



Analytics Accelerator

リリース 1.3

EnterpriseDB Corporation

2025 年 10 月 20 日

目次

第 1 章	Introduction to EDB Postgres® Analytics Accelerator	1
1.1	分析アクセラレーターのナビゲート	1
1.2	概念的な基礎	1
1.3	EDB コア分析テクノロジー	2
1.4	Use Anywhere マニュアル/リファレンス	2
1.5	PGD での使用マニュアル/リファレンス	2
1.6	ハイブリッドマネージャーでの使用マニュアル/リファレンス	2
1.7	How-Tos : ランブックに合わせた	3
1.8	どこから始めるべきか	3
1.8.1	レベル 100	3
1.8.2	レベル 200	4
1.8.3	レベル 300	4
第 2 章	Getting Started	7
2.1	Analytics Generic Concepts	7
2.1.1	アーキテクチャ	7
2.1.2	主要テクノロジー	8
2.1.3	コアコンセプト	9
2.1.4	追加の概念	11
2.1.5	次に探索する	12
2.2	Analytics Accelerator Concepts	12
2.2.1	Postgres での分析に対する EDB のビジョン	12
2.2.2	主要な EDB ソリューションとテクノロジー	12
2.2.3	アナリティクスと AI / ML ワークロード	14
2.2.4	EDB が製品全体にアナリティクスを実装する方法	14
2.2.5	次に探索する	15
2.3	Analytics Terminology	15
2.3.1	Apache Iceberg	15
2.3.2	デルタレイク	15
2.3.3	データレイクハウス	16
2.3.4	EDB Postgres Lakehouse クラスター	16
2.3.5	EDB Postgres Distributed PGD	16
2.3.6	階層化テーブル	16
2.3.7	PGAA アナリティクスアクセラレーター	17
2.3.8	PGFS Postgres ファイルシステム	17
2.3.9	AutoPartition	17
2.3.10	BDR 分析テーブル	17
2.3.11	氷山カタログ	18
2.3.12	Lakekeeper	18
2.3.13	オープンテーブルフォーマット	18

2.3.14	ベクトル化されたクエリエンジン	18
2.3.15	データ階層化	19
2.3.16	ストレージとコンピューティングの分離	19
2.3.17	次のステップ	19
2.4	Analytics Accelerator Architecture	19
2.4.1	アーキテクチャの概要	19
2.4.2	コアコンポーネント	20
2.4.3	アーキテクチャレイヤー	21
2.4.4	データフローパターン	21
2.4.5	パフォーマンスアーキテクチャ	22
2.4.6	高可用性と復元力	23
2.4.7	セキュリティアーキテクチャ	23
2.4.8	展開パターン	24
2.5	Analytics Accelerator Quick Start	25
2.5.1	はじめに	25
2.5.2	はじめに	25
2.5.3	Lakehouse ノードの動作	28
2.5.4	ベンチマークデータセットを検査する	29
2.5.5	ベンチマークデータセットを照会する	29
2.6	Reference - EDB Postgres® AI Lakehouse	30
2.6.1	Benchmarking datasets	30
2.6.2	Queries	31
2.6.3	Delta Lake Table tools	31
2.6.4	DirectScan and fallback modes	32
2.6.5	User management	33
2.6.6	Loading data	33
2.6.7	PGAA functions reference	34
2.6.8	PGAA functions reference	35
2.6.9	Supported AWS instances	36
第 3 章	Use Anywhere	37
3.1	EDB Postgres Lakehouse Architecture	37
3.1.1	EDB Postgres Lakehouse とは	37
3.1.2	レイクハウスアーキテクチャから恩恵を受ける人	37
3.1.3	Lakehouse アーキテクチャを展開する方法	38
3.1.4	アーキテクチャコンポーネント	39
3.1.5	パフォーマンス特性	40
3.1.6	統合パターン	41
3.1.7	運用上の考慮事項	42
3.2	Apache Iceberg Integration With Analytics Accelerator	42
3.2.1	Apache Iceberg とは	43
3.2.2	Iceberg の統合を使用する人	43
3.2.3	アナリティクスアクセラレーターで Iceberg を使用する方法	43
3.2.4	Iceberg アーキテクチャを理解する	44
3.2.5	スキーマ進化機能	45
3.2.6	カタログ統合パターン	46
3.2.7	パフォーマンス最適化戦略	47

3.2.8	一般的な統合パターン	47
3.2.9	運用上の考慮事項	48
3.3	Delta Lake Integration with Analytics Accelerator	49
3.3.1	レイクハウスのコンテキストでのデルタ湖の理解	49
3.3.2	アーキテクチャの概要	49
3.3.3	デルタレイクテーブルフォーマット	49
3.3.4	統合機能	50
3.3.5	実際のアプリケーション	50
3.3.6	パフォーマンスの最適化	51
3.3.7	移行の考慮事項	52
3.3.8	エコシステムの統合	52
第 4 章	Use With PGD	53
4.1	Tiered Tables With Analytics Accelerator	53
4.1.1	階層テーブルとは	53
4.1.2	階層テーブルの恩恵を受ける人	53
4.1.3	階層化テーブルの仕組み	53
4.1.4	アーキテクチャの概要	56
4.1.5	パフォーマンス特性	57
4.1.6	一般的な実装パターン	57
4.1.7	運用上の考慮事項	58
第 5 章	Learning	61
5.1	Analytics Accelerator learning resources	61
5.1.1	ドキュメントカテゴリ	61
5.1.2	次のステップ	62
5.2	Analytics Accelerator Learning Paths	62
5.2.1	これらのラーニングパスの使用方法	62
5.2.2	ラーニングパスレベル	63
5.2.3	EDB トレーニングと認定資格	64
5.2.4	追加のガイダンス	64
5.2.5	次のステップ	64
5.3	Analytics Accelerator 101 Foundational concepts	65
5.3.1	学ぶこと	65
5.3.2	始める前に	65
5.3.3	推奨される学習パス	65
5.3.4	次のステップ	65
5.4	Analytics Accelerator 201 Practical application and core solutions	65
5.4.1	学ぶこと	66
5.4.2	始める前に	66
5.4.3	推奨される学習パス	66
5.4.4	次のステップ	66
5.5	Analytics Accelerator 301 Advanced techniques and optimization	66
5.5.1	学ぶこと	67
5.5.2	始める前に	67
5.5.3	推奨される学習パス	67
5.5.4	次のステップ	67

5.6	Analytics Accelerator for your role a persona-based guide	68
5.6.1	自分のロールをすばやく見つける	68
5.6.2	データベース管理者 DBA	68
5.6.3	DevOps エンジニア / サイト信頼性エンジニア SRE	69
5.6.4	データサイエンティスト / データアナリスト	70
5.6.5	アプリケーション開発者	70
第 6 章	How To Hybrid Manager	73
6.1	Create A Lakehouse Cluster	73
6.1.1	Lakehouse クラスターを作成する理由	73
6.1.2	ゴール	73
6.1.3	前提条件	73
6.1.4	ステップ	74
6.1.5	次にできること	75
6.1.6	関連トピック	75
第 7 章	How To Playbook	77
7.1	How To Get Setup for Lakehouse analytics	77
7.2	Lakehouse アナリティクスの設定	77
7.2.1	ゴール	77
7.2.2	要件	77
7.2.3	Lakekeeper カタログを作成する	79
7.2.4	バケット URL を特定します	79
7.2.5	サニティチェック	80
7.2.6	次のステップ	80
7.2.7	テイクアウト	80
7.3	How-To Read/Write Without Catalog	81
7.3.1	前提条件	81
7.3.2	セットアップ: 分析インフラストラクチャの構成	81
7.3.3	読み取り専用分析: 既存のデータへのアクセス	83
7.3.4	トランザクションテーブル分析オフロード	84
7.3.5	高度なテーブル管理操作	86
7.3.6	分析オフロードを使用した AutoPartition	87
7.3.7	トラブルシューティングと検証	92
7.4	How-To Integrate with Third-Party Iceberg Catalogs	93
7.4.1	概要	93
7.4.2	コンテキストと目標	93
7.4.3	前提条件	93
7.4.4	実際のアプリケーション	94
7.4.5	構成手順	94
7.4.6	検証とテスト	96
7.4.7	既知の制限	97
7.4.8	障害対応	97
7.4.9	次のステップ	97
7.5	How-To Lakehouse Read With/Without A Catalog	98
7.5.1	概要	98
7.5.2	前提条件	98

7.5.3	構成アプローチ	98
7.5.4	AWS S3 テーブルの統合	98
7.5.5	デルタレイクへの直接アクセス	99
7.5.6	氷山への直接アクセス	101
7.5.7	カタログ添付ファイル	101
7.5.8	クエリーの検証	102
7.5.9	ベストプラクティス	103
7.5.10	障害対応	104
7.5.11	次のステップ	104
7.6	Configure Analytics Storage and Data Tiering with PGAA and PGD	104
7.6.1	目次	104
7.6.2	パート7 外部ツールの統合	105
7.6.3	ステップの概要	106
7.6.4	前提条件	106
7.6.5	セットアップ: 分析インフラストラクチャの構成	107
7.6.6	読み取り専用分析: 既存のデータへのアクセス	109
7.6.7	トランザクションテーブル分析オフロード	111
7.6.8	高度なテーブル管理操作	113
7.6.9	分析オフロードを使用した AutoPartition	114
7.6.10	カタログベースの操作	119
7.6.11	トラブルシューティングと検証	125

第1章 Introduction to EDB Postgres[®] Analytics Accelerator

Analytics Accelerator PGAA を使用して、EDB Postgres に構築された分析機能を探索します。このアクセラレータは、コア概念を理解し、EDB Postgres[®] Lakehouse などの主要テクノロジーを探索し、EDB Hybrid ManagerHM を使用して分析を実装する方法を学習するのに役立ちます。

私たちは、最新のデータアーキテクチャとオープンスタンダードを Postgres の信頼性と柔軟性と統合して、貴重な洞察を導き出すのを支援します。

1.1 分析アクセラレーターのナビゲート

アクセラレータは、コンテンツを4つのエリアに編成します。

- **概念的な基礎** 分析原則と EDB のアプローチについての理解を深めます。
- **EDB コア分析テクノロジー** 分析サービスを強化する EDB ソリューションとテクノロジーについて学びます。
- **実践的なガイダンスとソリューション** ユースケース、ペルソナベースのガイド、ハウツー記事、チュートリアルを見つけます。
- **製品固有の実装** これらの分析機能がどのように表面化し、EDB Hybrid Manager などの EDB 製品で管理される方法に関するドキュメントにアクセスします。

1.2 概念的な基礎

最新のデータ分析と EDB のアプローチの背後にある原則と戦略を理解します。

- *Analytics Generic Concepts*

データアーキテクチャデータウェアハウス、データレイク、レイクハウスと基盤テクノロジーカラムナストレージ、ベクトル化エンジンなどについて学びます。

- *Analytics Accelerator Concepts*

Postgres[®] 分析に対する EDB のビジョンと、EDB がコアテクノロジーを活用する方法を探ります。

EDB 分析機能、デザインの選択、および以下のセクションにわたる高度なトピックの詳細な説明を確認します。

1.3 EDB コア分析テクノロジー

EDB の分析テクノロジーと、それらが Postgres[®] を拡張する方法について学びます。

- [EDB Postgres Lakehouse Architecture](#)

オブジェクトストレージの分析を有効にするための EDB Postgres[®] Lakehouse ソリューションとそのコンポーネントを確認します。

- [Apache Iceberg Integration With Analytics Accelerator](#)

EDB ソリューションが Apache Iceberg を使用して大規模な分析データセットを管理する方法を理解します。

- [Delta Lake Integration with Analytics Accelerator](#)

EDB Postgres[®] がデルタテーブルと相互作用して、信頼性の高いデータレイクを有効にする方法を学びます。

- [Tiered Tables With Analytics Accelerator](#)

EDB Postgres DistributedPGD および Lakehouse 機能を使用してストレージ階層全体でデータを管理し、コストとパフォーマンスを最適化します。

1.4 Use Anywhere マニュアル/リファレンス

- レイクハウスの概要 [EDB Postgres Lakehouse Architecture](#)
- オープンフォーマット [Apache Iceberg Integration With Analytics Accelerator](#) 、 [Delta Lake Integration with Analytics Accelerator](#)
- 保存場所 機能と構成については、 [PGAA functions reference](#) を参照してください
- 参照 [PGAA functions reference](#) 、 [PGAA functions reference](#) 、 [Direct scan](#) 、 [Inspect the benchmark datasets](#)

1.5 PGD での使用マニュアル/リファレンス

- 概念 [Tiered Tables With Analytics Accelerator](#)

1.6 ハイブリッドマネージャーでの使用マニュアル/リファレンス

- 準備中 [Getting setup for Lakehouse analytics](#)
- プロビジョニング [Create A Lakehouse Cluster](#)
- カタログ [Configure Analytics Storage and Data Tiering with PGAA and PGD](#)
- 相互運用性 [How-To Lakehouse Read With/Without A Catalog](#)

1.7 How-Tos : ランブックに合わせた

これらのガイドは、ランブックフローとコード例を反映しています。

— コアハウツー

- ハウツー *How-To Lakehouse Read With/Without A Catalog*
- ハウツー *How-To Read/Write Without Catalog*
- ハウツー *Configure Analytics Storage and Data Tiering with PGAA and PGD*
- ハウツー *How-To Integrate with Third-Party Iceberg Catalogs*

1.8 どこから始めるべきか

- *Analytics Generic Concepts* および *EDB Postgres Lakehouse Architecture* から始めて、コアアイデアを理解します。
- 外部データを実験する場合は、No Catalog のハウツーを使用してください。
- PGD /階層テーブルまたはカタログと統合する場合、PGD およびカタログのハウツーに従ってください。

Postgres Lakehouse は、多くのテクノロジーを使用して構築されています。

- PostgreSQL
- *Seafowl*、分析データベース
- *Apache DataFusion*、*Seafowl* が使用するクエリエンジン
- *Delta Lake* および具体的には *delta-rs* デルタテーブルのストレージおよび取得レイヤーを実装するため

1.8.1 レベル 100

Postgres Lakehouse について理解する最も重要な点は、ストレージをコンピューティングから分離することです。このデザインでは、それらを独立してスケーリングできます。これは、クエリが予測不能でスパイクになる可能性がある分析ワークロードに最適です。接続されているハードドライブにデータを保持するためだけに、マシンをほとんどアイドル状態に保つことはしたくないでしょう。代わりに、データをオブジェクトストレージおよび高圧縮可能な形式で保持し、必要に応じて照会に必要なコンピューティングのみをプロビジョニングできます。

コンピューティング側では、ベクトル化されたクエリエンジンは Lakehouse テーブルを照会するように最適化されていますが、完全な互換性のために Postgres にフォールバックします。

ストレージ側では、Lakehouse テーブルは、分析用に最適化された圧縮性の高い列型ストレージ形式を使用して保存されます。

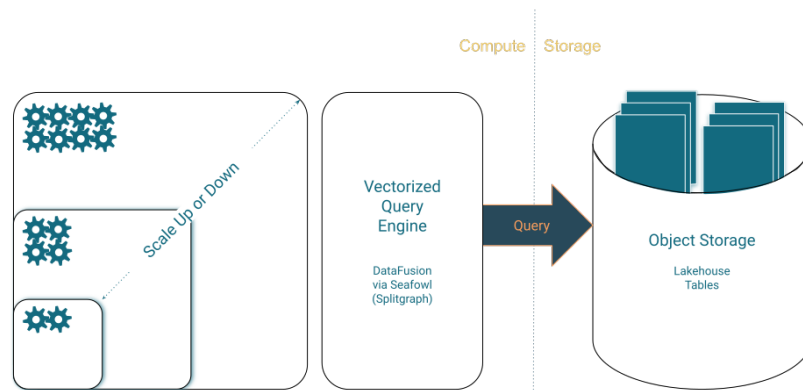


図 1: Level 100 Architecture

1.8.2 レベル 200

次に、これらのサービスがどのように連携するかを示すもう少し包括的な図を示します。

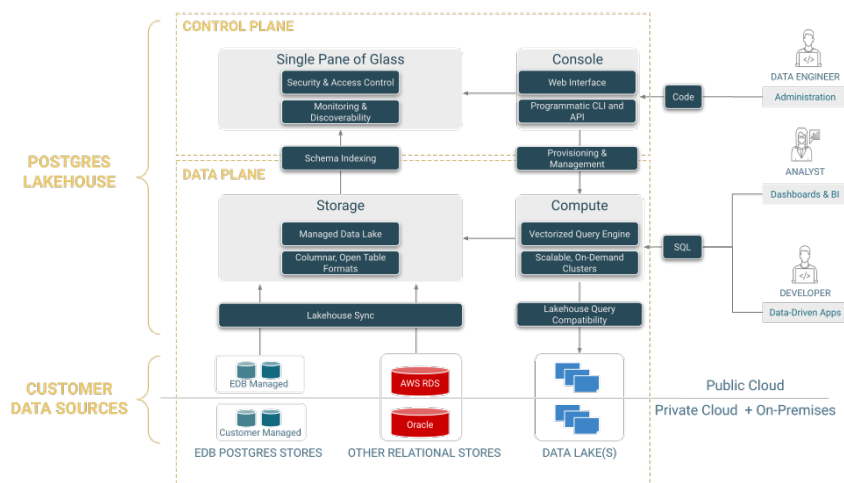


図 2: Level 200 Architecture

1.8.3 レベル 300

これは、「ボックスの内容」のより詳細な拡大ビューです。

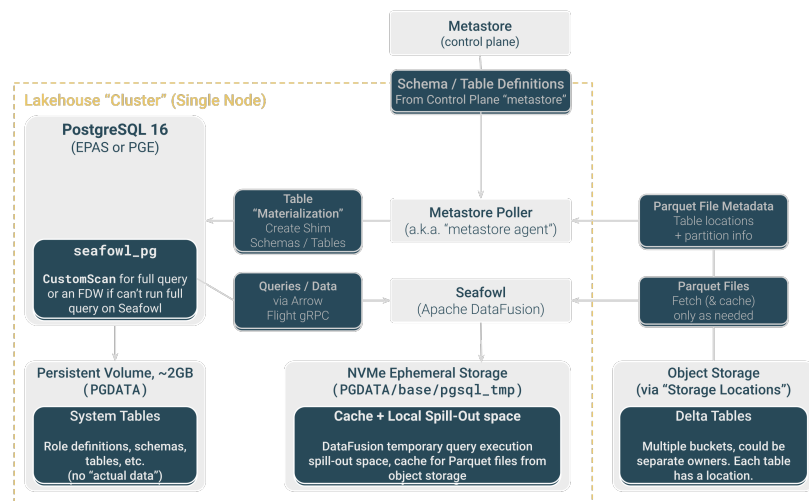


図 3: Level 300 Architecture

第2章 Getting Started

2.1 Analytics Generic Concepts

このハブでは、最新のデータ分析プラットフォームの基礎を形成する主要な業界概念、アーキテクチャ、およびテクノロジーについて説明します。

このページは、Analytics Hub の一部です。完全なナビゲーションについては、次をご覧ください [Analytics Hub — Analytics Accelerator Concepts — How-Tos \(Runbook-Aligned\)](#)

これらの概念を理解することは、EDB のアナリティクスアクセラレーターが実証済みのパターンに基づいてどのように構築されているかを理解するのに役立ちます。EDB がこれらのアイデアを実装する方法については、[Analytics Accelerator Concepts](#) にアクセスしてください。

2.1.1 アーキテクチャ

データウェアハウス

内容 構造化された SQL ベースの分析用に最適化された一元化リポジトリ。これらはスキーマオンライトを使用し、高速な分析クエリー用に設計されています。

理由 ビジネスインテリジェンスチームは、レポートと履歴分析のために信頼できる一貫したデータを必要としています。

方法 データは、ETL プロセスを介して、強い型付けと制約を使用してリレーショナルテーブルにロードされます。

使用する理由 既知の一貫したレポートニーズとデータ品質に対する強力なガバナンスに対応する高いパフォーマンスが必要な場合。

これが [Analytics Accelerator Concepts](#) に与える影響を学びます。

データレイク

内容 生データ構造化、半構造化、非構造化のための柔軟なリポジトリ。スキーマオンリード。

理由 データレイクを使用すると、すべてのデータを低コストでネイティブ形式で保存できます。

方法 データはファイルとしてオブジェクトストレージ S3、GCS などにロードされます。多くの場合 Parquet または ORC 形式で。

使用する理由 使用可能なすべてのデータを完全なユースケースを知る前にキャプチャし、後で柔軟な処理を実行する必要がある場合。

[More on data lakes](#)

これが [Analytics Accelerator Concepts](#) に与える影響を学びます。

データレイクハウス

内容 データレイクの柔軟性とデータウェアハウスのパフォーマンスと信頼性を組み合わせた最新のアーキテクチャ。

理由 組織は、大量のデータを移動せずに、データサイエンスと BI の両方に対応する 1 つのシステムを望んでいます。

方法 データは、構造、トランザクション、メタデータを追加するオープンテーブル形式を使用してレイクオブジェクトストレージに保存されます。

使用する理由 スケーラブルで低コストのストレージが必要だが、ACID 保証と高パフォーマンスのクエリーも必要な場合。

Lakehouse architecture paper (Armbrust et al.)

Apache Iceberg

Delta Lake

Apache Hudi

これが *Analytics Accelerator Concepts* にどのような影響を与えるかを学びます。

2.1.2 主要テクノロジー

カラムナストレージ形式

内容 行ではなく列をまとめて保存するデータ形式。

理由 ほとんどの分析クエリは、列のサブセットにのみアクセスします。列を直接読み取ると、パフォーマンスが向上し、I/O が削減されます。

方法 一般的な形式は [Apache Parquet](#) および [Apache ORC](#) で、圧縮と列ブルーニングをサポートしています。

使用する理由 大規模で読み取り負荷の高い分析クエリを実行し、最大のスキャン速度が必要な場合。

これが *Analytics Accelerator Concepts* でどのように使用されるかを確認してください。

ベクトル化されたクエリエンジン

内容 行ごとではなく列方向のバッチベクトルでデータを処理するクエリエンジン。

理由 カラムナバッチは CPU キャッシュ動作と SIMD 命令と一致し、分析を大幅に高速化します。

方法 [Apache DataFusion](#) や [Velox](#) のようなエンジンは、ベクトル化された実行を実装します。

使用する理由 分析ワークロードが CPU バウンドであり、パフォーマンスが重要な場合。

これが *Analytics Accelerator Concepts* でどのように使用されるかを確認してください。

ストレージとコンピューティングの分離

内容 コンピューティングクエリエンジンとストレージオブジェクトストレージが分離されたアーキテクチャ。

理由 これにより、それらを個別にスケーリングでき、コストとパフォーマンスを最適化できます。

方法 アナリティクスアクセラレーターのようなシステムは、データにオブジェクトストレージ S3 などを使用し、クエリにステートレスコンピューティングノードを使用します。

使用する理由 コンピューティングニーズがバーストまたは予測不能で、オーバープロビジョニングを回避したい場合。

これが *Analytics Accelerator Concepts* でどのように使用されるかを確認してください。

オープンテーブルフォーマット

内容 データベース機能 ACID トランザクション、スキーマの進化をデータレイクに提供するフォーマット。

理由 オープンテーブル形式がないと、レイクデータはルーズファイルのセットです。構造とガバナンスが必要です。

方法

- **Apache Iceberg** はメタデータレイヤーとマニフェストファイルを使用します
- **Delta Lake** はトランザクションログを使用します **delta-rs**
- **Apache Hudi** はストリームファースト更新に焦点を当てています

使用する理由 レイクベースのデータの強力なデータ管理が必要な場合。

これが *Analytics Accelerator Concepts* でどのように使用されるかを確認してください。

2.1.3 コアコンセプト

OLAP / OLTP

内容

- OLAP 大規模なデータセットの分析クエリ集計、トレンド
- OLTP 大容量、小規模トランザクション更新、挿入

理由 ワークロードが異なると、パフォーマンスプロファイルも異なります。

方法 OLAP は、カラム型ストレージとパラレル実行で最適化されます。OLTP は、行ベースのストレージとトランザクション保証を使用します。

使用する理由 ワークロードの種類ごとに適切なアーキテクチャを選択するため。

これが *Analytics Accelerator Concepts* でどのように使用されるかを確認してください。

ETL / ELT

内容

- ETL 抽出、変換、ロード
- ELT 抽出、ロード、変換レイクまたはウェアハウスで

理由 ELT は、最新のレイクハウスの大規模な生データを処理する機能を利用します。

方法 データはオブジェクトストレージにステージングされ、クエリエンジンまたはパイプラインを使用して変換されます。

使用する理由 データの取り込みを簡素化し、パイプラインをより柔軟にするため。

これが *Analytics Accelerator Concepts* でどのように使用されるかを確認してください。

データ階層化

内容 アクセスパターンに基づいて、データをホット、ウォーム、コールドに分類します。

理由 すべてのデータが高速ストレージを必要とするわけではありません。コールドデータをより安価なストレージに移動すると、コストを節約できます。

方法 データはストレージタイプ高速 SSD、オブジェクトストレージを超えて物理的に分離されます。

使用する理由 ストレージコストとパフォーマンスを最適化するため。

これが *Analytics Accelerator Concepts* でどのように使用されるかを見てください。

セマンティック検索

内容 キーワードだけでなく、意味を理解する検索。

理由 ユーザーは、よりスマートな検索エクスペリエンスを期待しています。

方法 データはベクトル埋め込みとして表され、類似性検索を使用して比較されます。

使用する理由 AI と最新のアプリの検索アプリケーションをサポートするため。

これが *Analytics Accelerator Concepts* でどのように使用されるかを確認してください。

ベクター埋め込み

内容 高次元空間におけるデータテキスト、画像、音声の数値表現。

理由 完全一致を超えた類似性比較を有効にします。

方法 FAISS、Pinecone、および多くの ML モデルのようなライブラリ、および多くの ML モデルは、エンベディングを生成および使用します。

使用する理由 AI / ML の検索および取得タスク用。

これが *Analytics Accelerator Concepts* でどのように使用されるかを確認してください。

取得拡張生成 RAG

内容 LLM が回答する前に外部ソースからドキュメントを取得する AI パターン。

理由 LLM 応答をより正確に、データに基づいたものにします。

方法 ベクトル検索は、関連するエンベディングを取得します。LLM は、それらをプロンプトと組み合わせます。

使用する理由 AI 駆動型アプリの事実の精度を向上させるため。

Intro to RAG

これが *Analytics Accelerator Concepts* でどのように使用されるかを確認してください。

2.1.4 追加の概念

マルチエンジンの相互運用性

内容 マルチプルのクエリエンジンがレイク内の共有データにアクセスできるアーキテクチャ。

理由 組織は柔軟性を求めています。さまざまなユースケースに応じたさまざまなツール SQL エンジン、データサイエンスノートブック、ストリーミングエンジン。

方法 オープンテーブルフォーマット Iceberg、Delta Lake はマルチエンジンアクセスを有効にします。

- Spark はデータを書き込みます → Trino、Postgres Lakehouse、および他のエンジンがそれを照会できます。
- PGD データをオフロード → Lakehouse クラスターと外部エンジンが分析できます。

使用する場合 ベンダーロックインなしで柔軟なデータアーキテクチャを構築します。

データの鮮度とレイテンシー

内容 湖に着くとすぐに、最小限の遅延でデータを照会する機能。

理由 ビジネスユーザーは、新しいデータへのほぼリアルタイムのアクセスをますます期待しています。

方法 Lakehouse アーキテクチャはこれをサポートしています。

- ELT パイプラインは、レイクデータを継続的に更新できます。
- オープンテーブルフォーマットは、アトミックコミットとトランザクションの可視性をサポートしています。
- ベクトル化された Lakehouse エンジンは、更新されたデータをすぐに照会できます。

使用する場合 完全なウェアハウス ETL サイクルを必要とせずに、業務データまたはストリーミングデータの分析。

2.1.5 次に探索する

- [Analytics Accelerator Concepts](#)
- [Analytics How-To Guides](#)
- [Analytics Terminology](#)

2.2 Analytics Accelerator Concepts

このページでは、EDB が **アナリティクスアクセラレータ PGAA** および **ハイブリッドマネージャー HM** を介して最新の分析機能を提供する方法について説明します。

このページは、Analytics Hub の一部です。完全なナビゲーションについては、次をご覧ください [Analytics Hub — Analytics Accelerator Concepts — How-Tos \(Runbook-Aligned\)](#)

確立された業界パターンとテクノロジーに基づいて構築されています。このスペースを初めて使用する場合は、次のことから始めます。

- [Analytics Terminology](#) — 使用される重要な用語を学習する
- [Analytics Generic Concepts](#) — 一般的なアーキテクチャとパターンを理解する

2.2.1 Postgres での分析に対する EDB のビジョン

EDB は、アナリティクスアクセラレーター PGAA を介して最新の分析パターンを実装し、Postgres が運用ワークロードと分析ワークロードの両方の統合プラットフォームとして機能できるようにします。

重要な原則

- **運用データに近い** 分析を実行し、データの移動と遅延を削減します
- オブジェクトストレージの高速分析には **オープンフォーマット** と **ベクトル化クエリエンジン** を使用
- **データ階層化** をサポートして、コストとパフォーマンスのバランスをとります
- **コンピューティングとストレージ** を分離して、独立したスケーリングを可能にする
- **ハイブリッドマネージャー** を介してスタック全体を管理する

関連概念 [Data lakehouse](#)

2.2.2 主要な EDB ソリューションとテクノロジー

EDB Postgres Lakehouse

Lakehouse アーキテクチャにより、EDB Postgres はオブジェクトストレージのデータを照会できます。

- レイクハウスノードは、**オープンテーブルフォーマット** のデータを照会します [Iceberg](#)、[Delta Lake](#)
- [Apache DataFusion](#) を活用したベクトル化されたクエリーの実行

