Engineer

変更データキャプチャ

データベースの変更を読み取り、他のシステムへ伝える仕組み

どのようなものか

変更データキャプチャは、通常CDCと略し、データベースで起きた変更を読み取り、発生に応じて他のシステムへ伝える仕組みです。挿入、更新、削除のそれぞれがイベントになり、別のシステムがそれに応じて処理できます。たとえば、最新の状態を保つレポート用のコピー、検索インデックス、注文が入ったことを知る必要のある別のサービスです。変更が起きるデータベースが引き続き正本を管理し、他のシステムはそれに追従します。

変更を見つける方法は、大きく二つあります。単純な方法は、問い合わせることです。数分ごとにクエリを実行し、「最終更新」のタイムスタンプが前回の実行時刻より新しい行を探します。もう一つは、データベース自身が保持する変更記録を読む方法です。PostgreSQLは先行書き込みログ（WAL）に記録しており、文書ではその原則を、データファイルへの変更は、それを記述するログレコードが永続ストレージへ書き出された後にのみ行われる、と説明しています。ログにはすでにすべての変更が発生順に残っているため、それを読めばテーブルへ一切問い合わせずに変更履歴全体を得られます。この二つ目の方法、ログベースのCDCが、現在この用語で通常指すものです。

PostgreSQLでは、この仕組みを論理デコーディングと呼びます。文書では、「データベースのテーブルに対する永続的な変更をすべて取り出す」ことで、内部構造を知らなくても読める形式にすると定義しています。三つの要素が連携します。サーバーはWALに十分な詳細を書き込む必要があり、それを制御するのが `wal_level` です。既定値の `replica` はサーバー間でデータをコピーするには十分で、 `logical` はデコーディングに必要な情報を追加します。レプリケーションスロットは、一つのデータベースからの変更ストリームを表し、元の順序で読み取り側へ渡すとともに、そのクライアントがどこまで読んだかを記憶します。出力プラグインは、WALの内部レコードをクライアントが求める形式へ変換します。

多くのチームは、読み取りクライアントを自作しません。DebeziumはApache Kafka Connect向けに、各データベースのCDC機構から変更を取得するコネクターを提供しています。PostgreSQLコネクターは、文書でPostgreSQL組み込みの出力プラグインと説明されている `pgoutput` か、Debeziumコミュニティーが保守するプラグイン `decoderbufs` を使って読み取ります。DebeziumはJVMアプリケーション内のライブラリーとしても、Debezium Serverとしても実行できます。後者はAmazon Kinesis、Google Cloud Pub/Sub、Apache Pulsarなどへ変更イベントを送ります。

概念を判断に生かす

「変更データキャプチャとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 月末に誰かが表計算ファイルをコピーしなくても、決まった時刻に必要な場所へ数値が届く仕組みをつくります。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 14部門を支える共通インフラ計画 — 14 共通インフラ計画の対象となった研究部門

14部門を支える共通インフラ計画 お問い合わせ →

なぜ重要なのか

ポーリングのほうが低コストに見え、変更の少ない小さなテーブルでは実際にそうである場合も少なくありません。Debeziumの文書は、ログを読むことで実現でき、ポーリングではできないことを挙げています。ログは変更をすべて捕捉しますが、クエリに見えるのは実行時点の行の状態だけなので、二回の実行の間に三回変わった行も一回しか現れません。削除も捕捉できますが、クエリでは削除済みの行は次の結果から消えるだけです。どの書き込み経路でも忘れず設定しなければならない「最終更新」フィールドのような追加の列を、データモデルに設ける必要もありません。行の変更前の状態や、変更したトランザクションのIDなどのメタデータも運べます。そして、頻繁なポーリングによるCPU負荷をかけず、ごく短い遅延で変更を届けられます。PostgreSQLとMySQLについて、Debeziumはこの遅延をミリ秒単位としています。

二つ目の理由は、データベースと他のシステムが受け取るメッセージの整合性です。Debeziumは比較対象となる別の方法を二重書き込みと呼びます。アプリケーションが変更をデータベースへ保存し、その後で変更を知らせるメッセージを送る方法です。この二つの手順の間でプロセスが停止すると、データベースとメッセージストリームが食い違い、どちらが誤っているかを示す記録もありません。CDCは二回目の書き込みを不要にします。アプリケーションは一度だけコミットし、メッセージはそのコミット済みの内容から作られます。アウトボックスパターンは、さらに一歩進めます。変更と同じトランザクションでメッセージ自

