Citus で時系列データの Postgres をスケーリングする方法

Written by **Burak Velioglu**
October 22, 2021

時系列データを大規模に管理するのは困難な場合があります。PostgreSQL は、インデックス、COPY、SQL などの多くの強力なデータ処理機能を提供しますが、時系列データはデータ量が多く増加し続ける性格があるので、時間の経過とともにデータベースの速度が低下する可能性があります。

幸いなことに、Postgres にはこの問題に対する組み込みの解決策があります: 時間範囲によるテーブルのパーティションへの分割。

Postgres の宣言型パーティション機能は、時系列ワークロードのクエリとデータ取り込み時間を短縮するのに役立ちます。範囲パーティションを使用すると、テーブルを作成し、範囲（通常は時間範囲）に基づく小さなパーティションに分割できます。各クエリははるかに小さなチャンクを処理するだけで良くなるため、クエリのパフォーマンスが向上します。ただし Postgres サーバーのメモリ、CPU、およびストレージリソースによる性能の限界を超えるものではありません。

利点として、パーティションをクラスター全体に分散させることで、パーティション化された Postgres テーブルをスケールアウトして膨大な量のデータを処理できることがあります。どのようにするのかというと、Postgres の Citus 拡張を使用します。つまり、Citus を使用すると、分散時間パーティションテーブルを作成できます。ノードのディスク領域を節約するために、パーティションのインデックスを有効にしたままパーティションを圧縮することもできます。さらに良いことに、最新の Citus 10.2 オープンソースリリースでは、PostgreSQL でパーティションを管理するのがはるかに簡単になります。

この投稿は、Postgres を Citus と pg_cron と共に時系列データに適用し、PostgreSQL を分散時系列データベースに効果的に変換するための「ハウツー」ガイドです。Postgres と Citus を一緒に使用することで、アプリケーション

は増え続ける膨大な量の時系列データの処理において、パフォーマンスがより向上し作業が楽になります。

この記事で説明する時系列データベース機能

- パーティション: Postgres のネイティブパーティション機能を使用し、大きな時系列テーブルを小さな時間パーティションに分割する方法
- 簡単なパーティション管理: Citus の新機能を使用してパーティションの管理を簡素化する方法
- 圧縮: Citus Columnar を使用して古いパーティションを圧縮し、ストレージを節約し、クエリのパフォーマンスを向上させる方法
- 自動化: `pg_cron` 拡張機能を使用してパーティション管理をスケジュールおよび自動化する方法
- シャーディング: 単一ノードの Citus で Postgres のパーティションをシャーディングする方法
- ノード間での分散: シャードパーティションを Citus データベースクラスターのノードに分散して、高いパフォーマンスと拡張を実現する方法

Postgres の組み込みパーティション機能を時系列データに適用する

Postgres の組み込みパーティション機能は、時系列データの管理において非常に便利です。

Postgres テーブルを時刻の列の範囲ごとにパーティションに分割することで（それによって時系列パーティションテーブルを作成することで）、単一の巨大なテーブルではなく、はるかに小さなパーティションテーブルと、はるかに小さなインデックスを持つテーブルを持つことができます。