- **Parquet** などのカラムナストレージ形式
- コンピューティングとストレージの分離

関連概念

- Columnar storage formats
- Vectorized query engines
- Separation of storage and compute
- Open table formats

ベクトル化されたクエリの最適化

アナリティクスアクセラレーター PGAA は、レイクハウスノードに [Apache DataFusion](#) を埋め込みます。

- データを **カラムバッチ** として処理
- **SIMD** 命令を使用してパフォーマンスを高速化
- **Parquet** 形式のデータ用に最適化

関連概念 [Vectorized query engines](#)

EDB Postgres 分散 PGD および階層テーブル

EDB Postgres 分散 PGD は階層テーブルを強化します。

- **BDR AutoPartition** は時間ベースのパーティショニングを管理します
- コールドデータは自動的に **オブジェクトストレージにオフロード** Iceberg テーブル
- ホットデータはトランザクション PGD ノードに残る

これにより、操作パフォーマンスとコスト効率の高い履歴データアクセスのバランスがとれます。

関連概念

- Data tiering
- [ETL / ELT](#)

高度なクエリ最適化

アナリティクスアクセラレーター PGAA は、分析のためのコア Postgres 機能を向上させます。

- 並列クエリ処理の強化
- 分析ワークロード用に最適化された結合アルゴリズム

- 高性能 OLAP クエリのためのベクトル化エンジンとの緊密な統合

関連概念 [OLAP / OLTP](#)

2.2.3 アナリティクスと AI / ML ワークロード

Analytics Accelerator は、従来の分析と AI / ML データパイプラインをサポートします。

- Lakehouse ノードは、モデルのトレーニングで使用される大規模なデータセットへの効率的なアクセスを提供します
- 階層化データパイプラインは、費用対効果の高い AI / ML ステージングパターンをサポート
- オープンテーブルフォーマットにより、ベクトル検索および**取得拡張生成 RAG** ワークフローとの統合が可能

関連概念

- Semantic search
- Vector embeddings
- Retrieval augmented generation (RAG)

2.2.4 EDB が製品全体にアナリティクスを実装する方法

ハイブリッドマネージャー HM

ハイブリッドマネージャーはコントロールプレーンとして機能します。

- レイクハウスクラスター のプロビジョニングと管理
- 階層テーブル の構成と監視
- ストレージ場所、 カタログ接続、 およびコンピューティングノード の管理

EDB Postgres Advanced Server / Extended Server

EDB Postgres Advanced Server は以下を追加します

- パラレル クエリー機能の向上
- 分析に役立つ高度な SQL 機能
- PGAA 機能のネイティブサポート

EDB Postgres Distributed PGD

PGD は、次の基盤を提供します。

- 高可用性トランザクションワークロード
- データの階層化とオフロード階層化テーブル
- Lakehouse クラスターと統合するスケーラブルなアーキテクチャ

2.2.5 次に探索する

- [Analytics Terminology](#)
- [Analytics Generic Concepts](#)
- [Analytics How-To Guides](#)
- [Analytics in Hybrid Manager](#)

2.3 Analytics Terminology

このページでは、アナリティクスアクセラレーター PGAA およびハイブリッドマネージャー HM 分析機能で使用される重要な用語を定義します。

概念、ハウツー、機能ガイドを探索する際のクイックリファレンスとして使用します。

2.3.1 Apache Iceberg

オブジェクトストレージに保存される大規模な分析データセットのオープンテーブル形式。スキーマの進化、タイムトラベル、および多くの分析エンジン Spark、Trino、Flink、Postgres との相互運用性を提供します。

[Apache Iceberg Integration With Analytics Accelerator](#)

2.3.2 デルタレイク

ACID トランザクションと信頼性をデータレイクに追加するオープンテーブル形式とストレージレイヤー `_delta_log` トランザクションログを使用して Parquet ファイルに構築されています。

[Delta Lake Integration with Analytics Accelerator](#)

2.3.3 データレイクハウス

データレイクとデータウェアハウスの要素を組み合わせた最新のアーキテクチャ。

- オブジェクトストレージにデータを保存する
- オープンテーブルフォーマットのサポート Iceberg、Delta Lake
- ベクトル化されたクエリエンジンを使用して、高速 SQL 分析を提供します
- ストレージをコンピューティングから分離する

関連概念 Data lakehouse

2.3.4 EDB Postgres Lakehouse クラスター

オブジェクトストレージに保存されているデータに高速 SQL クエリを実行するためにハイブリッドマネージャーによってプロビジョニングされた管理対象分析コンピューティングクラスター。

- Iceberg および Delta Lake フォーマットをサポート
- ベクトル化されたクエリエンジン Apache DataFusion を使用します
- スケーラビリティとコスト効率のためにストレージとコンピューティングを分離

EDB Postgres Lakehouse Architecture

2.3.5 EDB Postgres Distributed PGD

Postgres の高度な分散バージョン

- 高可用性とマルチマスターリプリケーションを提供します
- 高度なデータ階層化パターン階層化テーブルを有効にします
- Lakehouse クラスターとシームレスに動作して、スケーラブルな分析を実現

Learn more

2.3.6 階層化テーブル

PGD **AutoPartition** を使用して大規模な時間ベースのデータセットを管理する機能。

- **ホットデータ** は PGD トランザクションクラスターに残ります
- **コールドデータ** は Apache Iceberg テーブルとしてオブジェクトストレージにオフロードされます
- クエリーはホットデータとコールドデータの両方に透過的にアクセスできます

Tiered Tables With Analytics Accelerator

2.3.7 PGAA アナリティクスアクセラレーター

Postgres で次のことを可能にする EDB 機能。

- クエリオープンテーブルフォーマット Iceberg、Delta Lake
- オブジェクトストレージにマップする **外部テーブル** を定義する
- Power Lakehouse クラスタクエリー
- PGD 階層テーブルのオフロードとクエリーを有効にする

See how-tos

2.3.8 PGFS Postgres ファイルシステム

Postgres がオブジェクトストレージへのアクセスに使用できる**ストレージの場所**を定義する拡張機能。

- Iceberg と Delta Lake へのアクセスに PGAA および PGD によって使用
- AWS S3、Google Cloud Storage、および互換性のあるサービスをサポート
- オブジェクトストレージを照会する必要がある各レイクハウスまたは PGD クラスタで構成する必要があります

See how-tos

2.3.9 AutoPartition

PGD 双方向レプリケーションの機能

- PGD テーブルの時間ベースのパーティションを自動的に作成します
- `analytics_offload_period` で構成されている場合、古いパーティションをオブジェクトストレージに自動的にオフロード

See how-tos

2.3.10 BDR 分析テーブル

内部 PGD コンセプト

- `analytics_table` ビューは、階層テーブルオフロード用にマークされた PGD テーブルを追跡します
- 各テーブルの状態とオフロードの進行状況を追跡します

階層テーブルの動作の監視と検証に役立ちます。

See how-tos

2.3.11 氷山カタログ

Iceberg テーブルのスキーマ、場所、バージョンを追跡するメタデータサービス

- 複数のエンジン Spark、Trino、Postgres 間の完全な相互運用性に必要
- HM 管理 Lakekeeper または外部 AWS Glue、Nessie、Polarisなどで管理できます
- PGAA と PGD は両方とも、Iceberg REST カタログへの接続をサポート

See how-tos

2.3.12 Lakekeeper

HM 管理の Iceberg カタログサービス

- Iceberg テーブルの集中カタログを提供します
- カタログ統合による PGD 階層テーブルのオフロードをサポート
- カタログベースのテーブル検出を介して Lakehouse クラスタークエリをサポート
- Iceberg REST カタログ API を使用

See how-tos

2.3.13 オープンテーブルフォーマット

分析に使用される標準化されたファイルレイアウトとメタデータ形式

- **Apache Iceberg**
- **デルタレイク**

同じデータ Spark、Trino、Postgres Lakehouse などへのマルチエンジンアクセスを有効にします。

[Learn more](#)

2.3.14 ベクトル化されたクエリエンジン

行ごとではなく **列バッチ** でデータを処理する分析エンジン

- 分析クエリを高速化します
- 最新の CPU 機能 SIMD を利用
- Lakehouse クラスタークエリパフォーマンスを強化

EDB Lakehouse クラスターは、このために [Apache DataFusion](#) を埋め込みます。

2.3.15 データ階層化

ホット、ウォーム、およびコールドデータをさまざまなストレージ階層に保存する方法

- ホット → 高速トランザクション PGD ノード
- コールド → オブジェクトストレージ Iceberg テーブル
- 階層テーブルは、Postgres + HM でこのパターンを自動化します

関連概念 Data tiering

2.3.16 ストレージとコンピューティングの分離

Lakehouse アーキテクチャで使用されるパターン

- オブジェクトストレージにデータを保存する 独立してスケーリング
- ストレージとは無関係に分析コンピューティングレイクハウスクラスターをスケールする

費用効率の高い、柔軟な分析アーキテクチャを有効にします。

関連概念 Separation of storage and compute

2.3.17 次のステップ

重要な用語を理解したところで、

- *Analytics Accelerator Concepts*
- *EDB Postgres Lakehouse Architecture*
- *Tiered Tables With Analytics Accelerator*
- See How-To Guides

2.4 Analytics Accelerator Architecture

2.4.1 アーキテクチャの概要

Analytics Accelerator は、PostgreSQL の機能を拡張して、トランザクションおよび分析ワークロードのための統合プラットフォームを作成します。このアーキテクチャは、運用可能な PostgreSQL データベースをレイクハウーストレージパターンと組み合わせて、組織が複雑な ETL パイプラインやデータの重複なしで複数のストレージ階層にわたってデータを照会できるようにします。

このシステムは、3つの基本原則で動作します。弾力的なスケーリングのためのコンピューティングとストレージの分離、ホットおよびコールドデータ層にわたる透過的なクエリアクセス、およびエコシステムの相互運用性のためのオープンテーブルフォーマットとの互換性。これらの原則により、PostgreSQL の使いやすい SQL インターフェイスと操作特性を維持しながら、ペタバイト規模での費用対効果の高い分析が可能になります。

2.4.2 コアコンポーネント

EDB Postgres Distributed Foundation

EDB Postgres Distributed PGD は、ビルトインの高可用性、ホリゾンタル・スケーリング、および自動化されたデータライフサイクル管理を備えたトランザクション基盤を提供します。PGD ノードは、完全な ACID コンプライアンスで運用ワークロードを処理し、AutoPartition はデータ階層化に不可欠な時間ベースのパーティショニング戦略を管理します。

分散アーキテクチャにより、メンテナンス操作中のゼロダウンタイムが保証され、ディザスタリカバリーのための地理的分散が提供されます。書き込み操作は、マルチマスターレプリケーションを介して一貫性を維持し、読み取り操作は利用可能なノード全体で水平にスケーリングします。

Lakehouse クエリエンジン

専用の Lakehouse ノードは、オブジェクトストレージ内のデータに対して分析クエリを実行します。これらのステートレス計算ノードは、Apache DataFusion のベクトル化実行エンジンを活用して、列型データ形式を効率的に処理します。トランザクションノードからの分離により、分析ワークロードが運用パフォーマンスに影響を与えません。

クエリエンジンは、パーティションプルーニング、述語プッシュダウン、適応クエリ実行などの高度な最適化を実装します。これらの最適化により、データのスキャンが大幅に削減され、大規模なデータセットでのインタラクティブなクエリパフォーマンスが可能になります。

ストレージ抽象化レイヤー

PostgreSQL ファイルシステム PGFS は、AWS S3、Azure Blob Storage、Google Cloud Storage、オンプレミスオブジェクトストアを含むさまざまなストレージバックエンドへの統一アクセスを提供します。この抽象化は、認証、ネットワークの復元力、およびパフォーマンスの最適化を透過的に処理します。

ストレージレイヤーは、複数の同時ストレージ場所をサポートし、クラウドプロバイダーにまたがるクエリー、またはクラウドデータとオンプレミスデータを組み合わせたクエリーを可能にします。インテリジェントなキャッシュは、メタデータ操作を削減し、頻繁にアクセスされるデータのクエリレイテンシーを改善します。

テーブルフォーマットの統合

Apache Iceberg と Delta Lake のネイティブサポートにより、より広範なデータエコシステムとの相互運用性が可能になります。アナリティクスアクセラレーター PGAA は、テーブルのメタデータを読み取り、手動構成なしでスキーマ、パーティショニング、およびファイルの場所を理解します。

Iceberg の場合、システムはスキーマの進化、非表示のパーティショニング、タイムトラベルクエリを含む完全な機能をサポートします。デルタレイクのサポートは読み取り操作に焦点を当てており、既存の Spark ベースのデータレイクへのアクセスを提供します。どちらの形式も外部ツールとの互換性を維持し、マルチプルのコンピューティングエンジン間でデータにアクセスし続けます。

2.4.3 アーキテクチャレイヤー

インジェストおよびトランザクションレイヤー

アプリケーションは、標準のデータベース接続を使用して PostgreSQL または PGD クラスターにデータを書き込みます。このレイヤーは、リアルタイムトランザクションを処理し、参照整合性を維持し、ACID 保証を介してデータの一貫性を保証します。

ストリーミングインジェストシステムは継続的なデータフローを提供し、バッチロードは定期的な更新を処理します。トランザクションレイヤーは、インテリジェントなパーティション化と階層化戦略を介して、履歴データの量に関係なく操作パフォーマンスを維持します。

ストレージとメタデータレイヤー

データは、構造とガバナンスを提供するオープンテーブル形式を使用してオブジェクトストレージにあります。Parquet ファイルは実際のデータを保存しますが、メタデータはスキーマ、パーティション、およびファイルの場所を追跡します。この分離により、データを書き換えずにスキーマの進化とタイムトラベルが可能になります。

オプションのカatalogサービスは、マルチエンジン環境の集中的なメタデータ管理を提供します。Catalog は、一貫したテーブルの検出を有効にし、同時の変更を調整し、さまざまなコンピューティングエンジンにわたって監査証跡を維持します。

クエリーと変換レイヤー

複数のクエリエンジンは、ワークロード要件に基づいて、同じ基になるデータにアクセスします。Analytics Accelerator はインタラクティブ SQL クエリーを処理し、Apache Spark は複雑な変換を処理し、機械学習プラットフォームはトレーニングデータを直接読み取ります。

クエリコーディネーションは、複数のエンジンが同時に動作する場合の一貫性を保証します。トランザクションの分離は競合を防止し、スナップショットセマンティクスは、分析の精度に不可欠な反復可能な読み取り保証を提供します。

アクセスと統合レイヤー

標準の PostgreSQL ワイヤプロトコルは、既存のツールおよびアプリケーションとの互換性を保証します。ビジネスインテリジェンスプラットフォーム、データサイエンスノートブック、およびカスタムアプリケーションは、専用のドライバーや変更なしで接続します。

REST API により、自動化と統合シナリオへのプログラムによるアクセスが可能になります。これらの API は、運用展開に重要な Catalog 管理、監視、および管理操作をサポートします。

2.4.4 データフローパターン

オペレーションから分析パイプライン

データは、トランザクション操作をサポートする PostgreSQL でライフサイクルを開始します。データが古び、アクセスパターンが運用から分析に移行すると、階層テーブルはパーティションをオブジェクトストレージに自動的に移行します。この移行は透過的に発生し、アプリケーションの変更は必要ありません。

最近のデータはミリ秒未満のクエリレイテンシーで PostgreSQL に残りますが、履歴データは費用対効果の高いオブジェクトストレージに移動します。クエリは自動的に両方の層にまたがり、保存場所に関係なくすべてのデータの完全な可視性を提供します。

レイクハウスへの直接アクセス

既存のデータレイクを持つ組織は、アナリティクスアクセラレーターを Iceberg または Delta Lake テーブルに直接接続します。このパターンでは、データの移動が不要になり、既存の投資の即時分析機能が有効になります。

システムはテーブルスキーマを自動的に検出し、PostgreSQL SQL を適切な操作に変換し、標準のデータベースプロトコルを介して結果を返します。このアプローチは、移行中または確立されたデータプラットフォームと統合する場合に特に価値があることがわかります。

マルチエンジンのコラボレーション

さまざまなエンジンが、共通のストレージを共有しながら、それぞれの強みに基づいてデータを処理します。Spark は、複雑な ETL 変換を処理して、結果を Iceberg テーブルとして書き込みます。Analytics Accelerator は、インタラクティブ分析のためにこれらのテーブルを照会します。機械学習プラットフォームは、モデルのトレーニング用に同じデータを読み取ります。

この協調的なアプローチにより、データの重複が排除され、分析ワークフロー全体の一貫性が保証されます。1つのエンジンによって行われた変更は、共有メタデータとスナップショットの分離を介して他のエンジンにすぐに表示されます。

2.4.5 パフォーマンスアーキテクチャ

クエリ最適化パイプライン

オプティマイザーはクエリを分析して、レイクハウスのワークロードに固有の最適化の機会を特定します。パーティションプルーニングは、クエリの実行が開始される前に無関係なデータを削除します。述語プッシュダウンはフィルターをストレージレイヤーに移動し、データ転送を最小限に抑えます。

コストベースの最適化では、実行プランを生成するときにネットワーク遅延、データの局所性、およびフォーマットの特性を考慮します。オプティマイザーは、実行中に検出された実際のデータ分布に基づいて戦略を適応させます。

キャッシング階層

複数のキャッシュレイヤーは、繰り返されるアクセスパターンを最適化します。メタデータキャッシュには、テーブルスキーマ、パーティション情報、およびファイルの場所が保存されます。データキャッシュは、頻繁にアクセスされる Parquet ファイルをローカルストレージに保持します。クエリ結果のキャッシュは、再計算なしで同一のクエリを提供します。

キャッシュの無効化は、テーブルフォーマットのバージョン管理を尊重し、基になる変更にもかかわらず一貫性を保証します。適応型キャッシュ管理は、アクセス頻度、クエリコスト、および使用可能なリソースに基づいて、保持を優先します。

パラレル実行

クエリはマルチプルのディメンションにわたって並列化され、スループットを最大化します。パーティションレベルの並列処理は、独立したパーティションを同時に処理します。ファイルレベルの並列処理は、パーティション内の複数のファイルを同時に読み取ります。列レベルの並列処理は、列指向の操作のベクトル化された実行を活用します。

実行フレームワークは、使用可能なリソースとクエリ特性に基づいて並列処理を動的に調整します。この適応により、ハードウェアの使用率を最大化しながら、リソースの枯渇を防ぎます。

2.4.6 高可用性と復元力

トランザクション層の可用性

PGD は、マルチマスターリプリケーションを介して継続的な可用性を提供します。ノードに障害が発生すると、データ損失ゼロの自動フェイルオーバーがトリガーされます。地理的な分散により、構成可能な復旧目標を使用したディザスターリカバリーが可能になります。

アップグレードやスキーマの変更を含むメンテナンス操作は、ローリング展開戦略を通じてダウンタイムなしで発生します。システムは、部分的な障害が発生しても完全な機能を維持し、ビジネスの継続性を保証します。

分析ティアの可用性

ステートレスレイクハウスノードは、レプリケーションではなく冗長性を介して可用性を提供します。ロードバランサーは、自動障害検出と再ルーティングを使用して、利用可能なノードにクエリーを分散します。

オブジェクトストレージの耐久性保証は、コンピューティング障害とは無関係にデータの可用性を保証します。複数のアベイラビリティゾーンとクロスリージョンレプリケーションは、重要なデータセットの復元力を追加します。

カタログの復元力

カタログサービスは、選択した実装に基づいて独自の可用性戦略を実装します。REST カタログは、HTTP ロードバランシングとバックエンドの冗長性を活用します。AWS S3 テーブルは、管理対象インフラストラクチャを介してサーバーレスの可用性を提供します。

カタログが利用できない場合は、テーブル検出に影響しますが、既知のテーブルのクエリ実行には影響しません。このアーキテクチャは、カタログのメンテナンスまたは障害中に、コア機能の維持を正常に低下させます。

2.4.7 セキュリティアーキテクチャ

アクセス制御フレームワーク

PostgreSQL のロールベースのアクセス制御は、レイクハウスリソースに拡張されます。テーブルレベルと列レベルの権限は、保存場所に関係なく透過的に適用されます。行レベルのセキュリティポリシーは、ユーザーコンテキストに基づいてデータを動的にフィルタリングします。

エンタープライズ認証システムとの統合により、シングルサインオン機能が提供されます。LDAP、Active Directory、および OAuth 実装により、プラットフォーム全体で一貫した ID 管理が保証されます。

暗号化戦略

包括的な暗号化は、ライフサイクル全体にわたってデータを保護します。Transport Layer Security は、クライアント、計算ノード、およびストレージ間で移動しているデータを暗号化します。サーバーサイドの暗号化は、クラウドプロバイダーまたはカスタマー管理キーを使用して保存データを保護します。

列レベルの暗号化は、機密フィールドの保護を追加します。暗号化された列は、セキュリティコンプライアンスを維持しながら、特殊なインデックスを介して照会可能なままです。

監査とコンプライアンス

詳細な監査ログは、コンプライアンス要件のすべてのデータアクセスを追跡します。クエリログは、ユーザー、タイムスタンプ、およびアクセスされたオブジェクトをキャプチャします。ストレージアクセスログは、すべてのオブジェクト操作を記録します。認証ログは、ログイン試行と特権の変更を追跡します。

GDPR、HIPAA、PCI DSS を含むコンプライアンスフレームワークは、セキュリティ実装をガイドします。データの所在地の制御は地理的なコンプライアンスを保証し、保持ポリシーはデータのライフサイクル管理を自動化します。

2.4.8 展開パターン

管理対象サービスの展開

Hybrid Manager は、自動化されたプロビジョニング、スケーリング、およびメンテナンスを備えたフルマネージドの Analytics Accelerator 展開を提供します。このプラットフォームは、インフラストラクチャ管理、セキュリティ更新、およびパフォーマンスの最適化を処理します。

自己管理展開

自己管理展開では、インフラストラクチャと構成を完全に制御できます。組織は、オンプレミス、プライベートクラウド、またはインフラストラクチャ・アズ・コードのアプローチを使用して展開します。

Kubernetes オペレーターは、展開とライフサイクル管理を簡素化します。Helm チャートは、一般的なシナリオのテンプレート化された構成を提供します。Terraform モジュールは、クラウドプロバイダー全体でインフラストラクチャの自動化を可能にします。

ハイブリッド展開

ハイブリッドパターンは、管理対象コントロールプレーンと自己管理データプレーンを組み合わせます。このアプローチは、操作の簡素化と規制対象業界で一般的なデータ主権要件のバランスをとります。

管理対象コントロールプレーンは、オーケストレーション、モニタリング、およびメタデータ管理を処理します。自己管理データプレーンは、プラットフォーム機能を活用しながら、組織の境界内でデータを維持します。

2.5 Analytics Accelerator Quick Start

このクイックスタートでは、次のことを行います。

1. Lakehouse ノードを作成します。
2. 優先する Postgres クライアントを使用してノードに接続します。
3. オブジェクトストレージ内のサンプルデータ TPC-H、TPC-DS、Clickbench、または 1BRC を照会します。

詳細および高度なユースケースについては、[PGAA functions reference](#) を参照してください。

2.5.1 はじめに

Postgres Lakehouse は、EDB Postgres® AI プラットフォームでプロビジョニングできる新しいタイプの Postgres クラスタ。実際には 1 つのノードです。これには、オブジェクトストレージの列指向レイクハウステーブルを高速照会するための Apache DataFusion に基づくベクトル化クエリエンジンが含まれています。Delta Lake protocol を使用します。

知っておくべきこと

詳細については、[Analytics Accelerator Concepts](#) および [PGAA functions reference](#) を参照してください。ただし、これらが知っておくべき主なポイントです。

- Postgres Lakehouse ノードは一時的なものです。
- Postgres Lakehouse ノードは、オブジェクトストレージのベンチマークデータセットを照会するように事前構成されています。
- Postgres Lakehouse は通常読み取り専用ですが、それ以外の場合は、すべての有効な Postgres クエリをサポートします。
- システムテーブルは書き込み可能なため、ユーザーとロールを作成できます。これらのテーブルは、接続されたブロックストレージに保存されるため、インスタンスが再起動されるまで持続し、バックアップ/復元操作の一部として保存されます。それ以外の場合、Lakehouse テーブルは最終的にオブジェクトストレージに保存されるため、バックアップの一部ではありません。

基本アーキテクチャ

この図は、Lakehouse ノードのボックスの内容を示しています。

2.5.2 はじめに

HCP ポータルにログインしてプロジェクトを作成したら、クラスタを作成できます。

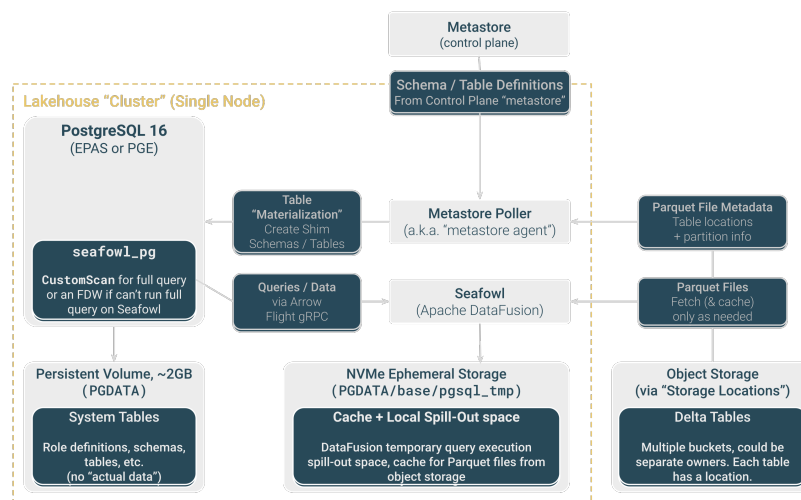


図 1: Level 300 Architecture

Lakehouse ノードを作成する

プロジェクトページで、**Create New > Lakehouse Analytics** を選択します。

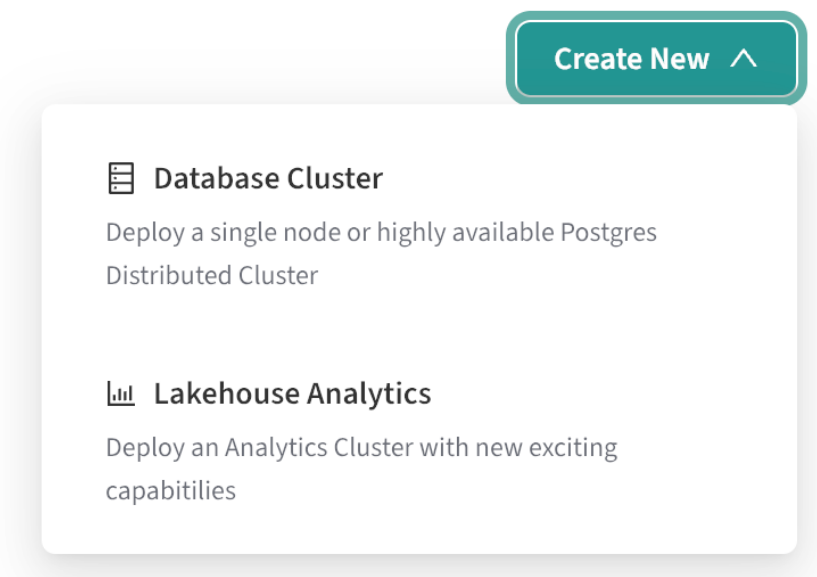


図 2: Create Lakehouse Node Dropdown

Create Analytical Cluster ページが開きます。ここで、テンプレートビルドまたはカスタムビルドを選択できます。

Custom Build を選択し、**Start from Scratch** を選択します。**Create Analytical Cluster** ページが開き、クラスターのサイズと構成を選択できます。

次に、**Clusters** ページに戻ります。ここで、クラスターの作成時に、クラスターのステータスを確認できます。クラスターのプロビジョニングには 10～15 分かかる場合があります。

いくつかの注意事項

← Back **Create Analytical Cluster** [Support](#) | [EDB Postgres® AI Docs](#)

[Choose Path](#)

Choose Your Path

Use a template or build from scratch.

No Templates Available, Yet...

No templates are available, but you can create your own! Start building a new template below.

Create New Template

Custom Build

Customize every aspect of your cluster, from primary settings to backups. Best for advanced users needing flexibility.

Start from Scratch

図 3: Create Lakehouse Node Wizard Step 1

← Back **Create Analytical Cluster** [Support](#) | [EDB Postgres® AI Docs](#)

[Cluster Settings](#) [Additional Settings](#)

Cluster Name

Enter a name for your cluster

Password

Passwords must be at least 12 characters.

Enter a password

We do not store your password and cannot retrieve it. If you forget your password, you can simply change it on the "Edit Cluster" page or from Clusters Action menu.

Tags

[Add Tag](#)

Deployment Location

Location

managed-devspatcher

Database Type

Please reference [this page](#) for more information on database types.

Postgres Type

PostgreSQL EDB Postgres Extended Server EDB Postgres Advanced Server

Postgres Version Software Image

Select... Select...

