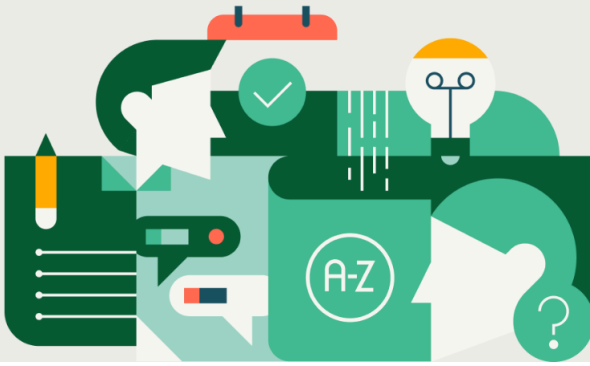
databricks

Databricks 試験ガイド

Databricks Certified データエンジニア Associate 更新試験



試験ガイドのフィードバックを提供する

情報の更新

更新	日付
試験公開日	Jun 15, 2026
試験ガイド更新日	Aug 6, 2026

試験日の2週間前に、当社のウェブページで試験ガイドを確認し、最新バージョンの試験に向けて準備していることをご確認ください。

この試験ガイドの目的

この試験ガイドの目的は、試験の概要と出題範囲をお伝えし、受験準備が整っているかどうかを判断するお手伝いをすることです。このドキュメントは、試験に変更があった際に更新され（変更の発効日も記載されます）、自信を持って準備を進めていただけるようになっています。このガイドは、以上に記載された発効日時点で有効な試験バージョンを対象としています。

対象者の説明

この Databricks Certified Data Engineer Associate Renewal 試験は、Databricks Platform を活用してデータエンジニアリングの基礎的なタスクを実行する個人の能力を評価します。本試験では、Databricks Certified Data Engineer Associate がデータ取り込み、データロード、データ変換、およびモデリングに関連するタスクについての知識を評価します。具体的には、PySpark / SQL を使用した抽出, 変換, ロード（ETL）タスクの実行、LakeFlow Jobs の活用、CI/CD などの能力が含まれます。さらに、本試験ではトラブルシューティング、モニタリング、および最適化手法に関する理解、ならびに Databricks Platform 内でガバナンスとセキュリティを実現する方法についての知識も評価します。

試験について

項目	詳細
採点対象の問題数	採点対象の多肢選択式問題 25 問
採点対象外の問題	なし
出題される問題の総数	採点対象の問題 25 問
制限時間	60 分
受験料	200 米ドル（現地の法律に基づき、適用される税金が別途かかります）
受験方法	オンライン監督付き試験のみ
提供言語	英語、日本語、ブラジルポルトガル語（pt-BR）、韓国語
補助資料	使用不可
前提条件	この再認定試験は、期限切れの、または有効期限が2か月以内に迫っている既存の Databricks Certified Data Engineer Associate 認定を保有している方が受験できます。
推奨される経験	試験の出題範囲に沿った Databricks Platform での実務経験を 1 年以上有することを強く推奨します
有効期間	2年

再認定：認定ステータスを維持するには、**2**年ごとに再認定が必要です。再認定を受けるには、**Data Engineer Associate**更新試験または現行の完全な試験を受験することができます。

採点対象外のコンテンツ：試験には、将来の使用に向けて統計的な情報を収集するために、採点対象外の問題が含まれる場合があります。これらの問題は試験上で識別されず、スコアに影響しません。また、これらの問題に対応するための追加時間が時間制限に考慮されています。

推奨される準備

トレーニング

- インストラクター主導：**Databricks** を使ったデータエンジニアリング
- 自己学習型（**Databricks Academy** で利用可能）：
    - LakeFlow Connect によるデータ取り込み
    - Lakeflow Jobs でワークロードをデプロイする
    - データエンジニアリングのための DevOps エッセンシャル
    - Apache Spark™ を使用したデータパイプラインの構築 宣言型パイプライン
    - Databricks でのデータガバナンス入門

AI Prep ガイド - すべての Databricks 試験

Databricks AI Prep Guide ガイドでは、すでにお使いのAIチャットボットを任意のDatabricks認定資格向けのパーソナライズされた個別指導ツールに変える方法をご紹介します。公式試験ガイドを活用してAIを準備し、AIが提供しがちな古い回答を見極め、自分の弱点を発見し、すべてをDatabricksの各試験で求められる実践的な演習と組み合わせる方法を解説します。

