databricks

Databricks 試験ガイド

Databricks 認定
データエンジニアプロフェッショナル



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	2026年11月20日
試験ガイド更新日	2026年11月20日

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場合（およびその変更の発効日）に更新されますので、自信を持って準備を進めることができます。このガイドは、上記の発効日時点で公開されている試験バージョンを対象としています。最新バージョンをご確認いただくため、受験の2週間前に再度ご確認ください。

対象者の説明

Databricks Certified Data Engineering Professional試験は、Databricks Platform上で本番運用グレードのデータエンジニアリングソリューションを構築、最適化、および維持するための受験者の高度なスキルを評価します。成功した受験者は、Databricks SQL、Unity Catalog、Auto Loader、Lakeflow Pipelines、Databricks コンピュート（サーバーレスを含む）、Lakeflow Jobs、Lakehouse Federation、OpenSharing（D2DおよびDatabricks-to-Open）、Clean Rooms、メダリオンアーキテクチャなどのコアプラットフォーム機能にわたる専門知識を示します。この試験に合格した方は、セキュアで信頼性が高くコスト効率に優れたETL Pipelinesを設計し、PythonおよびSQLを使用して多様なソースからの複雑なデータを処理し、スキーマ管理、オブザーバビリティ、ガバナンス、コンプライアンス、およびパフォーマンス最適化においてベストプラクティスを適用することが期待されます。また、受験者はストリーミングワークロードの実装、ワークフローのオーケストレーション、キャッシング、デバッグ技術、最適化技術、DevOpsおよびCI/CDの活用、Databricks

CLIやREST APIなどのツールを使用したデプロイ、AI FunctionsおよびDeclarative Automation Bundlesの適用についても評価されます。この認定試験に合格した方は、Databricksおよびその関連ツールを使用して高度なデータエンジニアリングタスクを完遂することが期待されます。

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 60 問
採点対象外の問題	採点対象外の問題が含まれる場合があります（後述の「採点対象外のコンテンツ」を参照）
出題される問題の総数	採点対象の問題 60 問＋採点対象外の問題
制限時間	120 分（採点対象外の問題に対応するための追加時間を含みます）
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またはテストセンター
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	なし（ただし、関連トレーニングの受講を強く推奨します）
推奨される経験	試験の出題範囲に記載されたデータエンジニアリングタスクにおける1年以上の実務経験を強く推奨します
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、現行の本番試験を受験する必要があります。

採点対象外のコンテンツ：試験には、将来の利用のための統計的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制限時間に加算されております。

推奨される準備

トレーニング

- インストラクター主導：Advanced Data Engineering with Databricks
- 自己学習型（Databricks Academy で利用可能）：

- Apache Spark™ Declarative パイプラインの高度なテクニック
- Databricks Data Privacy
- Databricks パフォーマンス最適化
- Declarative Automation Bundles を使用した自動デプロイメント

AI Prep ガイド - すべての Databricks 試験

Databricks AI Prep Guide ガイドでは、普段お使いの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チューターに変える方法をご紹介します。公式試験ガイドを活用し、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、すべてのDatabricks試験で求められる実践的な演習と組み合わせる方法をお伝えします。

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（上記の AI Prep ガイドセクションを参照）をご覧ください。AI Prep ガイドには「Step 5 — Hands-on minimum」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対応した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 で完了できる小さな内容です。

Databricksドキュメント

準備を進める中で、この試験でカバーされるプラットフォーム領域に関する公式の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、非推奨の古い内容ではなく、最新の機能を学んでいること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: Python および SQL を使用したデータ処理コードの開発 (23%)

