snippet file.
C. Define net revenue once in a Unity Catalog metric view for a governed, reusable definition, and use materialized views for the pre-computed aggregates the dashboards read.
D. Store the metric as a Python UDF copied into each notebook.

Answer Key & Explanations

#	Answer	Explanation
1	C	A materialized view incrementally maintains the full aggregate and can be refreshed on a schedule, so it correctly reflects late-arriving updates to prior days; a streaming table processes appended rows incrementally and does not recompute a full aggregate over changing history.
2	A	A watermark lets the engine drop state for events beyond the allowed lateness (bounding state growth), while the query's checkpoint persists offsets and state so a restart resumes exactly-once without double-counting.
3	B	Lakeflow Connect provides managed database connectors that ingest CDC — including deletes — from SQL Server, MySQL, and PostgreSQL with minimal code; the other options are full reloads or do not apply to an operational RDBMS.
4	A	The VARIANT type stores semi-structured data efficiently and supports schema-flexible querying via parse_json and variant_get, avoiding a rigid pre-declared schema.
5	C	System tables such as system.billing.usage (joined with compute/pricing system tables) are governed, queryable records ideal for cost attribution and workload analysis; the other sources are not queryable at scale or are not governed tables.
6	B	Deletion vectors mark updated and deleted rows without immediately rewriting entire data files, reducing write amplification from frequent small MERGE operations while keeping reads correct; compaction reconciles them later.
7	D	An ABAC policy keyed on a governed tag applies a column mask wherever the 'pii' tag is present, including tables created later, so the rule scales centrally — unlike per-table masks or duplicated views.
8	A	Unity Catalog privileges are inherited down the hierarchy: a grant on a catalog applies to its schemas and their objects, including objects created after the grant.
9	B	A Declarative Automation Bundle (databricks.yml) declares resources with environment targets and overrides and deploys them via the CLI, enabling reproducible, CI/CD-friendly deployment across environments.
10	C	A metric view provides one governed, reusable definition of net revenue; materialized views pre-compute the aggregates dashboards read, solving both the consistency and performance problems.