ハンズオン演習

何を練習すればよいかわからない場合は、AI Prep ガイド（以上の「AI Prep ガイド」セクションを参照）をご確認ください。このガイドには「ステップ 5 — ハンズオンの最低要件」というプロンプトが含まれており、AI チャットボットがこの試験の目標にマッピングされた 6〜10 個のハンズオンタスクの個別チェックリストを生成します。各タスクは Databricks Free Edition で完了できる規模です。

Databricksドキュメント

準備を進める際は、この試験でカバーされるプラットフォーム領域について、公式のDatabricksドキュメントを学習してください。それは最も新しく、権威あるソースです。各トピックをどの程度深く学習するかを判断し、古い情報ではなく最新の非推奨でない機能を習得しているかどうかを確認するために活用してください。
公式ドキュメント: Databricksドキュメント

試験の概要

セクション1：データ取り込みとロード（32%）

- COPY INTO を使用して、クラウドオブジェクトストレージから Unity Catalog で管理された Delta テーブルへの増分ファイル取り込みを実行します。
- Auto Loader を使用して、Unity Catalog で管理された Delta テーブルにファイルを増分取り込みします。
- Lakeflow Connect を設定して、多様なエンタープライズソースから Unity Catalog で管理されたテーブルにデータを確実に取り込みます。
- ノートブックでJDBC/ODBCまたはRESTクライアントを使用して、データをクラウドストレージに格納するか、Unity Catalogで管理されたテーブルに直接格納します。
- ボリューム、頻度、データタイプ、およびガバナンスの要件に基づいて、Auto Loader、Lakeflow Connect パートナーコネクター、およびその他のインジェスト方法の優先順位を決定します。
- LakeFlow Connect とマネージドコネクターを使用して、半構造化データおよび非構造化データ（例：JSON やネストされたデータ）を Delta テーブルに取り込みます。

セクション2：データ変換とモデリング（24%）

- PySpark/SQL を使用してブロンズテーブルを読み込み、データクリーニングを適用する（null の処理、データ型のデータの標準化）とともに、結果をシルバーテーブルに書き込みます。

- 一般的なSparkチューニングパラメータを特定し、パフォーマンスへの全般的な影響を理解する。
- Unity Catalog におけるゴールドレイヤーのオブジェクト（Materialized Views、ビュー、Streaming Tables、通常のテーブル）を区別し、BI/アナリティクスへの活用方法を理解する。
- データ品質チェックと検証ルールを適用する して、信頼性の高いシルバーおよびゴールドデータセットを確保します。

セクション 3: Lakeflow Jobs の操作 (12%)

- LakeFlow Jobsを使用して、マルチステップのデータワークフローをオーケストレーションするための制御フロー（タスクの再試行、分岐、ループ）を実装します。
- DAGベースのワークフローを使用して、LakeFlow Jobs で一般的なタスクタイプと依存関係を設定する
- LakeFlow Jobs のスケジュールとトリガーを設定します。これには、スケジュール済み、ファイル到着、テーブル更新、および継続的なトリガーが含まれます。

セクション 4: CI/CD の実装 (12%)

- Git Folders を使用して Databricks ワークスペース UI でコードを管理する：ブランチの作成・切り替え、コミット、プッシュ、プルリクエストの作成。
- Declarative Automation Bundle の環境ターゲット（dev、test、prod）を使用して、コードベースを各環境に昇格させます。
- Databricks CLI を使用して Declarative Automation Bundles (DABs) を検証およびデプロイします。

セクション5: トラブルシューティング、モニタリング、および最適化 (12%)

- Lakeflow Jobs の実行履歴ビューを使用して、ジョブのパフォーマンスの傾向を特定します。
- LakeFlow Jobs の UI を使用して、パイプラインの健全性、ジョブのステータス、DAG タスクグラフ、実行時間、および失敗率を監視します。
- マネージド Delta Lake 機能としてのリキッドクラスタリングと Predictive Optimization の目的および利点を特定する。

セクション 6: ガバナンスとセキュリティ (8%)

- 列レベルのマスキングと行レベルのセキュリティの目的を理解し、それらを適用すべき状況を特定する。
- Unity Catalog の ABAC ポリシーの機能と、それが集中型データアクセス制御をどのようにサポートするかを理解する。

サンプル問題

サンプル問題の目的は、試験ガイドに記載されている目標を示し、各目標に対応するサンプル問題を提供することです。試験ガイドには試験で出題される可能性のある目標が記載されています。最善の準備方法は、上記の試験日程に対応した試験の全概要を確認することです。