- Declarative Automation Bundles（旧称 Databricks Asset Bundles / DABs）に対して、モジュール開発と CI/CD インテグレーションをサポートするスケーラブルな Python プロジェクト構造を実装します。
- サーバーレス、パイプライン、およびバンドルで展開された環境全体にわたって、外部ライブラリ（PyPI、ローカルホイール、ソースアーカイブ）の依存関係の競合とインストールの失敗に対してトラブルシューティング手法を適用する。
- 指定された制約に対して、Pandas、Python、および SQL を使用したユーザー定義関数（UDF）を開発します。Unity Catalog 関数も含みます。
- Lakeflow Pipelines と Auto Loader を使用して、ストリーミングデータ向けの本番運用に対応したデータパイプラインを作成します。
- UI、API、CLI を使用して、LakeFlow Jobs で ETL ワークロードを作成し、自動化します。
- 指定されたレイテンシ、コスト、および更新要件に対して、ストリーミングテーブルとマテリアライズドビューのどちらかを選択します。

- Lakeflow Pipelines において、`stored_as_scd_type` を使用した SCD Type 1 および SCD Type 2 を含む、AUTO CDC APIs（旧称 APPLY CHANGES）を用いた CDC パイプラインを実装します。
- 運用上の制約を考慮したスケーラブルな ETL のために、Spark 構造化ストリーミングと Apache Spark™ Declarative Pipelines のどちらを使用するかを選択します。
- If/Else 条件や For Each ループなどの制御フロー演算子を使用する LakeFlow Jobs を作成します。
- 環境と依存関係に適切なコンピュートと構成を選択する — サーバーレスコンピュート（サーバーレス環境、依存関係の管理、パフォーマンスモード）、高メモリのノートブックタスク、自動最適化設定（例：リトライの禁止）を含む。
- `assertDataFrameEqual`、`assertSchemaEqual`、`DataFrame.transform`、およびテストフレームワークを使用してユニットテストと統合テストを開発し、コードの正確性を確保します。
- ウォーターマーク、出力モード、foreachBatch、およびフォールトトレラントな正確に1回の状態リカバリのためのチェックポイントを含む、構造化ストリーミングのステートフル処理セマンティクスを適用する。

セクション2: データ取り込みと取得 (12%)

- Delta Lake、Parquet、JSON、Iceberg、CSV、バイナリなど多様なデータ形式を、メッセージバス（Kafka、Kinesis、Pub/Sub）やクラウドストレージといった多様なソースから取り込むデータ取り込みパイプラインを開発します。
- Lakeflow Pipelines を使用して、Delta または Iceberg をターゲットテーブル形式とするインクリメンタル CDC パイプラインを設定します。
- Lakeflow Connect を使用して、SQL Server、MySQL、PostgreSQL などのリレーショナルデータベースソースから CDC インジェストパイプラインを設定します。
- プライバシーを保護したコラボレーションのために、OpenSharing（D2D および Databricks-to-Open）と Clean Rooms を実装します。
- サポートされているソースシステム全体にわたるガバナンスを備えたLakehouse Federationを設定し、UCパーミッションおよび接続レベルの認証情報を適用します。

セクション 3: データ操作 (12%)

- Spark SQL および PySpark を使用して、ウィンドウ関数、結合、集計などの高度なデータ変換処理を適用する大規模データセットの処理を行います。
- Variant データ型および関連する関数（例：`parse_json`、`variant_get`）を使用して、モデルを開発し、半構造化データをクエリーします。
- AI Functions（ai_queryを含む）を適用して、エンリッチメントおよび分類タスクのためにデータパイプライン内でモデル推論を実行します。
- Lakeflow Pipelines でデータ品質の期待値を実装し、不正なレコードを隔離、削除、またはエラーとして処理します。

セクション 4: モニタリングとアラート (10%)

- コスト解析、監査、ワークロードのモニタリングのために、Databricks システムテーブル（billing、コンピュート、access、lakeflow）を使用します。
- Databricks REST APIs / CLI および SDK を使用して、ジョブとパイプラインのモニタリングと分析を行います。
- パイプラインのイベントログを使用して、パイプラインの健全性とデータ品質を監視します。

- Databricks Lakehouse のアラートを使用して、管理されたビジネスメトリクス、データ品質および異常シグナル、使用状況とコスト、SQL warehouse/クエリーのヘルス、監査およびセキュリティイベント、AI エージェントの品質、Lakeflow Job の分岐を監視します。
- Lakeflow Jobs の UI と Jobs API を使用して、ジョブのステータスとパフォーマンスメトリクスを監視します。

