Delta Lake シリーズ

Delta Lake の基礎と性能

Delta Lake で機械学習と BI における
データの信頼性を高める

databricks

このeBookの概要

Databricks の Delta Lake の eBook シリーズは、データを扱う方が Delta Lake のフル機能を理解して利活用するための支援を目的として提供されています。

この eBook **「Delta Lake シリーズ：Delta Lake の基礎と性能」**では、Delta Lake の基礎と性能を詳しく解説します。

学べる内容

Delta Lake が提供する機能の詳細と、その機能が性能を大幅に向上させる仕組みを理解できます。

databricks

目次

Delta Lake とは

Delta Lake は、データの信頼性を高め、迅速な分析をクラウドのデータレイクにもたらす統合データ管理システムです。既存のデータレイク上で動作し、Apache Spark™ API と完全な互換性があります。

Databricks では、Delta Lake がデータレイクにもたらす信頼性、性能、ライフサイクル管理を目の当たりにしてきました。私たちのお客様は Delta Lake を活用して、不正形式のデータ取り込みの回避、コンプライアンスに対応するデータ削除、データ取得時のデータ修正など、さまざまな課題を解決しています。

Delta Lake は、高品質データをデータレイクに迅速にもたらし、セキュアでスケーラブルなクラウドサービスで、チームによるデータの利活用を加速させます。

Chapter

01

Delta Lake の基礎：信頼性と性能が重要である理由

01

Delta Lake の基礎：信頼性と性能が重要である理由

データの信頼性に関しては、プログラムの実行速度の性能が最も重要です。Delta Lake が提供する ACID のトランザクション保護により、信頼性と性能を得ることができます。

Delta Lake を使用すると、ストリーミングとバッチ処理を同時に行うことができます。また、CRUD 処理を実行でき、使用する VM が少なくなったため、コストを節約できます。バッチジョブでもストリーミングを活用することで、データエンジニアリングのパイプラインのメンテナンスが容易になります。

Delta Lake は、HDFS とクラウドオブジェクトストレージ上に構築されたデータレイクに信頼性をもたらすストレージレイヤーです。これにより、書き込み間の楽観的同時実行制御と、書き込み中の一貫性のある読み込みのためのスナップショットアイソレーションによって ACID トランザクションが提供されます。また、Delta Lake は、ロールバックやレポートの再現を容易にするためのデータのバージョニング機能を内蔵しています。

この章では、データレイクに共通するいくつかの課題と、それに対処する Delta Lake の機能を紹介します。

データレイクの課題

データレイクは、現代のデータアーキテクチャに共通する要素です。データレイクは、組織が収集・採掘しようとする大量のデータの中心的な取り込みポイントとして機能します。データの範囲を把握する上では良いステップですが、次のような共通の問題に直面します。

1. データレイクへの読み込みと書き込みは信頼性が低い
データエンジニアは、データレイクへの安全でない書き込みの問題にしばしば遭遇します。読み手が書き込み中に常に一貫したデータを見られるようにするために、回避策を構築しなければなりません。

2. データレイクのデータ品質は低い
構造化されていないデータをデータレイクにダンプすることは簡単ですが、これはデータ品質を犠牲にしています。スキーマとデータを検証するメカニズムがなければ、データレイクはデータ品質の低さに悩まされます。その結果、このデータを採掘しようとするアナリティクスプロジェクトも失敗に終わります。

3. データ量が増えるとパフォーマンスが低下
データレイクにダンプされるデータ量が増えると、ファイルやディレクトリの数も増えます。データを処理するビッグデータジョブやクエリエンジンは、メタデータ操作の処理に非常に多くの時間を費やします。この問題は、ストリーミングジョブや多数の同時バッチジョブを処理する場合に顕著になります。

4. データレイク内のレコードを修正、更新、削除することは困難
エンジニアは、パーティションやテーブル全体を読み取り、データを修正して書き戻すために、複雑なパイプラインを構築する必要があります。このようなパイプラインは効率が悪く、メンテナンスも大変です。

このような課題があるため、多くのビッグデータプロジェクトでは、ビジョンの実現に失敗したり、時には完全に失敗したりしています。データの品質を確保しながら、データ実務者が既存のデータレイクを活用できるようにするソリューションが必要です。

Delta Lake の主な機能

Delta Lake は前述の問題に対処し、データレイクの構築方法を簡素化します。Delta Lake は以下のような主要な機能を提供しています。

ACID トランザクション

Delta Lake は複数の書き込み間に ACID トランザクションを提供します。すべての書き込みはトランザクションであり、トランザクションログには書き込みのシリアルオーダーが記録されます。トランザクションログはファイルレベルで書き込みを追跡し、楽観的同時実行性を使用します。これは、同じファイルを変更しようとする複数の書き込みはそれほど頻繁には起こらないので、データレイクには理想的です。競合が発生した場合、Delta Lake は同時修正例外をスローし、ユーザーがそれらを処理してジョブを再試行できるようにします。また、Delta Lake は可能な限り最高レベルの分離（シリアライズ可能な分離）を提供しており、エンジニアはディレクトリやテーブルへの書き込みを継続的に行い、コンシューマーは同じディレクトリやテーブルからの読み込みを継続的に行うことができます。読者は、読み取りを開始した時点で存在していた最新のスナップショットを見ることができます。

スキーマの管理

Delta Lake は、書き込まれる DataFrame のスキーマがテーブルのスキーマと互換性があるかどうかを自動的に検証します。テーブルに存在するが DataFrame にないカラムは null に設定されます。テーブルに存在しない追加のカラムが DataFrame に存在する場合、この操作は例外をスローします。Delta Lake には、明示的に新しいカラムを追加する DDL と、スキーマを自動的に更新する機能があります。