16 is the minimum supported version for Lakehouse clusters

Instance Size

CPU (0.1-15 Cores) Memory (0.1-10 Gi)

Cores Gi

Cluster Summary

Cluster Type	Single Node
Cluster Nodes	1 node
Cluster Name	--
Tags	--
Deployment Location	managed-devspatcher
Postgres Type	--
Postgres Version	--
CPU Cores	--
Memory	--
Networking	Private
Backups Retention	30 Days
Scheduled Backup Time	21:49

図 4: Create Lakehouse Node Wizard Step 2

- 独自のパスワードを入力する必要があります。安全なパスワードを作成し、パスワードマネージャーに保存します。HCP はパスワードを保存しません。
- EDB Postgres Advanced Server または EDB Postgres Extended Server を選択できます。
- NVMe ドライブ上のすべてのデータは、クラスターがシャットダウンすると失われます。
- システムテーブルの永続データユーザー、ロールなどは、接続されたブロックストレージデバイスに保存され、再起動またはバックアップ/復元サイクル後も存続します。
- Postgres 16 のみがサポートされています。

2.5.3 Lakehouse ノードの動作

ノードに接続します

EDB Postgres AI プラットフォームから他のクラスターに接続するのと同じ方法で、Postgres クライアントを使用して Lakehouse ノードに接続できます。クラスターの詳細ページに移動し、その接続文字列をコピーします。

たとえば、`.pgpass blob` を `~/.pgpass` にコピーします。`$YOUR_PASSWORD` を、クラスター起動時に指定したパスワードに置き換えてください。次に、接続文字列をコピーし、`psql` または `pgcli` の引数として使用できます。

通常、任意の Postgres クライアントを使用してデータベースに接続できます。私たちはすべてのイントロスペクションクエリーが機能することを期待しています。そうでないクエリーが見つかった場合、それはバグです。

コンストレインを理解する

- すべてのクラスターは EDB Postgres Advanced Server または EDB Postgres Extended Server を使用します。したがって、接続したときにインストール内のこれらのフレーバーからボイラープレートテーブルが表示されることが予想されます。
- クエリ可能データベンチマークデータセットのようなは、デルタテーブルとしてオブジェクトストレージに保存されます。すべてのクラスターは、1~1000 のスケールファクターで TPC-H、TPC-DS、クリックベンチ内のベンチマークデータを含むストレージバケットをポイントするようにプリロードされています。
- 現在 AWS のみがサポートされています。Bring your own account BYOA はサポートされていません。
- EDB Postgres AI アカウントでアクティブ化されている任意のリージョンにクラスターを展開できます。各リージョンには、ベンチマークデータのコピーを含むバケットがあります。したがって、クラスターを起動すると、クラスターに最も近い場所のベンチマークデータが使用されます。
- クラスターは一時的なものです。ロール、ユーザー、付与などのシステムテーブルのデータを除き、データはハードドライブに保存されません。クラスターを再起動するか、クラスターをバックアップして復元すると、これらのシステムテーブルが復元されます。ただし、オブジェクトストレージのデータは変更されないままです。
- クラスターは、オブジェクトストレージ内のデータの読み取り専用クエリーをサポートしていますが、ユーザーなどを作成するためにシステムテーブルへの書き込みクエリーをサポートしています。オブジェクトストレージに直接書き込むことはできません。テーブルを作成することはできません。

- 独自のデータをオブジェクトストレージにロードする場合は、[Reference - Bring your own data](#) を参照してください。

2.5.4 ベンチマークデータセットを検査する

ベンチマークデータセットを検査します。すべてのクラスターには、すぐに使用できるベンチマークデータがあります。pgcli を使用している場合、`\dn` を実行して、使用可能なテーブルを表示できます。

使用可能なベンチマークデータセットは次のとおりです。

- TPC-H、スケールファクター 1、10、100、および 1000
- TPC-DS、スケールファクター 1、10、100、および 1000
- ClickBench、1 億行
- 10 億行チャレンジ

ベンチマークデータセットの詳細については、[Reference - Available benchmarking datasets](#) を参照してください。

2.5.5 ベンチマークデータセットを照会する

いくつかの基本的なクエリーを実行してみることができます。

```
edb_admin> select count(*) from clickbench.hits;
+-----+
| count |
+-----+
| 99997497 |
+-----+
SELECT 1
Time: 0.945s

edb_admin> select count(*) from brc_1b.measurements;
+-----+
| count |
+-----+
| 1000000000 |
+-----+
SELECT 1
Time: 0.651s
```

!!!note

構文/クエリーの互換性に関する注意事項と情報については、[PGAA functions reference](#) を参照してください。
!!!

2.6 Reference - EDB Postgres® AI Lakehouse

HCP Lakehouse は、データに分析クエリを実行するための強力なツールです。Postgres の上に構築されているため、既存の Postgres の知識とツールを使用できます。

Lakehouse で最高のエクスペリエンスを得るには、*Analytics Accelerator Quick Start* に従ってベンチマークデータを照会します。次に、独自のデータと *How-To Lakehouse Read With/Without A Catalog* のロードを試みることができます。

このセクションでは、知っておくべき重要な事項のいくつかについて詳しく説明します。

- [Benchmarking datasets](#)
- [Queries in Lakehouse](#)
- [Delta Table tools](#)
- [DirectScan and fallback modes](#)
- [Users](#)
- [Loading data](#)
- [PGAA functions reference](#)

保存場所を管理するための PGFS 機能については、AI Factory PGFS documentation を参照してください。

2.6.1 Benchmarking datasets

Lakehouse ノードをプロビジョニングすると、サンプルベンチマークデータセットを含む AWS us-east-1 のパブリック S3 バケットをポイントするように事前構成されます。

注釈

Accessing these datasets can incur costs for data transfer and be subject to cross-region latencies.

スキーマ名で参照することにより、これらのデータセットのテーブルを照会できます。

Schema name	Dataset
<i>tpcd_sf_1</i>	TPC-DS, Scale Factor 1
<i>tpcd_sf_10</i>	TPC-DS, Scale Factor 10
<i>tpcd_sf_100</i>	TPC-DS, Scale Factor 100
<i>tpcd_sf_1000</i>	TPC-DS, Scale Factor 1000
<i>tpch_sf_1</i>	TPC-H, Scale Factor 1
<i>tpch_sf_10</i>	TPC-H, Scale Factor 10
<i>tpch_sf_100</i>	TPC-H, Scale Factor 100
<i>tpch_sf_1000</i>	TPC-H, Scale Factor 1000
<i>clickbench</i>	ClickBench, 100 million rows
<i>brc_1b</i>	Billion row challenge

!!!note “ClickBench データに関する注意事項”

データ列 `EventData` は、日付ではなく整数です。

ClickBench の列名には大文字が含まれるため、引用符で囲む必要がありますが、Postgres の引用符のない識別子は小文字と大文字が区別されません。例

```
select "Title" from clickbench.hits;

select Title from clickbench.hits; !!!
```

2.6.2 Queries

読み取り専用クエリーがサポートされています。オブジェクトストレージに直接書き込むことはできません。これには、すべての Postgres ビルトインファンクション、ステートメント、およびタイプが含まれます。また、展開することを選択したディストリビューションに応じて、EDB Postgres Advanced Server または EDB Postgres Extended Server が提供するものが含まれます。

データを挿入、更新、削除、または変更することはできません。CREATE TABLE は、ノードに通常の Postgres テーブルを作成するためにのみ使用できます。ノードは一時的なもので、終了すると破壊されます。CREATE TABLE... USING PGAA を使用して、S3 互換オブジェクトストレージのデルタテーブルを参照する外部テーブルを作成することもできます。

独自の ETL スクリプト、または Delta テーブルまたは Iceberg テーブルへの書き込みをサポートするサードパーティの ETL プロバイダーを使用して、アウトオブバンドでデータをバケットにロードする必要があります。

詳細については、[Loading data](#) を参照してください。

1 つの例外は、ユーザー、ロール、付与を保存するために使用される Postgres システムテーブルです。これらのテーブルは、バックアップとリストアに含まれるローカルブロックデバイスに保存されます。したがって、CREATE USER、CREATE ROLE、または GRANT USAGE を使用できます。これらのユーザー/ロール/付与は、再起動と復元後も残ります。

2.6.3 Delta Lake Table tools

Delta Lake テーブルの作成

deltalake Python ライブラリの使用

`deltalake` Python ライブラリを使用して、デルタテーブルを作成し、バケットに書き込むことができます。

lakehouse-loader ユーティリティの使用

EDB がこのタスク用に作成した `lakehouse-loader` ユーティリティを使用して、任意の Postgres インスタンスからストレージバケットの Lakehouse テーブルにデータをエクスポートすることもできます。

例、`lakehouse-loader` ユーティリティを使用する場合

```
export PGPASSWORD="..."
export AWS_ACCESS_KEY_ID="..."
export AWS_SECRET_ACCESS_KEY="..."

# export other AWS envvars

./lakehouse-loader postgres-to-delta postgres://test-user@localhost:5432/test-db -q
→ "SELECT * FROM some_table" s3://my-bucket/my_schema/my_table
```

このコードは、test-db データベースの some_table テーブルから my-bucket バケットの my_schema/my_table パスのデルタテーブルにデータをエクスポートします。

次に、my_schema/my_table パスのデルタテーブルを参照する外部テーブルを作成することにより、Lakehouse ノードでこのテーブルを照会できます。これを行う方法の詳細については、[How-To Lakehouse Read With/Without A Catalog](#) を参照してください。

2.6.4 DirectScan and fallback modes

DirectScan

Postgres Lakehouse は、クエリ全体を DataFusion、可能な場合にクエリを処理するために使用されるベクトル化クエリに「プッシュダウン」できる場合に最速です。将来的には、部分的なプッシュダウンのサポートを追加するため、より詳細な制御が可能になります。

Postgres Lakehouse は、2つのモードでクエリを実行できます。最初に、DataFusion に基づいた専用の列型データベースである Seafoam を使用して、クエリ全体を実行しようとします。たとえば、JSON のような PostgreSQL 固有の操作を使用するために Seafoam がクエリ全体を実行できない場合、Postgres Lakehouse は PostgreSQL エグゼキュータの使用にフォールバックし、Seafoam は完全なテーブルコンテンツをストリーミングします。

クエリが非常に遅い場合、それが起こっている可能性があります。

EXPLAIN を使用してクエリプランを確認する

クエリーで EXPLAIN を実行し、最上位のクエリーノードが DirectScan であることを確認することにより、どのモードが使用されているかを確認できます。例

```
explain select count from (select count(*) from tpch_sf_1.lineitem);

          QUERY PLAN
-----
→ -----
→ -----
→ -----
Aggregate (cost=167.52..167.55 rows=1 width=8)
  → Append (cost=0.00..165.01 rows=1001 width=0)
    → CompatScan on "16529" lineitem_2 (cost=100.00..150.00 rows=1000 width=0)
```

(次のページに続く)

(前のページからの続き)

```

SeafowlPlan: logical_plan
  TableScan: tpch_sf_1.lineitem projection=[l_orderkey, l_partkey, l_
→suppkey, l_linenummer, l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag,
  l_linestatus, l_shipdate, l_commitdate, l_receiptdate, l_shipinstruct, l_shipmode, l_
→comment]
(6 rows)

```

この場合、クエリーは PostgreSQL によって実行され、Seafowl はテーブルをスキャンする場合にのみ関係します下部の `CompatScan` を参照してください。この場合の修正は、内側の `COUNT(*)` 列に明示的に名前を付けることです。Seafowl は暗黙の名前 `count(*)` を付けるのに対し、PostgreSQL は `count` と呼ぶためです。

```

edb_admin=> explain select count from (select count(*) as count from tpch_sf_1.
→lineitem);

```

QUERY PLAN

```

-----
DirectScan: logical_plan
  Projection: COUNT(*) AS count
    Aggregate: groupBy=[[]], aggr=[[COUNT(UInt8(1)) AS COUNT(*)]]
      TableScan: tpch_sf_1.lineitem projection=[]

```

(4 rows)

ここでは、上部に `DirectScan` が表示されます。これは、Seafowl がクエリ全体を実行していることを意味します。

Seafowl で完全に実行できるようにクエリを変更するのに問題がある場合は、サポートチケットをオープンしてください。

2.6.5 User management

Lakehouse ノードをプロビジョニングするときは、パスワードを入力する必要があります。edb_admin ユーザーとしてログインするには、パスワードが必要です。edb_admin ユーザーは、ユーザー、ロール、および付与を作成できるスーパーユーザーアカウントです。

HCP はこのパスワードを保存しません。

edb_admin の資格情報を共有するか、それを使用して他のユーザーを作成し、それらの資格情報を配布できます。

2.6.6 Loading data

独自のデータの持ち込み

内部にデルタテーブルを含む任意の S3 バケットをレイクハウスノードに向けることが可能です。ただし、これにはいくつかの大きな注意事項があります。これらは将来のリリースで解決される予定です。

注意事項

- テーブルは場所内に [Delta Lake Tables](#) として保存する必要があります。
- *Delta Lake Table* または *Delta Table* は、いくつかの JSON メタデータを含む Parquet ファイルのフォルダーです。

バケットへのデータのロード

lakehouse-loader ユーティリティを使用して、任意の Postgres インスタンスからストレージバケットのデルタテーブルにデータをエクスポートできます。そのユーティリティを取得して使用方法の詳細については、[Delta Lake Table tools](#) を参照してください。

詳細については、[How-To Lakehouse Read With/Without A Catalog](#) を参照してください。

2.6.7 PGAA functions reference

概要

インスタンス上のすべての PGAA テーブル、その格納場所、およびパスをリストします。

パラメーター

なし。

Returns

Column	Type	Description
<i>schema_name</i>	<i>text</i>	テーブルのスキーマ
<i>table_name</i>	<i>text</i>	テーブルの名前
<i>storage_location_name</i>	<i>text</i>	テーブルの保存場所の名前
<i>path</i>	<i>text</i>	格納場所にあるテーブルへのパス
<i>managed_by</i>	<i>text</i>	テーブルを作成したユーザーの名前
<i>format</i>	<i>text</i>	テーブルのフォーマット現在 `delta` または `iceberg`

pgaa.lakehouse_table_stats()

概要

Lakehouse テーブルの最新バージョンのバイト単位のサイズと、すべてのバージョンの合計サイズを返します。

パラメーター

Parameter	Type	Description
<i>relation</i>	<i>regclass</i>	サイズを取得するテーブルの名前

Returns

Column	Type	Description
<i>latest_snapshot_size</i>	<i>bigint</i>	テーブルの最新のスナップショットのバイト単位のサイズ
<i>total_size</i>	<i>bigint</i>	テーブルのすべてのバージョンのバイト単位の合計サイズ

2.6.8 PGAA functions reference

pgaa.create_storage_location

概要

S3 互換のオブジェクト保存場所を参照する新しい保存場所を作成します。

パラメーター

Parameter	Type	Description
<i>name</i>	<i>text</i>	保存場所の名前
<i>uri</i>	<i>text</i>	S3 互換のオブジェクト保存場所の URI `s3://bucket-name` または `s3://bucket-name/prefix` など
<i>options</i>	<i>json</i>	保存場所のオプション設定

オプション

Option	Type	Description
<i>access_key_id</i>	<i>text</i>	保存場所のアクセスキー ID
<i>secret_access_key</i>	<i>text</i>	保存場所のシークレットアクセスキー
<i>session_token</i>	<i>text</i>	保存場所のセッショントークン
<i>region</i>	<i>text</i>	保存場所のリージョン
<i>endpoint</i>	<i>text</i>	保存場所のエンドポイント
<i>bucket</i>	<i>text</i>	保存場所のバケット
<i>use_http</i>	<i>boolean</i>	保存場所に HTTPS の代わりに HTTP を使用する

次のページに続く

表 6 – 前のページからの続き

Option	Type	Description
<i>skip_signature</i>	<i>boolean</i>	保存場所の署名検証をスキップする

2.6.9 Supported AWS instances

Lakehouse ノードを展開するときは、**m6id** インスタンスファミリーからインスタンスタイプを選択する必要があります。重要なのは、これらのインスタンスには、NVMe ドライブが接続されていることです。

インスタンスは一時的なものです。これらの NVMe ドライブは、クエリの処理中に*スピルアウト領域にのみ使用され、およびディスクにデルタテーブルをキャッシュするために使用されます。クラスターがシャットダウンすると、NVMe ドライブ上のすべてのデータが失われます。

システムテーブルは永続化されます。システムテーブルの永続データユーザー、ロール、*などは、接続されたブロックストレージデバイスに保存され、一時停止/再開サイクル後も存続します。

サポートされているインスタンス

API Name	Memory	vCPUs	Cores	Storage
<i>m6id.large</i>	8.0 GiB	2 vCPUs	1	118 GB NVMe SSD
<i>m6id.xlarge</i>	16.0 GiB	4 vCPUs	2	237 GB NVMe SSD
<i>m6id.2xlarge</i>	32.0 GiB	8 vCPUs	4	474 GB NVMe SSD
<i>m6id.4xlarge</i>	64.0 GiB	16 vCPUs	8	950 GB NVMe SSD
<i>m6id.8xlarge</i>	128.0 GiB	32 vCPUs	16	1900 GB NVMe SSD
<i>m6id.12xlarge</i>	192.0 GiB	48 vCPUs	24	2850 GB (2 * 1425 GB NVMe SSD)
<i>m6id.16xlarge</i>	256.0 GiB	64 vCPUs	32	3800 GB (2 * 1900 GB NVMe SSD)
<i>m6id.24xlarge</i>	384.0 GiB	96 vCPUs	48	5700 GB (4 * 1425 GB NVMe SSD)
<i>m6id.32xlarge</i>	512.0 GiB	128 vCPUs	64	7600 GB (4 * 1900 GB NVMe SSD)

第3章 Use Anywhere

3.1 EDB Postgres Lakehouse Architecture

3.1.1 EDB Postgres Lakehouse とは

EDB Postgres Lakehouse は、PostgreSQL の分析機能を拡張してデータレイクストレージで直接動作し、従来のデータベースの信頼性と最新のオブジェクトストレージのスケラビリティとコスト効率を組み合わせます。このアーキテクチャにより、組織は、高価な分析データベースにデータを移動せずに、S3、Azure Blob Storage、または Google Cloud Storage に保存されているペタバイト規模のデータセットに対して複雑な SQL 分析を実行できます。

このページは、Analytics Hub の一部です。完全なナビゲーションについては、次をご覧ください [Analytics Hub — Analytics Accelerator Concepts — How-Tos \(Runbook-Aligned\)](#)

レイクハウスアプローチは、データウェアハウスとデータレイク間の従来の分離を排除します。アナリティクスアクセラレーター PGAA は、システム全体で重複データを維持するのではなく、Apache Iceberg や Delta Lake のようなオープンテーブルフォーマットを使用して、データが存在する場所を直接照会し、オブジェクトストレージでの ACID トランザクション、スキーマの適用、およびタイムトラベル機能を提供します。

3.1.2 レイクハウスアーキテクチャから恩恵を受ける人

データベース管理者は、レイクハウス機能を活用して、費用対効果の高い階層化ストレージ戦略を実装し、完全なクエリアクセスを維持しながら、高価な PostgreSQL ストレージからオブジェクトストレージに履歴データを自動的にオフロードします。

データエンジニアは、オブジェクトストレージに 1 回だけ書き込む簡素化されたデータパイプラインを構築し、運用システムと分析システム間の複雑な ETL プロセスを排除します。レイクハウスアーキテクチャは、統一されたテーブル形式を介してバッチとストリーミングワークロードの両方をサポートします。

データアナリストは、使い慣れた SQL インターフェイスを介してすべての組織データにアクセスし、基になるストレージの複雑性を理解せずに、オペレーションの PostgreSQL データと履歴レイクハウステーブルの両方を照会します。

プラットフォームチームは、マルチプルコンピューティングエンジンをサポートする統合分析アーキテクチャを実装します。Spark は複雑な変換を処理し、Analytics Accelerator はインタラクティブクエリを提供し、すべてが同じ基になるデータにアクセスします。

3.1.3 Lakehouse アーキテクチャを展開する方法

基本的な展開パターン

ハイブリッドマネージャーまたは自己管理インストールを介して専用の Lakehouse ノードを展開します。

```
- - Configure storage location for lakehouse data
SELECT pgfs.create_storage_location(
    lakehouse_storage,
    s3://analytics-bucket/lakehouse,
    {"region": "us-east-1"}
);

- - Create external table over Iceberg data
CREATE TABLE sales_history () USING PGAA
WITH (
    pgaa.format = iceberg,
    pgaa.storage_location = lakehouse_storage,
    pgaa.path = warehouse/sales.db/transactions
);

- - Query seamlessly across storage tiers
SELECT
    DATE_TRUNC(month, sale_date) as month,
    SUM(amount) as revenue
FROM sales_history
WHERE sale_date >= 2024-01-01
GROUP BY 1;
```

統合 OLTP + OLAP アーキテクチャ

運用ワークロードと分析ワークロードを単一のプラットフォームに結合します。

```
- - Hot data in PostgreSQL
CREATE TABLE sales_current (
    sale_id BIGINT PRIMARY KEY,
    sale_date DATE,
    amount DECIMAL(10,2)
);

- - Cold data in lakehouse
CREATE TABLE sales_archive () USING PGAA
WITH (
    pgaa.format = iceberg,
    pgaa.managed_by = lakehouse_catalog,
    pgaa.catalog_table = sales_historical
);
```

(次のページに続く)

(前のページからの続き)

```
- - Unified view across tiers
CREATE VIEW sales_complete AS
SELECT * FROM sales_current
UNION ALL
SELECT * FROM sales_archive;
```

3.1.4 アーキテクチャコンポーネント

ステートレスレイクハウスノード

Lakehouse ノードは、データをローカルに保存せずに、分析ワークロード用の専用の計算リソースを提供します。このステートレスデザインにより、データ量ではなくクエリ要求に基づいて柔軟なスケーリングが可能になります。

各ノードは、アナリティクスアクセラレーター拡張機能で強化された完全な PostgreSQL スタックを実行します。使い慣れた PostgreSQL インターフェイスは、既存のツールやアプリケーションとの互換性を保証し、基になるベクトル化エンジンが分析クエリ用に最適化されたパフォーマンスを提供します。

ノードは共有メタデータを介して自動的に調整され、マルチプルのインスタンスにわたる並列クエリの実行が可能になります。ロードバランサーは、利用可能なノード全体にクエリーを分散し、高可用性とホリゾンタルスケーラビリティの両方を提供します。

ベクトル化されたクエリエンジン

Apache DataFusion は、ベクトル化実行エンジンを強化し、列状データを行ごとではなくバッチで処理します。このアプローチにより、CPU キャッシュ使用率が最大化され、分析操作を劇的に高速化する SIMD 最適化が可能になります。

エンジンは、フィルタとプロジェクションをストレージレイヤーに自動的にプッシュし、データ転送を最小限に抑えます。パーティションプルーニングにより、読み取りが開始される前にデータファイル全体が検討対象から除外されます。これらの最適化を組み合わせることで、PostgreSQL の互換性を維持しながら、専用の分析データベースに迫るクエリパフォーマンスを提供します。

アダプティブクエリーの実行は、実行中に発見された実際のデータ特性に基づいて、処理戦略を調整します。初期統計は計画をガイドしますが、エンジンは、クエリの進行に応じて結合順序と集計戦略を動的に最適化します。

ストレージ抽象化レイヤー

PostgreSQL File System PGFS は、多様なオブジェクトストレージシステムへの統一アクセスを提供します。この抽象化レイヤーは、認証、プロトコルの違い、およびパフォーマンスの最適化を透過的に処理します。

このシステムは、マルチプルのストレージバックエンドを同時にサポートし、クラウドプロバイダーにまたがる、またはクラウドとオンプレミスのストレージを組み合わせたクエリを有効にします。インテリジェントなキャッシュにより、基になるテーブル形式の一貫性要件を尊重しながら、繰り返しのメタデータ操作が削減されます。

接続ブーリングと再試行ロジックは、クラウド環境で一般的な一時的なネットワーク問題を処理します。ストレージレイヤーは、観察されたレイテンシーとスループット特性に基づいて、要求パターンを自動的に調整します。

テーブルフォーマットの統合

Apache Iceberg と Delta Lake のネイティブサポートにより、より広範なデータエコシステムとの相互運用性が可能になります。Analytics Accelerator は、テーブルのメタデータを読み取り、手動構成を必要とせずスキーマ、パーティション、およびファイルの場所を理解します。

Iceberg テーブルの場合、システムは、スキーマの進化、隠しパーティショニング、タイムトラベルクエリを含む完全な機能セットをサポートしています。カタログの統合により、複数のコンピューティングエンジン全体での自動テーブル検出とメタデータの同期が可能になります。

デルタレイクのサポートは現在読み取り操作に焦点を当てており、既存のデータレイク投資へのアクセスを提供します。トランザクションログの解析は透過的に発生し、デルタテーブルを標準の PostgreSQL リレーションとして表示します。

3.1.5 パフォーマンス特性

クエリ最適化戦略

クエリオプティマイザーは、実行プランを生成するときにレイクハウス固有の特性を理解します。テーブル形式のメタデータからの統計は、結合順序と集計戦略をガイドします。オプティマイザーは、パーティションプルーニングの有効性とメタデータ操作のオーバーヘッドのバランスをとります。

述語プッシュダウンは、単純なフィルターを超えて、複雑な式と結合を含めます。システムはクエリを書き換えて、ストレージレイヤーで実行される操作を最大化し、データの移動と計算要件を削減します。

コストベースの最適化では、ネットワーク遅延、データの局所性、およびフォーマット固有の特性を考慮します。プランは、データがキャッシュ、リージョンストレージに存在するか、またはクロスリージョン転送が必要かどうかに適応します。

キャッシングアーキテクチャ

複数のキャッシュレイヤーは、分析ワークロードで一般的な繰り返しアクセスパターンを最適化します。メタデータのキャッシュにより、カタログ検索とスキーマ検出操作の繰り返しが不要になります。ファイルレベルのキャッシュは、頻繁にアクセスされる Parquet ファイルをローカル SSD に保存します。

クエリ結果のキャッシュは、同じクエリが繰り返し実行されるダッシュボードとレポートワークロードに対応します。キャッシュ無効化メカニズムはテーブルフォーマットのバージョン管理を尊重し、基になるデータが変更された場合でも一貫性を保証します。

適応型キャッシュ管理は、アクセスパターン、クエリコスト、および使用可能なリソースに基づいて、保持を優先します。ホットデータはキャッシュされたままですが、コールドデータはエージアウトし、最適なリソース使用率を維持します。

スケーラビリティの考慮事項

水平スケーリングは、ノードを追加して、実行中のワークロードを中断せずにクエリの同時実行性の増加を処理します。ステートレスアーキテクチャにより、新しいノードはすぐにクエリ処理能力に貢献します。

垂直スケーリングは、より多くのメモリまたは CPU を必要とする複雑な分析クエリに対して個々のノードリソースを調整します。システムは、利用可能なリソースに基づいて並列処理を自動的に調整し、リソースを枯渇させることなく使用率を最大化します。

ストレージのスケーリングはオブジェクトストレージを介して独立して発生し、事実上無制限のデータの増加をサポートします。コンピューティングとストレージを分離することにより、組織は比例するコンピューティングコストなしで長年の履歴データを保持できます。

3.1.6 統合パターン

階層化ストレージの実装

組織は、PostgreSQL 操作テーブルとレイクハウス履歴テーブルを組み合わせることにより、階層化ストレージを実装します。トランザクションの一貫性が必要な最近のデータは PostgreSQL に残りますが、古いデータはオブジェクトストレージに移行します。

自動移行ポリシーは、経過期間、アクセスパターン、またはビジネスルールに基づいてデータを移動します。移行プロセスは、データの整合性を維持しながら、行指向の PostgreSQL データを、分析クエリ用に最適化された列状の Parquet ファイルに変換します。

透過的なクエリルーティングは、述語に基づいてクエリを適切な層に転送します。時間ベースのクエリは、両方の層からの結果を自動的に結合し、手動クエリ変更なしで完全な分析ビューを提供します。

マルチエンジンアナリティクス

レイクハウスアーキテクチャは、さまざまなワークロードタイプに専用のコンピューティングエンジンをサポートします。Apache Spark は複雑な ETL 変換を処理し、アナリティクスアクセラレーターはインタラクティブ SQL クエリーを提供し、機械学習プラットフォームはトレーニングデータに直接アクセスします。

共有テーブルフォーマットにより、どのエンジンが変更を実行するかに関係なく、一貫性が保証されます。トランザクションの分離により、異なるエンジンからの同時操作間の競合が防止されます。この専門化により、組織は単一の真実の情報源を維持しながら、各ジョブに最適なツールを使用できます。

ストリーミング分析

最新のストリーミングプラットフォームは、マイクロバッチコミットを使用してレイクハウステーブルに継続的に書き込みます。Analytics Accelerator は、スナップショット分離を介してほぼリアルタイムの可視性を提供し、進行中の更新にもかかわらず一貫したクエリ結果を保証します。

Apache Kafka や Amazon Kinesis のようなイベントストリーミングシステムは、耐久性のあるストレージと分析処理のためにレイクハウステーブルにデータを配信します。ストリーミングインジェストと SQL 分析を組み合わせることで、リアルタイムの不正検出、操作モニタリング、顧客行動分析などのユースケースが可能になります。

3.1.7 運用上の考慮事項

導入モデル

ハイブリッドマネージャーを介した **管理展開**は、自動化されたプロビジョニング、スケーリング、およびメンテナンスを提供します。プラットフォームは、ノードのライフサイクル、セキュリティ更新、およびパフォーマンスの最適化を透過的に処理します。

自己管理展開は、インフラストラクチャと構成の完全な制御を提供します。特定のコンプライアンス要件または既存の Kubernetes インフラストラクチャを持つ組織は、多くの場合、自己管理型の展開を選択します。

ハイブリッド展開は、管理対象コントロールプレーンと自己管理データプレーンを組み合わせます。このアプローチは、操作の簡素化と規制対象業界で一般的なデータ主権要件のバランスをとります。

モニタリング要件

効果的なモニタリングは、クエリパフォーマンス、リソース使用率、およびストレージアクセスパターンを追跡します。クエリ実行時間、スキャンされたデータ、およびパーティションブルーニングの有効性は、最適化の機会を示します。