質問 **1:**

目標：**COPY INTO** を使用して、クラウドオブジェクトストレージから **Unity Catalog** で管理された **Delta** テーブルへの増分ファイル取り込みを実行します。

データエンジニアは、外部のクラウドオブジェクトストレージの場所に新たに到着したCSVファイルのみを、Unity Catalogで管理されたDelta テーブルにロードする必要があります。これは、以前の実行ですでに取り込まれたファイルを再処理することなく行う必要があります。エンジニアはこのタスクにCOPY INTOを使用することにしました。

COPY INTO を使用したインクリメンタルファイルインジェストを正しく実装する設定はどれですか？

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')

問題 **2:**

目標：ボリューム、頻度、データタイプ、およびガバナンスの要件に基づいて、**Auto Loader**、**LakeFlow Connect** パートナーコネクター、およびその他のインジェスト方法の優先順位を決定する。

データエンジニアが、SaaS アプリケーション向けのインジェストパイプラインを設計しています。このアプリケーションは、1日を通じて不規則な間隔でクラウドオブジェクトストレージバケットに新しい JSON ファイルを配信します。ファイルのカウントは、オフピーク時には1時間あたり数件、ピーク時には1時間あたり数千件に及びます。データエンジニアは、このパイプラインが新しいファイルのみを自動的に検出して処理し、既にインジェスト済みのファイルを再処理することなく、低ボリューム期と高ボリューム期の両方にわたって適切にスケールできるようにする必要があります。

データエンジニアはどのインジェスチョン方法を使用すべきですか？

A. 15分ごとに実行されるスケジュール済みバッチジョブを作成し、spark.read.json を使用してストレージパスからすべてのファイルを読み込み、手動で管理されたログテーブルと照合することで既に処理済みのファイルを除外する。

B. SaaS アプリケーションの API からデータを直接プルするように設定された LakeFlow Connect パートナーコネクターを使用する。これにより、ソースの構造化データに対する組み込みのスキーママッピングとガバナンス統合が提供されます。

C. cloudFiles 形式の Auto Loader を設定してクラウドストレージパスを監視し、ファイル通知またはディレクトリリスティングモードを使用して、内部チェックポイントを介して取り込み済みファイルを追跡します。

D. 固定スケジュールでクラウドストレージパスからファイルを読み込むために、Delta Lake ターゲットテーブルに対して COPY INTO を使用し、そのべき等なロードトラッキングによってすでに取り込み済みのファイルをスキップします。

問題 3:

目標: 一般的な**Spark**チューニングパラメーターを特定し、パフォーマンスへの全般的な影響を理解する。

データエンジニアが、大規模な Delta Lake テーブルを処理する Spark ジョブの実行速度が遅いことに気づきました。Spark UI を確認したところ、ほとんどのタスクは2秒以内に完了しているものの、少数のタスクが3分以上かかっており、ステージ全体が停滞していることが判明しました。このテーブルは、結合キーとして使用されている order_region カラムにデータの偏りがあります。エンジニアは、アダプティブクエリー実行のスキュー結合処理がクラスターの Spark 設定で明示的に無効化されていることを確認しました。

このパフォーマンスの問題に対処するために、データエンジニアはどの設定パラメーターを設定する必要がありますか？

A. クラスター内で利用可能なエグゼキューターコアの数と等しい値に `spark.sql.shuffle.partitions` を設定します。
B. `spark.sql.adaptive.skewJoin.enabled` を `true` に設定して、アダプティブクエリーエンジンがスキューしたパーティションを自動的に分割できるようにします。
C. 各タスクがソースファイルから読み取るバイト数を減らすために、`spark.sql.files.maxPartitionBytes` をより小さい値に設定します。
D. `spark.databricks.delta.optimizeWrite.enabled` を `true` に設定して、Delta Lake が結合の実行前に小さなファイルをマージするようにします。

問題 4:

目標：信頼性の高いシルバーおよびゴールドデータセットを確保するために、データ品質チェックと検証ルールを適用する。

データエンジニアが、ブロンズソースから顧客の取引レコードを取り込む LakeFlow パイプラインにおいて、シルバーレイヤーのテーブルを構築しています。このエンジニアは、2つのデータ品質ルールを適用する必要があります。transaction_id が null の取引は削除せずに隔離しなければならず、amount がゼロ以下の取引はパイプラインを直ちに失敗させなければなりません。