- 通常、テーブルが小さく、インデックスが小さいほど、クエリ応答が高速になります。

- 期間を分けたパーティションを持つことで、古いデータをドロップ/削除/期限切れにすることがより効率的にできます。

Postgres テーブルをパーティションに分割するには、まずパーティション化されたテーブルを作成する必要があります。パーティション化されたテーブルは仮想テーブルであり、独自のストレージはありません。パーティションを作成し、パーティション境界によって定義されたデータのサブセットを、`PARTITION BY RANGE` 構文を使用して格納する必要があります。パーティションテーブルにデータを取り込むと、Postgres は定義したパーティションキーに基づいて適切なパーティションにデータを格納します。

```
-- create a parent table partitioned by range
CREATE TABLE  time_series_events (event_time timestamp, event int, user_id int) PARTITION BY RANGE (event_time);

-- create partitions for that partitioned table
CREATE TABLE time_series_events_p2021_10_10 PARTITION OF time_series_events FOR VALUES FROM ('2021-10-10 00:00:00') TO ('2021-10-11 00:00:00');
CREATE TABLE time_series_events_p2021_10_11 PARTITION OF time_series_events FOR VALUES FROM ('2021-10-11 00:00:00') TO ('2021-10-12 00:00:00');
CREATE TABLE time_series_events_p2021_10_12 PARTITION OF time_series_events FOR VALUES FROM ('2021-10-12 00:00:00') TO ('2021-10-13 00:00:00');

-- insert rows into a partitioned table
INSERT INTO time_series_events VALUES('2021-10-10 12:00:00', 1, 2);
INSERT INTO time_series_events VALUES('2021-10-11 12:00:00', 1, 2);
INSERT INTO time_series_events VALUES('2021-10-12 12:00:00', 1, 2);
```

パーティションが不要になった場合は、通常の Postgres テーブルを削除するのと同様に手動で削除できます。

```
-- drop partitions
DROP TABLE time_series_events_p2021_10_10;
DROP TABLE time_series_events_p2021_10_11;
DROP TABLE time_series_events_p2021_10_12;
```

Postgres の組み込みパーティション機能を使用することで、Postgres を実行しているノードのリソースをより賢く利用することができますが、これらのパーティションを自分で管理するために時間を費やす必要があります。Postgres のパーティション管理を簡素化する新しい Citus UDF （ユーザー定義関数）を利用するのであれば、続きを読んでください。

新しい Citus 関数を使用してパーティション管理を簡素化する方法

Citus 10.2 では、Postgres の時間パーティションの管理方法を簡素化するために、2 つの新しいユーザー定義関数が追加されています：create_time_partitions と drop_old_time_partitions。これら 2 つの新しい Citus UDF を使用すると、時間パーティションを手動で作成または削除する必要がなくなりました。

2 つの新しい Citus UDF は共に、通常の Postgres テーブルと分散 Citus テーブルの両方で使用できます。

- `create_time_partitions(table_name regclass, partition_interval interval, end_at timestamp with time zone, start_from timestamp with time zone DEFAULT now())`: 指定されたテーブルと間隔に対して、指定された時間範囲に必要な数のパーティションを作成します。

- `drop_old_time_partitions(table_name regclass, older_than timestamp with time zone)`: 指定されたテーブルについて、指定されたタイムスタンプより古いパーティションをすべて削除します。

```
-- create partitions per day from 2021-10-10 to 2021-10-30
SELECT create_time_partitions(table_name:= 'time_series_events',
    partition_interval:= '1 day',
    end_at:= '2021-10-30',
    start_from:= '2021-10-10');
```

Citus の time_partitions ビューを使用して、クラスター上の時間パーティション化されたテーブルの詳細を取得できます。

```
-- check the details of partitions from time_partitions view
SELECT partition, from_value, to_value, access_method FROM time_partitions;

           partition            |     from_value      |       to_value
| access_method
--------------------------------+---------------------+---------------------
-+---------------
 time_series_events_p2021_10_10 | 2021-10-10 00:00:00 | 2021-10-11 00:00:00
| heap
 time_series_events_p2021_10_11 | 2021-10-11 00:00:00 | 2021-10-12 00:00:00
| heap
 time_series_events_p2021_10_12 | 2021-10-12 00:00:00 | 2021-10-13 00:00:00
| heap
 time_series_events_p2021_10_13 | 2021-10-13 00:00:00 | 2021-10-14 00:00:00
| heap
 time_series_events_p2021_10_14 | 2021-10-14 00:00:00 | 2021-10-15 00:00:00
| heap
 time_series_events_p2021_10_15 | 2021-10-15 00:00:00 | 2021-10-16 00:00:00
| heap
 time_series_events_p2021_10_16 | 2021-10-16 00:00:00 | 2021-10-17 00:00:00
| heap
 time_series_events_p2021_10_17 | 2021-10-17 00:00:00 | 2021-10-18 00:00:00
| heap
```

```
 time_series_events_p2021_10_18 | 2021-10-18 00:00:00 | 2021-10-19 00:00:00
| heap
 time_series_events_p2021_10_19 | 2021-10-19 00:00:00 | 2021-10-20 00:00:00
| heap
 time_series_events_p2021_10_20 | 2021-10-20 00:00:00 | 2021-10-21 00:00:00
| heap
 time_series_events_p2021_10_21 | 2021-10-21 00:00:00 | 2021-10-22 00:00:00
| heap
 time_series_events_p2021_10_22 | 2021-10-22 00:00:00 | 2021-10-23 00:00:00
| heap
 time_series_events_p2021_10_23 | 2021-10-23 00:00:00 | 2021-10-24 00:00:00
| heap
 time_series_events_p2021_10_24 | 2021-10-24 00:00:00 | 2021-10-25 00:00:00
| heap
 time_series_events_p2021_10_25 | 2021-10-25 00:00:00 | 2021-10-26 00:00:00
| heap
 time_series_events_p2021_10_26 | 2021-10-26 00:00:00 | 2021-10-27 00:00:00
| heap
 time_series_events_p2021_10_27 | 2021-10-27 00:00:00 | 2021-10-28 00:00:00
| heap
 time_series_events_p2021_10_28 | 2021-10-28 00:00:00 | 2021-10-29 00:00:00
| heap
 time_series_events_p2021_10_29 | 2021-10-29 00:00:00 | 2021-10-30 00:00:00
| heap
(20 rows)
```