セクション5: コスト・パフォーマンス最適化（15%）

- Unity Catalog のマネージドテーブル、Predictive Optimization、およびリキッドクラスタリングが、特定のワークロードに対する運用・メンテナンスのオーバーヘッドをどのように削減するかを説明してください。
- 指定されたテーブルアクセスパターンに対して、適切なDelta最適化手法（削除ベクター、リキッドクラスタリング、CLUSTER BY AUTO）を選択してください。
- キャッシュ（Delta cache）が同じデータの繰り返し読み取りに対するクエリーパフォーマンスをどのように向上させるかを理解します。
- チェンジデータフィード（CDF）を適用することで、行レベルの変更（更新/削除）を公開し、効率的なインクリメンタルなダウンストリーム処理を実現します。
- クエリープロファイルを使用して、データスキッピングの非効率性、結合戦略、シャッフル操作などのパフォーマンスボトルネックを特定します。
- 特定のテーブルサイズとクエリーパターンに対して、リキッドクラスタリングとパーティショニング/ZORDERを比較してください。

セクション6: データセキュリティとコンプライアンスの確保（8%）

- 最小権限のアクセス制御リスト（ACL）を適用して、Unity Catalog のセキュリティ保護可能なオブジェクトとワークスペースのリソースを保護します。
- ガバナンスされたタグを使用した属性ベースのアクセス制御（ABAC）ポリシーを適用する ことで、行フィルターと列マスクをスケールで適用します。
- 機密データに対して、匿名化および仮名化の手法（ハッシュ化、トークン化、抑制、汎化など）を適用する（例：列マスクおよび関連する Unity Catalog 機能を使用する）。
- Unity Catalog の機能を使用して、バッチおよびストリーミングワークロード全体にわたって PII の検出とマスキングコントロールを適用する、準拠したデータパイプラインを実装します。
- Delta Lake と Unity Catalog の機能を使用して、データ保持および削除ポリシー（例：GDPR の消去権、GDPR の忘れられる権利による削除）を満たすデータパージ戦略を実装します。

セクション 7: データガバナンス (5%)

- セキュリティ保護可能なオブジェクトにメタデータを追加してデータの検索性を向上させる仕組みとして、Unity Catalog のタグとコメントについての理解を示してください。
- カタログ、スキーマ、およびオブジェクト全体のアクセス制御を管理するメカニズムとして、Unity Catalog の権限継承モデルへの理解を示してください。

セクション 8: デバッグとデプロイ (10%)

- Spark UI、クラスターログ、システムテーブル、およびクエリープロファイルを使用して関連する診断情報を特定し、エラーのトラブルシューティングを行います。
- エラーを分析し、ジョブの修復とパラメーターのオーバーライドを使用して、失敗したジョブの実行を修正します。

- Declarative Automation Bundles（旧称 Databricks Asset Bundles）を使用して Databricks リソースをデプロイします。
- Databricks Git Folders（旧称 Repos）を使用して Git ベースの CI/CD Workflows を設定し、ノートブックとコードをデプロイします。

セクション 9: データモデリング (5%)

- パーティショニングをデータの粒度にマッピングし、クラスタリングをリレーションシップのアクセスパターンに合わせ、コンパクションによってバランスの取れたファイルサイズを維持することで、大規模なデータ資産に対してスケーラブルなDelta/Iceberg テーブルレイアウトを設計します。
- 分析ワークロード向けのディメンショナルモデルを設計し、事前計算済み集計のためにMaterialized Viewsを活用し、ガバナンスの効いた再利用可能なメトリクス定義のためにUnity Catalog Metric Viewsを活用することで、効率的なクエリと集計を実現します。

サンプル問題

この目的は、試験ガイドに記載されている目標をそのままお見せし、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、ご自身の受験日の試験アウトライン全体を以上でご確認いただくことです。

下のサンプル問題は新しい試験の概要に沿っています。試験日が**2026**年**10**月**9**日以前の場合、これらの問題は引き続き練習として役立ちますが、一部はご自身の試験の出題範囲に含まれない目標を扱っている場合があります。試験の概要が、出題内容の正式なリストです。