スケーラブルなメタデータ処理

Delta Lake は、テーブルやディレクトリのメタデータ情報をメタストアではなくトランザクションログに格納します。これにより、Delta Lake は大規模なディレクトリ内のファイルを一定時間でリストアップすることができ、データを読み込んでいる間に効率的に処理を行うことができます。

データのバージョニングとタイムトラベル

Delta Lake では、テーブルやディレクトリの以前のスナップショットを読み取ることができます。書き込み中にファイルが変更された場合、Delta Lake は新しいバージョンのファイルを作成し、古いバージョンを保存します。ユーザーがテーブルやディレクトリの古いバージョンを読みたい場合、Apache Spark の 読み込み API にタイムスタンプやバージョン番号を指定すると、Delta Lake はトランザクションログの情報に基づいて、そのタイムスタンプやバージョンの完全なスナップショットを構築します。これにより、ユーザーは実験やレポートを再現したり、必要に応じてテーブルを古いバージョンに戻したりすることができます。

バッチとストリーミングの統合シンク

バッチ書き込みとは別に、Delta Lake は Apache Spark の構造化ストリーミングを利用した効率的なストリーミングシンクとしても利用できます。ACID トランザクションやスケーラブルなメタデータ処理と組み合わせることで、効率的なストリーミングシンクは、複雑なストリーミングやバッチパイプラインを維持することなく、多くのニアリアルタイムアナリティクスのユースケースを可能にします。

レコードの更新と削除

Delta Lake は、マージ、更新、削除のデータ操作言語（DML）コマンドをサポートします。これにより、エンジニアはデータレイク内のレコードのアップサート（upsert）や削除を簡単に行うことができ、変更データの取り込みや GDPR のユースケースを簡素化することができます。Delta Lake はファイルレベルの粒度でデータを追跡・修正するため、パーティションやテーブル全体を読み込んで上書きするよりもはるかに効率的です。

データ期待値（近日更新予定）

Delta Lake は、テーブルやディレクトリに期待されるデータを設定するための新しい API もサポートします。エンジニアはブーリアン条件を指定して、データ期待値を処理するために深刻度を調整することができます。Apache Spark のジョブがテーブルやディレクトリに書き込むと、Delta Lake は自動的にレコードを検証し、違反があった場合には、指定されたシビアリティに基づいてレコードを処理します。

Chapter

02

トランザクションログの仕組み

02

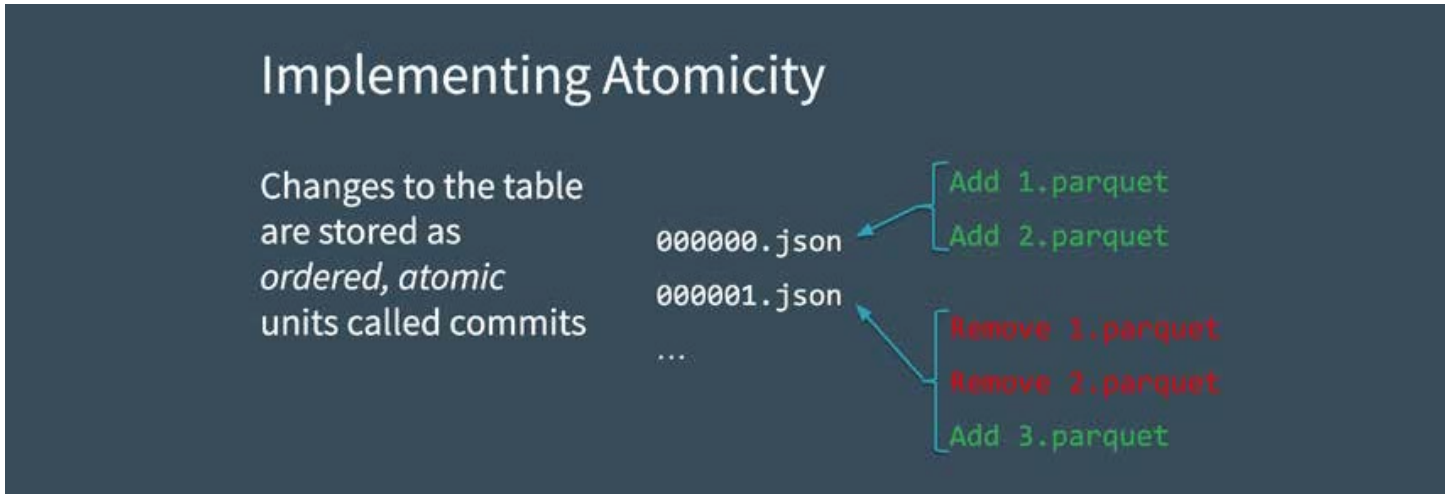
トランザクションログの仕組み

トランザクションログは、ACID トランザクション、スケーラブルなメタデータ処理、時間移動など、最も重要な機能の多くを実行する共通のスレッドであるため、Delta Lake を理解するための鍵となります。Delta Lake のトランザクションログは、Delta Lake テーブルが開始されてから実行されたすべてのトランザクションを順番に記録したものです。

Delta Lake は Apache Spark 上に構築されており、テーブルの複数のユーザー（読み込み、書き込み）が同時に作業できるようになっています。ユーザーに常に正しいデータのビューを表示するために、トランザクションログは、単一のソースとして機能します。これは、ユーザーがテーブルに加えたすべての変更を追跡する中央のリポジトリです。

ユーザーが初めて Delta Lake のテーブルを読み込んだときや、前回読み込んだときから変更されているオープンテーブルに対して新しいクエリを実行したとき、Spark はトランザクションログをチェックしてテーブルに新しいトランザクションがポストされたかどうかを確認します。次に、Spark はエンドユーザーのテーブルを新しい変更で更新します。これにより、ユーザーのテーブルのバージョンが常に最新のクエリの時点でのマスターレコードと同期され、ユーザーがテーブルに対して異なる競合した変更を行うことができなくなります。