ストレージメトリックは、パーティショニング戦略とキャッシュポリシーを通知するアクセスパターンを明らかにします。データアクセスのホットスポットは、マテリアライズドビューまたは代替パーティションスキームの候補を示唆します。

リソース使用率メトリックは、キャパシティプランニングとスケーリングの決定をガイドします。CPU、メモリ、およびネットワークの使用率パターンは、ワークロードがコンピューティング、メモリ、または I/O バウンドであるかどうかを示します。

セキュリティアーキテクチャ

レイクハウスセキュリティモデルは、マルチプルのレイヤーにわたって多層防御を実装します。PostgreSQL のロールベースのアクセス制御はユーザー権限を管理し、オブジェクトストレージ IAM ポリシーはデータアクセスを制御します。

暗号化は、クラウドプロバイダーの暗号化サービスまたはカスタマー管理キーを使用して、保存データを保護します。トランスポートレイヤーセキュリティは、計算ノードとストレージシステム間で移動しているデータを暗号化します。

監査ログは、コンプライアンスとセキュリティ分析のためのすべてのデータアクセスを追跡します。クエリログ、ストレージアクセスログ、および認証イベントは、システムの使用状況を完全に可視化します。

3.2 Apache Iceberg Integration With Analytics Accelerator

用語 Hybrid ManagerHM は、環境全体で Postgres、分析、AI をプロビジョニングおよび操作するための EDB のコントロールプレーンです。アナリティクスアクセラレータ PGAA は、レイクハウス、階層化、およびベクトル化クエリ機能を使用して Postgres を拡張する EDB の分析機能を指します。

3.2.1 Apache Iceberg とは

Apache Iceberg は、オブジェクトストレージに構築されたデータレイクにデータベースのような信頼性をもたらすオープンテーブル形式です。従来のファイルベースのアプローチとは異なり、Iceberg は、Parquet や ORC などの標準フォーマットとの互換性を維持しながら、ACID トランザクション、スキーマの進化、およびタイムトラベル機能を提供します。Analytics Accelerator PGAA は、Iceberg を活用して、オブジェクトストレージからデータを移動せずに、ペタバイト規模のデータレイクに対する PostgreSQL クエリーを有効にします。

このページは、Analytics Hub の一部です。完全なナビゲーションについては、次をご覧ください [Analytics Hub — Analytics Accelerator Concepts — How-Tos \(Runbook-Aligned\)](#)

3.2.2 Iceberg の統合を使用する人

データエンジニアは、スキーマの進化と同時の読み取り/書き込み操作を必要とする信頼性の高いデータパイプラインのために Iceberg テーブルを実装します。彼らは、クエリの一貫性を維持しながらストリーミング取り込みを処理する Iceberg の機能を評価しています。

データアナリストは、使い慣れた SQL 構文を使用して、アナリティクスアクセラレーターを介して Iceberg テーブルを照会し、基になるストレージの複雑さを理解せずにタイムトラベルクエリを介して履歴データにアクセスします。

プラットフォームチームは、Spark、Trino、および PostgreSQL が同じデータセットに一貫してアクセスする必要があるマルチエンジン分析環境の基盤として Iceberg を展開します。

データサイエンティストは、Iceberg のバージョンニング機能を活用して実験を再現し、特定の時点でトレーニングデータにアクセスし、モデルの再現性を保証します。

3.2.3 アナリティクスアクセラレーターで Iceberg を使用方法

基本的なセットアップ

Analytics Accelerator を Iceberg カタログに接続して、既存のテーブルを検出および照会します。

```
- - Add Iceberg catalog
SELECT pgaa.add_catalog(
  data_lake,
  iceberg-rest,
  {"url": "https://catalog.company.com",
   "warehouse": "analytics",
   "token": "auth_token"}
);

- - Attach catalog for querying
SELECT pgaa.attach_catalog(data_lake);

- - Query Iceberg tables directly
SELECT * FROM data_lake.sales.transactions
WHERE transaction_date >= 2024-01-01;
```

外部テーブルの作成

Iceberg データを参照する PostgreSQL テーブルを定義します。

```
CREATE TABLE customer_events () USING PGAA
WITH (
  pgaa.format = iceberg,
  pgaa.managed_by = data_lake,
  pgaa.catalog_namespace = events,
  pgaa.catalog_table = customer_activity
);
```

タイムトラベルクエリ

別のコピーを維持せずに履歴データにアクセスします。

```
- - Query data as of specific timestamp
SELECT COUNT(*) FROM sales_facts
FOR SYSTEM_TIME AS OF 2024-12-31 23:59:59;

- - Compare current with historical state
SELECT
  current.product_id,
  current.price as current_price,
  historical.price as year_end_price
FROM sales_facts current
JOIN sales_facts FOR SYSTEM_TIME AS OF 2024-12-31 historical
  ON current.product_id = historical.product_id
WHERE current.price != historical.price;
```

3.2.4 Iceberg アーキテクチャを理解する

メタデータ構造

Iceberg は、大規模なパフォーマンスを維持しながら高度な機能を有効にする 3 層メタデータアーキテクチャを実装しています。

メタデータファイルは、スキーマ、パーティショニング、スナップショット履歴を含む現在のテーブルの状態を追跡します。テーブルを変更するたびに、新しいメタデータファイルが作成され、タイムトラベル用に以前のバージョンが保存されます。

マニフェストリストは、パーティションブローニングの統計を使用してデータファイルのコレクションを編成します。アナリティクスアクセラレーターはこれらの統計を使用して、クエリの実行が開始される前に不必要なファイルのスキャンを排除します。

マニフェストファイルには、個々のデータファイルの列レベルの統計が含まれています。これらの統計により、述語のプッシュダウンが有効になり、アナリティクスアクセラレーターは、一致する行を含めることができないファイルをスキップできます。

隠しパーティション

Iceberg は、非表示のパーティショニングを介してユーザーからパーティションの複雑さを抽象化します。クエリーは自然な列値を使用しますが、Iceberg はこれらをパーティションフィルターに自動的に変換します。

```
- - User writes simple query
SELECT * FROM orders WHERE order_date = 2024-03-15;

- - Iceberg automatically prunes to day=2024-03-15 partition
- - without user knowing partitioning scheme
```

この抽象化により、クエリーを変更せずにパーティションの進化が可能になります。組織は、既存のアプリケーションを中断せずに、データ量の増加に応じてパーティショニング戦略を調整できます。

トランザクションモデル

Iceberg は、スナップショット分離を使用してオプティミスティック同時実行制御を実装します。ライターは、アトミックに可視になる新しいスナップショットを作成し、リーダーが同時変更にもかかわらず一貫したテーブル状態を観察できるようにします。

Analytics Accelerator は、このモデルを活用して、反復可能なクエリー結果を提供します。ストリーミングシステムが新しいデータを書き込む場合でも、長時間実行される分析クエリーはスナップショットにアクセスし続け、読み取り-書き込みの競合を排除します。

3.2.5 スキーマ進化機能

Iceberg は、アナリティクスアクセラレーターの標準 SQL コマンドを介して公開される、データを書き換えることのない包括的なスキーマの変更をサポートしています。

安全なスキーマの変更

データファイルに手を加えずに列を追加、名前変更、またはドロップします。

```
- - Add column (instant operation)
ALTER TABLE customer_events ADD COLUMN region VARCHAR;

- - Rename column (metadata only)
ALTER TABLE customer_events RENAME COLUMN usr_id TO user_id;

- - Drop column (marks as deleted, data retained)
ALTER TABLE customer_events DROP COLUMN deprecated_field;
```

タイププロモーション

要件の進化に応じて、データ型を安全に拡張します。

```
- - Promote int to bigint
ALTER TABLE transactions ALTER COLUMN amount TYPE BIGINT;

- - Promote float to double
ALTER TABLE metrics ALTER COLUMN value TYPE DOUBLE PRECISION;
```

3.2.6 カタログ統合パターン

Analytics Accelerator は、それぞれが異なる運用要件に適した複数の Iceberg カタログ実装をサポートしています。

REST カタログ

REST カタログは、HTTP プロトコルを介してベンダーニュートラルな統合を提供します。AWS Glue、Tabular、およびカスタム実装は、標準化された REST API を介して Iceberg テーブルを公開します。

```
SELECT pgaa.add_catalog(
  aws_catalog,
  iceberg-rest,
  [{"url": "https://glue.us-east-1.amazonaws.com",
    "warehouse": "production",
    "credential": "aws_iam"}]
);
```

AWS S3 テーブル

AWS S3 テーブルとのネイティブ統合により、サーバーレスの Iceberg カタログ機能が提供されます。

```
SELECT pgaa.add_catalog(
  s3_tables,
  iceberg-s3tables,
  [{"arn": "arn:aws:s3tables:us-east-1:123456:bucket/analytics",
    "region": "us-east-1"}]
);
```

プロジェクトネッシー

Nessie は、Git のような分岐を Iceberg テーブルに追加し、実験的分析を可能にします。

```
- - Query production branch
SELECT * FROM catalog.main.sales_facts;
```

(次のページに続く)

(前のページからの続き)

```
- - Query experimental branch
SELECT * FROM catalog.experiment_branch.sales_facts;
```

3.2.7 パフォーマンス最適化戦略

ファイル構成

最適なファイルサイズにより、並列処理と I/O 効率のバランスが取れます。通常の圧縮を介して 128～512 MB の Parquet ファイルをターゲットにします。

```
- - Compact small files for better performance
CALL pgaa.compact_table(catalog.schema.table,
  target_file_size => 256MB);
```

ソートの最適化

ファイル内のデータを並べ替えると、圧縮が向上し、効率的なスキップが可能になります。

```
- - Z-order by frequently filtered columns
CALL pgaa.zorder_table(catalog.schema.table,
  columns => ARRAY[customer_id, order_date]);
```

メタデータ管理

定期的なメンテナンスにより、メタデータの効率が維持されます。

```
- - Expire old snapshots
CALL pgaa.expire_snapshots(catalog.schema.table,
  older_than => CURRENT_TIMESTAMP - INTERVAL 30 days);

- - Remove orphan files
CALL pgaa.remove_orphan_files(catalog.schema.table);
```

3.2.8 一般的な統合パターン

階層化ストレージアーキテクチャ

PostgreSQL の運用データと Iceberg の履歴データを組み合わせます。

```
- - Unified view across tiers
CREATE VIEW sales_unified AS
SELECT * FROM postgres.sales_current
UNION ALL
SELECT * FROM iceberg.sales_historical;
```

マルチエンジンプロセッシング

Iceberg のオープンフォーマットにより、ツールの特化が可能になります。

- **Apache Spark** 複雑な ETL 変換
- **アナリティクスアクセラレーター** 対話型 SQL クエリー
- **Trino** システム全体のフェデレーションクエリー
- **Flink** ストリーム処理とリアルタイム分析

ストリーミングデータの統合

一貫したスナップショットを使用してストリーミングデータを処理します。

```
- - Query latest streaming data
SELECT COUNT(*) FROM events
WHERE event_time >= CURRENT_TIMESTAMP - INTERVAL 1 hour;

- - Streaming platforms write continuously
- - Analytics Accelerator queries see consistent snapshots
```

3.2.9 運用上の考慮事項

モニタリング要件

最適なパフォーマンスのために主要なメトリックを追跡します。

- **スナップショット数** メタデータの増加を示します
- **パーティションごとのファイル数** 圧縮ニーズを特定します
- **メタデータ操作のレイテンシー** カタログパフォーマンスを明らかにします
- **クエリブルーニングの有効性** パーティショニング戦略を検証します

キャパシティプランニング

Iceberg はストレージをコンピューティングから分離し、独立したスケーリングを有効にします。

- **ストレージ** データ量とスナップショットの保持に応じて増加します
- **コンピューティング** 同時クエリの負荷に応じてスケーリング
- **メタデータ** テーブルの進化に応じて定期的なメンテナンスが必要

移行戦略

Iceberg を段階的に採用して、リスクを最小限に抑えます。

1. **パイロットフェーズ** 検証のために非クリティカルデータセットを変換する
2. **プロダクション移行** 高価値の分析テーブルを移動する

3. 完全な採用 すべての分析データを Iceberg で標準化する

3.3 Delta Lake Integration with Analytics Accelerator

3.3.1 レイクハウスのコンテキストでのデルタ湖の理解

デルタレイクは、アナリティクスアクセラレーター PGAA のレイクハウスアーキテクチャ内の基礎的なテーブルフォーマットとして機能し、大規模な分析ワークロードに ACID トランザクションとタイムトラベル機能を提供します。この統合により、PostgreSQL はオブジェクトストレージシステムに保存されている Delta Lake テーブルを直接照会できるため、運用データベースと分析システム間の従来の ETL パイプラインが不要になります。

このページは、Analytics Hub の一部です。完全なナビゲーションについては、次をご覧ください [Analytics Hub — Analytics Accelerator Concepts — How-Tos \(Runbook-Aligned\)](#)

Analytics Accelerator の Lakehouse Connector は、ネイティブの Delta Lake 読み取り機能を実装し、組織はデータを移行せずに既存の Delta Lake 投資を活用できます。このアプローチでは、より広範な Spark エコシステムとの互換性を維持しながら、分析クエリ用に PostgreSQL の使い慣れた SQL インターフェイスを提供します。

3.3.2 アーキテクチャの概要

Analytics Accelerator アーキテクチャは、Lakehouse Connector コンポーネントを介してアクセスできる、オープンレイクハウスエコシステム内に Delta Lake テーブルを配置します。このコネクタは、PostgreSQL クエリを Delta Lake の Parquet ファイルとトランザクションログに対する最適化された操作に変換します。

システムアーキテクチャはコンピューティングをストレージから分離し、ワークロード要件に基づいて独立したスケーリングを可能にします。クエリ処理は、Delta Lake のストレージ形式用に特別に設計された列指向データの最適化を実装する分析エンジン内で発生します。この分離により、従来の行ベースの処理と比較して、クエリパフォーマンスの大幅な向上を実現しながら、完全な SQL 互換性を維持します。

3.3.3 デルタレイクテーブルフォーマット

Delta Lake は、`_delta_log` ディレクトリに保存される付随トランザクションログを含む Parquet ファイルとしてデータを構造化します。このトランザクションログは、すべてのテーブル変更の完全な履歴を維持し、規制コンプライアンスに不可欠なポイントインタイムクエリと監査機能を有効にします。

このフォーマットはオプティミスティック同時実行制御を実装し、ライターが変更を実行しているときに複数のリーダーがデータにアクセスできます。このデザインにより、データの更新中に分析ワークロードをブロックすることなく、一貫したクエリ結果が保証されます。Analytics Accelerator は、これらの一貫性保証を活用して、基になるデータが変更された場合でも、信頼性の高いクエリ結果を提供します。

Delta Lake テーブル内のファイル構成は、クエリパターンに合わせた構成可能なパーティションスキームに従います。一般的なパーティショニング戦略には、時系列データの日付ベースのパーティショニングまたはリージョン分析の地理的パーティショニングが含まれます。Analytics Accelerator のクエリオプティマイザーは、関連するパーティションを自動的に特定し、データのスキャンを大幅に削減します。

3.3.4 統合機能

Analytics Accelerator は、Delta Lake テーブルとの双方向統合を提供しますが、現在の運用展開は主に読み取り操作に焦点を当てています。Lakehouse Connector は、メタデータの同期、スキーマ検出、およびクエリ変換を透過的に処理します。

読み取り操作

Delta Lake テーブルに対するクエリの実行は、標準の PostgreSQL SQL 構文を介して発生します。システムは Delta Lake メタデータからテーブルスキーマを自動的に検出し、手動のスキーマ定義の必要性を排除します。列統計とパーティション情報はクエリの最適化をガイドし、効率的な実行プランを保証します。

タイムトラベルクエリは、Delta Lake のトランザクションログを使用して履歴テーブルバージョンにアクセスします。この機能は、規制報告、データ復旧、および変更分析にとって非常に貴重であることがわかります。組織は、別の履歴コピーを維持することなく、任意の時点で存在したデータを照会できます。

書き込み操作

Analytics Accelerator の主な焦点は引き続き読み取りの最適化ですが、アーキテクチャは Delta Lake テーブルへの将来の書き込み機能をサポートします。現在の実装では、通常、操作上の書き込みに EDB PostgreSQL 分散 PGD を使用し、分析処理のために Delta Lake への定期的な同期を行います。

3.3.5 実際のアプリケーション

統合分析プラットフォーム

組織は Delta Lake でアナリティクスアクセラレーターを実装して、業務データと履歴データにまたがる統合分析プラットフォームを作成します。リアルタイムの操作データは PostgreSQL にありますが、履歴データは Delta Lake テーブルとして費用対効果の高いオブジェクトストレージに存在します。クエリは両方のデータソースに透過的にアクセスし、データを移動せずに完全な分析ビューを提供します。

金融サービス組織は、リスク分析にこのパターンを使用し、PostgreSQL の現在の取引ポジションと Delta Lake の長年の履歴市場データを組み合わせます。統合クエリインターフェイスにより、フェデレーションクエリシステムの複雑さなしで、複数の期間にまたがる複雑なリスク計算が可能になります。

データレイクのモダナイゼーション

未加工のファイル形式に基づいて構築された既存のデータレイクは、Delta Lake のトランザクションの一貫性とスキーマの適用の恩恵を受けます。Analytics Accelerator は、これらの最新のデータレイクのクエリレイヤーを提供し、従来のデータベースに精通したビジネスアナリストの SQL アクセスを可能にします。

小売組織は、生のイベントログを Delta Lake 形式に変換することにより、顧客分析を最新化します。Analytics Accelerator により、マーケティングチームは、専門的なビッグデータのスキルを必要とするのではなく、使い慣れた SQL を使用して顧客の行動パターンを照会できます。

ストリーミング分析

デルタレイクのストリーミングインジェストのサポートとアナリティクスアクセラレーターのクエリ機能との組み合わせにより、ストリーミングデータのほぼリアルタイムの分析が可能になります。イベントストリームは Delta Lake に継続的に書き込みますが、アナリティクスアクセラレーターは、進行中の更新にもかかわらず、一貫したクエリ結果を提供します。

製造会社は、センサーデータが Delta Lake テーブルにストリーミングされる生産ラインモニタリングにこのパターンを実装します。品質管理チームは、アナリティクスアクセラレーターを介して最近の生産メトリックを照会し、履歴分析は長期的な傾向とパターンを特定します。

3.3.6 パフォーマンスの最適化

デルタレイクの効果的な統合には、フォーマットに固有のパフォーマンス特性と最適化戦略を理解する必要があります。

Z オーダーの最適化

Delta Lake は、Parquet ファイル内の関連データを同じ場所に配置する技術である Z オーダーをサポートしています。この最適化により、Z オーダー列のフィルターのクエリパフォーマンスが劇的に向上します。Analytics Accelerator のクエリプランナーは、Z オーダーの列を認識し、データの局所性の利点を最大化する実行プランを生成します。

組織は通常、WHERE 句で頻繁に使用される高カーディナリティ列を Z オーダーします。トランザクションテーブルの顧客 ID またはイベントログのタイムスタンプフィールドは、クエリ時間を 50~80%削減できる一般的な Z オーダー候補を表します。

ファイル圧縮

ストリーミングインジェストによって作成された小さなファイルは、メタデータのオーバーヘッドを発生させ、読み取り効率を低下させます。通常の圧縮は、これらのファイルを最適なサイズのユニットに統合します。Delta Lake テーブルが 128~512 MB のファイルを維持する場合、Analytics Accelerator のパフォーマンスが大幅に向上します。

圧縮プロセスはバックグラウンドメンテナンスタスクとして実行され、通常は使用率の低い期間にスケジュールされます。組織は、ほとんどの分析ワークロードに適した毎日のコンパクションを使用して、圧縮頻度と書き込みワークロード要件のバランスをとります。

キャッシュ戦略

アナリティクスアクセラレーターは、Delta Lake クエリの複数のキャッシュレイヤーを実装します。メタデータキャッシュは、反復的なトランザクションログ解析を排除しますが、データキャッシュは、頻繁にアクセスされる Parquet ファイルをローカルに保存します。これらのキャッシュは透過的に動作し、Delta Lake テーブルが更新されると自動的に無効になります。

マテリアライズドビューは、複雑な集計用の追加キャッシュレイヤーを提供します。マテリアライズドビューは、大規模な Delta Lake テーブルを繰り返しスキャンするのではなく、事前計算された結果を PostgreSQL に保存します。このアプローチは、何百万もの行を概要統計に集計するダッシュボードクエリに特に適しています。

3.3.7 移行の考慮事項

既存のデータウェアハウスまたはデータレイク実装から Delta Lake に移行する組織は、主要なアーキテクチャの違いと移行戦略を理解する必要があります。

従来のデータウェアハウスから

データウェアハウスの移行は、通常、既存のシステムで現在の運用データを維持しながら、デルタレイクへの履歴データのアーカイブから始まります。移行期間中にアナリティクスアクセラレータークエリーは両方のシステムにまたがり、既存のプロセスを中断せずにユニファイドアクセスを提供します。

移行プロセスは、アナリティクスアクセラレーターの互換性レイヤーを介して既存の SQL クエリーを保持します。クエリライトは、ほとんどの標準の SQL 操作で自動的に発生しますが、複雑なウェアハウス固有の機能は変更が必要な場合があります。

Raw Data Lakes から

トランザクションの一貫性を欠く生ファイル形式は、Delta Lake 変換から大きな利益をもたらします。変換プロセスは、可能な場合は既存の Parquet データを保存しながら、トランザクションログとスキーマ適用を追加します。Analytics Accelerator は、変換されたテーブルへの SQL アクセスをすぐに有効にし、組織全体のデータアクセスを民主化します。

組織は通常、最初に高価値のデータセットを変換し、広範な移行を続行する前にクエリパフォーマンスとデータ品質を検証します。このインクリメンタルなアプローチにより、リスクを最小限に抑えながら、データの一貫性とクエリ機能の向上から即時の価値を提供します。

3.3.8 エコシステムの統合

Analytics Accelerator の Delta Lake サポートは、より広範なレイクハウスエコシステムとの互換性を維持し、最新のデータアーキテクチャに不可欠なマルチエンジンアクセスパターンを有効にします。

Apache Spark は、データエンジニアリングのワークロードを提供し続け、変換されたデータを Delta Lake テーブルに書き込み、Analytics Accelerator が分析処理のために照会します。この責任の分割により、各エンジンの強みを活用しながら、Delta Lake での単一の真実のソースを維持します。

ビジネスインテリジェンスツールは、標準の PostgreSQL ドライバーを介して Analytics Accelerator に接続し、専用のコネクタを使用せずに Delta Lake データにアクセスします。この互換性、既存のレポートインフラストラクチャへの投資を維持しながら、分析機能をレイクハウスデータに拡張します。

機械学習プラットフォームは、Delta Lake テーブルからトレーニングデータを直接読み取り、モデルトレーニングと分析レポートの一貫性を保証します。Analytics Accelerator は、SQL クエリを介してデータ検証と品質チェックを提供し、モデルのトレーニングを開始する前にデータの問題を特定します。

第4章 Use With PGD

4.1 Tiered Tables With Analytics Accelerator

4.1.1 階層テーブルとは

階層テーブルは、すべてのデータに対する透過的なクエリアクセスを維持しながら、履歴パーティションを高価なトランザクションストレージから費用対効果の高いオブジェクトストレージに移動することにより、データのライフサイクルを自動的に管理します。この機能は、EDB Postgres 分散 PGD パーティショニングと Analytics Accelerator PGAA レイクハウス統合を組み合わせ、増大するデータセットのパフォーマンスとコストの両方を最適化します。

このページは、Analytics Hub の一部です。完全なナビゲーションについては、次をご覧ください [Analytics Hub — Analytics Accelerator Concepts — How-Tos \(Runbook-Aligned\)](#)

システムは、トランザクションパフォーマンスを必要とする「ホット」データと、主に分析またはコンプライアンス目的でアクセスされる「コールド」データを区別します。ホットデータは PostgreSQL に残り、最適な OLTP パフォーマンスを実現し、コールドデータはオブジェクトストレージの Apache Iceberg または Delta Lake 形式に移動し、一般的な時系列ワークロードのトランザクションデータベースサイズを 70～90%削減します。

4.1.2 階層テーブルの恩恵を受ける人

データベース管理者は、階層化ストレージを実装して、データのアクセシビリティを犠牲にすることなくデータベースの成長を制御します。リーンでパフォーマンスの高いトランザクションシステムを維持しながら、完全なデータ保持を通じて規制コンプライアンスを保証します。

金融サービスチームは、規制要件ごとに 7～10 年間保持する必要がある取引履歴の階層化を使用しますが、最初の決済期間後にはほとんどアクセスされません。

IoT プラットフォームオペレーターは、継続的に到着するセンサーデータを管理しますが、数日または数週間後に運用の関連性を失い、分析ワークロードに自然に移行します。

小売組織の階層注文履歴。最近の注文はカスタマーサービスにリアルタイムのアクセスを必要とし、履歴データは傾向分析と予測に役立ちます。

4.1.3 階層化テーブルの仕組み

基本構成

BDR AutoPartition を使用して時間パーティションテーブルで階層化ストレージを有効にし、パーティションが存在すると分析オフロードを構成します。これは、Runbook フローを反映しています。

パーティションテーブルを作成し、AutoPartition を初期化します。オフロードはまだありません。

```

- - Create a time-partitioned table
CREATE TABLE partitioned_table (
  key BIGINT,
  inserted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  revenue DOUBLE PRECISION,
  comment TEXT,
  PRIMARY KEY (key, inserted_at)
) PARTITION BY RANGE(inserted_at);

- - Create yearly partitions (no offload yet)
SELECT bdr.autopartition (
  relation := partitioned_table,
  partition_increment := 1 year,
  partition_initial_lowerbound := 2010-01-01 00:00:00,
  managed_locally := TRUE
);

- - Wait for partitions to be created across nodes
SELECT bdr.autopartition_wait_for_partitions_on_all_nodes(partitioned_table::regclass,
  → CURRENT_TIMESTAMP::text);

```

データを設定し、コールドパーティションのオフロードを有効にします。

```

- - Insert sample data
INSERT INTO partitioned_table (
  SELECT
    g,
    2010-01-01 00:00:00::timestamp + INTERVAL 1 second *
      (EXTRACT (EPOCH FROM (CURRENT_TIMESTAMP - 2010-01-01 00:00:00::timestamp)) /
  → 1000000 * g),
    g * 0.001,
    encode(sha256(g::text::bytea), hex)
  FROM generate_series(1, 1000000) g
);

- - Reconfigure AutoPartition to offload partitions older than 1 year
SELECT bdr.autopartition (
  relation := partitioned_table,
  partition_increment := 1 year,
  partition_initial_lowerbound := 2010-01-01 00:00:00,
  managed_locally := TRUE,
  analytics_offload_period := 1 year
);

```

オフロードパターン

AutoPartition は、時間ベースの階層化を自動的に処理します。テーブル全体の手動/カタログなしオフロードには、分析レプリケーションを使用します。

```
- - Example: offload an entire transactional table (no catalog)
CREATE TABLE transactional_table (
  key BIGINT,
  inserted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  revenue DOUBLE PRECISION,
  comment TEXT,
  PRIMARY KEY (key, inserted_at)
);

- - Enable replication to analytics storage
ALTER TABLE transactional_table SET (pgd.replicate_to_analytics = TRUE);

- - Optionally convert to PGAA to read directly from Iceberg
ALTER TABLE transactional_table
SET ACCESS METHOD PGAA,
SET (
  pgaa.auto_truncate = true,
  pgaa.format = iceberg,
  pgaa.storage_location = appliance-bucket,
  pgaa.path = public.transactional_table
);
```

部分オフロードは、データを別のテーブルに分割し、そのテーブルのみをオフロードすることにより実現できます。

```
BEGIN;
CREATE TABLE transactional_table_before_2020 AS
  SELECT * FROM transactional_table
  WHERE inserted_at < 2020-01-01;
DELETE FROM transactional_table WHERE inserted_at < 2020-01-01;
COMMIT;

ALTER TABLE transactional_table_before_2020 ADD PRIMARY KEY (key, inserted_at);
ALTER TABLE transactional_table_before_2020 SET (pgd.replicate_to_analytics = TRUE);

- - (Optional) convert to PGAA for direct Iceberg reads
ALTER TABLE transactional_table_before_2020
SET ACCESS METHOD PGAA,
SET (
  pgaa.auto_truncate = true,
  pgaa.format = iceberg,
  pgaa.storage_location = appliance-bucket,
```

(次のページに続く)

```
pgaa.path = public.transactional_table_before_2020  
);
```

復元オペレーション

テーブルをトランザクションストレージに戻し、分析レプリケーションを無効にします。

```
ALTER TABLE transactional_table  
SET ACCESS METHOD heap,  
SET (pgd.replicate_to_analytics = FALSE);
```

4.1.4 アーキテクチャの概要

ストレージ階層

このアーキテクチャは、さまざまなアクセスパターンに最適化された3つの異なるストレージ階層を実装しています。

ホットティア PostgreSQL は、ミリ秒未満のレイテンシーと完全なトランザクション機能を必要とする現在の操作データを保存します。この層には、ワークロードの特性に応じて、数日から数か月の最近のデータが含まれています。

コールド層オブジェクトストレージは、分析クエリ用に最適化された列指向形式で履歴データを保持します。データは完全に照会可能なままですが、より高いレイテンシーはコンプライアンスおよび分析のユーザースペースで許容されます。

クエリ処理

階層テーブルは、PostgreSQL のパーティションごとのクエリプランニングを活用して、クロス層クエリを最適化します。プランナーは述語を分析して、アクセスが必要なパーティションを判断し、無関係なホットおよびコールドパーティションを自動的にプルーニングします。