これらの要件を満たす制約宣言を2つ選んでください。

A. 同じブロンズソースから、反転したフィルターを使用して2つのStreaming Tablesを作成します。有効な行（transaction_id IS NOT NULL）はメインテーブルへ、transaction_idがnullの行は検疫テーブルへ送られます。
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. 同じブロンズソースから、反転したフィルターを使用して2つのMaterialized Viewsを作成します。有効な行（`transaction_id IS NOT NULL`）はメインビューに、`transaction_id`がnullの行は検疫ビューに格納されます。
E. CONSTRAINT valid_transaction_id EXPECT (transaction_id IS NOT NULL) ON VIOLATION FAIL UPDATE

問題 **5:**

目標**:** スケジュール、ファイル到着、テーブル更新、継続的なトリガーを含む、**LakeFlow Jobs** のスケジュールとトリガーを設定します。

データエンジニアは、クラウドオブジェクトストレージの場所に新しいファイルが到着するたびに、固定の時間間隔でポーリングすることなく、LakeFlow Jobs のジョブを自動的にトリガーする必要があります。

データエンジニアは、ジョブに設定するトリガータイプを選択するためにどのアプローチを使用すべきですか？

A. File-arrival トリガーを使用します。これは指定されたクラウドストレージパスを監視し、そのパスで新しいファイルが検出されたときにジョブの実行を開始します。
B. Continuous トリガーを使用します。これにより、ジョブが継続的に実行され続け、個別の実行なしに新しいファイルを継続的なストリームとして処理します。
C. 1分間のcron間隔でScheduledトリガーを使用します。これにより、ストレージパスを1分ごとにポーリングすることで、リアルタイムに近いファイル検出を実現します。
D. Unity Catalog に登録された Delta テーブルに新しいデータが書き込まれるたびにジョブの実行を起動する、テーブル更新トリガーを使用します。

質問 6

目標：***Spark UI*** のステージレベルのメトリクスを解釈することで、データスキュー、シャッフル、ディスクスピルなどの一般的なパフォーマンスのボトルネックを特定する。

データエンジニアが、新しいデータソースがオンボーディングされた後にバッチジョブの期間が2倍になったことに気づきました。Spark UI では、最も長いステージにおいて、ほとんどのタスクが30秒以内に完了する一方、1つのタスクが10分以上かかっています。そのステージのタスクサマリーには、Min/Median のシャッフル読み取りが約400 MB であるのに対し、Max のシャッフル読み取りが5 GB を超えることが示されています。エンジニアは、adaptive query execution のスキュー結合処理がクラスターの Spark 設定で明示的に無効化されていることを確認しました。

どのソリューションがジョブのランタイムを短縮しますか？

A. クラスターのサイズを拡大してエグゼキューターを追加し、遅いタスクをより速く完了させる
B. スキュー結合処理を含むアダプティブクエリー実行が有効になっていることを確認し、ランタイム時に過大なパーティションを自動的に分割できるようにします
C. より少ないタスクにより多くの処理をまとめるために、spark.sql.shuffle.partitions を小さくする。
D. 結合の前にソルトキーを使用してデータセットを手動で再パーティション化し、スキューされたキーを均等に分散させます。

質問 7

目標：***Lakeflow Connect*** とマネージドコネクターを使用して、半構造化データおよび非構造化データ（***JSON*** やネストされたデータなど）を ***Delta*** テーブルに取り込む。

データエンジニアが、SaaS ソースから JSON レコードを Unity Catalog の Delta テーブルに取り込むために、LakeFlow Connect のマネージドコネクターを設定する必要があります。ソースのペイロードには、取り込み実行のたびに可変のサブキーを持つ `order_details` というネストされたオブジェクトフィールドが含まれています。エンジニアは、手動でスキーマを更新することなく、コネクターが新しいサブキーをターゲットの Delta テーブルに自動的に取り込めるようにしたいと考えています。

どの設定が要件を満たしていますか？

A. コネクタのインジェスト設定でスキーマ進化モードを 'rescue' に設定し、認識されないサブキーを含むレコードがターゲットスキーマを拡張するのではなく、別のレスキューデータカラムに書き込まれるようにします。
B. ターゲットの Delta テーブルの TBLPROPERTIES で 'mergeSchema' オプションを有効にし、コネクタレベルの設定なしに、書き込み時にコネクタが受信スキーマの変更をマージするようにします。
C. コネクタのインジェスチョン設定でスキーマ進化モードを 'failOnNewColumns' に設定し、インジェスチョンの実行ごとに手動で ALTER TABLE ADD COLUMN を実行して新しいサブキーを取り込みます。
D. マネージドコネクタの自動スキーマ進化を利用することで、新たに検出されたサブキーが手動介入なしに次のパイプライン実行時にターゲットテーブルのスキーマにマージされます。