この章では、Delta Lake トランザクションログによる、複数の同時読み取り／書き込みの問題に対する優れた解決方法を詳しく説明します。



操作を完全に完了させるための原子性の実装

原子性とは、ACID トランザクションの 4 つのプロパティのうちの 1 つで、データレイク上で実行される操作（INSERT や UPDATE など）が完全に完了するか、または全く完了しないことを保証するものです。このプロパティがなければ、ハードウエアの故障やソフトウエアのバグでデータが部分的にしかテーブルに書き込まれず、データが乱雑になったり、データが破損したりすることはあまりにも簡単です。

トランザクションログは、Delta Lake が原子性の保証を提供するためのメカニズムです。トランザクションログに記録されていなければ、何も起こらなかったことになります。完全かつ完全に実行されたトランザクションのみを記録し、その記録を唯一の真実のソースとして使用することで、Delta Lake のトランザクションログは、ユーザーが自分のデータについて推論し、ペタバイト規模での基本的な信頼性について安心感を持つことを可能にします。

複数の同時読み書きへの対応

しかし、Delta Lake は複数の同時読み書きにどのように対処しているのでしょうか？Delta Lake は Apache Spark を使用しているので、複数のユーザーが一度にテーブルを変更することは不可能ではありません。これらの状況を処理するために、Delta Lake は**楽観的同時実行制御**を採用しています。

楽観的同時実行制御は、異なるユーザーがテーブルに行った変更が互いに競合せずに完了することを前提とした同時実行トランザクションの処理方法です。ペタバイトのデータを扱う場合、ユーザーがデータの異なる部分で作業する可能性が高いため、信じられないほど高速で、競合しないトランザクションを同時に完了させることができます。

もちろん、楽観的同時実行制御を行っていても、ユーザーがデータの同じ部分を同時に変更しようとすることがあります。幸いなことに、Delta Lake にはそのような場合のためのプロトコルがあります。Delta Lake はこのような場合には、相互除外ルールを実装し、楽観的に競合を解決しようとします。

このプロトコルにより、Delta Lake は ACID の分離の原理を実現し、複数の同時書き込みの後のテーブルの状態が、連続的に書き込みが行われた場合と同じであることを保証します。

Delta Lake のテーブル上で行われたすべてのトランザクションが直接ディスクに保存されるため、このプロセスは耐久性という ACID の特性を満たしており、システム障害が発生した場合でも持続します。

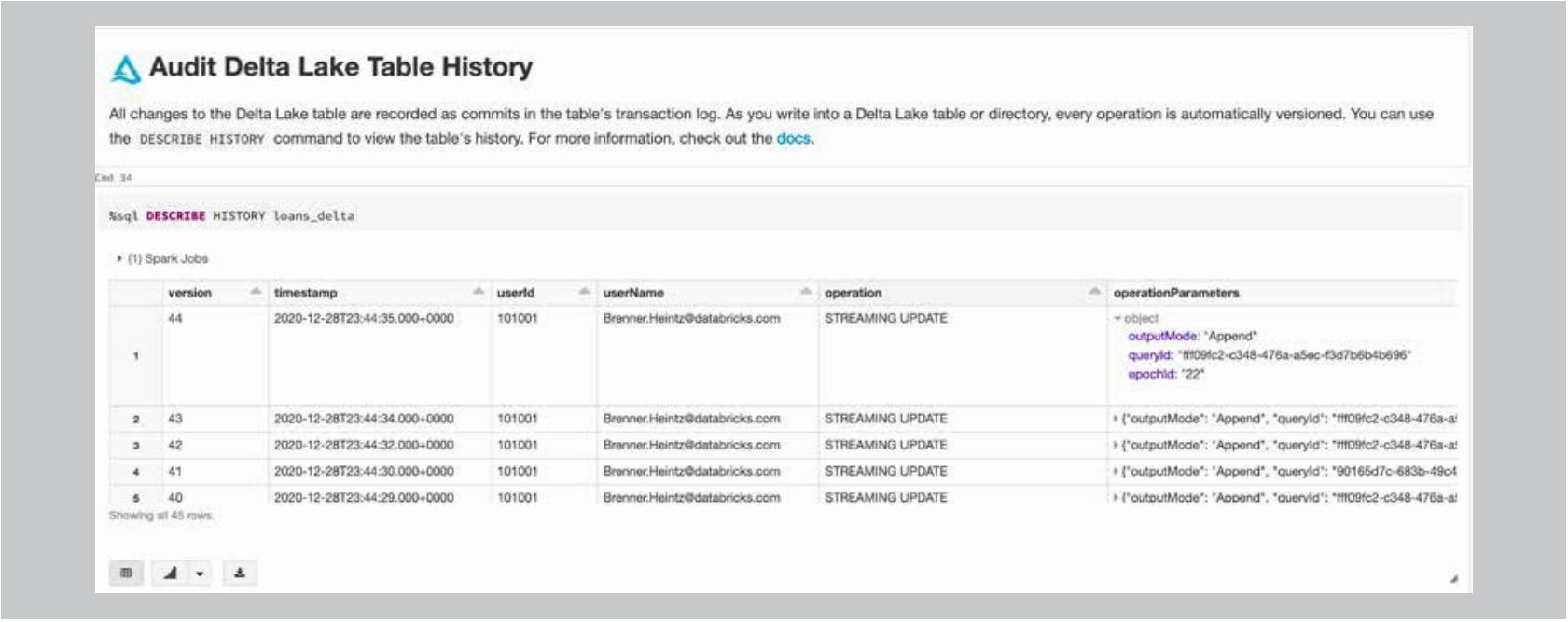
タイムトラベル、データの系統、デバッグ

すべてのテーブルは、Delta Lake のトランザクションログに記録されたすべてのコミットの合計の結果です。トランザクションログには、テーブルの元の状態から現在の状態に至るまでの詳細な手順が記載されています。したがって、オリジナルのテーブルから開始し、その時点以降に行われたコミットのみを処理することで、任意の時点でのテーブルの状態を再現することができます。この強力な機能は、「タイムトラベル」、または「データのバージョン管理」として知られており、様々な状況で救世主となる可能性があります。