クエリが階層にまたがる場合、システムは各ストレージレイヤーに対して並列サブクエリを実行します。ホット層クエリは標準の PostgreSQL 処理を介して実行されますが、コールド層クエリは Analytics Accelerator のベクトル化エンジンを利用します。結果は透過的にマージされ、統合されたデータセットがアプリケーションに表示されます。

時間ベースの述語は、パーティション境界に自然に整列するため、特に効率的であることがわかります。前四半期のデータのクエリは、それぞれがストレージ特性に合わせて最適に処理される1つのホットパーティションと2つのコールドパーティションにアクセスする場合があります。

メタデータ管理

システムは、パーティションの場所、オフロードタイムスタンプ、およびストレージ統計を追跡する包括的なメタデータを維持します。このメタデータは、インテリジェントなクエリルーティングを有効にし、ストレージ使用率の可視性を提供します。

PGD は、アクティブ、オフロード中、オフロード、復元を含むパーティションのライフサイクル状態を追跡します。状態遷移はアトミックに発生し、クラスターノード全体で一貫した可視性を保証します。失敗したオフロード操作は自動的に再試行またはロールバックし、データの整合性を維持します。

ストレージ統計は、オフロードによって達成されたサイズの削減を追跡し、コスト削減の定量化に役立ちます。圧縮率は、データの特徴と選択したカラムナー形式に応じて、通常 3:1~10:1 の範囲です。

4.1.5 パフォーマンス特性

ホットティアの最適化

トランザクションストレージから履歴データを削除すると、すぐにパフォーマンスの向上が得られます。テーブルが小さいほど、インデックスの効率が向上し、バキューム操作が高速になり、キャッシュヒット率が向上します。組織は通常、階層化を実装した後、トランザクションクエリパフォーマンスの 20~40 の向上を観察します。

データセットサイズの削減は、バックアップおよびリカバリー操作に特にメリットがあります。データベースが小さいほど、バックアップが速く、必要なストレージが少なく、ディザスターリカバリーシナリオでの復元がより迅速に行われます。

コールド層のパフォーマンス

コールドデータに対する分析クエリーは、圧縮、辞書エンコード、ベクトル化処理を含むカラム型ストレージの最適化を活用します。個々の行検索の実行は PostgreSQL よりも遅くなりますが、分析集計は、多くの場合、カラム指向の効率のため、より速く実行されます。

パーティションのマージは、何千もの履歴パーティションを処理する際のメタデータのオーバーヘッドに対処します。複数の日次パーティションは、オブジェクトストレージの月次または年次パーティションにマージできるため、クエリ機能を維持しながらメタデータ操作を削減できます。

クロスティアクエリー

ホット層とコールド層にまたがるクエリーは、データ分散に基づいたパフォーマンス特性を示します。最小限のコールドデータフィルタリングで主にホットデータにアクセスするクエリーは、PostgreSQL に近いパフォーマンスを維持します。長年のコールドデータと最近のホットデータを集計する分析クエリは、階層をまたがる並列処理のメリットをもたらします。

システムは、頻繁にアクセスされるコールドパーティションのインテリジェントなキャッシュを実装します。メタデータキャッシュは、繰り返しの検出操作を排除し、データキャッシュは、よくアクセスされる Parquet ファイルをローカルに保存します。

4.1.6 一般的な実装パターン

規制コンプライアンス

金融サービス組織は、階層化を実装して、規制の保持要件をコスト効率よく満たします。たとえば、トランザクションデータは操作クエリをサポートして 90 日間 PostgreSQL に残り、7 年間の保持のために Iceberg に自動的にオフロードできます。

実装は、すべての層の移行のタイムスタンプ追跡を使用して完全な監査証跡を維持します。コンプライアンスクエリは履歴データに透過的にアクセスし、手動データ組み立てなしで保持期間全体のレポートを生成します。

時系列分析

毎日数百万のセンサーの読み取りを受信する IoT プラットフォームは、積極的な階層化ポリシーを実装しています。7 日より古い生のセンサーデータはオブジェクトストレージにオフロードされますが、集計されたメトリックは PostgreSQL に無期限に残ります。

このパターンは、操作ダッシュボードが最近の生データを表示し、履歴分析が集計された概要を使用して動作する一般的なクエリパターンを最適化します。階層化境界は自然なアクセスパターンと一致し、階層間のクエリを最小限に抑えます。

季節的な景気循環

強い季節パターンを持つ小売組織は、動的な階層化ポリシーを実装します。ピークシーズンの注文データはより長くホットなままで、Returns 期間の延長と顧客サービスのニーズをサポートします。

オフシーズンデータは 30 日後に積極的にオフロードされますが、ピークシーズンデータは 90 日間のホットリテンションを保持します。この適応的なアプローチにより、年間を介してストレージコストとビジネス要件のバランスがとれます。

4.1.7 運用上の考慮事項

モニタリング要件

効果的な階層化には、パーティションの成長率、オフロード成功率、および階層全体のクエリパターンを監視する必要があります。ダッシュボードはストレージ使用傾向を追跡し、階層化からメリットを受けるテーブルを特定し、コスト削減を検証します。

クエリパフォーマンスモニタリングは、階層ローカルクエリとクロス階層クエリを区別します。クロス層のクエリ頻度の増加は、調整されたリテンションポリシーまたはマテリアライズドビュー層をブリッジする必要があることを示している可能性があります。

キャパシティプランニング

階層アーキテクチャでは、両方のストレージ階層の容量計画が必要です。ホット層のキャパシティは、データの取り込み率と保持期間によって異なります。コールド層の容量は継続的に増加しますが、オフロードポリシーに基づいて予測可能な速度で増加します。

層間のネットワーク帯域幅は、オフロードとリストア操作に影響を与えます。組織は通常、メンテナンス期間中にオフロード操作をスケジュールして、運用ワークロードへの影響を最小限に抑えます。

メンテナンス操作

定期的なメンテナンスには、コールドストレージでのパーティションのマージ、孤立したファイルのクリーンアップ、メタデータの最適化が含まれます。これらの操作は通常、アクティビティの少ない期間にスケジュールされたジョブとして実行されます。

バックアップ戦略は、階層化されたデータ分散を考慮する必要があります。ホット層バックアップはオペレーションリカバリーのために頻繁に発生しますが、コールド層バックアップはオブジェクトストレージの耐久性保証に依存する場合があります。

可逆性の保証

すべてのオフロード操作は、リストア操作を介したリバーサルをサポートしています。この可逆性により、運用要件の変更に対応できることがわかり、階層化戦略を実装する際に自信が得られます。

システムはオフロード履歴を維持し、ポイント インタイムリストアを可能にします。組織は、特定の日付に存在したデータを復元でき、履歴データが一時的にトランザクションアクセスを必要とするシナリオをサポートします。

第5章 Learning

5.1 Analytics Accelerator learning resources

このガイドを使用して、Postgres®上に構築されたアナリティクスアクセラレータ PGAA のすべてのドキュメントをナビゲートします。中核概念を理解する、ステップバイステップのチュートリアルに従う、特定の問題を解決する、または自分のロールに分析を適用する方法を検討したいかにかかわらず、このガイドは、適切なリソースを見つけるのに役立ちます。

ハブクイックリンク [Analytics Hub](#) — [Analytics Accelerator Concepts](#) — [How-Tos \(Runbook-Aligned\)](#)

ドキュメントは、効果的に学習し、効率的に情報を見つけられるように構成されています。

5.1.1 ドキュメントカテゴリ

構造化された学習パス

構造化された学習パスに従って、基本的な概念から高度な実装に進みます。

ラーニングパスの探索 [Analytics Accelerator Learning Paths](#)

- [Analytics Accelerator 101 Foundational concepts](#)
- [Analytics Accelerator 201 Practical application and core solutions](#)
- [Analytics Accelerator 301 Advanced techniques and optimization](#)

概念的な説明

分析の原則、最新のデータアーキテクチャ、およびアナリティクスアクセラレーターのテクノロジーと戦略に関する基礎知識を構築します。

Explore [Analytics Generic Concepts](#) および [Analytics Accelerator Concepts](#)

汎用分析概念、アナリティクスアクセラレータ PGAA ビジョンをカバーし、EDB Postgres Lakehouse、Apache Iceberg、Delta Lake、階層テーブル、アーキテクチャパターンなどのコアテクノロジーを深く掘り下げています。

ハウツーガイド

アナリティクスアクセラレータに関連する一般的なタスクの簡潔なステップバイステップの指示を見つけます。

Explore [Analytics Accelerator how-to guides index](#)

ユースケースとソリューション

Analytics Accelerator がどのようにビジネス上の問題を解決し、業界全体のアーキテクチャパターンをサポートするかをご覧ください。

Explore Analytics in Hybrid Manager

ペルソナベースのガイダンス

技術的なロールに合わせて調整された情報と学習パスにアクセスします。

Explore *Analytics Accelerator for your role a persona-based guide*

ハイブリッドマネージャー HM のアナリティクスアクセラレーター

EDB の管理対象サービスのユーザーを対象に、このセクションでは、アナリティクスアクセラレーターの概念とテクノロジーがハイブリッドマネージャーで実装、プロビジョニング、および管理される方法に関するドキュメントを提供します。

Explore Analytics in Hybrid Manager

5.1.2 次のステップ

- *Analytics Accelerator Learning Paths* をフォローして専門知識を構築します
 - *Analytics Generic Concepts* および *Analytics Accelerator Concepts* で基礎的な知識を構築する
 - How-to guides で学んだことを適用する
 - Analytics in Hybrid Manager を介して実践的な例を探索する
 - *Persona-based guides* でのロールに合わせたガイダンスを取得します
 - HM ユーザーの場合 Analytics in Hybrid Manager の詳細
-

この学習ガイドを使用して、アナリティクスアクセラレータードキュメントを介したパスを図示します。コア概念から始める場合も、高度なアーキテクチャを深く掘り下げる場合も。

5.2 Analytics Accelerator Learning Paths

これらのラーニングパスを使用して、基本的な概念から高度な実装、最適化テクニックに至るまで、知識を段階的に構築します。

5.2.1 これらのラーニングパスの使用方法

現在の知識と目標に合ったレベルを選択してください。各レベルは、概念的な説明、ハウツー記事、実践的なチュートリアルを含む厳選されたガイドにリンクしています。

高度なトピックと認定資格については、公式 EDB トレーニングプログラムをご覧ください。

5.2.2 ラーニングパスレベル

レベル 101 アナリティクスアクセラレーターの基礎

対象者 アナリティクスアクセラレーターを初めて使用するユーザー、最新のデータ分析の概念を初めて使用するユーザー、または復習を求めているユーザー。

フォーカス コアアイデア、重要な用語、アナリティクスアクセラレーターランドスケープ、および分析アプローチの背後にある理由。

学習する内容

- データウェアハウス、データレイク、およびレイクハウスパラダイムの基本的な概念
- 列型ストレージ、ベクトル化されたクエリエンジン、オープンテーブルフォーマットなどのキーワード Apache Iceberg、Delta Lake
- Postgres と EDB Postgres Lakehouse などのコアソリューションを使用したアナリティクスアクセラレータービジョンの概要
- 分析のためにストレージとコンピューティングを分離するメリット

ここから始めます [Analytics Generic Concepts](#)

レベル 201 アナリティクスアクセラレーターの適用 — コアテクノロジーとユースケース

対象者 基礎的な知識を持つユーザー、コアテクノロジーを実際に適用する準備ができているユーザー。DBA、データエンジニア、開発者に適しています。

フォーカス EDB Postgres Lakehouse、PGD 分析階層テーブルを含む、および Iceberg/Delta Lake の統合の実践的な使用。基本的な使用例と構成。

学習する内容

- EDB Postgres Lakehouse クラスターの設定と照会
- Apache Iceberg および Delta Lake テーブルを使用した作業
- EDB Postgres DistributedPGD を使用して基本的な階層テーブル構成を実装する
- BI ツールをアナリティクスアクセラレーターに接続する
- 分析を BI レポート、履歴データ分析、およびオペレーション分析に適用する

旅を続けます [Analytics Accelerator Concepts + How-to guides](#)

レベル 301 アドバンスドアナリティクスアクセラレーター、最適化、およびアーキテクチャ

対象者 経験豊富なユーザー、アーキテクト、およびシニアエンジニア。

重点 高度な機能、パフォーマンスチューニング、複雑なアーキテクチャ、スケーラビリティ、セキュリティ、およびトラブルシューティング。

学習する内容

- EDB Postgres Lakehouse、PGAA、および PGFS の高度な構成
- ベクトル化されたクエリとオブジェクトストレージアクセスのパフォーマンスチューニング
- 最適なコストとパフォーマンスを実現するための PGD を使用して複雑な階層テーブル戦略を設計および実装します
- 高度な Iceberg カタログを管理し、サードパーティのカatalogを統合
- アナリティクスアクセラレーターと他のデータフレームワーク Spark、ストリーミングプラットフォームを組み合わせたソリューションを設計する
- 複雑な分析クエリパフォーマンスとデータパイプラインのトラブルシューティング
- 分析環境にセキュリティのベストプラクティスを適用する

専門知識を深めましょう [Analytics Accelerator Architecture](#) および [PGAA functions reference](#) を参照してください

5.2.3 EDB トレーニングと認定資格

専門家による指導と公式認定については、EDB のトレーニングプログラムと認定資格をご覧ください。

- [Analytics Accelerator Learning Paths](#) 近日公開予定
- [Analytics Accelerator Learning Paths](#) 近日公開予定

5.2.4 追加のガイダンス

- ロールベースのガイダンスジョブファンクションに合わせたリソースについては、[Analytics Accelerator for your role a persona-based guide](#) を参照してください。
- ソリューション指向の例 実際の例については、Analytics in Hybrid Manager にアクセスしてください。

5.2.5 次のステップ

- [Analytics Generic Concepts](#) で基礎を構築する
- [Analytics Accelerator Concepts](#) で EDB のアプローチを学ぶ
- How-to guides で知識を適用する
- Analytics in Hybrid Manager を介して業界アプリケーションを探索する

これらの学習パスを使用して、基礎知識から高度な最適化とアーキテクチャに至るまで、アナリティクスアクセラレーターの専門知識を進歩させます。

5.3 Analytics Accelerator 101 Foundational concepts

ここから始めて、アナリティクスアクセラレーターの基礎となる中心的なアイデアを学びましょう。このレベルは、最新の分析概念を初めて使用する人、または復習を求めている人向けに設計されています。

5.3.1 学ぶこと

- データウェアハウス、データレイク、およびレイクハウスアーキテクチャの基本的な概念
- カラムナストレージとベクトル化されたクエリエンジン
- オープンテーブルフォーマット Apache Iceberg と Delta Lake
- EDB Analytics Accelerator のビジョンとコアアーキテクチャ
- 分析のためにストレージとコンピューティングを分離することの価値

5.3.2 始める前に

- *Analytics Generic Concepts*
- *Analytics Accelerator Concepts*

5.3.3 推奨される学習パス

1. *Analytics Generic Concepts*
 2. *Analytics Accelerator Concepts*
 3. *Apache Iceberg Integration With Analytics Accelerator*
 4. *Delta Lake Integration with Analytics Accelerator*
- *EDB Postgres Lakehouse Architecture*

5.3.4 次のステップ

101 が完了したら、次のことに進みます。

- *Analytics Accelerator 201 Practical application and core solutions*
- *Analytics Accelerator Learning Paths*

5.4 Analytics Accelerator 201 Practical application and core solutions

基礎的な知識をもとに、主要なアナリティクスアクセラレーターテクノロジーを環境に適用し始めます。

5.4.1 学ぶこと

- EDB Postgres Lakehouse クラスターをセットアップおよび照会する方法
- Apache Iceberg および Delta Lake テーブルを操作する方法
- PGD を使用して階層テーブルを実装する方法
- BI ツールをアナリティクスアクセラレーターに接続する方法
- これらのテクノロジーを BI レポート、履歴データ分析、およびオペレーション分析に適用する方法

5.4.2 始める前に

- *Analytics Accelerator 101 Foundational concepts*
- *Analytics Generic Concepts*
- *Analytics Accelerator Concepts*

5.4.3 推奨される学習パス

1. *EDB Postgres Lakehouse Architecture*
2. *Apache Iceberg Integration With Analytics Accelerator*
3. *Delta Lake Integration with Analytics Accelerator*
4. *Tiered Tables With Analytics Accelerator*
5. *Create A Lakehouse Cluster*
6. *Configure Analytics Storage and Data Tiering with PGAA and PGD*
7. *How-To Lakehouse Read With/Without A Catalog*
8. *How-To Read/Write Without Catalog*
9. *How-To Integrate with Third-Party Iceberg Catalogs*

5.4.4 次のステップ

201 が完了したら、次のことに進みます。

- *Analytics Accelerator 301 Advanced techniques and optimization*
- Reference index

5.5 Analytics Accelerator 301 Advanced techniques and optimization

高度な最適化とアーキテクチャパターンで専門知識を深める準備ができている経験豊富なユーザー、アーキテクト、エンジニア向け。

5.5.1 学ぶこと

- EDB Postgres Lakehouse、PGAA、および PGFS の高度な構成
- ベクトル化されたクエリとオブジェクトストレージアクセスのパフォーマンスチューニング
- 最適化されたコストとパフォーマンスのための複雑な階層テーブル戦略
- 高度な Iceberg カタログの管理とサードパーティのカタログの統合
- Analytics Accelerator と他のデータフレームワーク Spark、ストリーミングプラットフォームを組み合わせたソリューションのアーキテクチャ
- 複雑な分析クエリとデータパイプラインのトラブルシューティング
- 分析環境のセキュリティのベストプラクティス

5.5.2 始める前に

- *Analytics Accelerator 201 Practical application and core solutions*
- *Analytics Accelerator Concepts*
- *Tiered Tables With Analytics Accelerator*

5.5.3 推奨される学習パス

1. *Analytics Accelerator Architecture*
2. *Apache Iceberg Integration With Analytics Accelerator*
3. *Delta Lake Integration with Analytics Accelerator*
4. *DirectScan and fallback modes*
5. *PGAA functions reference*
6. *PGAA functions reference*
7. *Benchmarking datasets*
8. *User management*
9. *Analytics Accelerator for your role a persona-based guide*

5.5.4 次のステップ

公式のトレーニングと認定資格を検討します。

- EDB training and certification

5.6 Analytics Accelerator for your role a persona-based guide

このガイドを使用して、特定のロールに EDB Postgres® AI および分析機能を活用する方法を理解します。EDB Postgres AI は、幅広い分析ニーズに対応する強力なプラットフォームを提供します。このガイドは、目標に合わせた機能に焦点を当てるのに役立ちます。

5.6.1 自分のロールをすばやく見つける

- Database administrator (DBA)
- DevOps engineer / site reliability engineer (SRE)
- Data scientist / data analyst
- Application developer

EDB Hybrid ManagerHM 内でこれらのソリューションを実装および管理する方法の詳細については、Analytics in Hybrid Manager を参照してください。

5.6.2 データベース管理者 DBA

データベースインフラのパフォーマンス、信頼性、コスト効率、および管理のしやすさに焦点を当てます。

主要な目標とアナリティクスアクセラレーターソリューション

パフォーマンスのモニタリングとチューニング

遅いクエリー、ボトルネック、および非効率なリソースの使用を特定します。HM ダッシュボードを使用して、EDB Postgres Lakehouse ノードと PGD クラスターのパフォーマンスをモニタします。

ストレージ管理とコストの最適化

大規模なデータベースと履歴データをコスト効率よく管理します。PGD および EDB Postgres Lakehouse で階層化テーブルを使用して、コールドデータをオブジェクトストレージに Iceberg テーブルとしてオフロードします。*Tiered Tables With Analytics Accelerator*。HM を介して階層テーブルを管理します。

キャパシティプランニング

ストレージとコンピューティングのニーズを予測します。Lakehouse および HM ダッシュボードを介して、履歴的な成長パターンと現在の使用率を分析します。Lakehouse のコンピューティングとストレージの分離を活用して、柔軟なスケーリングを実現します。

分析データのバックアップ リカバリー

オブジェクトストレージのデータと関連するメタデータを保護します。Iceberg のカタログバックアップ戦略と HM バックアップオプションに従います。

分析のためのデータのアクセシビリティの確保

データアナリストやデータサイエンティストにパフォーマンス的で安全なアクセスを提供します。Lakehouse ノードと PGD オフロードを使用して、トランザクションシステムへの影響を最小限に抑えます。

推奨されるリソース

- EDB Postgres Distributed PGD ドキュメント
- *EDB Postgres Lakehouse Architecture*
- Analytics in Hybrid Manager

5.6.3 DevOps エンジニア / サイト信頼性エンジニア SRE

自動化、インフラストラクチャ管理、スケーラビリティ、信頼性、オブザーバビリティに焦点を当てます。

主要な目標とアナリティクスアクセラレーターソリューション

自動プロビジョニングと管理

ハイブリッドマネージャーを使用して、コンソール、API、または CLI を介して Lakehouse クラスターと関連リソースをプロビジョニングします。 *Create A Lakehouse Cluster* 。

スケーラビリティと弾力性

需要に合わせて Lakehouse コンピューティングリソースを個別にスケーリングします。

モニタリングとオブザーバビリティ

分析システムのパフォーマンス、状態、およびコストを可視化します。HM ダッシュボードとロギングを使用します。

データパイプラインと分析アプリケーションのための CI/CD

スキーマの変更 Iceberg の進化と PGD を介したデータのオフロードを CI/CD パイプラインに統合します。

クラウドリソースのコスト管理

オブジェクトストレージと分析コンピューティングのコストを追跡および最適化します。HM リソースの洞察を使用します。

推奨されるリソース

- Analytics in Hybrid Manager
- *EDB Postgres Lakehouse Architecture*
- EDB API および CLI ドキュメント

5.6.4 データサイエンティスト / データアナリスト

洞察の抽出、モデルの構築、およびビジネスの質問への回答に焦点を当てます。

主要な目標とアナリティクスアクセラレーターソリューション

大規模で多様なデータセットへのアクセス

Lakehouse ノードを介して、Iceberg または Delta Lake テーブルとして保存されている大規模なデータセットを照会します。Python、R、SQL クライアント、および BI ツールを使用します。

データの準備と探索

Lakehouse ノードを介して SQL でデータのフィルタリング、集計、および変換を実行します。

オープンテーブルフォーマットの使用

Iceberg および Delta Lake テーブルのスキーマの進化、タイムトラベル、および ACID トランザクションを活用します。 [Apache Iceberg Integration With Analytics Accelerator](#)。 [Delta Lake Integration with Analytics Accelerator](#)。

データサイエンスツールおよびフレームワークとの統合

Lakehouse のクエリ結果を pandas、R データフレームにロードするか、Spark、TensorFlow、または PyTorch で使用します。

クエリパフォーマンス

Lakehouse ノードのベクトル化クエリエンジンを使用して分析 SQL クエリを高速化します。

推奨されるリソース

- [EDB Postgres Lakehouse Architecture](#)
- [How-To Lakehouse Read With/Without A Catalog](#)
- BI ツールの接続

5.6.5 アプリケーション開発者

データ駆動型アプリケーションの構築とインサイトの埋め込みに焦点を当てます。

主要な目標とアナリティクスアクセラレーターソリューション

分析機能を使用したアプリケーションの構築

Lakehouse ノードまたは PGD リードレプリカを使用して、アプリケーションでレポート、ダッシュボード、およびデータ探索を提供します。

大規模なレポートクエリの効率的な処理

大規模な集計および履歴データのクエリをレイクハウスノードにオフロードします。階層化テーブルを使用して、履歴クエリを分析階層に転送します。

統合プラットフォームからのデータへのアクセス

運用データと分析データの両方への一貫した SQL アクセスを提供します。

AI / ML 対応アプリケーションの開発

Lakehouse ノードを使用して ML モデルデータを保存およびアクセスします。

推奨されるリソース

- [Postgres SQL ドキュメントとクライアントライブラリ](#)
- [EDB Postgres Lakehouse Architecture](#)
- [Tiered Tables With Analytics Accelerator](#)

より詳細なガイダンスについては、これらのリソースを参照してください。HM 固有の実装手順については、[Analytics in Hybrid Manager](#) を参照してください。

第6章 How To Hybrid Manager

6.1 Create A Lakehouse Cluster

このガイドでは、ハイブリッドマネージャー HM で **レイクハウスクラスター**を作成する方法について説明します。Lakehouse クラスターは、オブジェクトストレージに保存されている **Apache Iceberg** または **Delta Lake** テーブルに対する高速クエリーに**スケーラブルな分析コンピューティング**を提供します。

アーキテクチャの背景については、[Lakehouse clusters in Hybrid Manager](#) を参照してください。

6.1.1 Lakehouse クラスターを作成する理由

Lakehouse クラスターを作成することにより

- オブジェクトストレージ内の **大規模なデータセット**で高速 **Postgres SQL** クエリーを実行できる機能のロックを解除します。
- PGD または 外部 Iceberg/Delta Lake テーブル からオフロードされた階層テーブルを照会できます。
- レイクハウスデータの **BI ツール**と**アドホック分析**のコンピューティングを提供します。

これは、ハイブリッドマネージャーで分析アーキテクチャを構築するための基礎的な手順です。

6.1.2 ゴール

このハウツーを完了すると、次のことができるようになります。

- Lakehouse クラスターを展開します。
- 標準の Postgres クライアントを使用して接続します。
- Iceberg および Delta Lake テーブルを照会するためのストレージとカタログを構成します。
- クラスターを使用して、既存のまたは新しく定義されたレイクハウステーブルを照会します。

6.1.3 前提条件

始める前に

- Lakehouse クラスターを作成する権限を持つ Hybrid Manager 環境にアクセスしていること。
- 利用可能なオブジェクトストレージがあること S3、GCS、MinIO など。
- 使用可能な Iceberg または Delta Lake テーブルがあるか、クラスターの作成後にそれらを定義する予定があります。
- Iceberg テーブルを使用する場合、Iceberg カタログを準備しています。

- *Configure Analytics Storage and Data Tiering with PGAA and PGD*

6.1.4 ステップ

ステップ 1 クラスターの作成を開始する

- HM ダッシュボードで、**Create New** をクリックします。
- **Lakehouse Analytics** または **分析クラスター** を選択します。
- オープンレイクハウスフォーマットを照会するための分析クラスターを作成していることを確認します。

ステップ 2 自分の道を選択してください

以下を選択するように求められます。

- **テンプレート** — 事前構成されたテンプレートを使用します利用可能な場合。
- **カスタムビルド** — 完全な構成制御には **Start from Scratch** をクリックしますほとんどの場合にお勧めします。

ステップ 3 クラスター設定を構成する

クラスター設定

- **クラスター名** — 一意の名前たとえば `sales_analytics_lakehouse` を入力します。
- **Password** — 強力なスーパーユーザーパスワードを入力します。
- **タグ** — オプションでタグを追加します `environment:production`、`project:q3-reporting` など。
- **展開場所** — クラウドプロバイダーとリージョンを選択します。
- **データベースタイプ** — EDB Postgres Extended Server または Advanced Server を選択します。
- **Postgres バージョン** — Lakehouse 拡張機能を含むサポートされているバージョンを選択します。
- **インスタンスサイズ** — ワークロードに適切なコンピューティングサイズを選択します。

追加設定

- **ネットワーク** — VPC、サブネット、セキュリティグループ、IP ホワイトリストを構成します。
- **バックアップ** — 必要に応じてバックアップ設定を構成します。

ステップ 4 確認と作成

- **クラスターの概要** を確認します。
- すべての設定を確認します。
- **Create Cluster** をクリックします。
- HM ダッシュボードで進行状況を監視します。

6.1.5 次にできること

Lakehouse クラスターがプロビジョニングされたため、次のことができます。

- クラスターに接続して SQL クエリーを実行します。HM 提供の接続の詳細で標準の Postgres クライアント `psql`、`DBeaver`、`pgAdmin` を使用します。
- データにアクセスするようにストレージとカタログを構成します
- Configure a PGFS storage location — Delta Lake またはファイルシステムベースの Iceberg を照会する場合。
- *Configure Analytics Storage and Data Tiering with PGAA and PGD* —カタログ管理の Iceberg テーブルを照会する場合、または PGD 階層化テーブルを使用する場合。
- レイクハウステーブルの定義と照会
- *How-To Lakehouse Read With/Without A Catalog*
- PGD からオフロードされたクエリ階層テーブル
- *Tiered Tables With Analytics Accelerator* の概念を学習する

6.1.6 関連トピック

- *Lakehouse clusters in Hybrid Manager*
- Analytics in Hybrid Manager

第7章 How To Playbook

7.1 How To Get Setup for Lakehouse analytics

7.2 Lakehouse アナリティクスの設定

このドキュメントでは、Delta Lake および Iceberg テーブルで読み取り専用分析を実行する前に必要な前提条件と設定手順について説明します。

必要なイメージバージョン、環境変数、CA 証明書、カタログ構成、およびオブジェクトストレージバケットを識別する方法について説明します。

7.2.1 ゴール

- 必要なオペランドと拡張バージョンの確認
- 環境変数と資格情報を確立する
- アプライアンス CA 証明書の取得
- アプライアンスでプロビジョニングされたオブジェクトストレージバケットの特定

7.2.2 要件

次のオペランドイメージが必要です。

- PGD 6.0 Expanded スナップショットバージョン
- PGAA 1.2+

拡張機能のバージョンを確認します。

```
SELECT name, installed_version, default_version
FROM pg_available_extensions
WHERE name = pgaa;
\set
```

`default\_version` は `1.2+` である必要があります。

>

→ 注ご使用の環境がスナップショットリポジトリまたは事前構築されたスナップショットオペランドイメージを必要とする場合は、次のページに続く

(次のページに続く)

```
-- --
```

```
## 仮定と変数
```

このガイドでは、PGD6 クラスターがハイブリッドマネージャーで既にプロビジョニングされていることを前提としています。プロビジョニングは範囲外です。

EPAS または PGE では、`-pgdx` イメージバージョンを使用する必要があります。Vanilla PostgreSQL は **サポートされていません**。