質問 **1**

目標**:** 指定されたレイテンシ、コスト、および更新要件に対して、ストリーミングテーブルとマテリアライズドビューのどちらかを選択する。

LakeFlow パイプラインのゴールドレイヤーオブジェクトは、ブロンズソースから日次注文合計を集計します。下流の BI ダッシュボードはスケジュールに従ってこれを読み取り、集計は新たに追加された行だけでなく、過去の日付への遅延到着更新を含む完全な履歴を常に反映している必要があります。

エンジニアはどのオブジェクトを定義する必要がありますか？

A. ストリーミングテーブル。各ソース行をちょうど1回処理し、あらゆる集計において最もコストが低いオプションであるためです。
B. 一時ビュー。ストレージコストをかけずに、クエリー時に常に最新のデータを反映するためです。
C. マテリアライズドビューは、完全な集計結果をインクリメンタルに維持し、スケジュールに従って更新でき、遅延到着したデータの変更を反映するため、適切です。
D. foreachBatch マージを使用したストリーミングテーブル。Materialized Views は集計を実行できないためです。

質問 2

目標: ウォーターマーク、出力モード、**foreachBatch**、およびフォールトトレラントな **exactly-once** 状態回復のためのチェックポイントを含む、構造化ストリーミングのステートフル処理セマンティクスを適用する。

構造化ストリーミングのクエリーは、イベント時間ウィンドウ集計を実行します。非常に遅延したイベントによって引き起こされる状態の増大を制限する必要があり、かつドライバーの障害後に結果を二重カウントすることなく処理を再開できなければなりません。

どの構成が両方の要件を満たしていますか？

A. イベント時刻カラムにウォーターマークを設定し、再起動時にオフセットと状態を復元するためにクエリーのチェックポイントに依存します。
B. 正確性のためにすべての状態を保持するには、ウォーターマークなしの complete 出力モードを使用してください。
C. リカバリを高速化するために、spark.sql.shuffle.partitions を増やし、チェックポインティングを無効にします。
D. 集計処理を `foreachBatch` でラップし、ソースのオフセットを Delta テーブルに手動でコミットします。

問題 3

目標: **LakeFlow Connect** を使用して、**SQL Server**、**MySQL**、**PostgreSQL** などのリレーショナルデータベースソースから **CDC** インジェストパイプラインを設定する。

データエンジニアは、運用中の PostgreSQL データベースからの挿入、更新、削除の変更イベントを、カスタムコードを最小限に抑えながら lakehouse へ継続的にレプリケートする必要があります。

どのアプローチが要件に適合していますか？

A. 毎晩の JDBC フルテーブル読み取りをスケジュールし、毎晩ターゲットテーブルを上書きします。
B. PostgreSQL のチェンジデータキャプチャを lakehouse に取り込むために、LakeFlow Connect のマネージドコネクターを設定します。
C. データベースをCSVとしてクラウドストレージにエクスポートし、Auto Loaderでファイルを取り込みます。
D. OpenSharing を使用して、PostgreSQL テーブルを lakehouse と直接共有します。

質問 4

目標: **VARIANT** データ型および関連する関数（例: **parse_json**、**variant_get**）を使用して、モデルを開発し、半構造化データをクエリーします。

ブロンズテーブルは、形状が異なる生のJSONペイロードを単一の文字列カラムに格納しています。エンジニアは、固定スキーマを事前に宣言することなく、効率的なストレージと高速なフィールド抽出を実現したいと考えています。

何をすべきですか？

A. `parse_json` を使用してペイロードを Variant カラムにパースし、`variant_get` およびコロンパスアクセスを使用してフィールドを抽出します。
B. 固定のSTRUCTスキーマを使用して、JSONをキーごとに1行になるように展開します。
C. ペイロードを STRING として保持し、クエリー時に各フィールドに対して regexp_extract を使用します。
D. 取り込み前に、JSON を厳密な固定スキーマを持つ Parquet に変換します。

質問 5

目標: コスト解析、監査、ワークロードモニタリングのために、**Databricks** システムテーブル（**billing**、コンピュート、**access**、**LakeFlow**）を使用します。