詳細については、Introducing Delta Time Travel for Large- Scale Data Lakes と、Getting Data Ready for Data Science With Delta Lake and MLflow を参照してください。

テーブルに加えられたすべての変更の決定的な記録として、Delta Lake のトランザクションログは、ガバナンス、監査、コンプライアンスの目的に有用な検証可能なデータの系譜をユーザーに提供します。また、不注意な変更やパイプライン内のバグの発生源を、それを引き起こした正確なアクションに遡って追跡するために使用することもできます。ユーザーは DESCRIBE HISTORY コマンドを実行して、行われた変更のメタデータを見ることができます。

Delta Lake のトランザクションログについての詳細は、
ブログ記事、技術トークをご覧ください。



Chapter

03

スキーマの適用と展開

03 スキーマの適用と展開

ビジネス上の問題や要件が時間の経過とともに変化すると、データの構造も変化します。Delta Lake では、新しいカラムやオブジェクトの追加が簡単に行えます。ユーザーは簡単なセマンティクスにアクセスして、テーブルのスキーマを制御できます。同時に、ユーザーが誤ってテーブルを使えなくしたり、不要なデータでテーブルを汚したりすることを防ぐために、スキーマの適用の重要性を強調しておくことも重要です。これにより、リッチデータの新しいカラムが属するときに自動的に新しいカラムを追加できます。

スキーマの適用は、テーブルと互換性のない新しいカラムやその他のスキーマの変更を拒否します。このような高い基準を設定して維持することで、アナリストやエンジニアは、データが最高レベルの整合性を持っていることを信頼し、それについて明快に推論することができ、より良いビジネス上の意思決定を行うことができます。

コインの裏返しとして、スキーマの展開は、意図したスキーマの変更が自動的に行われるように簡単にすることで、適用を補完します。結局のところ、カラムを追加することは難しくないはずです。

スキーマの適用はスキーマの展開の陽に対する陰です。これらの機能を一緒に使うと、ノイズを遮断して信号に合わせることがこれまで以上に簡単になります。

テーブルスキーマを理解する

Apache Spark のすべての DataFrame にはスキーマが含まれており、データタイプやカラム、メタデータなどのデータの形状を定義する青写真です。Delta Lake では、テーブルのスキーマは JSON 形式でトランザクションログ内に保存されます。

スキスキーマの適用とは？

スキーマの適用（スキーマ検証）は、テーブルのスキーマに一致しないテーブルへの書き込みを拒否することでデータの品質を保証する Delta Lake のセーフガードです。

忙しいレストランのフロントデスクのマネージャーが予約のみを受け付けているように、テーブルに挿入されたデータの各列が期待される列のリストに載っているかどうか（それぞれが「予約」を持っているかどうか）をチェックし、リストに載っていない列を持つ書き込みを拒否します。

スキーマの適用の機能方法

Delta Lake は書き込み時にスキーマ検証を使用します。つまり、テーブルへのすべての新規書き込みは、書き込み時にターゲットテーブルのスキーマとの互換性をチェックします。スキーマが互換性がない場合、Delta Lake はトランザクションを完全にキャンセルし（データは書き込まれません）、例外を発生させてユーザーに不一致を知らせます。

テーブルへの書き込みが互換性があるかどうかを判断するために、Delta Lake は以下のルールを使用します。書き込まれる DataFrame には次のものを含めることはできません。

ターゲットテーブルのスキーマに存在しない追加のカラム

逆に、入力データにテーブルのすべてのカラムが含まれていなくても問題ありません。それらのカラムには単にヌル値が割り当てられます。

ターゲットテーブルのカラムデータタイプとは異なるカラムデータタイプ

ターゲットテーブルのカラムに StringType データが含まれていても、DataFrame の対応するカラムに IntegerType データが含まれている場合、スキーマ の適用は例外を発生させ、書き込み操作が行われないようにします。

大文字小文字のみで異なるカラム名

つまり、"Foo" と "foo" のようなカラムを同じテーブルで定義することはできません。Spark は大文字と小文字を区別するモードと区別しないモード（デフォルト）がありますが、Delta Lake はスキーマを格納する際に大文字と小文字を区別しません。Parquet は、カラム情報を格納して返すときに大文字と小文字を区別します。潜在的なミスやデータの破損、損失の問題（個人的にDatabricksで経験したことがあります）を避けるために、この制限を追加することにしました。

Delta Lake は自動的に新しいカラムを追加するのではなく、スキーマを強制して書き込みを停止します。不一致の原因となったカラムを特定するために、Spark はスタックトレースで両方のスキーマをプリントアウトして比較します。

スキーマの適用はどのように役立つか？

このような厳しいチェックであるため、スキーマの適用は、生産や消費の準備ができたクリーンで完全に変換されたデータセットのゲートキーパーとして使用するための優れたツールです。一般的には、次のデータを直接フィードするテーブルに適用されます。

- **機械学習アルゴリズム**
- **BI ダッシュボード**
- **データ分析と可視化ツール**
- **高度に構造化され、強力に型付けされた、セマンティックスキーマを必要とするあらゆる生産システム**