後で使用するために次の変数を定義します。

```
.. csv-table::
:header: Variable, Meaning, Example
:widths: 12, 25, 15
:align: left
:class: longtable

`PROJECT_ID`, Hybrid Manager Project ID, `prj_RAI8WE2xivmeK4qm`
`APPL_URL`, Hybrid Manager base URL, `https://portal.eks-04250637-main.edbaiplatform.
→enterprisedb.network`
`API_PROV`, Hybrid Manager token with "editor" project permissions, `baak_abcdef...`
`API_WRITE`, Hybrid Manager token with "write_catalog" project permissions, `baak_
→ghijkl...`
`PARAMS_URI`, Managed catalog API URL, `https://portal.eks-04250637-main.../catalog/`
`PARAMS_WAREHOUSE`, Managed catalog warehouse name (PyIceberg), `prj_RAI8WE2xivmeK4qm-
→catalog`
`PARAMS_WAREHOUSE_ID`, Managed catalog warehouse UUID (PGAA), `cd8c29f6-20fa-11f0-97cd-
→cfc1662c48a8`
`PARAMS_CONFIG`, Managed catalog config URL (PySpark), `https://portal.eks.../config?
→warehouse=prj_RAI8WE2xivmeK4qm-catalog`
`BUCKET_URL`, URL of the bucket used by Hybrid Manager for offloads, `s3://eks-04220339-
→main-edb-postgres`
```

```
-- --
```

```
## アプライアンスCA 証明書の取得
```

この証明書は、サンプルデータのロードスクリプトに必要です。

```
```bash
```

```
Using kube_context
```

(前のページからの続き)

```
kubectl --context "$kube_context" get secret -n cert-manager global-ca-secret -o \
→yaml | yq .data."ca.crt" | base64 -d > ca.crt

On RHOS or avoiding kube_context

kubectl get secret -n cert-manager global-ca-secret -o yaml | yq .data."ca.crt" | \
→base64 -d > ca.crt

On RHOS GCP where cert-manager may not exist

kubectl get secret -n edbpgai-bootstrap beaconator-ca-bundle -o json | jq -r .data.
→"public.crt" | base64 --decode > ca.crt
```

**説明** CA 証明書は、ハイブリッドマネージャーアプライアンスの API とサービスに接続するときに信頼を確立するために必要です。

### 7.2.3 Lakekeeper カタログを作成する

API\_PROV トークンを使用します。

1. ハイブリッドマネージャー UI で、**Catalogs**→**Add Project Catalog** に移動します。
2. カタログ設定を使用して JSON をコピーします。例

```
{
 "uri": "https://portal-uat-rke.rke.hcp.enterprisedb.network/api/iceberg/prj_
→efpRstCQ1l6m2lGm",
 "warehouse": "prj_efpRstCQ1l6m2lGm-catalog-iXFkMR",
 "warehouseId": "d1d1d4ce-850a-11f0-88fb-af8509011340",
 "config": "https://portal-uat-rke.rke.hcp.enterprisedb.network/api/iceberg/prj_
→efpRstCQ1l6m2lGm/v1/config?warehouse=prj_efpRstCQ1l6m2lGm-catalog-iXFkMR"
}
```

**説明** これらのパラメーターは、後で PGAA を管理対象カタログまたは PyIceberg や PySpark などの外部ツールにリンクするときに使用されます。

RHOS では、pgaa.add\_catalog のようなクラスター内 SQL ステートメントのパブリック URL の代わりに内部 **Kubernetes サービス URL** を使用します。

### 7.2.4 バケット URL を特定します

アプライアンスは、独自のオブジェクトストレージバケットをプロビジョニングします。CSP に基づいて URL を構成します。

- AWS ポータル URL の eks-XXXX 部分を使用し、-main-edb-postgres を追加します。例

```
https://portal.eks-03242048-v11x.edbaipatform.enterprisedb.network
→ s3://eks-03242048-main-edb-postgres
```

- GCP ポータル URL の gke-XXXX 部分を使用し、-main-edb-object-storage を追加します。例

```
https://portal.gke-05151425-v12x.edbaipatform2.enterprisedb.network
→ gs://gke-05151425-main-edb-object-storage
```

- RHOS Barman バックアップまたは edb-object-storage シークレットを検査します。

```
kubectl get secret edb-object-storage -o yaml | yq .data.bucket_name | base64 -d
```

説明 PGAA コマンドおよびサンプルデータスクリプトでストレージを参照するには、BUCKET\_URL 値が必要です。

---

## 7.2.5 サニティチェック

パブリックバケットへのアクセスを確認するオプション

```
aws s3 ls s3://beacon-analytics-demo-data-us-east-1-dev/iceberg-example/default.db --
→ recursive --no-sign-request
```

次のようなファイルが表示されるはずです。

```
2025-03-18 08:44:38 902 iceberg-example/default.db/iceberg_table/data/000000-0-
→ ..
2025-03-18 08:44:38 4340 iceberg-example/default.db/iceberg_table/metadata/4ef9.
→ ..
2025-03-18 08:44:38 1823 iceberg-example/default.db/iceberg_table/metadata/snap.
→ ..
```

説明 これは、サンプルの Iceberg データセットが存在し、テストのためにアクセスできることを確認します。

---

## 7.2.6 次のステップ

- *How-To Lakehouse Read With/Without A Catalog* に進みます。
  - または、*How-To Read/Write Without Catalog* を試してください。
- 

## 7.2.7 テイクアウト

- 準備された環境が後続のハウツーカタログ、リーダー、オフロードを加速します
- CA バンドルとバケット URL は、データアクセスとツールの基礎です

## 7.3 How-To Read/Write Without Catalog

**目的** PGD 高可用性を使用して EDB PGAA を構成して、コスト最適化とパフォーマンスのスケーリングのための自動データ階層化、分析オフロード、およびレイクハウスアーキテクチャを実装します。

**範囲** 初期セットアップから高度なパーティション管理までの完全なワークフローデルタレイクとアイスバーグの統合、自動階層化戦略、ストレージ最適化分析を含みます。

### 7.3.1 前提条件

#### 技術的要件

PGD 高可用性を備えた PostgreSQL は、カタログ管理、パーティション操作、およびストレージ構成の管理者権限で動作している必要があります。PGAA 拡張機能は、S3 互換エンドポイントを含む外部ストレージシステムにネットワーク接続してインストールする必要があります。十分な帯域幅は、オフロードプロセス中のストレージ階層間のデータ転送操作をサポートします。

#### 知識要件

PostgreSQL のパーティショニングの概念、システムカタログ構造、および分散データベース管理を理解することにより、効果的な実装が可能になります。オブジェクトストレージシステム、レイクハウスアーキテクチャ、および分析ワークロードパターンに精通していると、構成上の決定を最適化するのに役立ちます。SQL パフォーマンス分析の基本的な知識は、トラブルシューティングと最適化アクティビティをサポートします。

#### 環境設定

マルチプルのノードを備えたアクティブ PGD クラスターは高可用性の基盤を提供し、PGAA 拡張機能は分析ストレージの統合を可能にします。ネットワークセキュリティ構成は、適切なファイアウォールと認証設定を使用して外部ストレージエンドポイントへのアクセスを許可する必要があります。監視インフラストラクチャは、バックグラウンドタスクの処理とストレージ使用パターンを追跡する必要があります。

### 7.3.2 セットアップ: 分析インフラストラクチャの構成

#### ステップ 1 PGAA 拡張機能とパフォーマンス設定を初期化する

テストおよび運用展開シナリオに最適化された設定で分析インフラストラクチャを構成します。

```
- - Setup PGD configuration for efficient testing cycles
SET bdr.taskmgr_nap_time = 10000; -- AutoPartition worker runs every 10s vs default 290s
SET pgaa.max_replication_lag_s = 5; -- Analytics replication lag threshold
SET pgaa.flush_task_interval_s = 5; -- Frequent storage flush for testing

- - Initialize PGAA extension
CREATE EXTENSION IF NOT EXISTS pgaa CASCADE;
```

パフォーマンス設定は、開発およびテストフェーズ中のバックグラウンドタスク処理を加速します。実稼働環境は、システムリソースと操作要件に対してこれらの設定を評価する必要があります。PGAA 拡張機能は、分析ストレージ統合とレイクハウステーブル管理の基盤を提供します。

## ステップ 2 ストレージの場所を構成する

適切なセキュリティ構成を使用して、読み取り専用の分析アクセスと書き込み操作の両方の保存場所を確立します。ご覧ください

```
-- Create read-only storage location for existing Delta Lake data
SELECT bdr.replicate_ddl_command($$
 SELECT pgfs.create_storage_location(
 biganimal-sample-data,
 s3://beacon-analytics-demo-data-us-east-1-prod,
 {"aws_skip_signature": "true"}
);
$$);

-- Create development storage location for Iceberg tables
SELECT bdr.replicate_ddl_command($$
 SELECT pgfs.create_storage_location(
 biganimal-sample-data-dev,
 s3://beacon-analytics-demo-data-us-east-1-dev,
 {"aws_skip_signature": "true"}
);
$$);
```

PGFS (Postgres File System)。PGFS ストレージ場所構成は、読み取り専用分析アクセスと書き込み操作の両方の基盤を確立します。aws\_skip\_signature 設定はパブリックバケットに適用されますが、本番環境では、IAM ロールまたは資格情報ファイルを介した適切な AWS 資格情報構成が必要です。

## ステップ 3 アナリティクスの書き込み場所を構成する

自動データオフロード操作のプライマリ分析保存場所を設定します。Getting setup for Lakehouse analytics を参照してください。

```
-- Create appliance storage location with dynamic path construction
SELECT bdr.replicate_ddl_command($$
 SELECT pgfs.create_storage_location(
 appliance-bucket,
 $BUCKET_URL || / ||
 (SELECT node_group_name FROM bdr.node_group WHERE node_group_name != world LIMIT
→1) ||
 /analytics-offload
)
 $$);
```

(次のページに続く)

(前のページからの続き)

```

- - Validate storage location accessibility
CREATE TABLE sentinel_sl_read_test () USING PGAA
WITH (pgaa.format = iceberg, pgaa.storage_location = appliance-bucket, pgaa.path = ↵
↵nonexistent);
- - Should return ERROR: No valid Iceberg metadata file found (confirms bucket access)

- - Set as primary analytics write location
SELECT bdr.alter_node_group_option(
 (SELECT node_group_name FROM bdr.node_group WHERE node_group_name != world LIMIT 1),
 analytics_storage_location,
 appliance-bucket
);

```

動的パス構成には、マルチテナント環境での競合を防ぐための PGD ノードグループ名が含まれます。意図的なエラーテストを介した保存場所の検証により、適切な認証とネットワーク接続が確認されます。分析の保存場所を設定すると、レイクハウス操作の Seafoal クエリエンジンがアクティブになります。

#### ステップ 4 Seafoal クエリエンジンのアクティブ化を確認する

分析クエリエンジンが起動し、レイクハウス操作の準備ができていることを確認します。

```

- - Verify Seafoal query engine is running
SELECT application_name FROM pg_stat_activity WHERE application_name LIKE %seafoal%;
- - Must show pgd seafoal_manager for proper operation

```

Seafoal クエリエンジンは、カラムナストレージ形式全体で分析ワークロードの最適化された実行を提供します。

### 7.3.3 読み取り専用分析: 既存のデータへのアクセス

Iceberg フォーマット仕様により、スキーマの進化、パーティション管理、タイムトラベル機能を含む完全なメタデータサポートを備えた Apache Iceberg テーブルにアクセスできます。

#### ステップ 5 ストレージ特性を分析する

分析機能を使用して、レイクハウステーブルのストレージ使用率とパフォーマンス特性を理解します。

```

- - Analyze storage statistics for Delta Lake tables
SELECT * FROM pgaa.lakehouse_table_stats(lineitem::regclass);

- - Analyze storage statistics for Iceberg tables
SELECT * FROM pgaa.lakehouse_table_stats(iceberg_table::regclass);

```

ストレージ統計は、圧縮率、ファイル構成、およびクエリパフォーマンス特性に関する洞察を提供し、最適化の決定を通知します。これらのメトリックは、分析ワークロードのキャパシティプランニングとパフォーマンス調整をサポートします。

### 7.3.4 トランザクションテーブル分析オフロード

#### ステップ 9 トランザクションテーブルを作成してデータを入力する

オフロードテストと検証用の代表データを使用してトランザクションテーブルを確立します。

```
- - Create transactional table for offload testing
CREATE TABLE transactional_table (
 key BIGINT,
 inserted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
 revenue DOUBLE PRECISION,
 comment TEXT,
 PRIMARY KEY (key, inserted_at)
);

- - Insert test data with temporal distribution
INSERT INTO transactional_table (
 SELECT
 g,
 2010-01-01 00:00:00::timestamp + INTERVAL 1 second *
 (EXTRACT (EPOCH FROM (CURRENT_TIMESTAMP - 2010-01-01 00:00:00::timestamp)) /
→10000000 * g),
 g * 0.001,
 encode(sha256(g::text::bytea), hex)
 FROM generate_series(1, 10000000) g
);
```

データ生成は、パフォーマンステストに十分なボリュームを提供しながら、現実的な時間分布パターンを作成します。SHA256 ハッシュ生成は、実稼働システムで見つかった可変長テキストコンテンツをシミュレートします。

#### ステップ 10 分析オフロードを構成する

コストの最適化とパフォーマンスのスケーリングのために、トランザクションデータの分析ストレージへの自動レプリケーションを有効にします。

```
- - Enable analytics replication for transactional table
ALTER TABLE transactional_table SET (pgd.replicate_to_analytics = TRUE);

- - Monitor replication progress
SELECT * FROM bdr.analytics_table;

- - Wait until table appears in results (~120s for initial sync)
```

アナリティクスレプリケーションは、構成された保存場所にトランザクションデータの Iceberg 表現を作成します。初期同期プロセスはフォーマット変換を処理し、データの一貫性のために進行中のレプリケーションストリームを確立します。

### ステップ 11 アナリティクスリーダーインターフェイスを作成する

専用のリーダーテーブルを介して、分析層データへの直接アクセスを確立します。

```
- - Create reader interface for analytics data
CREATE TABLE transactional_table_analytics_reader ()
USING PGAA
WITH (
 pgaa.format = iceberg,
 pgaa.storage_location = appliance-bucket,
 pgaa.path = public.transactional_table
);

- - Verify data integrity across storage tiers
SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_revenue
FROM transactional_table_analytics_reader;

- - Should return: count=10000000, total_revenue=5000000500
```

分析リーダーは、クエリの透明性を維持しながら、費用対効果の高いストレージへの直接アクセスを提供します。データの完全性検証により、フォーマット変換とレプリケーションプロセスが成功したことが確認されます。

### ステップ 12 トランザクションテーブルをアナリティクス専用に変換する

ローカルデータを破棄しながら、分析ストレージを直接使用するようにトランザクションテーブルを変換します。

```
- - Convert to analytics-only access
ALTER TABLE transactional_table
SET ACCESS METHOD PGAA,
SET (
 pgaa.auto_truncate = true,
 pgaa.format = iceberg,
 pgaa.storage_location = appliance-bucket,
 pgaa.path = public.transactional_table
);

- - Verify conversion maintained data access
SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_revenue FROM transactional_
→table;
```

変換プロセスは、分析層を介した透過的な SQL アクセスを維持しながら、ローカルストレージのオーバーヘッドを排除します。自動切り捨ては、分析ティアの可用性を確認した後、ローカルデータを削除します。

### 7.3.5 高度なテーブル管理操作

#### ステップ 13 アナリティクスからトランザクションストレージにテーブルを復元する

ローカルの高パフォーマンスのアクセスが必要なシナリオの分析オフロードの可逆性を示します。

```
- - Restore table to transactional storage
ALTER TABLE transactional_table SET ACCESS METHOD heap, SET (pgd.replicate_to_
→analytics = FALSE);

- - Verify restoration with query plan analysis
EXPLAIN SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_revenue FROM
→transactional_table;

- - Should show Parallel Seq Scan indicating local heap access
```

復元機能は、アクセスパターンまたはパフォーマンス要件を変更するための柔軟性を提供します。クエリプラン分析により、並列処理機能を備えたローカル PostgreSQL ヒープストレージへの移行が確認されました。

#### ステップ 14 分析同期を使用してデータの更新を処理する

ストレージ階層全体で一貫性を維持するデータ変更ワークフローを示します。

```
- - Update data in transactional table
UPDATE transactional_table SET revenue = revenue * 2;

- - Re-enable analytics replication to propagate changes
ALTER TABLE transactional_table SET (pgd.replicate_to_analytics = TRUE);

- - Wait for synchronization (~120s) then verify changes propagated
SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_revenue
FROM transactional_table_analytics_reader;

- - Should show doubled revenue: total_revenue=10000001000
```

更新の伝播は、データの変更を効率的に処理しながら、トランザクション層と分析層間の一貫性を維持します。同期プロセスは、現在のデータを使用して分析表現を再作成します。

#### ステップ 15 部分テーブルオフロードを実装する

テーブルの履歴部分のみが分析ストレージに移動する選択的なデータアーカイブを実行します。

```
- - Perform partial offload with consistent transaction boundary
BEGIN;
CREATE TABLE transactional_table_before_2020 AS (
 SELECT * FROM transactional_table
 WHERE DATE_PART(year, inserted_at) < 2020
);
```

(次のページに続く)

(前のページからの続き)

```

DELETE FROM transactional_table WHERE DATE_PART(year, inserted_at) < 2020;
COMMIT;

-- Configure partial table for analytics offload
ALTER TABLE transactional_table_before_2020 ADD PRIMARY KEY (key, inserted_at);
ALTER TABLE transactional_table_before_2020 SET (pgd.replicate_to_analytics = TRUE);

-- Convert to analytics-only access
ALTER TABLE transactional_table_before_2020
SET ACCESS METHOD PGAA,
SET (
 pgaa.auto_truncate = true,
 pgaa.format = iceberg,
 pgaa.storage_location = appliance-bucket,
 pgaa.path = public.transactional_table_before_2020
);

-- Verify temporal data distribution
SELECT
 COUNT(*) FILTER (WHERE inserted_at < 2020-01-01) AS before_2020,
 COUNT(*) FILTER (WHERE inserted_at >= 2020-01-01) AS after_2020
FROM transactional_table_before_2020;

```

部分オフロードにより、期間が異なるとストレージ階層要件が異なる場合でも、きめの細かいデータライフサイクル管理が可能になります。トランザクション境界は、アーカイブプロセス中のデータの一貫性を保証します。

### 7.3.6 分析オフロードを使用した AutoPartition

#### ステップ 16 自動パーティションを構成する

スケーラブルなデータ組織のための自動ライフサイクル管理を使用して時間ベースのパーティショニングを確立します。

```

-- Create partitioned table structure
CREATE TABLE partitioned_table (
 key BIGINT,
 inserted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
 revenue DOUBLE PRECISION,
 comment TEXT,
 PRIMARY KEY (key, inserted_at)
) PARTITION BY RANGE(inserted_at);

-- Configure AutoPartition without immediate offload
SELECT bdr.autopartition (

```

(次のページに続く)

(前のページからの続き)

```

relation := partitioned_table,
partition_increment := 1 year,
partition_initial_lowerbound := 2010-01-01 00:00:00,
managed_locally := TRUE
);

-- Wait for partition creation across cluster
SELECT bdr.autopartition_wait_for_partitions_on_all_nodes(partitioned_table::regclass,
→ CURRENT_TIMESTAMP::text);

```

AutoPartition 構成は、一般的なデータアクセスパターンに合わせた境界を持つ年次パーティションを作成します。初期構成ではオフロード設定を除外して、セットアップおよびデータの読み込みフェーズ中の早すぎるデータの移行を防ぎます。

### ステップ 17 テストデータのロードとベースラインストレージの分析

パーティションに代表的なデータを入力し、最適化分析のベースラインストレージメトリックを確立します。

```

-- Load test data across temporal partitions
INSERT INTO partitioned_table (
 SELECT
 g,
 2010-01-01 00:00:00::timestamp + INTERVAL 1 second *
 (EXTRACT (EPOCH FROM (CURRENT_TIMESTAMP - 2010-01-01 00:00:00::timestamp)) /
→ 1000000 * g),
 g * 0.001,
 encode(sha256(g::text::bytea), hex)
 FROM generate_series(1, 1000000) g
);

```

データのロードにより、予測可能なコストモデリングとパフォーマンス分析を可能にする一貫したストレージ特性を備えた年間パーティションごとに約 65,000 レコードが作成されます。

### ステップ 18 ベースラインストレージ分析

包括的なストレージ分析機能を作成し、比較のためにプレオフロードメトリックをキャプチャします。

```

-- Create optimized row counting function
CREATE OR REPLACE FUNCTION temp_get_table_rowcount(schema_name text, table_name text)
RETURNS bigint AS $$
DECLARE count bigint;
BEGIN
 -- Uses DirectScan for PGAA tables (1.8s vs 20s without optimization)
 EXECUTE FORMAT(SELECT COUNT(*) FROM %I.%I, schema_name, table_name) INTO count;
 RETURN count;

```

(次のページに続く)

(前のページからの続き)

```

END $$ LANGUAGE plpgsql;

-- Comprehensive partition storage analysis
WITH partitioned_table_info AS (
 SELECT
 nspc.nspname AS schema_name,
 child.relname AS table_name,
 pg_catalog.pg_get_expr(child.relpartbound, child.oid) AS partition_constraint,
 pg_get_partition_constraintdef(child.oid) AS partition_constraint_full
 FROM pg_inherits inh
 JOIN pg_class parent ON inh.inhparent = parent.oid
 JOIN pg_class child ON inh.inhrelid = child.oid
 JOIN pg_namespace nspp ON parent.relnamespace = nspp.oid
 JOIN pg_namespace nspc ON child.relnamespace = nspc.oid
 WHERE parent.relname = partitioned_table AND nspp.nspname = public
),

pgaa_tables AS (
 SELECT
 m.schema_name, m.table_name, m.format,
 CASE
 WHEN m.storage_location_name IS NOT NULL THEN FORMAT(pgfs://%s/%s, m.storage_
 ↳location_name, m.path)
 WHEN (m.managed_by IS NOT NULL AND m.catalog_namespace IS NOT NULL AND m.
 ↳catalog_table IS NOT NULL)
 THEN FORMAT(%s.%s.%s, m.managed_by, m.catalog_namespace, m.catalog_table)
 END AS path,
 temp_get_table_rowcount(m.schema_name, m.table_name) AS num_rows,
 s.total_size AS total_size,
 pti.partition_constraint
 FROM pgaa.get_all_analytics_table_settings() m
 LEFT JOIN LATERAL pgaa.lakehouse_table_stats(FORMAT(%I.%I, m.schema_name, m.table_
 ↳name)::regclass) s ON TRUE
 JOIN partitioned_table_info pti ON m.schema_name = pti.schema_name AND m.table_name_
 ↳= pti.table_name
 WHERE m.storage_location_name IS NOT NULL OR (m.managed_by IS NOT NULL AND m.
 ↳catalog_namespace IS NOT NULL AND m.catalog_table IS NOT NULL)
),

local_tables AS (
 SELECT
 pn.nspname AS schema_name, pc.relname AS table_name,
 temp_get_table_rowcount(pn.nspname, pc.relname) AS num_rows,
 pg_total_relation_size(FORMAT(%I.%I, pn.nspname, pc.relname)::regclass) AS total_
 ↳size,

```

(次のページに続く)

```

 pti.partition_constraint
FROM pg_class pc
 JOIN pg_namespace pn ON pc.relnamespace = pn.oid
 JOIN pg_am am ON am.oid = pc.relam
 JOIN partitioned_table_info pti ON pn.nspname = pti.schema_name AND pc.relname =
→pti.table_name
WHERE pn.nspname NOT IN (pg_catalog, information_schema, pg_toast, pg_internal,
→pgaa, pgfs, bdr)
 AND pc.relkind = r AND am.amname != pgaa
),

all_leaf_tables AS (
 SELECT schema_name, table_name, format AS pgaa_format, path AS pgaa_path, num_rows,
→total_size, partition_constraint FROM pgaa_tables
 UNION
 SELECT schema_name, table_name, NULL AS pgaa_format, NULL AS pgaa_path, num_rows,
→total_size, partition_constraint FROM local_tables
),

final_table_info AS (
 SELECT
 schema_name, table_name,
 CASE
 WHEN pgaa_format = iceberg THEN iceberg
 WHEN pgaa_format = delta THEN delta
 ELSE heap
 END AS type,
 COALESCE(pgaa_path, local) AS location,
 num_rows, total_size AS total_size_bytes,
 SUBSTRING(partition_constraint, FROM []((^[^]+)) AS start_bound,
 SUBSTRING(partition_constraint, TO []((^[^]+)) AS end_bound
 FROM all_leaf_tables
 ORDER BY partition_constraint ASC
)

SELECT * FROM final_table_info;

```

ベースライン結果すべてのパーティションは、履歴パーティションごとに約 10MB、高価なトランザクションストレージの完全なデータセットの合計～154MB の type = 'heap' および location = 'local' を示します。

### ステップ 19 自動分析オフロードを構成する

クエリの透明性を維持しながら、データの経過時間のしきい値に基づいて自動パーティション階層化を有効にします。

```
- - Configure analytics offload for partitions older than one year
SELECT bdr.autopartition (
 relation := partitioned_table,
 partition_increment := 1 year,
 partition_initial_lowerbound := 2010-01-01 00:00:00,
 managed_locally := TRUE,
 analytics_offload_period := 1 year
);
```

1 年間のしきい値は、履歴パーティションを費用対効果の高い分析ティアに移行しながら、最近のデータを高性能ストレージに維持することにより、操作アクセス要件とストレージコストの最適化のバランスをとります。

### ステップ 20 自動オフロード操作を監視する

バックグラウンドの移行の進行状況を追跡し、ストレージ階層の移行の完了ステータスを特定します。

```
- - Monitor offload work queue until completion
SELECT
 wqs.workid,
 wis.ap_wi_started_at,
 wis.ap_wi_finished_at,
 wis.ap_wi_status,
 "partition",
 sql
FROM bdr.taskmgr_local_workitem_status wis
 RIGHT JOIN bdr.taskmgr_local_work_queue_status wqs ON wis.ap_wi_workid = wqs.workid
WHERE relname = public.partitioned_table
AND sql NOT LIKE %bdr%.autopartition_create_partition%
ORDER BY ap_wi_finished_at DESC;
```

すべての `ap_wi_finished_at` 値が NOT NULL、正常に完了したことを示すまで監視します。処理には、通常、14 の履歴パーティション 2010～2025 年に実行の場合 2023 年に 10 分以上が必要です。

### ステップ 21 ストレージ階層の分散を検証する

ストレージ階層の割り当て分析とローカルストレージの削減検証を通じて、移行が成功したことを確認します。