データエンジニアのチームは、先月の DBU 支出を特定のジョブおよび SQL ウェアハウスに帰属させ、最もコストの高いワークロードを特定する必要があります。その際、Unity Catalog ですでに利用可能なガバナンス済みデータを使用します。

どのソースをクエリーすべきですか？

A. 各クラスターの Spark UI イベントタイムライン。
B. ワークスペースからダウンロードされたクラスターごとのドライバーログ。
C. Unity Catalog 内の `system.billing.usage` テーブルと、コンピュートおよび価格のシステムテーブルを結合したもの。
D. アカウントコンソールの請求書PDF エクスポート。

問題 6

目標: 特定のテーブルアクセスパターンに対して、適切な **Delta** 最適化手法（削除ベクター、リキッドクラスタリング、**CLUSTER BY AUTO**）を選択する。

データエンジニアは、大規模な Delta テーブルが頻繁に小さなマージ操作を受け取り、個々の行を更新・削除していることに気づきました。チームは、変更のたびにデータファイル全体を書き換えることなく、読み取りを高速に保ちたいと考えています。

ライトアンプリフィケーション（書き込み増幅）に対処する手法はどれですか？

A. テーブルをプライマリキーでパーティション分割します。

B. 削除ベクターを有効にすることで、ファイル全体をすぐに書き換えることなく、更新と削除が行レベルでマークされるようにします。
C. すべての MERGE の後に、保持期間をゼロ時間に設定して VACUUM を実行します。
D. バックグラウンドでのファイル書き換えを防ぐために、Predictive Optimization を無効にします。

質問 7

目標: ガバナンスされたタグを使用した属性ベースのアクセス制御（**ABAC**）ポリシーを適用する、スケールでの行フィルターと列マスクを適用するために。

あるデータエンジニアは、PII を含む数百のテーブルを持つ組織で働いており、各テーブルを個別に変更することなく、現在および将来のすべてのテーブルにわたって 'pii' とタグ付けされた列をマスクするための、一元管理された単一のルールを求めています。

何を実装すべきですか？

A. 既存のすべてのテーブルに対して、テーブルごとに `ALTER COLUMN SET MASK` ステートメントを実行します。
B. 各スキーマに行フィルター関数をアタッチします。
C. マスキングロジックを含むすべてのベーステーブルを複製する動的ビューを作成する。
D. 管理対象の 'pii' タグが付与された列にカラムマスクを適用する ABAC ポリシーを定義する。

質問 8

目標：カタログ、スキーマ、およびオブジェクト全体のアクセス制御を管理するメカニズムとして、**Unity Catalog** の権限継承モデルへの理解を示す。

データエンジニアが、アナリストのグループに対してカタログへの SELECT 権限を付与します。その後、そのカタログに新しいスキーマとテーブルが作成されます。

競合するグラントがないと仮定した場合、アナリストは新しいテーブルにどのようなアクセス権を持ちますか？

A. カタログレベルで付与された権限は、後から作成されたものも含め、その中のスキーマおよびオブジェクトに継承されるため、新しいテーブルに対して SELECT を実行できます。
B. 新しいテーブルごとに再度 SELECT が付与されるまで、アクセス権はありません。
C. 権限が付与された時点で存在していたテーブルのみを SELECT できます。
D. 新しいテーブルの所有権が自動的に付与されます。

質問 9

目標：**Declarative Automation Bundles**（旧称 **Databricks Asset Bundles**）を使用して **Databricks** リソースをデプロイする。

データエンジニアは、同じジョブとパイプラインの定義を、環境固有のオーバーライドとCI/CDを使用して、ソースコントロールからdev、staging、prodに同一の形で展開されることを望んでいます。

どのアプローチが要件を満たしていますか？

A. 各ワークスペースのUIでジョブを手動で再作成し、その後JSONをエクスポートおよびインポートします。
B. databricks.yml バンドル内でリソースをターゲットごとのオーバーライドとともに定義し、`databricks bundle deploy -t <target>` を使用してデプロイします。
C. ワークスペース間でノートブックをコピーするには、workspace import CLIのみを使用します。
D. 構成のドリフトを避けるために、すべての環境で単一の共有ワークスペースを使用します。