この最後のハードルに備えてデータを準備するために、多くのユーザーは、テーブルに構造を段階的に追加するシンプルなマルチホップアーキテクチャを採用しています。詳細については、Delta Lake を使用した機械学習の実運用をご覧ください。

スキーマの展開とは？

スキーマの展開は、時間の経過とともに変化するデータに対応するために、テーブルの現在のスキーマを簡単に変更することができる機能です。一般的には、追加や上書き操作を行う際に使用され、1つ以上の新しいカラムを含むようにスキーマを自動的に適応させます。

スキーマの展開はどのように機能するのか？

前節の例に続いて、開発者はスキーマの不一致により拒否された新しいカラムを簡単に追加することができます。スキーマの展開は、以下の例のように `.write` または `.writeStream` Spark コマンドに `.option('mergeSchema', true')` を追加することで有効になります。

```
# Add the mergeSchema option
loans.write.format(“delta”) \
.option(“mergeSchema”, “true”) \
.mode(“append”) \
.save(DELTALAKE_SILVER_PATH)
```

クエリに `mergeSchema` オプションを含めると、DataFrame には存在するがターゲット テーブルには存在しないカラムは、書き込みトランザクションの一部として自動的にスキーマの最後に追加されます。入れ子になったフィールドを追加することもでき、これらのフィールドもそれぞれの構造体カラムの最後に追加されます。

データエンジニアやサイエンティストはこのオプションを使用して、古いカラムに依存している既存のモデルを壊すことなく、既存の ML 生産テーブルに新しいカラム（新しく追跡されたメトリックや今月の売上高のカラム）を追加することができます。

以下のタイプのスキーマ変更は、テーブルの追加または上書き中にスキーマを展開させることができます。

- 新しいカラムの追加（これは最も一般的なシナリオです）
- NullType からデータ型の変更 → 他のタイプ、または ByteType → ShortType → IntegerType からのアップキャスト

その他の変更点は、スキーマ展開の対象外で、**`.option("overwriteSchema","true")`** を追加してスキーマとデータを上書きする必要があります。これらの変更には次のようなものがあります。

- **カラムのドロップ**
- **既存のカラムのデータ型を変更する C (in place)**
- **カラム名を大文字小文字のみ異なる名前でリネーム（例："Foo" と "foo"）**

最後に、Spark 3.0 のリリースでは、明示的な DDL（ALTER TABLE を使用）が完全にサポートされ、テーブルスキーマに対して以下のアクションを実行できるようになりました。

- カラムの追加
- コラムのコメントを変更する
- トランザクションログの保持期間の設定など、テーブルの動作を定義するテーブルプロパティの設定

スキーマの展開はどのように役立つのか？

スキーマの展開は、テーブルのスキーマを変更する場合にいつでも使用できます（あるはずのないカラムを誤って DataFrame に追加してしまった場合とは異なります）。スキーマを移行する最も簡単な方法は、明示的に宣言しなくても、正しいカラム名とデータ型を自動的に追加することです。

まとめ

スキーマの適用は、テーブルと互換性のない新しいカラムやその他のスキーマの変更を拒否します。このような高い基準を設定して維持することで、アナリストやエンジニアは、データが最高レベルの整合性を持っていることを信頼し、それについて明快に推論することができ、より良いビジネス上の意思決定を行うことができます。

コインの裏返しとして、スキーマの展開は、意図したスキーマの変更を自動的に簡単に行うことができるようにすることで、適用を補完します。結局のところ、カラムを追加するのは難しくないはずです。

スキーマの適用はスキーマの展開の陽に対する陰です。これらの機能を一緒に使うと、ノイズを遮断して信号に合わせることがこれまで以上に簡単になります。

スキーマの適用と展開についての詳細は、ブログ記事、技術トークをご覧ください。

Chapter

04

Delta Lake のデータ操作言語（DML）の内部構造

04

Delta Lake のデータ操作言語（DML）の内部構造

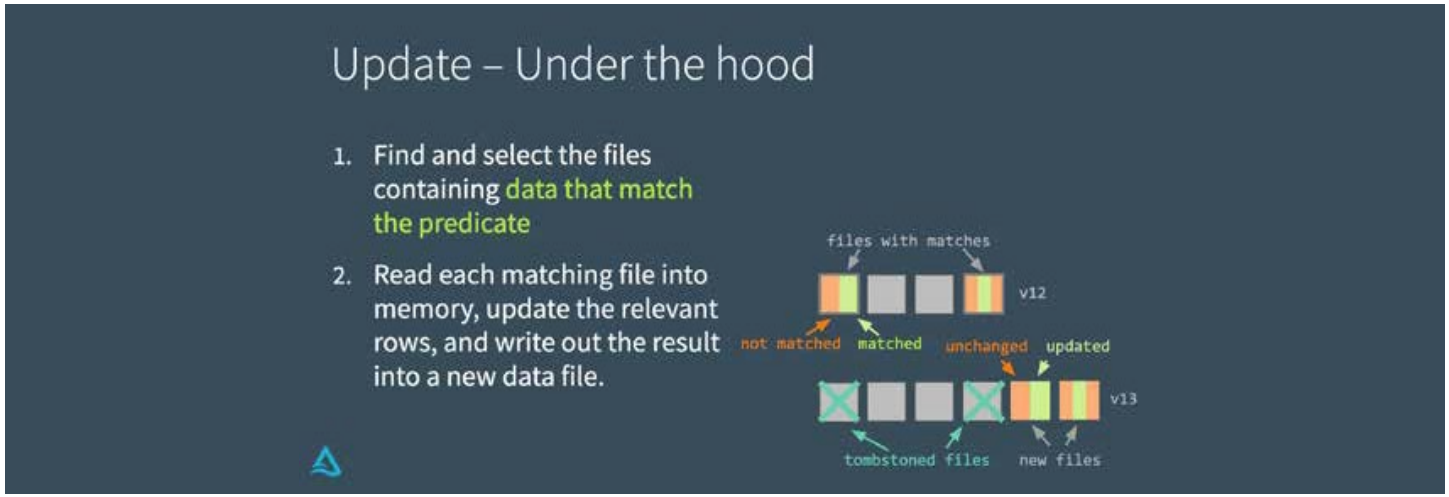
Delta Lake は、UPDATE、DELETE、MERGE などのデータ操作言語（DML）コマンドをサポートしています。これらのコマンドは、変更データの取り込み（CDC）、監査およびガバナンス、GDPR/CCPA ワークフローなどを簡素化します。