時系列ワークロードでは、古いデータが不要になったら、古いデータをドロップ（または削除、期限切れ）するのが一般的です。パーティション化すると、Postgres がドロップしたすべてのデータを読み取る必要がなくなるため、古いデータを削除するのが非常に効率的になります。特定のしきい値より古いパーティションの削除を容易にするために、Citus 10.2 では UDF `drop_old_time_partitions` が導入されました。

```
-- drop partitions older than 2021-10-15
CALL drop_old_time_partitions(table_name:= 'time_series_events',
older_than:= '2021-10-15');

-- check the details of partitions from time_partitions view
SELECT partition, from_value, to_value, access_method FROM time_partitions;

           partition            |     from_value      |      to_value
| access_method
--------------------------------+---------------------+---------------------
-+---------------
 time_series_events_p2021_10_15 | 2021-10-15 00:00:00 | 2021-10-16 00:00:00
| heap
 time_series_events_p2021_10_16 | 2021-10-16 00:00:00 | 2021-10-17 00:00:00
| heap
 time_series_events_p2021_10_17 | 2021-10-17 00:00:00 | 2021-10-18 00:00:00
| heap
 time_series_events_p2021_10_18 | 2021-10-18 00:00:00 | 2021-10-19 00:00:00
| heap
 time_series_events_p2021_10_19 | 2021-10-19 00:00:00 | 2021-10-20 00:00:00
| heap
 time_series_events_p2021_10_20 | 2021-10-20 00:00:00 | 2021-10-21 00:00:00
| heap
 time_series_events_p2021_10_21 | 2021-10-21 00:00:00 | 2021-10-22 00:00:00
| heap
 time_series_events_p2021_10_22 | 2021-10-22 00:00:00 | 2021-10-23 00:00:00
| heap
 time_series_events_p2021_10_23 | 2021-10-23 00:00:00 | 2021-10-24 00:00:00
| heap
 time_series_events_p2021_10_24 | 2021-10-24 00:00:00 | 2021-10-25 00:00:00
| heap
 time_series_events_p2021_10_25 | 2021-10-25 00:00:00 | 2021-10-26 00:00:00
| heap
 time_series_events_p2021_10_26 | 2021-10-26 00:00:00 | 2021-10-27 00:00:00
| heap
```

```
 time_series_events_p2021_10_27 | 2021-10-27 00:00:00 | 2021-10-28 00:00:00
| heap
 time_series_events_p2021_10_28 | 2021-10-28 00:00:00 | 2021-10-29 00:00:00
| heap
 time_series_events_p2021_10_29 | 2021-10-29 00:00:00 | 2021-10-30 00:00:00
| heap
(15 rows)
```