質問 10

目標：効率的なクエリと集計を実現するために、事前計算された集計のために**Materialized Views**を活用し、ガバナンスが効いた再利用可能なメトリクス定義のために**Unity Catalog Metric Views**を活用して、分析ワークロード向けのディメンショナルモデルを設計する。

データエンジニアリングチームは、各ダッシュボードでわずかに異なる SQL を使用して「純収益」を計算しており、レポート間で数値に一貫性がない状態です。また、クエリー時に大規模な結合と集計を毎回再計算するため、同じダッシュボードの動作も遅くなっています。チームは、メトリクスの唯一の権威ある定義と、ダッシュボードが直接読み取れる事前集計済みの結果を必要としています。

エンジニアはどの実装を選択すべきですか？

A. 純収益の計算方法を説明するコメントを各ダッシュボードに追加する。
B. すべてのアナリストに、共有された SQL スニペットファイルへの編集アクセス権を付与します。
C. Unity Catalog のメトリクスビューでネット収益を一度定義してガバナンスの効いた再利用可能な定義を作成し、ダッシュボードが読み取る事前計算済みの集計には Materialized Views を使用します。
D. メトリクスを Python UDF として各ノートブックにコピーして保存します。

解答キーと解説

#	解答	解説
1	C	マテリアライズドビューは集計全体をインクリメンタルに維持し、スケジュールに従ってリフレッシュできるため、過去の日付に遅れて到着した更新も正しく反映されます。一方、ストリーミングテーブルは追加された行をインクリメンタルに処理するもので、変化する履歴全体に対して集計を再計算することはありません。
2	A	ウォーターマークにより、エンジンは許容される遅延を超えたイベントのステートを破棄できます（ステートの増大を抑制）。また、クエリーのチェックポイントがオフセットとステートを永続化するため、再起動後も二重カウントすることなく exactly-once で処理を再開できます。
3	B	Lakeflow Connect は、SQL Server、MySQL、PostgreSQL から、削除を含む CDC を最小限のコードで取り込むマネージドデータベースコネクターを提供します。その他の選択肢は、フルリロードであるか、運用中の RDBMS には適用できません。
4	A	VARIANT 型は半構造化データを効率的に格納し、parse_json と variant_get によるスキーマに柔軟なクエリーをサポートするため、厳密なスキーマを事前に宣言する必要がありません。
5	C	system.billing.usage などのシステムテーブル（コンピュートや価格のシステムテーブルと結合）は、ガバナンスされたクエリー可能なレコードであり、コストの帰属分析やワークロード分析に最適です。その他のソースは、大規模にクエリーできないか、ガバナンスされたテーブルではありません。
6	B	削除ベクターは、データファイル全体をすぐに書き換えることなく、更新・削除された行をマークします。これにより、読み取りの正確性を保ちながら、頻繁な小規模 MERGE 操作による書き込み増幅を軽減できます。マークされた行は、後でコンパクションによって整理されます。
7	D	ガバナンスされたタグをキーとする ABAC ポリシーは、後から作成されたテーブルも含め、'pii' タグが付与されているすべての箇所に列マスクを適用するため、テーブルごとのマスクや複製されたビューとは異なり、ルールを一元的にスケールさせることができます。
8	A	Unity Catalog の権限は階層の下位へ継承されます。カタログに対する権限付与は、そのカタログ内のスキーマおよびオブジェクトに適用され、権限付与後に作成されたオブジェクトも対象となります。
9	B	Declarative Automation Bundle（databricks.yml）は、環境ごとのターゲットとオーバーライドを指定してリソースを宣言し、CLI を介してデプロイします。これにより、環境間で再現性が高く、CI/CD に適したデプロイが可能になります。
10	C	メトリクスビューは純収益について、ガバナンスされた再利用可能な単一の定義を提供し、マテリアライズドビューはダッシュボードが読み取る集計を事前計算します。これにより、一貫性とパフォーマンスの両方の問題を解決できます。