この章では、これらのデータ操作言語（DML）コマンドの使い方を説明し、Delta Lake が舞台裏で何をしているのかを説明し、それぞれのパフォーマンスチューニングのヒントを提供します。

Delta Lake データ操作言語（DML）：UPDATE

UPDATE 操作を使用して、述語としても知られるファイル条件に一致する行を選択的に更新することができます。以下のコードは、UPDATE 文の一部として各タイプの述語を使用する方法を示しています。Delta Lake は Python、Scala、SQL の API を提供していますが、この eBook では、SQL コードのみを掲載します。

```
-- Update events
UPDATE events SET eventType='click' WHERE buttonPress = 1
```



UPDATE：内部構造

Delta Lake は、2 つのステップでテーブルの UPDATE を実行します。

1. 述語に一致し、更新が必要なデータを含むファイルを検索して選択します。Delta Lake は、このプロセスを高速化するために、可能な限りデータのスキップを使用します。
2. 各マッチングファイルをメモリに読み込み、関連する行を更新し、結果を新しいデータファイルに書き出します。

Delta Lake が UPDATE を正常に実行すると、トランザクションログにコミットが追加され、今後は古いデータファイルの代わりに新しいデータファイルが使用されることを示します。古いデータファイルは削除されません。代わりに、 "tombstoned"（トゥームストーン）状態になります。表の古いバージョンに適用されたデータファイルとして記録され、現在のバージョンには適用されません。Delta Lake では、これを利用してデータのバージョン管理やタイムトラベルを行うことができます。

UPDATE＋Delta Lake のタイムトラベル = 簡単デバッグ

古いデータファイルを保存しておくことはデバッグに非常に便利です。時間を比較することができます。テーブルを誤って更新してしまい、何が起こったのかを把握したい場合は、テーブルの 2 つのバージョンを簡単に比較して変更箇所を確認できます。

```
SELECT * FROM events VERSION AS OF 11 EXCEPT ALL SELECT
* FROM mytable VERSION AS OF 12
```

UPDATE：パフォーマンスチューニングのヒント

Delta Lake の UPDATE コマンドのパフォーマンスを向上させる主な方法は、検索空間を絞り込むための述語を追加することです。より詳細な検索を行えば行うほど、Delta Lake がスキャンや修正を行う必要のあるファイル数は少なくなります。

Delta Lakeデータ操作言語（DML）：DELETE

DELETE コマンドを使用して、述語（フィルタリング条件）に基づいて行を選択的に削除することができます。

```
DELETE FROM events WHERE date < '2017-01-01'
```

誤った DELETE 操作を元に戻したい場合は、タイムトラベルを使用してテーブルを元に戻すことができます。

DELETE：内部構造

DELETE は、内部構造の UPDATE と同じように動作します。Delta Lake はデータに対して 2 回のスキャンを行います。最初のスキャンでは、述語の条件に一致する行を含むデータ ファイルを識別します。2 回目のスキャンでは、一致するデータファイルをメモリに読み込み、その時点で Delta Lake は問題の行を削除してから、新しくクリーンなデータをディスクに書き出します。

Delta Lake が DELETE 操作を正常に完了した後、古いデータファイルは完全に削除されるわけではありません。古いファイルは、テーブルの以前のバージョンにタイムトラベルするために必要になる可能性があるため、すぐには削除されません。特定の期間よりも古いファイルを削除したい場合は、VACUUM コマンドを使用することができます。