それでは、Postgres のテーブルをパーティション分割した後に使用できる、特に規模の課題に対処している場合に使用できる、より高度な機能をいくつか見てみましょう。

Citus Columnar で古いパーティションを圧縮する方法（インデックスもサポートされるようになりました）

Citus 10 以降では、列ストレージを使用して Postgres テーブル内のデータを圧縮できます。列圧縮と Postgres の時間パーティションを組み合わせることで、古いパーティションのディスク使用量を簡単に減らすことができます。パーティションを列形式で格納すると、クエリが不要な列をスキップできるため、分析クエリのパフォーマンスも向上します。Postgres の列圧縮をまだチェックしていない場合は、Jeff の列ストレージについての投稿から始めて詳細な説明を読むことができます。または、Citus の列圧縮の使用方法に関するこのデモビデオをご覧ください。

UDF `alter_old_partitions_set_access_method` を使用すると、アクセス方式を `heap` から `columnar` に変換して、指定されたしきい値より古いパーティションを圧縮できます。また、その逆に、アクセス方式を `columnar` から `heap` に変換して圧縮解除することもできます。

- `alter_old_partitions_set_access_method(parent_table_name regclass, older_than timestamp with time zone, new_access_method name)`: 指定されたテーブルに対して、指定されたしきい値より古いすべてのパーティションを、圧縮または解凍します。

```
-- compress partitions older than 2021-10-20
CALL alter_old_partitions_set_access_method('time_series_events',  '2021-
10-20', 'columnar');

-- check the details of partitions from time_partitions view
SELECT partition, from_value, to_value, access_method FROM time_partitions;

           partition            |     from_value      |      to_value
| access_method
--------------------------------+---------------------+---------------------
-+---------------
 time_series_events_p2021_10_15 | 2021-10-15 00:00:00 | 2021-10-16 00:00:00
| columnar
 time_series_events_p2021_10_16 | 2021-10-16 00:00:00 | 2021-10-17 00:00:00
| columnar
 time_series_events_p2021_10_17 | 2021-10-17 00:00:00 | 2021-10-18 00:00:00
| columnar
 time_series_events_p2021_10_18 | 2021-10-18 00:00:00 | 2021-10-19 00:00:00
| columnar
 time_series_events_p2021_10_19 | 2021-10-19 00:00:00 | 2021-10-20 00:00:00
| columnar
 time_series_events_p2021_10_20 | 2021-10-20 00:00:00 | 2021-10-21 00:00:00
| heap
 time_series_events_p2021_10_21 | 2021-10-21 00:00:00 | 2021-10-22 00:00:00
| heap
 time_series_events_p2021_10_22 | 2021-10-22 00:00:00 | 2021-10-23 00:00:00
| heap
 time_series_events_p2021_10_23 | 2021-10-23 00:00:00 | 2021-10-24 00:00:00
| heap
 time_series_events_p2021_10_24 | 2021-10-24 00:00:00 | 2021-10-25 00:00:00
| heap
 time_series_events_p2021_10_25 | 2021-10-25 00:00:00 | 2021-10-26 00:00:00
| heap
 time_series_events_p2021_10_26 | 2021-10-26 00:00:00 | 2021-10-27 00:00:00
| heap
```

```
 time_series_events_p2021_10_27 | 2021-10-27 00:00:00 | 2021-10-28 00:00:00
| heap
 time_series_events_p2021_10_28 | 2021-10-28 00:00:00 | 2021-10-29 00:00:00
| heap
 time_series_events_p2021_10_29 | 2021-10-29 00:00:00 | 2021-10-30 00:00:00
| heap
(15 rows)
```

圧縮されたパーティション上のデータを更新または削除することは（少なくともまだ）できないので `alter_table_set_access_method`（テーブルを圧縮/解凍するための別の Citus の UDF です）を使用して、`heap` を最後の引数に指定してパーティションをまず解凍します。パーティションが圧縮解除され、行ベースのストレージ（ヒープと呼ばれる）に戻ったら、データを更新または削除できます。その後、`alter_table_set_access_method` を呼び出し、最後の引数に `columnar` を指定することで、パーティションを再度圧縮できます。