```
- - Examine storage tier assignments
dt+
- - Historical partitions should show "pgaa" with "0 bytes"
- - Recent partitions should show "heap" with actual storage allocation
```

予想される結果は、「pgaa」アクセス方法とゼロローカルストレージを使用した履歴パーティションを示し、分析層への正常な移行を確認します。最近のパーティションは、ローカルストレージとの「ヒープ」アクセスを維持して、最適な操作パフォーマンスを実現します。

ステップ 22 移行後のストレージ分析と ROI 計算

包括的なストレージ分析を実行して、最適化結果を定量化し、投資収益率を計算します。

- - Run same **storage** analysis query as Step 18 to capture post-migration metrics

- - Verify **data** integrity across **storage** tiers

**SELECT COUNT(\*) AS count**, ROUND(**SUM**(revenue)) **AS** total\_revenue **FROM** partitioned\_table;

- - Should maintain: **count=10000000**, **total\_revenue=5000000500**

ストレージの最適化の結果

Metric		Before Migration		After Migration	Savings
Local Storage (Historical)		144MB		16MB	128MB (89% reduction)
Storage Cost Profile		100%	premium storage	11% premium, 89% standard	60-80% cost reduction
Query Performance		Uniform fast access		Recent fast, historical acceptable	Optimized by access pattern
Administrative head	Over-head	Manual	management	Automated lifecycle	Eliminated manual intervention

自動階層化は、ストレージ階層全体で透過的なクエリアクセスを維持しながら、インフラストラクチャコストの大幅な削減を実現します。最近のデータは、操作要件に合わせて高性能ストレージに残りますが、履歴データは、費用対効果の高い分析ストレージ用に最適化されます。

7.3.7 トラブルシューティングと検証

一般的な問題と解決策

ストレージロケーションアクセス資格情報構成と外部ストレージエンドポイントへのネットワーク接続を確認します。意図的なエラーテストを使用して、バケットのアクセス可能性と認証を確認します。

**オフロード処理遅延**バックグラウンドタスクキューをモニタし、開発環境に合わせて `bdr.taskmgr_nap_time` を調整します。実稼働システムは、リソース最適化のデフォルト設定を維持する必要があります。

**データの整合性に関する懸念**行数と集計計算を使用して、ストレージ階層の移行全体でのデータの一貫性を確認します。分析ストレージは、フォーマット変換プロセスを通じて完全なデータの忠実性を維持します。

**クエリパフォーマンスのバリエーション**レイテンシーよりスループットについて最適化された分析ストレージを使用して、ストレージ階層間で異なるパフォーマンス特性が予想されます。ローカルストレージの最近のデータは、最適な応答時間を維持します。

## パフォーマンスモニタリング

ストレージ使用傾向、階層全体のクエリ応答時間、およびバックグラウンドタスクの完了率を追跡します。オフロード操作中の外部ストレージシステムへのネットワーク接続とリソースの使用率を監視します。

アクセスパターンの定期的な分析により、オフロードしきい値を最適化する機会、またはビジネス要件に基づいてさまざまなライフサイクルポリシーを必要とするパーティションを特定する可能性があります。

包括的な構成は、ハイブリッドストレージアーキテクチャ全体で透過的な SQL アクセスを維持しながら、運用パフォーマンス要件とストレージコストの最適化のバランスを保つ自動データ階層化機能を提供します。

## 7.4 How-To Integrate with Third-Party Iceberg Catalogs

### 7.4.1 概要

このガイドでは、サードパーティの Apache Iceberg カタログと連携するように PostgreSQL Advanced Analytics PGAA を構成し、分析ワークロードのシームレスなデータレイクの統合を可能にする手順を提供します。PGAA は、外部カタログサービスを介した Iceberg テーブルの読み取りおよび書き込みをサポートし、PostgreSQL 展開と並行して既存のデータレイクインフラストラクチャを活用できます。

### 7.4.2 コンテキストと目標

#### このガイドの内容

このドキュメントでは、次の方法を説明します。

- サードパーティの Iceberg カタログに接続するように PGAA を構成する
- PostgreSQL と Iceberg テーブル間の双方向データフローを有効にする
- PGD Postgres Distributed オフロードのカタログ統合を設定する
- 外部分析ツールとの互換性を確保する

#### サポートされているカタログタイプ

PGAA は現在以下との統合をサポートしています。

- **Snowflake Open Data Catalog Apache Polaris** 手動 OAuth トークン管理との互換性を確認しました
- **AWS S3 テーブル** テーブルバケットを使用したネイティブ統合

### 7.4.3 前提条件

この統合を開始する前に、次のことを確認してください。

#### 必要なコンポーネント

- 適切なライセンスを使用したアクティブな PGAA インストール
- BDR 双方向レプリケーションが構成された PostgreSQL クラスター
- PGAA カタログ管理機能への管理アクセス

### Snowflake Polaris 統合の場合

- Snowflake の試用版または実稼働アカウント
- 組織の AWS アカウントで設定された AWS S3 バケット
- S3 バケットにアクセスするための Snowflake の IT 承認
- Snowflake からの OAuth クライアント資格情報

### AWS S3 テーブル統合の場合

- S3 テーブルが有効になった AWS アカウント
- S3 テーブル管理のための適切な IAM 権限
- テーブルバケットとして構成された S3 バケット

## 7.4.4 実際のアプリケーション

この統合により、いくつかの重要なユースケースが可能になります。

1. 分析オフロード運用中の PostgreSQL データベースから費用対効果の高いオブジェクトストレージに履歴データを移動します
2. データレイク統合データを移動せずに PostgreSQL から Iceberg テーブルを直接照会します
3. ハイブリッドアナリティクスリアルタイムの PostgreSQL データと統合クエリでの Iceberg データの履歴を結合します
4. クロスプラットフォーム分析共通の Iceberg カタログを介して PostgreSQL と他の分析エンジン間でデータを共有する

## 7.4.5 構成手順

### Snowflake Polaris 統合のセットアップ

#### ステップ 1 OAuth トークンを取得する

現在の制限のため、OAuth トークンは手動で取得する必要があります、有効期間は 1 時間です。次の Python スクリプトを使用して、トークンを取得します。

```
from pyiceberg.catalog import load_catalog
from pyiceberg.catalog.rest import RestCatalog

Replace with your Snowflake OAuth credentials

CLIENT_ID = "your_client_id"
CLIENT_SECRET = "your_client_secret"
TOKEN = f"{CLIENT_ID}:{CLIENT_SECRET}"

Replace with your Snowflake Polaris endpoint
```

(次のページに続く)

(前のページからの続き)

```
URI = "https://your-account.snowflakecomputing.com/polaris/api/catalog"

catalog = RestCatalog(
 name="PGAA Test",
 warehouse="pgaa_test",
 uri=URI,
 credential=TOKEN,
 scope="PRINCIPAL_ROLE:read_write_all"
)

This will print your temporary token

print(catalog.properties["token"])
```

**重要** トークン出力を保存します。次のステップで必要になります。

## ステップ 2 PGAA カタログを構成する

次の SQL コマンドを実行して、PGAA に Snowflake Polaris カタログを追加および構成します。

```
-- Remove existing catalog if present
SELECT bdr.replicate_ddl_command($$
 SELECT pgaa.delete_catalog(snowflake_polaris);
$$);

-- Add new catalog configuration
SELECT bdr.replicate_ddl_command($$
 SELECT pgaa.add_catalog(
 snowflake_polaris,
 iceberg-rest,
 {
 "url": "https://your-account.snowflakecomputing.com/polaris/api/catalog",
 "token": "YOUR_OAUTH_TOKEN_HERE",
 "warehouse": "pgaa_test"
 }
);
$$);

-- Import catalog metadata
SELECT bdr.replicate_ddl_command($$
 SELECT pgaa.import_catalog(snowflake_polaris);
$$);
```

### ステップ 3 書き込み権限を構成する

BDR ノードグループを構成することにより、カタログへの書き込み操作を有効にします。

```

SELECT bdr.alter_node_group_option(
 (SELECT node_group_name FROM bdr.node_group
 WHERE node_group_name != world LIMIT 1),
 analytics_write_catalog,
 snowflake_polaris
);

```

### AWS S3 テーブル統合のセットアップ

S3 テーブル統合の場合、[AWS S3 Tables Getting Started Guide](#) に従ってテーブルバケットを作成し、PGAA の構成時にカタログタイプとして iceberg-s3tables を使用します。

```

SELECT bdr.replicate_ddl_command($$
 SELECT pgaa.add_catalog(
 s3_tables_catalog,
 iceberg-s3tables,
 {"bucket": "your-table-bucket",
 "region": "us-east-1",
 "access_key": "YOUR_ACCESS_KEY",
 "secret_key": "YOUR_SECRET_KEY"}
);
$$);

```

## 7.4.6 検証とテスト

構成後、次の操作をテストして、統合を確認します。

#### カタログからの読み取り

```

- - List available tables
SELECT * FROM pgaa.catalog_tables(your_catalog_name);

- - Query an Iceberg table
SELECT * FROM your_catalog_name.schema_name.table_name LIMIT 10;

```

#### カタログへの書き込み

```

- - Create a new Iceberg table
CREATE TABLE your_catalog_name.schema_name.new_table AS
SELECT * FROM your_postgresql_table;

```

## PGD オフロード

PGD が保持ポリシーに従って、構成済みのカタログにデータを正常にオフロードできることを確認します。

## 外部ツールの互換性

サードパーティの分析ツールが、PGAA を介して作成されたテーブルとビューの両方を読み取ることができることをテストします。

### 7.4.7 既知の制限

- **トークン管理** Snowflake Polaris トークンは 1 時間後に有効期限が切れるため、手動で更新する必要があります
- **名前付けの制限** 名前にスペースが含まれるカタログはクエリエラーが発生する場合があります
- **OAuth フロー** OAuth クライアント資格情報フローは、PGAA の外部で手動で実行する必要があります

### 7.4.8 障害対応

#### よくある問題

##### トークン有効期限エラー

- **症状** 約 1 時間後にクエリが認証エラーで失敗します
- **解決策** OAuth トークンを再生成し、カタログ構成を更新します

##### カタログのインポートの失敗

- **症状** `pgaa.import_catalog()` はエラーを返します
- **解決策** ネットワーク接続を確認し、カタログ URL がすべての PostgreSQL ノードからアクセスできることを確認します

##### クエリパフォーマンスの問題

- **症状** Iceberg テーブルにアクセスする際のクエリパフォーマンスの低下
- **解決策** 適切なパーティションプルーニングを保証し、頻繁にアクセスされるデータのマテリアライズドビューの作成を検討します

### 7.4.9 次のステップ

カタログ統合が正常に構成された後

1. 実稼働環境のトークンリフレッシュ手順を確立する
2. カatalog接続の監視を構成する
3. 自動オフロードのためのデータライフサイクルポリシーを実装する
4. PostgreSQL と Iceberg の混合ワークロードのクエリパターンを最適化する

## 7.5 How-To Lakehouse Read With/Without A Catalog

### 7.5.1 概要

このガイドでは、カタログ統合の有無にかかわらず、データレイクフォーマットにアクセスするために PGAA Lakehouse クラスターを構成する方法を説明します。PGAA は、Delta Lake および Apache Iceberg テーブルへの直接アクセスをサポートし、データを移動せずにオブジェクトストレージで分析ワークロードを有効にします。

### 7.5.2 前提条件

#### システム要件

- PGAA Lakehouse クラスターがプロビジョニングされアクセス可能
- PostgreSQL File System PGFS バージョン 2.0.1 以降を推奨
- カタログおよびストレージ構成の管理者権限

#### アクセス要件

- カタログ統合の場合 ターゲットカタログサービスの有効な資格情報
- 直接アクセスの場合ターゲット S3 バケットまたはストレージ場所の読み取り権限
- ストレージエンドポイントへのネットワーク接続

### 7.5.3 構成アプローチ

PGAA は、レイクハウスデータアクセスの 3 つの主要なアプローチをサポートしています。

1. **カタログ管理アクセス** 外部カタログサービスとの完全な統合
2. **ダイレクトデルタレイクアクセス** カタログ依存関係なしでデルタテーブルを照会します
3. **Iceberg のダイレクトアクセス** メタデータファイルを使用して Iceberg テーブルを照会します

### 7.5.4 AWS S3 テーブルの統合

#### Python クライアント構成

AWS S3 テーブルでは、Python クライアントの REST API の互換性が必要です。次のように PyIceberg を構成します。

```
from pyiceberg.catalog import load_catalog

Define S3 Tables parameters

REGION = "eu-north-1" # Your S3 Tables bucket region
ARN = "arn:aws:s3tables:eu-north-1:0123456789:bucket/your-bucket"
```

(次のページに続く)

(前のページからの続き)

```
s3tables_catalog = load_catalog(
 "s3tables_catalog",
 **{
 "type": "rest",
 "warehouse": ARN,
 "uri": f"https://s3tables.{REGION}.amazonaws.com/iceberg",
 "rest.sigv4-enabled": "true",
 "rest.signing-name": "s3tables",
 "rest.signing-region": REGION,
 }
)
```

注 PyIceberg バージョン ≤ 0.9.1 は、各 API 呼び出しで資格情報の解決を実行し、カタログ定義中に S3 オプションを直接渡すことはできません。

## PGAA ネイティブ構成

PGAA はネイティブの Rust S3 Tables クライアントを実装し、REST プロキシのオーバーヘッドを排除します。

```
SELECT bdr.replicate_ddl_command($$
 SELECT pgaa.delete_catalog(s3tables_catalog);
 SELECT pgaa.add_catalog(
 s3tables_catalog,
 iceberg-s3tables,
 {
 "arn": "arn:aws:s3tables:eu-north-1:0123456789:bucket/your-bucket",
 "region": "eu-north-1"
 }
);
 SELECT pgaa.import_catalog(s3tables_catalog);
$$);
```

## 主な違い

- カタログタイプ iceberg-s3tables iceberg-rest ではありません
- 必須パラメーター ARN とリージョンのみ
- 認証 AWS 資格情報チェーンプロファイル、環境変数、IMDS
- 明示的な資格情報はサポートされていません。SDK 資格情報の解決に依存する

## 7.5.5 デルタレイクへの直接アクセス

### ストレージ場所の設定

Delta Lake テーブルの格納場所を構成します。

```

- - PGFS 2.0.1 and later
SELECT pgfs.create_storage_location(
 biganimal-sample-data,
 s3://beacon-analytics-demo-data-us-east-1-prod,
 {"aws_skip_signature": "true"}
);

- - Legacy syntax (before PGFS 2.0.1)
SELECT pgfs.create_storage_location(
 biganimal-sample-data,
 s3://beacon-analytics-demo-data-eu-west-1-prod,
 NULL, -- managed storage location ID (unused)
 {"aws_skip_signature": "true"}, -- options
 {} -- credentials
);

```

### デルタテーブルの作成

PGAA テーブルアクセス方法を使用して Delta Lake テーブルをマップします。

```

- - Create customer table
CREATE TABLE public.customer () USING PGAA
WITH (
 pgaa.storage_location = biganimal-sample-data,
 pgaa.path = tpch_sf_1/customer
);

- - Verify table access
SELECT count(*) FROM public.customer;

- - Create additional TPC-H tables
CREATE TABLE public.lineitem () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/lineitem);

CREATE TABLE public.orders () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/orders);

CREATE TABLE public.nation () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/nation);

```

### 大規模なデータセットの操作

スケールテスト用に別のスキーマを構成します。

```

CREATE SCHEMA tpch;

-- Create 1TB scale factor tables
CREATE TABLE tpch.lineitem () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1000/
→lineitem);

CREATE TABLE tpch.customer () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1000/
→customer);

-- Additional tables follow same pattern...

```

## 7.5.6 氷山への直接アクセス

### Iceberg ストレージの構成

```

-- Create storage location for Iceberg data
SELECT pgfs.create_storage_location(
 biganimal-sample-data-dev,
 s3://beacon-analytics-demo-data-us-east-1-dev,
 {"aws_skip_signature": "true"}
);

-- Create Iceberg table reference
CREATE TABLE iceberg_table () USING PGAA
WITH (
 pgaa.storage_location = biganimal-sample-data-dev,
 pgaa.path = iceberg-example/default.db/iceberg_table,
 pgaa.format = iceberg -- Explicitly specify format
);

-- Query Iceberg data
SELECT * FROM iceberg_table ORDER BY key ASC;

```

## 7.5.7 カタログ添付ファイル

### 既存のカタログへの接続

構成済みのカタログからデータを接続および照会します。

```

-- Add REST catalog
SELECT pgaa.add_catalog(
 lakekeeper-test,
 iceberg-rest,

```

(次のページに続く)

(前のページからの続き)

```
{
 "url": "https://catalog-endpoint.example.com",
 "token": "your-api-token",
 "warehouse": "warehouse-id",
 "danger_accept_invalid_certs": "true"
}
);

-- Attach catalog to current session
SELECT pgaa.attach_catalog(lakekeeper-test);

-- Query catalog tables
SELECT COUNT(*) FROM tpch_sf_1.lineitem;
```

### カタログアクセスの検証

```
-- Check available schemas
SELECT * FROM information_schema.schemata
WHERE schema_name LIKE tpch%;

-- Verify offloaded data views
SELECT * FROM partitioned_table_offloaded LIMIT 10;
```

## 7.5.8 クエリーの検証

### サンプル分析クエリー

複雑な結合を使用した構成を検証する

```
SELECT
 c_custkey,
 c_name,
 sum(l_extendedprice * (1 - l_discount)) AS revenue,
 c_acctbal,
 n_name,
 c_address,
 c_phone,
 c_comment
FROM
 customer,
 orders,
 lineitem,
 nation
WHERE
```

(次のページに続く)

(前のページからの続き)

```
c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate >= CAST(1993-10-01 AS date)
AND o_orderdate < CAST(1994-01-01 AS date)
AND l_returnflag = R
AND c_nationkey = n_nationkey
GROUP BY
 c_custkey, c_name, c_acctbal,
 c_phone, n_name, c_address, c_comment
ORDER BY
 revenue DESC
LIMIT 20;
```

#### 予想される結果

- Customer#000057040 収益 734235.2455
- Customer#000143347 収益 721002.6948
- Customer#000060838 収益 679127.3077

## 7.5.9 ベストプラクティス

### ストレージ構成

1. 可能な場合は、PGFS 2.0.1+の簡素化された構文を使用します
2. 最適なパフォーマンスを得るためにリージョンに適切なエンドポイントを構成する
3. AWS 資格情報チェーンを介して適切な資格情報管理を実装する

### カタログ管理

1. REST カタログの OAuth トークンを定期的に更新する
2. 可能な場合はネイティブクライアントを使用します S3 テーブル
3. 構成変更後のカタログメタデータのインポート

### パフォーマンスの最適化

1. 大規模なデータセットを適切に分割する
2. 頻繁にアクセスされるデータのマテリアライズドビューを作成する
3. 最適化の機会についてクエリ実行プランを監視する

## 7.5.10 障害対応

### よくある問題

#### ストレージアクセスエラー

- S3 バケットの権限とネットワーク接続を確認する
- AWS 認証の資格情報チェーン構成を確認する

#### カタログ接続の障害

- カタログエンドポイントの URL と資格情報を検証する
- 適切な SSL 証明書検証設定を確認します

#### クエリパフォーマンスの低下

- テーブル統計とパーティションプルーニングを確認します
- データの局所性とキャッシュ戦略を検討する

## 7.5.11 次のステップ

1. レイクハウスクエリのモニタリングを構成する
2. マテリアライズドビューのデータ更新スケジュールを確立する
3. インテリジェントなキャッシュを介したコスト最適化を実装する
4. カタログアクセス制御のガバナンスポリシーを開発する

## 7.6 Configure Analytics Storage and Data Tiering with PGAA and PGD

### 7.6.1 目次

#### パート 1 基礎のセットアップ

- Step 1: Initialize PGAA Extension and Performance Settings
- Step 2: Configure Storage Locations
- Step 3: Configure Analytics Write Location
- Step 4: Verify Seafoam Query Engine Activation

#### パート 2 読み取り専用分析既存のデータへのアクセス

- Step 5: Delta Lake Table Access
- Step 6: Execute Analytics Queries Across Delta Lake Data
- Step 7: Iceberg Table Access
- Step 8: Analyze Storage Characteristics

### パート 3 トランザクションテーブル分析のオフロード

- Step 9: Create and Populate Transactional Table
- Step 10: Configure Analytics Offload
- Step 11: Create Analytics Reader Interface
- Step 12: Convert Transactional Table to Analytics-Only ]

### パート 4: 高度なテーブル管理操作

- Step 13: Restore Table from Analytics to Transactional Storage
- Step 14: Handle Data Updates with Analytics Synchronization
- Step 15: Implement Partial Table Offload

### パート 5 分析オフロードを使用した **AutoPartition**

- Step 16: Configure Automated Partitioning
- Step 17: Load Test Data and Analyze Baseline Storage
- Step 18: Baseline Storage Analysis
- Step 19: Configure Automated Analytics Offload
- Step 20: Monitor Automated Offload Operations
- Step 21: Validate Storage Tier Distribution
- Step 22: Post-Migration Storage Analysis and ROI Calculation

### パート 6 カタログベースの操作

- Step 23: Environment Cleanup and Catalog Setup
- Step 24: External Data Loading and Catalog Configuration
- Distributed Catalog Access
- Step 26: Dynamic Catalog Discovery and Management
- Catalog-Managed Analytics Operations
- Cross-Node Data Access Validation

## 7.6.2 パート 7 外部ツールの統合

- *PyIceberg Catalog Access*
- Unified Cold Storage View

### 7.6.3 ステップの概要

このガイドでは、6つの進行的なフェーズを介して完全な分析ストレージアーキテクチャを実装します。

**基礎ステップ 1~4** PGAA インフラストラクチャを構成し、ストレージの場所を確立し、レイクハウス操作のクエリエンジンをアクティブにします。

**読み取り専用分析ステップ 5~8** データを移動せずに既存の Delta Lake および Iceberg データセットにアクセスし、即時分析機能を示します。

**基本的なオフロードステップ 9-12** データの完全性検証とフォーマット変換を使用して、分析ストレージへのトランザクションテーブルオフロードを実装します。

**高度な管理ステップ 13-15** 複雑なシナリオの双方向データの移動、更新の同期、および選択的なアーカイブ戦略をマスターします。

**\*\*自動階層化ステップ 16~22** 自動化されたライフサイクル管理を使用して AutoPartition を展開し、ストレージ最適化の前後のメトリックをキャプチャします。

**カタログ統合ステップ 23-28** エンタープライズ規模のデータ共有、マルチノードアクセス、およびサードパーティツールの互換性のために外部カタログを統合します。

---

### 7.6.4 前提条件

#### 技術的要件

PGD 高可用性を備えた PostgreSQL は、カタログ管理、パーティション操作、およびストレージ構成の管理者権限で動作している必要があります。PGAA 拡張機能は、S3 互換エンドポイントを含む外部ストレージシステムにネットワーク接続してインストールする必要があります。十分な帯域幅は、オフロードプロセス中のストレージ階層間のデータ転送操作をサポートします。

#### 知識要件

PostgreSQL のパーティショニングの概念、システムカタログ構造、および分散データベース管理を理解することにより、効果的な実装が可能になります。オブジェクトストレージシステム、レイクハウスアーキテクチャ、および分析ワークロードパターンに精通していると、構成上の決定を最適化するのに役立ちます。SQL パフォーマンス分析の基本的な知識は、トラブルシューティングと最適化アクティビティをサポートします。

#### 環境設定

マルチプルのノードを備えたアクティブ PGD クラスターは高可用性の基盤を提供し、PGAA 拡張機能は分析ストレージの統合を可能にします。ネットワークセキュリティ構成は、適切なファイアウォールと認証設定を使用して外部ストレージエンドポイントへのアクセスを許可する必要があります。監視インフラストラクチャは、バックグラウンドタスクの処理とストレージ使用パターンを追跡する必要があります。

## 7.6.5 セットアップ: 分析インフラストラクチャの構成

### ステップ 1 PGAA 拡張機能とパフォーマンス設定を初期化する

テストおよび運用展開シナリオに最適化された設定で分析インフラストラクチャを構成します。

```
-- Setup PGD configuration for efficient testing cycles
SET bdr.taskmgr_nap_time = 10000; -- AutoPartition worker runs every 10s vs default 290s
SET pgaa.max_replication_lag_s = 5; -- Analytics replication lag threshold
SET pgaa.flush_task_interval_s = 5; -- Frequent storage flush for testing

-- Initialize PGAA extension
CREATE EXTENSION IF NOT EXISTS pgaa CASCADE;
```

パフォーマンス設定は、開発およびテストフェーズ中のバックグラウンドタスク処理を加速します。実稼働環境は、システムリソースと操作要件に対してこれらの設定を評価する必要があります。PGAA 拡張機能は、分析ストレージ統合とレイクハウステーブル管理の基盤を提供します。

### ステップ 2 ストレージの場所を構成する

適切なセキュリティ構成を使用して、読み取り専用の分析アクセスと書き込み操作の両方の保存場所を確立します。

```
-- Create read-only storage location for existing Delta Lake data
SELECT bdr.replicate_ddl_command(
 SELECT pgfs.create_storage_location(
 biganimal-sample-data,
 s3://beacon-analytics-demo-data-us-east-1-prod,
 {"aws_skip_signature": "true"}
);
);

-- Create development storage location for Iceberg tables
SELECT bdr.replicate_ddl_command(
 SELECT pgfs.create_storage_location(
 biganimal-sample-data-dev,
 s3://beacon-analytics-demo-data-us-east-1-dev,
 {"aws_skip_signature": "true"}
);
);
```

ストレージ場所の構成は、読み取り専用の分析アクセスと書き込み操作の両方の基盤を確立します。`aws_skip_signature` 設定はパブリックバケットに適用されますが、本番環境では、IAM ロールまたは資格情報ファイルを介した適切な AWS 資格情報構成が必要です。

### ステップ 3 アナリティクスの書き込み場所を構成する

自動データオフロード操作用のプライマリ分析保存場所を設定します。

```
- - Create appliance storage location with dynamic path construction
SELECT bdr.replicate_ddl_command($$
 SELECT pgfs.create_storage_location(
 appliance-bucket,
 $BUCKET_URL || / ||
 (SELECT node_group_name FROM bdr.node_group WHERE node_group_name != world LIMIT
→1) ||
 /analytics-offload
)
 $$);

- - Validate storage location accessibility
CREATE TABLE sentinel_sl_read_test () USING PGAA
WITH (pgaa.format = iceberg, pgaa.storage_location = appliance-bucket, pgaa.path =
→nonexistent);
- - Should return ERROR: No valid Iceberg metadata file found (confirms bucket access)

- - Set as primary analytics write location
SELECT bdr.alter_node_group_option(
 (SELECT node_group_name FROM bdr.node_group WHERE node_group_name != world LIMIT 1),
 analytics_storage_location,
 appliance-bucket
);
```

動的パス構成には、マルチテナント環境での競合を防ぐための PGD ノードグループ名が含まれます。意図的なエラーテストを介した保存場所の検証により、適切な認証とネットワーク接続が確認されます。分析の保存場所を設定すると、レイクハウス操作の Seafoal クエリエンジンがアクティブになります。

### ステップ 4 Seafoal クエリエンジンのアクティブ化を確認する

分析クエリエンジンが起動し、レイクハウス操作の準備ができていることを確認します。

```
- - Verify Seafoal query engine is running
SELECT application_name FROM pg_stat_activity WHERE application_name LIKE %seafoal%;
- - Must show pgd seafoal_manager for proper operation
```

Seafoal クエリエンジンは、カラムナストレージ形式全体で分析ワークロードの最適化された実行を提供します。レイクハウスアーキテクチャのパフォーマンスの最適化には、適切なアクティベーションが不可欠です。

## 7.6.6 読み取り専用分析: 既存のデータへのアクセス

### ステップ 5 デルタレイクテーブルアクセス

データの移動や ETL プロセスなしで、既存の Delta Lake データセットへの直接アクセスを提供する分析テーブルを作成します。

```
-- Create Delta Lake analytics tables pointing to existing data
CREATE TABLE public.customer () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/customer);

CREATE TABLE public.lineitem () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/lineitem);

CREATE TABLE public.nation () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/nation);

CREATE TABLE public.orders () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/orders);

CREATE TABLE public.part () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/part);

CREATE TABLE public.partsupp () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/partsupp);

CREATE TABLE public.region () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/region);

CREATE TABLE public.supplier () USING PGAA
WITH (pgaa.storage_location = biganimal-sample-data, pgaa.path = tpch_sf_1/supplier);
```

これらのテーブル定義は、データの移行や形式変換を必要とせずに、既存の Delta Lake データへの PostgreSQL インターフェイスを作成します。PGAA アクセス方法は、Delta Lake フォーマットの解析とクエリ最適化を自動的に処理します。

### ステップ 6 Delta Lake データ全体で分析クエリを実行する

レイクハウスのパフォーマンス特性を示す複雑なマルチテーブルクエリを介して分析機能を検証します。

```
-- Complex analytics query spanning multiple Delta Lake tables
SELECT
 c_custkey, c_name,
 sum(l_extendedprice * (1 - l_discount)) AS revenue,
 c_acctbal, n_name, c_address, c_phone, c_comment
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey
```

(次のページに続く)

(前のページからの続き)

```

 AND l_orderkey = o_orderkey
 AND o_orderdate >= CAST(1993-10-01 AS date)
 AND o_orderdate < CAST(1994-01-01 AS date)
 AND l_returnflag = R
 AND c_nationkey = n_nationkey
GROUP BY c_custkey, c_name, c_acctbal, c_phone, n_name, c_address, c_comment
ORDER BY revenue DESC
LIMIT 20;

```

このクエリは、使いやすい SQL 構文を維持しながら、複数の Delta Lake テーブルにわたる複雑な結合操作を示しています。クエリの実行は Seafoal 分析エンジンを活用して、列指向データ形式全体でパフォーマンスを最適化します。

## ステップ 7 Iceberg テーブルアクセス

ダイレクトストレージの場所マッピングを介して Apache Iceberg テーブルへのアクセスを構成します。

```

-- Create Iceberg analytics table
CREATE TABLE iceberg_table () USING PGAA
WITH (
 pgaa.storage_location = biganimal-sample-data-dev,
 pgaa.path = iceberg-example/default.db/iceberg_table,
 pgaa.format = iceberg
);

-- Query Iceberg data
SELECT * FROM iceberg_table ORDER BY key ASC;

```

Iceberg フォーマット仕様により、スキーマの進化、パーティション管理、タイムトラベル機能を含む完全なメタデータサポートを備えた Apache Iceberg テーブルにアクセスできます。明示的なフォーマット宣言により、Iceberg メタデータ構造の適切な解析が保証されます。

## ステップ 8 ストレージ特性を分析する

分析機能を使用して、レイクハウステーブルのストレージ使用率とパフォーマンス特性を理解します。

```

-- Analyze storage statistics for Delta Lake tables
SELECT * FROM pgaa.lakehouse_table_stats(lineitem::regclass);

-- Analyze storage statistics for Iceberg tables
SELECT * FROM pgaa.lakehouse_table_stats(iceberg_table::regclass);

```

ストレージ統計は、圧縮率、ファイル構成、およびクエリパフォーマンス特性に関する洞察を提供し、最適化の決定を通知します。これらのメトリックは、分析ワークロードのキャパシティプランニングとパフォーマンス調整をサポートします。

## 7.6.7 トランザクションテーブル分析オフロード

### ステップ 9 トランザクションテーブルを作成してデータを入力する

オフロードテストと検証用の代表データを使用してトランザクションテーブルを確立します。

```
-- Create transactional table for offload testing
CREATE TABLE transactional_table (
 key BIGINT,
 inserted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
 revenue DOUBLE PRECISION,
 comment TEXT,
 PRIMARY KEY (key, inserted_at)
);

-- Insert test data with temporal distribution
INSERT INTO transactional_table (
 SELECT
 g,
 2010-01-01 00:00:00::timestamp + INTERVAL 1 second *
 (EXTRACT (EPOCH FROM (CURRENT_TIMESTAMP - 2010-01-01 00:00:00::timestamp)) /
→10000000 * g),
 g * 0.001,
 encode(sha256(g::text::bytea), hex)
 FROM generate_series(1, 10000000) g
);
```

データ生成は、パフォーマンステストに十分なボリュームを提供しながら、現実的な時間分布パターンを作成します。SHA256 ハッシュ生成は、実稼働システムで見つかった可変長テキストコンテンツをシミュレートします。

### ステップ 10 分析オフロードを構成する

コストの最適化とパフォーマンスのスケーリングのために、トランザクションデータの分析ストレージへの自動レプリケーションを有効にします。