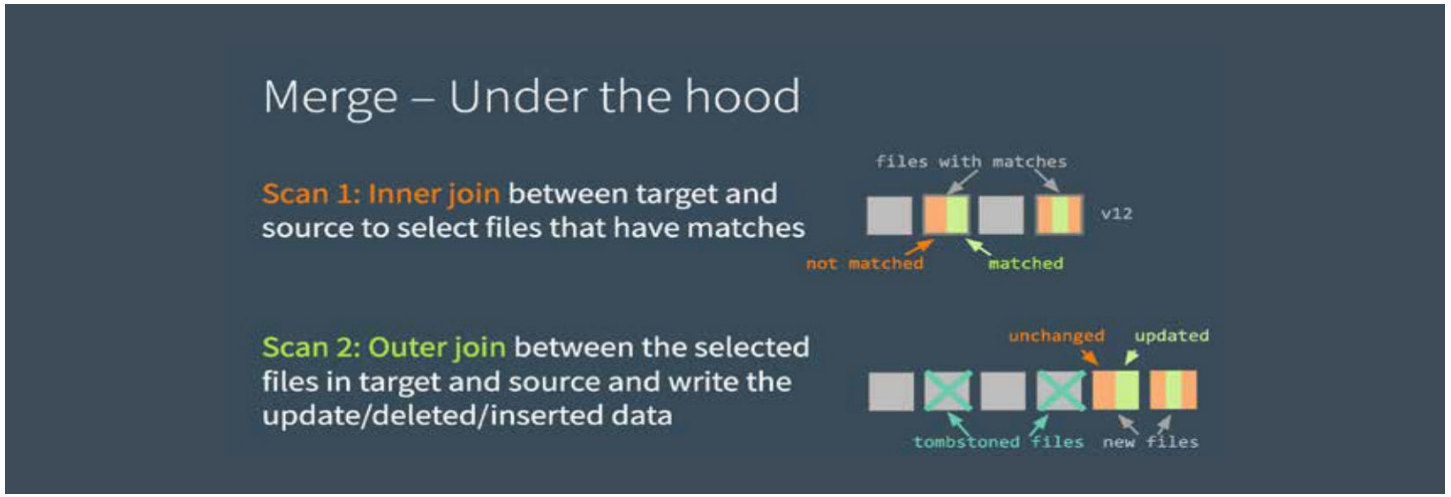
DELETE + VACUUM：古いデータファイルのクリーンアップ

VACUUM コマンドを実行すると、次の条件のデータファイルが永久に削除されます。

- アクティブなテーブルの一部ではなくなり、
- デフォルトでは 7 日間である保持のしきい値よりも古い

Delta Lake は自動的に古いファイルを VACUUM しません。以下に示すように、自分でコマンドを実行する必要があります。デフォルトの 7 日間とは異なる保存期間を指定したい場合は、パラメータとして指定することができます。

```
from delta.tables import * deltaTable.
# vacuum files older than 30 days(720 hours)
deltaTable.vacuum(720)
```



DELETE：パフォーマンスチューニングのヒント

UPDATE コマンドと同様に、Delta Lake での DELETE 操作のパフォーマンスを向上させる主な方法は、より多くの述語を追加して検索空間を絞り込むことです。Databricks 管理バージョンの Delta Lake には、データスキップの改善、ブルームフィルタの使用、Z オーダーの最適化（多次元クラスタリング）など、その他のパフォーマンス向上機能も搭載されています。Databricks による Z オーダーの最適化についての詳細はこちらをご覧ください。

Delta Lake データ操作言語（DML）：MERGE

Delta Lake MERGE コマンドを使用すると、UPDATE と INSERT を組み合わせたアップサートを実行できます。アップサートを理解するには、既存のテーブル（ターゲットテーブル）と、新規レコードと既存レコードの更新が混在するソーステーブルがあると想像してください。

アップサートの仕組み

- ソーステーブルのレコードがターゲットテーブルの既存のレコードと一致すると、Delta Lake はレコードを更新します。
- そのような一致がない場合、Delta Lake は新しいレコードを挿入します。

Delta Lake MERGE コマンドは、Parquet のような他の伝統的なデータフォーマットでは複雑で煩雑になりがちなワークフローを大幅に簡素化します。マージ/アップサートが便利な一般的なシナリオには、変更データの取得、GDPR/CCPA 準拠、セッション化、レコードの重複排除などがあります。

アップサートについては、ブログで詳しく解説しています。

- Databricks Delta を使ったデータレイクへの効率的なアップサート
- Python API を使用した Delta Lake のテーブルでのシンプルで信頼性の高いアップサートと削除
- Delta Lake のマージ操作と運用指標におけるスキーマの展開

MERGE：内部構造

Delta Lake は 2 ステップで MERGE を完成させます。

1. ターゲットテーブルとソーステーブル間で内部結合を実行し、一致するファイルをすべて選択します。
2. ターゲットテーブルとソーステーブルで選択されたファイル間で外部結合を実行し、更新/削除/挿入されたデータを書き出します。

UPDATE や DELETE と異なる主な点は、Delta Lake は結合を使用して MERGE を完了させている点です。この事実により、パフォーマンスを向上させようとする際に、いくつかのユニークな戦略を利用することができます。

MERGE：パフォーマンスチューニングのヒント

MERGE コマンドのパフォーマンスを向上させるには、マージを構成する 2 つの結合のうち、どちらの結合が速度を制限しているかを判断する必要があります。

内側の結合がボトルネックになっている場合（つまり、Delta Lake が書き換える必要のあるファイルを見つけるのに時間がかかりすぎる場合）、以下の戦略を試してみてください。

述語を追加して検索スペースを絞り込みます。

- シャッフルパーティションを調整します。
- ブロードキャスト参加のしきい値を調整します。
- "Delta Lake" では書き換えのためにファイル全体をコピーしなければならないので、表の小さなファイルが多い場合はコンパクトにしますが、大きすぎるファイルにはコンパクトにしないようにします。

Databricks の管理する Delta Lakeでは、Z オーダーの最適化を使用して更新の局所性を利用します。

一方、外部結合がボトルネックになっている場合（つまり、実際のファイル自体の書き換えに時間がかかりすぎる）、以下の戦略を試してみてください。

- シャッフルパーティションを調整します。
- 書き込み前に自動リパーティショニングを有効にしてファイルを削減（Databricks Delta Lake の Optimized Writes 使用）。

- ブロードキャストのしきい値を調整します。完全な外側結合を行っている場合、Spark はブロードキャスト結合を行うことができませんが、右外側結合を行っている場合、Spark はブロードキャスト結合を行うことができ、必要に応じてブロードキャストのしきい値を調整することができます。

- ソーステーブル／DataFrame をキャッシュする。
- ソーステーブルをキャッシュすることで 2 回目のスキャンを高速化することができますが、ターゲットテーブルをキャッシュしないように注意してください。

Delta Lake は、UPDATE、DELETE、MERGE INTO などのデータ操作言語（DML）コマンドをサポートしており、多くの一般的なビッグデータ操作のワークフローを大幅に簡素化します。この章では、Delta Lake でのこれらのコマンドの使用方法を説明し、それぞれのコマンドがどのように動作するかについての情報を共有し、パフォーマンスチューニングのヒントを提供しました。

