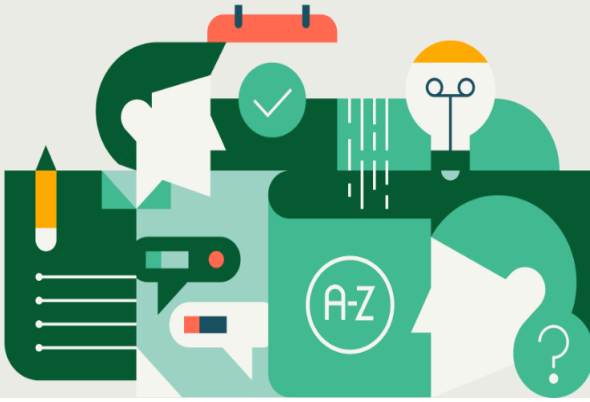
databricks

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



試験ガイドへのフィードバックを送信する

この試験ガイドの目的

この試験ガイドは、Databricks 認定データエンジニアプロフェッショナル試験の概要と出題範囲を示し、受験準備の状況を把握できるよう支援することを目的としています。本ドキュメントは、試験内容に変更があった場合（およびその変更が試験に反映される時期）に随時更新されますので、常に準備を整えておくことができます。最新版を確実に入手するため、受験の **2** 週間前に必ず再度ご確認ください。本バージョンは、**2025** 年 **11** 月 **30** 日時点で公開されている現行バージョンに対応しています。

対象者の説明

Databricks 認定データエンジニアリングプロフェッショナル試験は、Databricks Data Intelligence Platform 上で本番運用レベル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、維持するための、受験者の高度なスキルを検証します。合格者は、Delta Lake、Unity Catalog、Auto Loader、Lakeflow Spark Declarative Pipelines、Databricks コンピュート（サーバーレスを含む）、Lakeflow ジョブ、メダリオンアーキテクチャといった中核的なプラットフォーム機能全般にわたる専門知識を備えています。この認定資格では、安全で信頼性が高く、コスト効率に優れた ETL パイプラインを設計し、Python と SQL を用いて多様なソースの複雑なデータを処理し、スキーマ管理、可観測性、ガバナンス、パフォーマンス最適化におけるベストプラクティスを適用する能力を評価します。受験者はさらに、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、DevOps と CI/CD の活用、そして Databricks CLI、REST API、Asset Bundles といったツールによるデプロイについても評価されます。この認定試験の合格者には、Databricks とその関連ツールを用いて高度なデータエンジニアリングタスクを遂行できることが期待されます。

試験について

- 問題数: 採点対象となる多肢選択式問題 59 問
- 制限時間: 120 分
- 受験料: USD 200（現地の法律で必要とされる場合は、適用される税金が別途加算されます）
- 実施方法: オンライン監督試験およびテストセンターでの監督試験
- 試験補助資料: API ドキュメントを含め、いかなる試験補助資料も提供されません。
- 前提条件: 必須の前提条件はありません。ただし、関連コースの受講と、本試験ガイド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 年間の実務経験を強く推奨します。
- 有効期間: 2 年間

- 再認定: 認定ステータスを維持するには、2 年ごとの再認定が必要です。再認定を受けるには、現行バージョンの試験を受験する必要があります。再認定試験の準備については、以下の「試験に向けた準備」セクションをご確認ください。
- 採点対象外コンテンツ: 試験には、将来の利用に向けた統計情報を収集するための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フォーム上で識別されることはなく、スコアに影響することもありません。また、これらのコンテンツに対応するための追加時間が考慮されています。

推奨トレーニング

- 講師主導型: Advanced Data Engineering With Databricks (Databricks を活用した高度なデータエンジニアリング)
- セルフペース型(Databricks Academy で受講可能):
    - Databricks Streaming and Lakeflow Spark Declarative Pipelines (Databricks Streaming および Lakeflow Spark Declarative Pipelines)
    - Databricks Data Privacy (Databricksのデータプライバシー)
    - Databricks Performance Optimization (Databricksのパフォーマンス最適化)
    - Automated Deployment with Databricks Asset Bundle (Databricks Asset Bundle を使用した自動デプロイ)