```
-- Enable analytics replication for transactional table
ALTER TABLE transactional_table SET (pgd.replicate_to_analytics = TRUE);

-- Monitor replication progress
SELECT * FROM bdr.analytics_table;

-- Wait until table appears in results (~120s for initial sync)
```

アナリティクスレプリケーションは、構成された保存場所にトランザクションデータの Iceberg 表現を作成します。初期同期プロセスはフォーマット変換を処理し、データの一貫性のために進行中のレプリケーションストリームを確立します。

## ステップ 11 アナリティクスリーダーインターフェイスを作成する

専用のリーダーテーブルを介して、分析層データへの直接アクセスを確立します。

```
- - Create reader interface for analytics data
CREATE TABLE transactional_table_analytics_reader ()
USING PGAA
WITH (
 pgaa.format = iceberg,
 pgaa.storage_location = appliance-bucket,
 pgaa.path = public.transactional_table
);

- - Verify data integrity across storage tiers
SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_revenue
FROM transactional_table_analytics_reader;
- - Should return: count=1000000, total_revenue=500000500
```

分析リーダーは、クエリの透明性を維持しながら、費用対効果の高いストレージへの直接アクセスを提供します。データの完全性検証により、フォーマット変換とレプリケーションプロセスが成功したことが確認されます。

## ステップ 12 トランザクションテーブルをアナリティクス専用に変換する

ローカルデータを破棄しながら、分析ストレージを直接使用するようにトランザクションテーブルを変換します。

```
- - Convert to analytics-only access
ALTER TABLE transactional_table
SET ACCESS METHOD PGAA,
SET (
 pgaa.auto_truncate = true,
 pgaa.format = iceberg,
 pgaa.storage_location = appliance-bucket,
 pgaa.path = public.transactional_table
);

- - Verify conversion maintained data access
SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_revenue FROM transactional_
→table;
```

変換プロセスは、分析層を介した透過的な SQL アクセスを維持しながら、ローカルストレージのオーバーヘッドを排除します。自動切り捨ては、分析ティアの可用性を確認した後、ローカルデータを削除します。

## 7.6.8 高度なテーブル管理操作

### ステップ 13 アナリティクスからトランザクションストレージにテーブルを復元する

ローカルの高パフォーマンスのアクセスが必要なシナリオの分析オフロードの可逆性を示します。

```
- - Restore table to transactional storage
ALTER TABLE transactional_table SET ACCESS METHOD heap, SET (pgd.replicate_to_
→analytics = FALSE);

- - Verify restoration with query plan analysis
EXPLAIN SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_revenue FROM
→transactional_table;

- - Should show Parallel Seq Scan indicating local heap access
```

復元機能は、アクセスパターンまたはパフォーマンス要件を変更するための柔軟性を提供します。クエリプラン分析により、並列処理機能を備えたローカル PostgreSQL ヒープストレージへの移行が確認されました。

### ステップ 14 分析同期を使用してデータの更新を処理する

ストレージ階層全体で一貫性を維持するデータ変更ワークフローを示します。

```
- - Update data in transactional table
UPDATE transactional_table SET revenue = revenue * 2;

- - Re-enable analytics replication to propagate changes
ALTER TABLE transactional_table SET (pgd.replicate_to_analytics = TRUE);

- - Wait for synchronization (~120s) then verify changes propagated
SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_revenue
FROM transactional_table_analytics_reader;

- - Should show doubled revenue: total_revenue=10000001000
```

更新の伝播は、データの変更を効率的に処理しながら、トランザクション層と分析層間の一貫性を維持します。同期プロセスは、現在のデータを使用して分析表現を再作成します。

### ステップ 15 部分テーブルオフロードを実装する

テーブルの履歴部分のみが分析ストレージに移動する選択的なデータアーカイブを実行します。

```
- - Perform partial offload with consistent transaction boundary
BEGIN;
CREATE TABLE transactional_table_before_2020 AS (
 SELECT * FROM transactional_table
 WHERE DATE_PART(year, inserted_at) < 2020
);
```

(次のページに続く)

(前のページからの続き)

```

DELETE FROM transactional_table WHERE DATE_PART(year, inserted_at) < 2020;
COMMIT;

-- Configure partial table for analytics offload
ALTER TABLE transactional_table_before_2020 ADD PRIMARY KEY (key, inserted_at);
ALTER TABLE transactional_table_before_2020 SET (pgd.replicate_to_analytics = TRUE);

-- Convert to analytics-only access
ALTER TABLE transactional_table_before_2020
SET ACCESS METHOD PGAA,
SET (
 pgaa.auto_truncate = true,
 pgaa.format = iceberg,
 pgaa.storage_location = appliance-bucket,
 pgaa.path = public.transactional_table_before_2020
);

-- Verify temporal data distribution
SELECT
 COUNT(*) FILTER (WHERE inserted_at < 2020-01-01) AS before_2020,
 COUNT(*) FILTER (WHERE inserted_at >= 2020-01-01) AS after_2020
FROM transactional_table_before_2020;

```

部分オフロードにより、期間が異なるとストレージ階層要件が異なる場合でも、きめの細かいデータライフサイクル管理が可能になります。トランザクション境界は、アーカイブプロセス中のデータの一貫性を保証します。

## 7.6.9 分析オフロードを使用した AutoPartition

### ステップ 16 自動パーティションを構成する

スケーラブルなデータ組織のための自動ライフサイクル管理を使用して時間ベースのパーティショニングを確立します。

```

-- Create partitioned table structure
CREATE TABLE partitioned_table (
 key BIGINT,
 inserted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
 revenue DOUBLE PRECISION,
 comment TEXT,
 PRIMARY KEY (key, inserted_at)
) PARTITION BY RANGE(inserted_at);

-- Configure AutoPartition without immediate offload
SELECT bdr.autopartition (

```

(次のページに続く)

(前のページからの続き)

```

relation := partitioned_table,
partition_increment := 1 year,
partition_initial_lowerbound := 2010-01-01 00:00:00,
managed_locally := TRUE
);

-- Wait for partition creation across cluster
SELECT bdr.autopartition_wait_for_partitions_on_all_nodes(partitioned_table::regclass,
↪ CURRENT_TIMESTAMP::text);

```

AutoPartition 構成は、一般的なデータアクセスパターンに合わせた境界を持つ年次パーティションを作成します。初期構成ではオフロード設定を除外して、セットアップおよびデータの読み込みフェーズ中の早すぎるデータの移行を防ぎます。

### ステップ 17 テストデータのロードとベースラインストレージの分析

パーティションに代表的なデータを入力し、最適化分析のベースラインストレージメトリックを確立します。

```

-- Load test data across temporal partitions
INSERT INTO partitioned_table (
 SELECT
 g,
 2010-01-01 00:00:00::timestamp + INTERVAL 1 second *
 (EXTRACT (EPOCH FROM (CURRENT_TIMESTAMP - 2010-01-01 00:00:00::timestamp)) /
↪ 1000000 * g),
 g * 0.001,
 encode(sha256(g::text::bytea), hex)
 FROM generate_series(1, 1000000) g
);

```

データのロードにより、予測可能なコストモデリングとパフォーマンス分析を可能にする一貫したストレージ特性を備えた年間パーティションごとに約 65,000 レコードが作成されます。

### ステップ 18 ベースラインストレージ分析

包括的なストレージ分析機能を作成し、比較のためにプレオフロードメトリックをキャプチャします。

```

-- Create optimized row counting function
CREATE OR REPLACE FUNCTION temp_get_table_rowcount(schema_name text, table_name text)
RETURNS bigint AS $$
DECLARE count bigint;
BEGIN
 -- Uses DirectScan for PGAA tables (1.8s vs 20s without optimization)
 EXECUTE FORMAT(SELECT COUNT(*) FROM %I.%I, schema_name, table_name) INTO count;
 RETURN count;

```

(次のページに続く)

```

END $$ LANGUAGE plpgsql;

-- Comprehensive partition storage analysis
WITH partitioned_table_info AS (
 SELECT
 nspc.nspname AS schema_name,
 child.relname AS table_name,
 pg_catalog.pg_get_expr(child.relpartbound, child.oid) AS partition_constraint,
 pg_get_partition_constraintdef(child.oid) AS partition_constraint_full
 FROM pg_inherits inh
 JOIN pg_class parent ON inh.inhparent = parent.oid
 JOIN pg_class child ON inh.inhrelid = child.oid
 JOIN pg_namespace nspp ON parent.relnamespace = nspp.oid
 JOIN pg_namespace nspc ON child.relnamespace = nspc.oid
 WHERE parent.relname = partitioned_table AND nspc.nspname = public
),

pgaa_tables AS (
 SELECT
 m.schema_name, m.table_name, m.format,
 CASE
 WHEN m.storage_location_name IS NOT NULL THEN FORMAT(pgfs://%s/%s, m.storage_
 ↳location_name, m.path)
 WHEN (m.managed_by IS NOT NULL AND m.catalog_namespace IS NOT NULL AND m.
 ↳catalog_table IS NOT NULL)
 THEN FORMAT(%s.%s.%s, m.managed_by, m.catalog_namespace, m.catalog_table)
 END AS path,
 temp_get_table_rowcount(m.schema_name, m.table_name) AS num_rows,
 s.total_size AS total_size,
 pti.partition_constraint
 FROM pgaa.get_all_analytics_table_settings() m
 LEFT JOIN LATERAL pgaa.lakehouse_table_stats(FORMAT(%I.%I, m.schema_name, m.table_
 ↳name)::regclass) s ON TRUE
 JOIN partitioned_table_info pti ON m.schema_name = pti.schema_name AND m.table_name_
 ↳= pti.table_name
 WHERE m.storage_location_name IS NOT NULL OR (m.managed_by IS NOT NULL AND m.
 ↳catalog_namespace IS NOT NULL AND m.catalog_table IS NOT NULL)
),

local_tables AS (
 SELECT
 pn.nspname AS schema_name, pc.relname AS table_name,
 temp_get_table_rowcount(pn.nspname, pc.relname) AS num_rows,
 pg_total_relation_size(FORMAT(%I.%I, pn.nspname, pc.relname)::regclass) AS total_
 ↳size,

```

(前のページからの続き)

```

 pti.partition_constraint
FROM pg_class pc
 JOIN pg_namespace pn ON pc.relnamespace = pn.oid
 JOIN pg_am am ON am.oid = pc.relam
 JOIN partitioned_table_info pti ON pn.nspname = pti.schema_name AND pc.relname =
→pti.table_name
WHERE pn.nspname NOT IN (pg_catalog, information_schema, pg_toast, pg_internal,
→pgaa, pgfs, bdr)
 AND pc.relkind = r AND am.amname != pgaa
),

all_leaf_tables AS (
 SELECT schema_name, table_name, format AS pgaa_format, path AS pgaa_path, num_rows,
→total_size, partition_constraint FROM pgaa_tables
 UNION
 SELECT schema_name, table_name, NULL AS pgaa_format, NULL AS pgaa_path, num_rows,
→total_size, partition_constraint FROM local_tables
),

final_table_info AS (
 SELECT
 schema_name, table_name,
 CASE
 WHEN pgaa_format = iceberg THEN iceberg
 WHEN pgaa_format = delta THEN delta
 ELSE heap
 END AS type,
 COALESCE(pgaa_path, local) AS location,
 num_rows, total_size AS total_size_bytes,
 SUBSTRING(partition_constraint, FROM []((^[^]+)) AS start_bound,
 SUBSTRING(partition_constraint, TO []((^[^]+)) AS end_bound
 FROM all_leaf_tables
 ORDER BY partition_constraint ASC
)

SELECT * FROM final_table_info;

```

ベースライン結果すべてのパーティションは、履歴パーティションごとに約 10MB、高価なトランザクションストレージの完全なデータセットの合計～154MB の type = 'heap' および location = 'local' を示します。

## ステップ 19 自動分析オフロードを構成する

クエリの透明性を維持しながら、データの経過時間のしきい値に基づいて自動パーティション階層化を有効にします。

```
- - Configure analytics offload for partitions older than one year
SELECT bdr.autopartition (
 relation := partitioned_table,
 partition_increment := 1 year,
 partition_initial_lowerbound := 2010-01-01 00:00:00,
 managed_locally := TRUE,
 analytics_offload_period := 1 year
);
```

1 年間のしきい値は、履歴パーティションを費用対効果の高い分析ティアに移行しながら、最近のデータを高性能ストレージに維持することにより、操作アクセス要件とストレージコストの最適化のバランスをとります。

## ステップ 20 自動オフロード操作を監視する

バックグラウンドの移行の進行状況を追跡し、ストレージ階層の移行の完了ステータスを特定します。

```
- - Monitor offload work queue until completion
SELECT
 wqs.workid,
 wis.ap_wi_started_at,
 wis.ap_wi_finished_at,
 wis.ap_wi_status,
 "partition",
 sql
FROM bdr.taskmgr_local_workitem_status wis
 RIGHT JOIN bdr.taskmgr_local_work_queue_status wqs ON wis.ap_wi_workid = wqs.workid
WHERE relname = public.partitioned_table
AND sql NOT LIKE %bdr%.autopartition_create_partition%
ORDER BY ap_wi_finished_at DESC;
```

すべての `ap_wi_finished_at` 値が NOT NULL、正常に完了したことを示すまで監視します。処理には、通常、14 の履歴パーティション 2010～2025 年に実行の場合 2023 年に 10 分以上が必要です。

## ステップ 21 ストレージ階層の分散を検証する

ストレージ階層の割り当て分析とローカルストレージの削減検証を通じて、移行が成功したことを確認します。

```
- - Examine storage tier assignments
dt+
- - Historical partitions should show "pgaa" with "0 bytes"
- - Recent partitions should show "heap" with actual storage allocation
```

予想される結果は、「pgaa」アクセス方法とゼロローカルストレージを使用した履歴パーティションを示し、分析層への正常な移行を確認します。最近のパーティションは、ローカルストレージとの「ヒープ」アクセスを維持して、最適な操作パフォーマンスを実現します。

## ステップ 22 移行後のストレージ分析と ROI 計算

包括的なストレージ分析を実行して、最適化結果を定量化し、投資収益率を計算します。

```
- - Run same storage analysis query as Step 18 to capture post-migration metrics
- - Verify data integrity across storage tiers
SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_revenue FROM partitioned_table;
- - Should maintain: count=10000000, total_revenue=5000000500
```

## ストレージの最適化の結果

Metric	Before Migration		After Migration	Savings
<b>Local Storage (Historical)</b>	144MB		16MB	<b>128MB (89% reduction)</b>
<b>Storage Cost Profile</b>	100%	premium storage	11% premium, 89% standard	<b>60-80% cost reduction</b>
<b>Query Performance</b>	Uniform fast access		Recent fast, historical acceptable	<b>Optimized by access pattern</b>
<b>Administrative Overhead</b>	Manual	management	Automated lifecycle	<b>Eliminated manual intervention</b>

自動階層化は、ストレージ階層全体で透過的なクエリアccessを維持しながら、インフラストラクチャコストの大幅な削減を実現します。最近のデータは、操作要件に合わせて高性能ストレージに残りますが、履歴データは、費用対効果の高い分析ストレージ用に最適化されます。

## 7.6.10 カタログベースの操作

### ステップ 23 環境のクリーンアップとカタログセットアップ

以前のテスト構成を削除し、外部カタログ統合を確立することにより、カタログベースの操作のための環境を準備します。

```
- - Clean up previous test configurations
DROP TABLE IF EXISTS partitioned_table CASCADE;
DROP TABLE IF EXISTS transactional_table CASCADE;
DROP TABLE IF EXISTS transactional_table_before_2020 CASCADE;
DROP TABLE IF EXISTS transactional_table_analytics_reader CASCADE;

- - Reset analytics storage location
SELECT bdr.alter_node_group_option(
 (SELECT node_group_name FROM bdr.node_group WHERE node_group_name != world LIMIT 1),
```

(次のページに続く)

(前のページからの続き)

```
analytics_storage_location,
NULL
);
```

環境のクリーンアップは、以前の構成から潜在的な競合を削除しながら、カタログ統合テストのクリーンな状態を保証します。

## ステップ 24 外部データのロードとカタログ構成

PyIceberg データローダーを使用してサンプルデータを外部カタログにロードし、カタログアクセス用に PGD クラスターを構成します。

```
Load sample data into HCP Lakekeeper catalog

python loader.py \
 --ca-cert ca.crt \
 --bucket beacon-analytics-demo-data-eu-west-1-prod \
 --prefix tpch_sf_1/lineitem \
 --endpoint $PARAMS_URI \
 --token $API_WRITE \
 --warehouse $PARAMS_WAREHOUSE \
 --namespace tpch_sf_1 \
 --table-name lineitem
```

外部データの読み込みにより、カタログに Iceberg テーブルが作成され、PostgreSQL インターフェイスからアクセスでき、ハイブリッドカタログ機能を示します。

```
- - Configure catalog connection with authentication
SELECT bdr.replicate_ddl_command($$
 SELECT pgaa.add_catalog(
 lakekeeper-test,
 iceberg-rest,
 {
 "url": "$PARAMS_URI",
 "token": "$API_WRITE",
 "warehouse": "$PARAMS_WAREHOUSE_ID",
 "danger_accept_invalid_certs": "true"
 }
)
$$);

- - Set catalog as analytics write destination
SELECT bdr.alter_node_group_option(
 (SELECT node_group_name FROM bdr.node_group WHERE node_group_name != world LIMIT 1),
 analytics_write_catalog,
```

(次のページに続く)

(前のページからの続き)

```
lakekeeper-test
);

-- Verify Seafoal activation for catalog operations
SELECT application_name FROM pg_stat_activity WHERE application_name LIKE %seafoal%;
-- Should show pgd seafoal_manager
```

カタログ構成は、分散分析操作のクエリエンジン機能をアクティブにしながら、外部の Iceberg カタログへの認証接続を確立します。

## ステップ 25 分散カタログ操作

外部カタログを接続し、分散 PGD クラスターノード全体でのデータのアクセス可能性を検証します。

```
-- Attach catalog for distributed access
SELECT bdr.replicate_ddl_command($$
 SELECT pgaa.attach_catalog(
 lakekeeper-test
)
$$);

-- Validate external data access
SELECT COUNT(*) FROM tpch_sf_1.lineitem;
-- Expected: 6001215 rows

-- Examine sample data structure
SELECT * FROM tpch_sf_1.lineitem LIMIT 1;

-- Test namespace isolation with local transactional table
CREATE TABLE tpch_sf_1.sentinel_tx(value TEXT);
INSERT INTO tpch_sf_1.sentinel_tx VALUES (still alive);
```

分散カタログアタッチメントは、ローカルテーブルと外部テーブル間の名前空間の分離を維持しながら、すべてのクラスターノードにわたって外部データへの均一なアクセスを提供します。センチネルテーブルは、カタログ操作がローカルトランザクションデータを干渉しないことを検証します。

## ステップ 26 動的カタログの検出と管理

動的なカタログコンテンツの更新と新しいテーブルの自動検出を示します。

```
Add additional table to external catalog

python loader.py \
 --ca-cert ca.crt \
 --bucket beacon-analytics-demo-data-eu-west-1-prod \
```

(次のページに続く)

(前のページからの続き)

```
--prefix tpch_sf_1/customer \
--endpoint $PARAMS_URI \
--token $API_WRITE \
--warehouse $PARAMS_WAREHOUSE \
--namespace tpch_sf_1 \
--table-name customer
```

edb\_notranlate\_27 sql-新しいテーブルのアクセシビリティを確認します SELECT \* FROM tpch\_sf\_1.customer LIMIT 1;

- - センチネルテーブルの保持の確認 SELECT \* FROM tpch\_sf\_1.sentinel\_tx; - ' still promote' を返す必要があります

Dynamic catalog discovery enables real-time integration of new datasets while  
 ↳preserving local table isolation. The catalog refresh mechanism supports evolving  
 ↳data architectures without requiring manual configuration updates.

```
``` sql --テストの干渉を防ぐために自動カタログ更新を無効にします SELECT bdr.replicate_ddl_
↳command($$ SELECT pgaa.detach_catalog('lakekeeper-test'); $$);
```

Catalog detachment prevents automatic refresh operations from interfering with subsequent testing while maintaining established table mappings for continued access.

Step 27: Catalog-Managed Analytics Operations

Execute the complete analytics workflow using catalog-managed storage instead of direct storage locations.

```
``` sql -カタログベースのオフロードのトランザクションテーブルを再作成します CREATE TABLEtrans-
actional_table(key BIGINT、insert_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP、収益 DOUBLE
PRECISION、comment TEXT、PRIMARY KEY(key、insert_at));
```

- - テストデータのロード INSERT INTO rotational\_table( SELECT g, '2010-01-01 00:00:00' ::times-  
tamp + INTERVAL '1second' (EXTRACT(EPOCH FROM (CURRENT\_TIMESTAMP - '2010-01-01  
00: 00:00' ::timestamp))/1000000 g), g \* 0.001, encode(sha256(g::text::bytea), 'hex' ) FROMgen-  
erate\_series(1, 1000000) g );
- - カタログへの分析レプリケーションを有効にする ALTER TABLEtransactional\_table SET  
pgd.replicate\_to\_analytics = TRUE)。
- - 初期同期の完了を待機します~120 秒 SELECT \* FROM bdr.analytics\_table;

Catalog-managed operations provide centralized metadata management while maintaining  
 ↳the same operational patterns as direct storage approaches. The external catalog  
 ↳handles schema registration and metadata coordination across different analytics  
 ↳tools.

```
``` sql --カタログ管理アナリティクスリーダーを作成します CREATE TABLE rotational_table_
↳analytics_reader() USING PGAA WITH ( pgaa.format = 'iceberg', pgaa.managed_by =
```

(次のページに続く)

(前のページからの続き)

```

→ 'lakekeeper-test', pgaa.catalog_namespace = 'public', pgaa.catalog_table = '
→ transactional_table' );

- - データの完全性を検証する SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_
→ revenue FROM transactional_table_analytics_reader; -- count=10000000、total_
→ revenue=5000000500 を返す必要があります

- - 元のテーブルをカタログ管理へのアクセスに変換する ALTER TABLE rotational_table SET
→ ACCESS METHOD PGAA, SET ( pgaa.format = 'iceberg', pgaa.managed_by = 'lakekeeper-
→ test', pgaa.catalog_namespace = 'public', pgaa.catalog_table = 'transactional_table
→ ', pgaa.auto_truncate = 'true' );

```

Catalog-managed table access provides the same functionality as direct storage while enabling integration with external analytics tools and providing centralized metadata management across different processing engines.

Step 28: Cross-Node Data Access Validation

Validate that catalog-managed data remains accessible across all PGD cluster nodes for high availability scenarios.

```
`` sql-テストのための非リーダーノードを特定します SELECT * FROM bdr.node_group_routing_summary;
```

- - 非リーダーノードに接続し、データアクセスを検証します - kubectl を使用するか、非リーダーノードへの直接接続を使用します - kubectl -n exec -c postgres -it - psql bdrdb

```
SELECT COUNT(*) FROM tpch_sf_1.lineitem; - 返す必要があります: 6001215
```

```
SELECT * FROM tpch_sf_1.lineitem LIMIT 1; - サンプルデータを返す必要があります
```

```
SELECT COUNT(*) AS count, ROUND(SUM(revenue)) AS total_revenue FROM transactional_table; -
count=10000000、total_revenue=5000000500 を返す必要があります
```

```

Cross-node validation confirms that catalog-managed data access functions correctly
→ across all cluster members, providing high availability for analytics workloads.
Distributed access patterns ensure consistent query results regardless of connection
→ endpoint.

```

```
## External Tool Integration
```

```
### Step 29: PyIceberg Catalog Access
```

```

Validate that external tools can access catalog-managed data created through PGD
→ analytics operations.

```

```

`` python pyiceberg.catalog からの load_catalog pyiceberg.catalog.rest からの load_
→ catalog import RestCatalog import os

```

```

URI = os.getenv("PARAMS_URI") TOKEN = os.getenv("API_WRITE") WAREHOUSE = os.getenv(
→ "PARAMS_WAREHOUSE")

```

(次のページに続く)

```

catalog = RestCatalog( name="PGAA Test", warehouse=WAREHOUSE, uri=URI, token=TOKEN,
ssl={"cabundle": "ca.crt"} )

# 名前空間全体で使用可能なすべてのテーブルをリストする

print("Available tasks:") for namespace in catalog.list_namespaces() : print(f"
→{namespace[0]}: ") for _, catalog.list_tables(namespace) のテーブル print(f" {table}"
)

# PGD で作成されたデータのアクセシビリティを検証する

table = catalog.load_table(("public", "transactional_table")) print(f"トランザクション
テーブル行数: {table.scan().count()}") print("サンプルレコード:", table.scan( limit=1).
→to_arrow())

# AutoPartition 操作からのパーティションデータを検証する

Partition_tables = [t for _, t in catalog.list_tables("public") if t.startswith(
→"partitioned_table")] if Partition_tables print(f"Partition {sample_partition} row
→count: {table.scan().count()}")

```

External tool validation confirms that data written by PGD analytics operations maintains full compatibility with standard Iceberg tooling, enabling integration with diverse analytics ecosystems.

Step 30: Unified Cold Storage View

Create and validate unified views of partitioned data that span both local and analytics storage tiers.

```

``` sql --ユニファイドパーティションビューが存在することを確認します SELECT COUNT(*) FROM pub-
lic.partitioned_table_offloaded;

```

- - ビューが存在しない場合は、カタログを再インポートします `SELECT bdr.replicate_ddl_command(`

```

 SELECT pgaa.import_catalog('lakekeeper - test')

```

```

);

```

- - ホット vs コールドデータの分布を分析する `DO $$ DECLARE total_count INTEGER DEFAULT 0; table_name TEXT; row_count INTEGER; cold_row_count INTEGER DEFAULT 0; BEGIN`
  - ローカルの「ホット」パーティションの行をカウントする `FOR table_name IN ( SELECT relname FROM pg_class WHERE relname LIKE 'partitioned_table%' AND relam = 2 -- ヒープアクセス方法 ) LOOP EXECUTE FORMAT(' SELECT COUNT( *) FROM public.%I ', table_name) INTO row_count; total_count := total_count + row_count; エンドループ。`
  - 分析「コールド」ビューの行をカウントします `EXECUTE 'SELECT COUNT(*) FROM Parti-  
tioned_table_offloaded' INTO cold_row_count;`

```
RAISE NOTICE 'ホットストレージ行 %', total_count; RAISE NOTICE 'Cold storage rows: %', cold_row_count; RAISE NOTICE 'Total rows: %', total_count + cold_row_count; END $$; ``
```

ユニファイドビューは、インテリジェントなデータ配置を通じてパフォーマンスの最適化を維持しながら、複数のストレージ階層にまたがる完全なデータセットへのシームレスなアクセスを提供します。このアーキテクチャにより、複雑なデータフェデレーション要件なしで、業務データと履歴データの両方にわたる包括的な分析が可能になります。

## 7.6.11 トラブルシューティングと検証

### パフォーマンスのモニタリングと最適化

クエリパフォーマンス特性はストレージ階層によって異なり、ローカル PostgreSQL ストレージは低レイテンシーのトランザクション操作用に最適化され、分析ストレージは高スループットの分析ワークロード用に最適化されます。クエリ実行プランを監視して、アクセスパターンを理解し、それに応じて最適化します。

`pgaa.lakehouse_table_stats` によるストレージ使用率分析は、圧縮率、ファイル構成、およびアクセスパターンに関する洞察を提供し、最適化の決定に情報を提供します。定期的な分析は、さらなるコスト削減またはパフォーマンス向上の機会を特定するのに役立ちます。

### 一般的な問題と解決策

**カタログ接続** 外部カタログエンドポイントへのネットワークアクセスを確認し、認証資格情報を検証します。意図的なエラーテストを使用して、適切な接続と権限を確認します。

**バックグラウンドタスク処理** `bdr.taskmgr_local_workitem_status` を介してワークキューのステータスを監視し、リソース最適化のための運用デフォルトを維持しながら、開発環境の処理間隔を調整します。

**ストレージ階層同期** 分散クラスターノード全体で一貫したストレージ階層の割り当てを保証し、行カウントと集計計算を介して移行操作後のデータの整合性を検証します。

**外部ツールの互換性** PyIceberg、PySpark、およびその他のエコシステムコンポーネントを使用した直接テストを介して、カタログ管理テーブルが外部分析ツールとの完全な互換性を維持していることを検証します。

### セキュリティと操作の考慮事項

運用展開には、適切な SSL 証明書の検証、安全な資格情報管理、および認証フローとデータ転送を保護するネットワークアクセス制御が必要です。定期的なセキュリティ監査では、アクセス権限と資格情報のローテーション手順を検証する必要があります。

バックアップとリカバリー戦略は、異なるバックアップメカニズムとリカバリー特性を備えた複数のストレージシステムに分散されたデータを考慮する必要があります。ストレージ階層全体で一貫したリカバリポイントを保証する手順を開発します。

キャパシティプランニングでは、即時のストレージ削減のメリットと、新しいデータがさまざまなストレージ階層を介して到着および古くなるにつれての長期的な成長パターンの両方を考慮する必要があります。コスト最適化の効果を監視し、実際のアクセスパターンに基づいてポリシーを調整します。

包括的な分析ストレージ構成は、ハイブリッドストレージアーキテクチャ全体で透過的な SQL アクセスを維持しながら、運用パフォーマンス要件とストレージコストの最適化のバランスを図る自動データ階層化機能を提供します。外部カタログとの統合により、エンタープライズ規模のデータ共有と分析ツールの

互換性が可能になり、トランザクションと分析ワークロードの両方のプライマリインターフェイスとして PostgreSQL を維持します。