質問 8

目標：ボリューム、頻度、データタイプ、およびガバナンスの要件に基づいて、***Auto Loader***、***LakeFlow Connect*** パートナーコネクター、およびその他のインジェスト方法の優先順位を判断する。

データエンジニアが、顧客所有の S3 バケットから Databricks の監査ログを取り込むダウンストリームパイプラインを構築しています。スキーマ推論とチェックポイントを実装する前に、配信フォーマット、一般的な取り込みレイテンシー、およびファイルが上書きされる可能性があるかどうかを把握したいと考えています。

Databricks の監査ログのストレージ動作はどのようなものですか？

A. ファイルはJSONとして配信され、配信開始から通常15分以内にイベントログが記録されます。また、新しい配信によって既存のファイルが上書きされる場合があります。
B. ファイルはCSV形式で配信され、1分未満のレイテンシが保証されます。また、イミュータビリティを保持するために、ファイルが書き込まれた後に上書きが行われることはありません。

C. ファイルは24時間を超える結果整合性を持つParquetとして配信され、ストリーミングインジェストを簡素化するために上書きが無効化されています。
D. ファイルはJSONとして配信され、週次バッチのサイクルで配信されます。また、上書きによって以前のコンテンツが完全に置き換えられ、追記は行われません。

質問 9

目的*: **Lakeflow Jobs*** の実行履歴ビューを使用して、ジョブのパフォーマンスの傾向を特定する。

あるデータエンジニアは、夜間ジョブの実行時間が過去数週間にわたって徐々に増加しているかどうかを視覚的に確認し、いずれかの実行が異常に長い時間を要した場合にアラートを受け取りたいと考えています。エンジニアはジョブの「実行」タブを開き、マトリックスビューに切り替えます。

マトリックスビューが期間のトレンドを特定するのにどのように役立つかを正しく説明しているのはどの記述ですか？

A. 各実行およびタスクバーの高さはその期間を視覚的に表しており、ジョブに設定された予想完了時間を超えた実行には警告インジケーターが表示されます。
B. マトリックスビューは、ローリング平均期間を自動的に計算し、それより2標準偏差以上高い実行にフラグを立てます。
C. マトリックスビューは、過去の期間とクラスターのDBUレートに基づいて、将来の実行に対するコスト見積もりを予測して生成します。
D. マトリックスビューは、曜日ごとにランをグループ化し、各グループの期間の中央値を棒グラフで表示します。

質問 10

目標*: **Git Folders*** を使用して ***Databricks*** ワークスペース ***UI*** でコードを管理する*:* ブランチの作成・切り替え、コミット、プッシュ、プルリクエストのオープン。

あるチームは、Databricks で ETL パイプラインをモジュール式にデプロイ、バージョン管理、オーケストレーションし、CI/CD と再現性を実現する方法を求めています。

この要件をサポートしている機能はどれですか？

A. Unity Catalog のモデルを使用して ETL ジョブを表現し、各モデルがパイプラインのコード成果物を格納し、CI/CD がジョブのタスクに紐付けられたモデルエイリアスを更新することでバージョンを昇格させます。
B. 変換ロジックをwheelライブラリとしてパッケージ化してUnity Catalog Volumesに保存し、環境をまたいで決定論的なデプロイメントを実現するためにJobsタスクにバインドします。
C. API ロジックをボリュームにマウントされたノートブック内にパッケージ化し、Jobs API v2 を使用してノートブックをトリガーし、バージョン管理システムとしてノートブックの改訂履歴に依存します。
D. DABs を使用してリソースとコードアセットを定義し、Git でバージョン管理を行い、自動化された CI/CD アクションを通じて環境全体にデプロイメントを推進します。