試験の概要

セクション **1: Python** と **SQL** を用いたデータ処理用コードの開発

- 開発における Python とツールの使用
- モジュール式の開発、デプロイの自動化、CI/CD 連携を可能にする、Databricks Asset Bundles (DABs) に最適化されたスケーラブルな Python プロジェクト構造を設計および実装する。
- PyPI パッケージ、ローカル Wheel、ソースアーカイブなど、Databricks におけるサードパーティ製外部ライブラリのインストールと依存関係を管理し、トラブルシューティングを行う。
- Pandas/Python UDF を用いてユーザー定義関数 (UDF) を開発する。
- Lakeflow Spark Declarative Pipelines、SQL、Apache Spark を用いた Databricks Platform上での ETL パイプラインの構築とテスト
- Lakeflow Spark Declarative Pipelines と Autoloader を用いて、バッチデータおよびストリーミングデータ向けの信頼性が高く本番運用に対応したデータパイプラインを構築および管理する。
- UI/API/CLI を介したジョブを用いて ETL ワークロードを作成および自動化する。
- マテリアライズドビューと比較した場合のストリーミングテーブルの利点と欠点を説明する。
- APPLY CHANGES API を使用して、Lakeflow Spark Declarative Pipelines における CDC を簡素化する。
- Spark 構造化ストリーミングと Lakeflow Spark Declarative Pipelines を比較し、スケーラブルな ETL パイプラインを構築するための最適なアプローチを判断する。
- 制御フロー演算子(例: if/else、for/each など)を使用するパイプラインコンポーネントを作成する。
- 環境と依存関係に適した構成、ノートブックタスク向けの高メモリ構成、リトライを許可しない自動最

適化構成を選択する。
- assertDataFrameEqual、assertSchemaEqual、DataFrame.transform、各種テストフレームワーク、および組み込みデバッガーを用いて、コードの正確性を担保するユニットテストと統合テストを開発する。

セクション **2:** データの取り込みと取得**:**
- メッセージバスやクラウドストレージなどの多様なソースから、Delta Lake、Parquet、ORC、AVRO、JSON、CSV、XML、Text、Binary を含むさまざまなデータ形式を効率的に取り込むデータ取り込みパイプラインを設計および実装する。
- Delta を用いて、バッチデータとストリーミングデータの両方を処理できる追記専用のデータパイプラインを作成する。

セクション **3:** データの変換、クレンジング、品質
- 効率的な Spark SQL コードと PySpark コードを記述し、ウィンドウ関数、結合、集計などの高度なデータ変換を適用して、大規模なデータセットを操作および分析する。
- Lakeflow Spark Declarative Pipelines、またはクラシックジョブにおける autoloader を用いて、不正なデータを隔離するプロセスを開発する。

セクション **4:** データ共有とフェデレーション
- Databricks 間共有 (D2D) を用いた Databricks デプロイ間、またはオープン共有プロトコル (D2O) を用いた外部プラットフォームとの間で、Delta Sharing を安全に実行する方法を示す。
- サポート対象のソースシステム全体にわたり、適切なガバナンスを備えた Lakehouse Federation を構成する。
- Delta Share を用いて、Lakehouse から任意のコンピューティングプラットフォームへライブデータを共有する。

セクション **5:** モニタリングとアラート
- モニタリング
    - システムテーブルを用いて、リソース使用率、コスト、監査、ワークロードのモニタリングに関する可観測性を確保する。
    - Query Profiler UI と Spark UI を用いてワークロードをモニタリングする。
    - Databricks REST API/Databricks CLI を用いてジョブとパイプラインをモニタリングする。
    - Lakeflow Spark Declarative Pipelines のイベントログを用いてパイプラインをモニタリングする。
- アラート
    - SQL Alerts を用いてデータ品質をモニタリングする。
    - Lakeflow Jobs UI と Jobs API を用いて、ジョブのステータスやパフォーマンスの問題に関する通知を設定する。

セクション **6:** コストとパフォーマンスの最適化
- Unity Catalog マネージドテーブルの使用が、運用上のオーバーヘッドと保守負担をどのように、そ

してなぜ軽減するのかを理解する。
- 削除Vectorsやリキッドクラスタリングなど、Delta の最適化技術を理解する。
- 大規模なデータセットに対するクエリのパフォーマンスを確保するために Databricks が用いる最適化技術（データスキッピング、ファイルプルーニングなど）を理解する。
- Change Data Feed (CDF) を適用して、ストリーミングテーブルの特定の制約に対処し、レイテンシを改善する。
- クエリプロファイルを用いてクエリを分析し、不適切なデータスキッピング、非効率な結合の種類、データシャッフルなどのボトルネックを特定する。