コードスニペットを含む、データ操作言語（DML）の内部の詳細は、ブログ記事をご覧ください。

Chatper

05

データスキップと Z オーダーによる迅速なペタバイト規模のデータ処理

05

データスキップと Z オーダーによる迅速なペタバイト規模のデータ処理

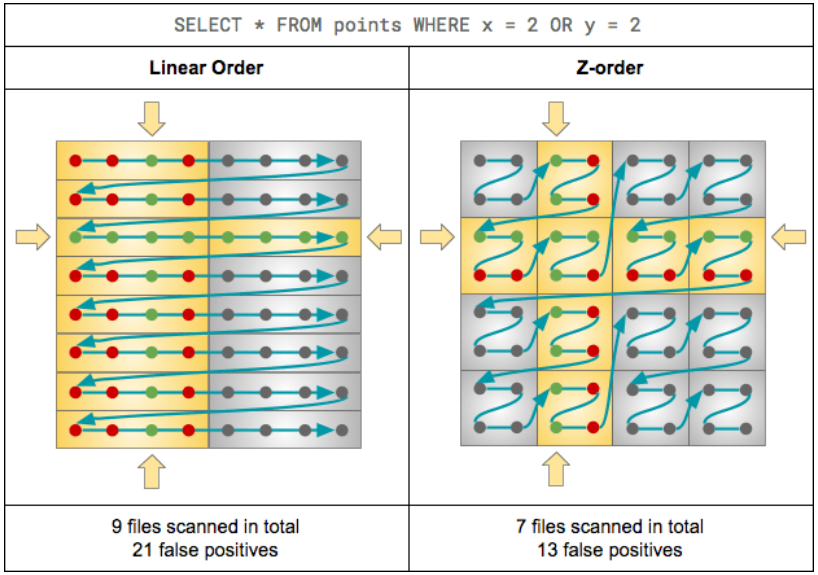
Delta Lake は、数秒でペタバイトのデータをふるいにかけることができます。この速度は、(1) データスキップ、(2) Z オーダーの 2 つの機能によるものです。

これらの機能を組み合わせることで、Databricks ランタイムは、スキャンする必要のあるデータ量を大幅に削減し、大規模な Delta テーブルに対する選択的なクエリに答えることができます。

Delta Lake のビルトインデータスキップ機能と Z オーダークラスタリング機能を使用してクエリに関係のないファイルをスキップすることで、大規模なクラウドデータレイクを数秒でクエリ実行できます。例えば、実際のサイバーセキュリティ分析のユースケースでは、504 TB のデータセットの 93.2% のレコードが典型的なクエリでスキップされ、クエリ時間が最大で 2 桁短縮されました。言い換えれば、Delta Lake はクエリを 100 倍も高速化することができます。

データスキップと Z オーダーの動作

Apple の Dominique Brezinski 氏と Databricks の Michael Armbrust 氏が、サイバーセキュリティの監視と脅威対応において、データエンジニアリングとデータサイエンスのための統合ソリューションとして Delta Lake をどのように使用するかを実演しました。基調講演「Threat Detection and Response at Scale」をご覧ください。



データスキップと Z オーダークラスタリングの使用

データスキップと Z オーダーは、膨大なデータセットに対して、針の穴に糸を通すようなクエリのパフォーマンスを向上させるために使用されます。データスキップは Delta Lake の自動機能で、SQL クエリやデータセット操作に "column op literal, where" という形式のフィルタが含まれている場合に自動的に実行されます。

- **column** は、トップレベルであれネストされたものであれ、Delta Lake テーブルの属性であり、データタイプは文字列 / 数値 / 日付 / タイムスタンプです。
- **op** は、バイナリ比較演算子、**StartWith / LIKE pattern%** または **IN <list_of_values>** です。
- リテラルは、カラムと同じデータ型の明示的な値のリストです。

AND / OR / NOT は、「リテラルオペ列」述語と同様にサポートされます。

上記の条件が満たされたときにデータスキップが開始されたとしても、必ずしも効果があるとは限りません。しかし、頻繁にフィルタリングするカラムがいくつかあって、それを高速にしたい場合は、以下のコマンドを実行することで、スキップの効果に関してデータレイアウトを明示的に最適化することができます。

```
OPTIMIZE <table> [WHERE <partition_filter>]
ZORDER BY (<column> [, …])
```

詳細を探る

パーティションのプルーニングとは別に、データウェアハウスの世界で使われているが、Spark には現在欠けているもう一つの一般的なテクニックは、小さなマテリアライズドアグリゲートに基づいた I/O プルーニングです。要するに、ある粒度での最小値や最大値など、I/O の粒度と相関のある単純な統計情報を記録しておくというものです。そして、不必要な I/O を避けるために、クエリ計画時にこれらの統計情報を活用したいと考えています。

これこそが、Delta Lake のデータスキップ機能の正体です。Delta Lake テーブルに新しいデータが挿入されると、サポートされているタイプのすべてのカラム（ネストされたものを含む）について、ファイルレベルの min/max 統計が収集されます。そして、そのテーブルに対してルックアップクエリが行われると、Delta Lake はまずこれらの統計情報を参照して、どのファイルを安全にスキップできるかを判断します。

サイバーセキュリティ分析への適用方法など、データスキップと Z オーダーの詳細は、ブログ記事をご覧ください。

次のステップ

この eBook では、Delta Lake とその機能が性能を向上させる仕組みについて解説しました。
このシリーズの他の eBook では、Delta Lake のリソースを詳しくご紹介しています。

この eBook の後続シリーズ

- Delta Lake シリーズ：機能
- Delta Lake シリーズ：レイクハウス
- Delta Lake シリーズ：ストリーミング
- Delta Lake シリーズ：顧客ユースケース

Delta Lake をさらに詳しく

- Databricks の Web サイト
- Databricks の無料トライアル
- Web セミナー：Delta Lake オープンソースの信頼性