解答キーと解説

#	解答	解説
1	A	COPY INTO は、Delta テーブルのトランザクションログ内で内部的なロード履歴を維持することにより、すでにロードされたファイルを追跡します。'force' が 'false'（デフォルト値）に設定されている場合、COPY INTO は以前に取り込まれたファイルをスキップするため、次回以降の実行では新しく到着したファイルのみが処理されます。
2	C	Auto Loader はまさにこのシナリオのために設計されています。クラウドオブジェクトストレージのパスを監視して新しいファイルを検出し、内部チェックポイントを使用してすでに処理されたファイルを追跡することで重複取り込みを防止します。また、ディレクトリリスティングモード（低ボリューム向け）またはファイル通知モード（高ボリューム・イベント駆動型の到着向け）を使用して、ファイルボリュームの変動に自動的にスケールします。
3	B	ジョインキーの値の分布に大きな偏り（スキュー）がある場合、少数のシャッフルパーティションが他のパーティションよりもはるかに多くの行を受け取るため、それらのタスクの実行時間が不均衡に長くなり、ステージが停滞します。Spark の Adaptive Query Execution（AQE）には、設定可能なサイズしきい値を超えるパーティションを検出して自動的に小さなサブパーティションに分割するスキュージョイン最適化が含まれており、エグゼキューター間で作業をより均等に分散させます。spark.sql.adaptive.skewJoin.enabled を true に設定すると、エンジニアが手動でデータを再パーティショニングしたりクエリを変更したりすることなく、実行時にこのメカニズムが有効になります。
4	A, C	A — 正しいパターンです。反転させたフィルターによって有効な行と無効な行を別々のテーブルに分割します。 C — FAIL UPDATE は違反発生時にパイプラインを確実に停止させるため、「直ちに失敗させる」という要件を満たします。
5	A	Lakeflow Jobs のファイル到着トリガー（file-arrival trigger）は、監視対象のクラウドストレージパスに新しいファイルが出現したときにジョブの実行を開始するように特別に設計されています。固定スケジュールではなく、ファイルの配置に連動したイベント駆動型の実行を行いたい場合に適切な選択肢です。
6	B	他のタスクが30秒未満であるのに対し、1つのタスクが10分以上かかり、最大シャッフル読み取り量（5 GB 以上）が中央値（約 400 MB）を大きく上回っているのは、典型的なパーティションスキューです。AQE は、パーティションが中央値サイズの5倍以上かつ 256 MB を超えている場合にスキューとしてフラグを立てますが、このケースは両方の条件を満たします。スキュージョイン処理を再度有効にすることで、AQE は過大なパーティションを自動的に分割できます。選択肢 1 は効果がありません。エグゼキューターの数に関係なく、スキューしたパーティション全体が依然として単一のタスクに割り当てられるためです。選択

		肢 3 は、より少ないタスクにより多くのデータを詰め込むため状況を悪化させます。選択肢 4（ソルティング）も有効ですが、AQE を再度有効にすれば不要になる手動のコード変更です。
7	D	マネージドコネクターは新しい列を自動的に処理します。手動のステップを踏むことなく、次回の実行時に新しいフィールドが自動検出されます。選択肢 4 は、設定名を捏造したり、ネストされたフィールドが「トップレベル」の列になると主張したりすることなく、この動作を正確に述べています。選択肢 1（レスキュー）は、スキーマを進化させるのではなく、意図的にスキーマの進化（schema evolution）を回避します。選択肢 2 は、コネクターが設定用として公開していない書き込み時のテーブルオプションである mergeSchema を誤って適用しています。選択肢 3 は手動の ALTER TABLE を必要としますが、これは要件によって除外されています。
8	A	Databricks の監査ログは JSON 形式で配信され、通常は配信開始から約 15 分以内に届きます。また、Databricks は既存のファイルを上書き（置換ではなく新しいイベントを追加）する場合があります。選択肢 1 のみがこれら 3 つの事実すべてと一致しています（他の選択肢は形式、レイテンシ、または上書きの挙動のいずれかが間違っています）。実践的な意味として、ファイルは最初の到着後に追加書き込みされる可能性があるため、下流の取り込みやチェックポイント処理では、ファイルを一度だけ処理するのではなく、再読み込みに対応できるように設計する必要があります。
9	A	マトリックスビューのバーの高さは実行／タスクの所要時間を視覚的に表しており、Databricks は実行がジョブに設定された予想完了時間を超過した際に警告インジケーターを表示します。どちらもドキュメントに記載されている実際の動作です。
10	D	現在 Declarative Automation Bundles（DABs）とも呼ばれる Declarative Automation Bundles は、まさにこの用途において Databricks がドキュメントで推奨しているアプローチです。ジョブ、パイプライン、その他のリソースを Git でバージョン管理するソースファイル（YAML）として定義し、CI/CD ツール（GitHub Actions、Azure DevOps、Jenkins など）を使用して開発、ステージング、本番の各環境にデプロイします。