Citus 10.2 以降、圧縮テーブルにインデックス[1]を設定することができます。パーティションテーブルの一部のパーティションが圧縮されていても、パーティションテーブルにインデックスを追加できるようになりました。

```
-- create index on a partitioned table with compressed partitions
CREATE INDEX index_on_partitioned_table ON time_series_events(user_id);
```

以下の UDF を使用すると、時間パーティションの管理が容易になります - `create_time_partitions`、`drop_old_time_partitions` と `alter_old_partitions_set_access_method` - `pg_cron` を使用してパーティション管理を完全に自動化できるようになりました。

pg_cron でパーティション管理を自動化する方法

パーティション管理を完全に自動化するには、私たちのチームによって作成および保守されるオープンソースの拡張機能である pg_cron を使用できます。

`pg_cron` を使用すると、Postgres で cron ベースのジョブをスケジュールできます。

`pg_cron` を使用して、これらの Citus 関数をパーティションの作成、削除、圧縮用にスケジュールし、Postgres パーティション管理を自動化できます。Marco の pg_cron の投稿をチェックして、その使用法と時間の経過に伴う進化の詳細な説明をご覧ください。

以下は、`pg_cron` を利用しパーティション管理を完全に自動化する例です。

```
-- schedule cron jobs to
-- create partitions for the next 7 days
SELECT cron.schedule('create-partitions',
    '@daily',
    $$SELECT create_time_partitions(table_name:='time_series_events',
            partition_interval:= '1 day',
            end_at:= now() + '7 days') $$);

-- compress partitions older than 5 days
SELECT cron.schedule('compress-partitions',
    '@daily',
    $$CALL alter_old_partitions_set_access_method('time_series_events',
            now() - interval '5 days', 'columnar') $$);

-- expire partitions older than 7 days
SELECT cron.schedule('expire-partitions',
    '@daily',
    $$CALL drop_old_time_partitions('time_series_events',
            now() - interval '7 days') $$);
```

上記でスケジュールされた UDF は、`cron.schedule` の 2 番目の引数が `@daily` として指定されているため、1 日に 1 回呼び出されることに注意してください。他のオプションについては、cron 構文で確認できます。

これらの UDF をスケジュールした後は、パーティションの管理について考える必要がなくなります。あなたの `pg_cron` のジョブが Citus と Postgres を自動的に活用します:

- 指定された期間のパーティションを作成し、
- 指定された圧縮しきい値より古いパーティションを圧縮し、
- アプリケーションの動作中に、指定された有効期限しきい値より古いパーティションを削除します。

より深く掘り下げたい場合は、Citus のドキュメントにある時系列データのユースケースガイドで、より詳細な説明を提供します。

Citus の UDF を使用してパーティションを管理し、pg_cron を使用してパーティションを自動化することで、時系列ワークロードを 1 つの Postgres ノードで手間のかからない方法で処理できます。ただし、データベースが大きくなるにつれてパフォーマンスの問題に遭遇し始める可能性があります。ここでシャーディング（クラスター全体にデータベースを分散する）を、次の 2 つのセクションで説明します。

単一ノードの Citus でパーティションをシャーディングする方法

時間の経過と共にデータベースの速度が低下する原因となる大量の時系列データを処理するには、シャーディングとパーティションを一緒に使用して、データを 2 次元に分割します。データを 2 次元に分割すると、データとインデックスのサイズがさらに小さくなります。Postgres をシャードするには、Citus を使用できます。また、Citus 10 以降では、Citus を使用して単一のノードで Postgres をシャーディングし、Citus から取得したクエリ並列処理を利用し、アプリケーションを「スケールアウト準備完了」にすることができます。または、Citus をもっと慣れ親しんだ方法で使用して、Postgres を複数のノードにまたがってシャーディングすることもできます。

ここでは、Citus を使用してパーティションを単一の Citus ノードでシャーディングする方法を探ってみましょう。次のセクションでは、Citus を使用してシャードパーティションを複数のノードに分散する方法を見ていきます。