体をアウトボックステーブルへ書き込み、CDCがそのテーブルの行を配信します。DebeziumのOutbox Event Routerは、その効果を、サービス自身の状態と他のサービスが消費するイベントとの不整合を避けることだと説明しています。

三つ目は、順序です。以下の例のように、レプリケーションスロットは変更を発生元のトランザクションごとにまとめ、元のサーバーでの順序を保って渡します。消費側がその順に適用すれば、一段階ずつ同じ状態を再構築できます。ETLでは、これにより差分ロードを確かなものにできます。夜間のジョブは、テーブル全体を読み直したり、一つのコード経路で更新を忘れたタイムスタンプ列に頼ったりせず、削除も含めて前回の実行以降の変更だけを正確に取得できます。

実際の活用例

最小の動作例は、PostgreSQLだけで実行できます。サーバー設定に `wal_level = logical` が必要で、この設定は再起動後に初めて有効になります。空きレプリケーションスロットも必要です。 `max_replication_slots` の既定値は10です。以下のセッションは、文書内の例と同様にスーパーユーザーで実行し、その例で使われている出力プラグイン `test_decoding` を用います。このプラグインは各変更を一行のテキストとして出力します。

```
-- postgresql.conf: wal_level = logical（再起動後に有効）
SELECT * FROM pg_create_logical_replication_slot('shop_cdc', 'test_decoding');

BEGIN;
INSERT INTO orders (id, status) VALUES (42, 'pending');
INSERT INTO outbox (id, event) VALUES (7, 'order 42 placed');
COMMIT;

SELECT data FROM pg_logical_slot_peek_changes('shop_cdc', NULL, NULL);
-- BEGIN 10301
-- table public.orders: INSERT: id[integer]:42 status[text]:'pending'
-- table public.outbox: INSERT: id[integer]:7 event[text]:'order 42 placed'
-- COMMIT 10301

SELECT data FROM pg_logical_slot_get_changes('shop_cdc', NULL, NULL);
-- 同じ四行を返して消費するため、二度目の呼び出しでは何も返らない

SELECT pg_drop_replication_slot('shop_cdc');
```

注文とそのアウトボックスメッセージは、それらを書き込んだトランザクション内にまとまって出てきます。 `pg_logical_slot_peek_changes` はスロットを進めずに読むため、次の呼び出しでも同じ行を返します。一方、

`pg_logical_slot_get_changes` はそれらを消費します。論理デコーディングはDDLを伝えないため、`CREATE TABLE` では空の `BEGIN` と `COMMIT` しか出ません。最も重要なのは最後の行です。放置したスロットは、サーバーの資源を確保し続けます。

本番では、Debeziumが読み取り役になります。PostgreSQLコネクターは初回起動時に、捕捉対象のテーブルについて整合性のあるスナップショットを取得し、その取得時点からの変更をストリーム配信します。`snapshot.mode` の既定値 `initial` では、保存済みのオフセットがない場合にだけスナップショットを取得するため、再起動時は最後に記録した位置から再開します。各イベントには、行の変更前後の状態を示す `before` と `after`、`source` ブロック、タイムスタンプ、`op` コードが含まれます。`c` は作成、`u` は更新、`d` は削除、`r` はスナップショット中に読み取った行、`t` は切り詰めを表します。削除後には、同じキーで値がnullのイベント、トゥームストーンも出力し、Kafkaのログコンパクションで、そのキーの以前のメッセージを除去できるようにします。

この記事の二つのパターンは、会社自身のページにも登場します。2022年から2024年にかけて、創業者は複数のPostgreSQL開発・本番環境にわたり、PL/pgSQLで600本のETL/ELTパイプラインを構築しました。CDCとタイムスタンプに基づく差分を使った、差分ロードも含まれます。最後のチケットを一度だけ売るというノートでは、アウトボックスを中心にフラッシュセールを設計しています。各手順は変更と同じトランザクションでメッセージをアウトボックスへ記録し、変更データキャプチャがそれを配信します。メッセージは複数回届く可能性があるため、各消費側は注文をキーとして冪等に動作します。