セクション **7:** データセキュリティとコンプライアンスの確保
- データセキュリティメカニズムの適用。
    - ACL を用いてワークスペースオブジェクトを保護し、最小権限の原則やポリシーの適用といった原則を徹底しながら、最小権限の原則を実施する。
    - 行フィルターと列マスクを用いて、機密性の高いテーブルデータをフィルタリングおよびマスキングする。
    - ハッシュ化、トークン化、抑制、一般化などの匿名化および仮名化の手法を機密データに適用する。
- コンプライアンスの確保
    - PII を検出してマスキングを適用し、データプライバシーを確保するコンプライアンス対応のバッチ&ストリーミングパイプラインを実装する。
    - データ保持ポリシーへの準拠を確保するデータパージソリューションを開発する。

セクション **8:** データガバナンス
- エンタープライズデータに関する説明やメタデータを作成/付与し、データの発見性を高める。
- Unity Catalog の権限継承モデルに関する理解を示す。

セクション **9:** デバッグとデプロイ
- デバッグとトラブルシューティング
    - Spark UI、クラスターログ、システムテーブル、クエリプロファイルを用いて、エラーのトラブルシューティングに必要な診断情報を特定する。
    - 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用いて、失敗したジョブ実行を修正する。
    - Lakeflow Spark Declarative Pipelines のイベントログと Spark UI を用いて、Lakeflow Spark Declarative Pipelines と Spark パイプラインをデバッグする。
- CI/CD のデプロイ
    - Databricks Asset Bundles を用いて Databricks リソースを構築およびデプロイする。
    - ノートブックとコードのデプロイにあたり、Databricks Git Folders を用いて Git ベースの CI/CD ワークフローを構成および連携する。

セクション **10:** データモデリング
- Delta Lake を用いて、大規模なデータセットを管理するためのスケーラブルなデータモデルを設計お

よび実装する。
- リキッドクラスタリングを用いて、データレイアウトの決定を簡素化し、クエリのパフォーマンスを最適化する。
- パーティショニングや ZOrder に対するリキッドクラスタリングの利点を特定する。
- 分析ワークロード向けのディメンションモデルを設計し、効率的なクエリと集計を実現する。

サンプル問題

これらの問題は、過去バージョンの試験から引退したものです。その目的は、試験ガイドに記載されている各目標を示し、その目標に沿ったサンプル問題を提供することにあります。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。認定試験に向けた最善の準備方法は、試験ガイドに記載された試験の概要を確認することです。

問題 **1**
目標: *Delta Lake* のカタログ*-Metastore*操作と *ACID* 準拠の動作を理解する。

Delta Lake テーブルが、次のクエリで作成されました。

```
CREATE TABLE prod.sales_by_stor
USING DELTA
LOCATION "/mnt/prod/sales_by_store"
```

元のクエリにタイプミスがあったことに気づき、以下のコードが実行されました。`ALTER TABLE prod.sales_by_stor RENAME TO prod.sales_by_store` 2 番目のコマンドを実行した後、どの結果が発生しますか？

A. 関連するすべてのファイルとメタデータが、単一の ACID トランザクションの中で削除され、再作成される。
B. テーブル名の変更が Delta トランザクションログに記録される。
C. 名前が変更されたテーブル用に、新しい Delta トランザクションログが作成される。
D. Metastore内のテーブル参照が更新される。

問題 **2**
目標: *Spark* 構造化ストリーミングの動作を理解し、本番環境で *SLA* に対応したパイプラインに最適なアプローチを判断する。

本番環境にデプロイされた構造化ストリーミングジョブで、1 日のピーク時間帯に遅延が発生しています。現在、通常の実行時には、各マイクロバッチのデータは 3 秒未満で処理されています。1 日のピーク時間帯になると、各マイクロバッチの実行時間が

非常に不安定になり、30 秒を超えることもあります。ストリーミング書き込みは現在、10 秒のトリガー間隔で構成されています。
他のすべての変数を一定に保ち、レコードを 10 秒未満で処理する必要があると仮定した場合、要件を満たす調整はどれですか？

A. トリガーを once オプションに設定し、Databricks jobを 8 秒ごとにクエリを実行するよう構成する。これにより、バックログされたすべてのレコードが各バッチで処理されることが保証される。
B. トリガー間隔を 5 秒に短縮する。バッチをより頻繁にトリガーすることで、レコードのバックログや、大きなバッチによるスピルの発生を防げる可能性がある。
C. トリガー間隔を 5 秒に短縮する。バッチをより頻繁にトリガーすることで、アイドル状態のエグゼキューターが、前のバッチの長時間実行タスクが完了する間に次のバッチの処理を開始できるようになる。
D. チェックポイントディレクトリを変更しない限り、トリガー間隔は変更できない。現在のストリーム状態を維持するため、シャッフルパーティションの数を増やして並列性を最大化する。

問題 **3**
目標: *Delta Lake* を用いて、大規模なデータセットを管理するためのスケーラブルなデータモデルを設計および実装する。

ユーザーによるコンテンツ投稿に関するメタデータを表す Delta Lake テーブルが、次のスキーマを持っています。

```
schema:

user_id LONG, post_text STRING, post_id STRING, longitude FLOAT, latitude
FLOAT, post_time TIMESTAMP, date DATE
```

上記のスキーマに基づくと、Delta テーブルをパーティショニングするのに適した列はどれですか？

A. post_id
B. post_time
C. date
D. user_id

問題 **4**
目標: *Unity Catalog* の権限継承モデルに関する理解を示す

user_ltv という名前のテーブルが、さまざまなチームのデータアナリストによって使用されるビューを作成するために使われています。ワークスペースのユーザーはグループに編成されており、これらのグループは ACL を用いたデータアクセスの設定に使用されます。
user_ltv テーブルは、次のスキーマを持っています。

```
email STRING, age INT, ltv INT
```

次のビュー定義が実行されます。

```
CREATE VIEW email_ltv AS
SELECT
CASE WHEN
is_member('marketing') THEN email
ELSE 'REDACTED'
END AS email,
ltv
FROM user_ltv
```

マーケティング グループのメンバーではないアナリストが、次のクエリを実行します。

```
SELECT * FROM email_ltv
```

このクエリの結果はどうなりますか？

A. email 列と ltv 列のみが返される。email 列は、各行に文字列 "REDACTED" を含む。
B. 3 つの列が返されるが、そのうちの 1 列は "REDACTED" という名前で、null 値のみを含む。
C. email 列と ltv 列のみが返される。email 列はすべて null 値を含む。
D. email 列と ltv 列が、user_ltv 内の値とともに返される。

問題 **5**
目標: 環境と依存関係に適した構成、ノートブックタスク向けの高メモリ構成、リトライを許可しない自動最適化構成を選択する。

ビジネスレポートチームは、ダッシュボード用のデータを 1 時間ごとに更新することを求めています。データの抽出、変換、ロードを行うこのパイプラインの 1 回の実行あたりの総処理時間は 10 分です。
通常の運用条件を想定した場合、最も低いコストでサービスレベルアグリーメントの要件を満たす構成はどれですか？

A. 専用のインタラクティブクラスター上で、1 時間に 1 回パイプラインを実行するようジョブをスケジュールする。
B. 新しいジョブクラスター上で、1 時間に 1 回パイプラインを実行するようジョブをスケジュールする。
C. トリガー間隔を 60 分に設定した構造化ストリーミングジョブをスケジュールする。
D. 指定したディレクトリに新しいデータが到着するたびに実行されるジョブを構成する。

問題 **6**
目標: ノートブック開発環境、変数管理、そして安全で構成可能なコードの作成について理解する。

セキュリティチームは、外部データベースへの接続に Databricks の secretsモジュールを活用できるかどうかを検討しています。
すべての Python 変数を文字列で定義した状態でコードをテストした後、彼らはパスワードを secretsモジュールにアップロードし、現在アクティブなユーザーに対して適切な権限を構成します。そして、コードを次のように変更します（他のすべての変数は変更しません）。

```
password = dbutils.secrets.get(scope="db_creds", key="jdbc_password")
print(password)
df = (spark
.read
.format("jdbc")
.option("url", connection)
.option("dbtable", tablename)
.option("user", username)
.option("password", password)
)
```