問題が起きるところ

スロットが保持する量に上限がありません。 スロットは、消費側が確認していないWALをすべて保持し、PostgreSQLの既定設定はその量を制限しません。`max_slot_wal_keep_size` は `-1` で、スロットによるWALの無制限な保持を許します。`idle_replication_slot_timeout` は `0` で、使われていないスロットの無効化を停止します。そのため、失敗したデプロイ中に停止したコネクターが、誰もがその存在を忘れた後までディスクを埋め続けることがあります。`pg_replication_slots` で非アクティブになったスロットを監視し、`max_slot_wal_keep_size` をディスクで許容できる量に設定します。ただし、保持上限を超えるほど遅れた消費側は、再開できなくなる可能性があります。復旧には新しいスロットが必要で、新しいスロットは現在から始まります。Debeziumは、スロットを削除して作り直すと、その間のイベントを気付かずに飛ばす場合があると警告しているため、その後は新たなスナップショットを取得します。

変更の少ないテーブルが、ログの保持を長引かせます。 Debeziumの文書は、より気付きにくい事例も説明しています。スロットは一つのデータベースに属し、コネクターは捕捉対象の変更を見たときだけ、確認済みの位置を進めます。対象テーブルがほとんど変わらない一方で、他のテーブルや同じサーバー上の他のデータベースに大量の書き込みがあると、位置はほぼ動かず、保持されたWALが増え続けます。対策はハートビートです。`heartbeat.interval.ms` で、設定間隔ごとにハートビートイベントを出力し、`heartbeat.action.query` で、捕捉対象のデータベース内の小さなテーブルへ定期的に書き込めます。そのテーブルは、コネクターが読むパブリケーションに含める必要があります。そうすれば、他の変更がないときもスロットが進みます。

消費側が、各変更は一度だけ届くと思い込んでいます。 論理スロットは通常の動作では各変更を一度だけ出力しますが、位置を保存するのはチェックポイント時だけなので、クラッシュ後には前の位置へ戻り、直近の変更を再送する場合があります。PostgreSQLの文書は、その影響への対処をクライアントに委ねています。各金額を累計に加える消費側なら、同じ変更を二重に数えてしまいます。各消費側を冪等にします。行の主キーをキーとするupsertとして変更を適用するか、処理済みイベントのIDを記録し、二回目の配信では何も変わらないようにします。

更新と削除が、変更前の値を伴わず届きます。 更新・削除イベントに以前の行のどの情報が含まれるかは、テーブルのレプリカアイデンティティーによって決まります。`REPLICA IDENTITY DEFAULT` では、Debeziumのイベントが保持する変更前の値は、多くても主キーの値だけであり、主キーのないテーブルについては更新イベントも削除イベントも生成しません。`REPLICA IDENTITY FULL` は、変更前の全列の値を `before` へ入れますが、代償があります。PostgreSQLの文書は、特に多数の `UPDATE` と `DELETE` を実行する場合、WALの量が増えると指摘しています。捕捉するすべてのテーブルに主キーを設け、古い値を消費側が必要とするテーブルだけを `FULL` に切り替えます。

参考資料

- Debeziumの機能：ポーリングや二重書き込みでは捉えられない、ログベースCDCで捕捉できるものと、スナップショット、フィルター、マスキングを説明します。
- DebeziumのPostgreSQLコネクター：出力プラグイン、スナップショット、変更イベントの形式、レプリカアイデンティティー、WALのディスク容量を説明します。
- 論理デコーディングの概念：レプリケーションスロット、出力プラグイン、不要なスロットを削除すべき理由を説明します。

- 論理デコーディングの例：スロットの作成、変更の参照と消費、最後のスロット削除を示します。
- レプリケーションのサーバー設定：`max_replication_slots`、`max_slot_wal_keep_size`、`idle_replication_slot_timeout` を説明します。
- Outbox Event Router：アウトボックスパターンと、Debeziumがアウトボックスの行をトピックへ振り分ける仕組みを説明します。

用語集

実際の制約に合わせて選ぶ

代表的なデータセット、そこから答えを得たい問い、現在のボトルネックをお知らせください。品質、処理量、復旧の測り方を合意できます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「変更データキャプチャとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く 連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/change-data-capture/