このコードを実行すると、何が起こりますか？

A. 外部テーブルへの接続は成功し、文字列 "REDACTED" が出力される。
B. 外部テーブルへの接続は成功し、パスワードの文字列値がプレーンテキストで出力される。
C. ノートブックにインタラクティブな入力ボックスが表示される。正しいパスワードが入力されれば接続は成功し、パスワードがプレーンテキストで出力される。
D. ノートブックにインタラクティブな入力ボックスが表示される。正しいパスワードが入力されれば接続は成功し、エンコードされたパスワードが DBFS に保存される。

問題 **7**
目標: 大規模なデータセットに対するクエリのパフォーマンスを確保するために *Databricks* が用いる最適化技術（データスキッピング、ファイルプルーニングなど）を理解する

あるデータ取り込みタスクでは、1 TB の JSON データセットを、ターゲットのパートファイルサイズ 512 MB で Parquet に書き出す必要があります。Delta Lake ではなく Parquet を使用しているため、Auto-Optimize（自動最適化）や Auto-Compaction（自動コンパクション）といった組み込みのファイルサイズ調整機能は使用できません。

データを並べ替えることなく機能するアプローチはどれですか？

A. データを取り込み、ナロー変換を実行し、2,048 パーティション (1TB*1024*1024/512) に再パーティショニングしてから、Parquet に書き込む。
B. spark.sql.adaptive.advisoryPartitionSizeInBytes を 512 MB に設定し、データを取り込み、ナロー変換を実行し、2,048 パーティションに coalesce する
(1TB*1024*1024/512)。その後、Parquet に書き込む。

C. spark.sql.files.maxPartitionBytes を 512 MB に設定し、データを取り込み、ナロー変換を実行してから、Parquet に書き込む。
D. spark.sql.shuffle.partitions を 2,048 パーティション (1TB*1024*1024/512) に設定し、データを取り込み、ナロー変換を実行し、データをソートして最適化し（これによりデータが自動的に再パーティショニングされる）、その後 Parquet に書き込む。

問題 **8**
目標: *Delta Lake* のクローンを適用し、シャロークローンとディープクローンがソース/ターゲットテーブルとどのように相互作用するかを学ぶ。

マーケティングチームは、集計テーブル内のデータを営業部門と共有したいと考えています。しかし、両チームが使用するフィールド名は一致しておらず、また、いくつかのマーケティング固有のフィールドは営業部門向けには承認されていません。
シンプルさを重視しつつ、この状況に対応するソリューションはどれですか？

A. 営業チーム向けに承認されたフィールドのみを選択するビューをマーケティングテーブル上に作成する。標準化すべきフィールドの名前には、営業部門の命名規則に合わせてエイリアスを付ける。
B. 必要なスキーマを持つ新しいテーブルを作成し、Delta Lake の DEEP CLONE 機能を用いて、一方のテーブルにコミットされた変更を対応するテーブルに同期する。
C. CTAS ステートメントを使用してマーケティングテーブルから派生テーブルを作成し、本番ジョブを構成して変更を伝播させる。
D. 現在の本番パイプラインに並行したテーブル書き込みを追加し、マーケティングテーブルとは必要に応じて異なる新しい営業テーブルを更新する。

問題 **9**
目標: 複数の依存関係を持つマルチタスクジョブを作成する。

ある Databricks Jobが、それぞれ Databricks notebooksである 3 つのタスクで構成されています。タスク A は他のタスクに依存しません。タスク B とタスク C は並列で実行され、それぞれがタスク A に対して直列の依存関係を持っています。

スケジュール実行の際にタスク A とタスク B は正常に完了したものの、タスク C が失敗した場合、結果として生じる状態はどれですか？

A. タスク A とタスク B に関連付けられたノートブックで表現されたすべてのロジックは正常に完了している。タスク C の一部の操作は正常に完了している可能性がある。
B. すべてのタスクが正常に完了しない限り、Lakehouse には変更がコミットされない。タスク C が失敗したため、すべてのコミットが自動的にロールバックされる。
C. タスク A とタスク B に関連付けられたノートブックで表現されたすべてのロジックは正常に完了している。タスク C で行われた変更は、タスクの失敗によりロールバックされる。

D. すべてのタスクは依存関係グラフとして管理されるため、すべてのタスクが正常に完了するまで、Lakehouse には変更がコミットされない。

解答

問題 *1*: *D*
問題 *2*: *B*
問題 *3*: *C*
問題 *4*: *A*
問題 *5*: *B*
問題 *6*: *A*
問題 *7*: *C*
問題 *8*: *A*
問題 *9*: *A*