Citus でシャーディングするために最初に行う必要があるのは、ディストリビューション列（シャーディング キーと呼ばれることもあります）を決定することです。データを 2 次元に分割し、Postgres パーティショニングと Citus シャーディングの両方を利用する場合、ディストリビューション列をパーティション列と同じにすることはできません。時系列データの場合、ほとんどの人が時間でパーティションに分割するため、アプリケーションに適したディストリビューション列を選択するだけで済みます。Citus ドキュメントのディストリビューション列の選択ガイドでは、ここで役立つガイダンスをいくつか紹介しています。

次に、テーブルをシャーディングするように Citus に指示するには、Citus の `create_distributed_table` 関数を使用する必要があります。単一のノードで Citus を実行している場合でも、テーブルのパーティションは選択したディストリビューション列によってシャードされます。この例では、`user_id` をディストリビューション列として使用します。

```
-- shard partitioned table to have sharded partitioned table
SELECT create_distributed_table('time_series_events', 'user_id');
```

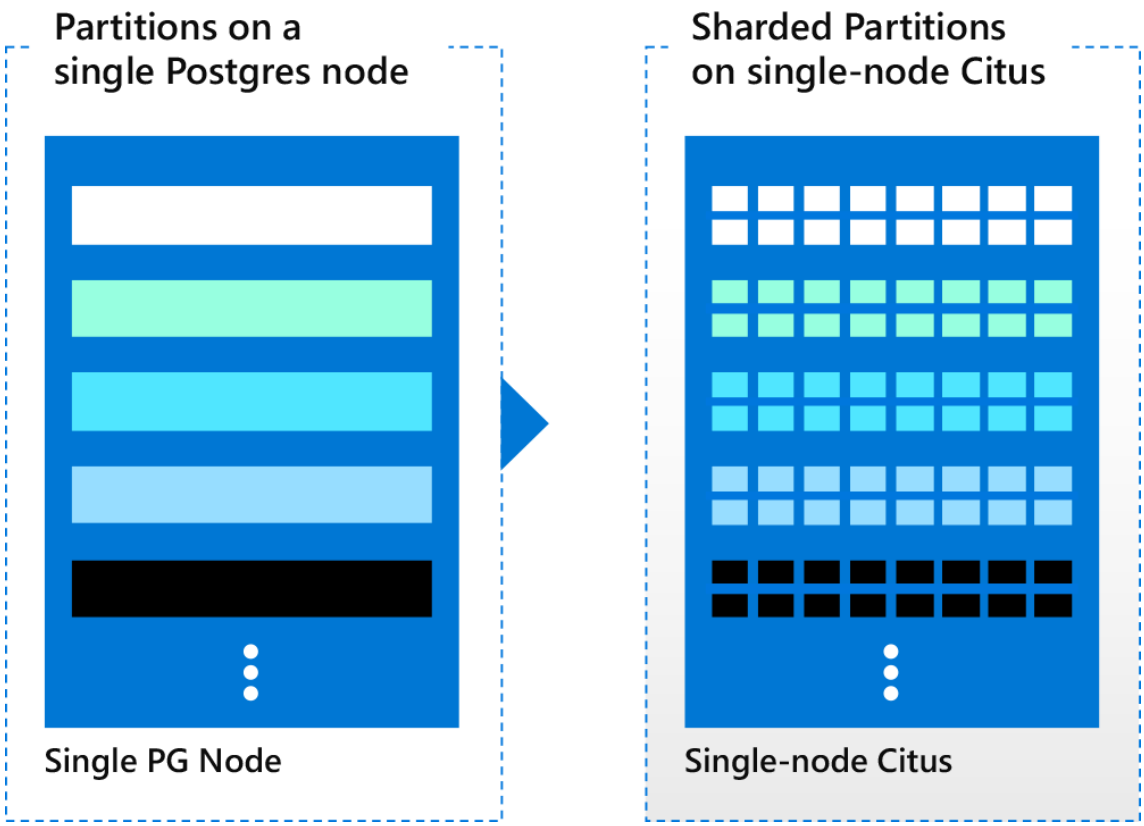

図 1: 左側はパーティション化されたテーブルを持つ単一の Postgres ノードです。右側はシャードされたパーティションを持つ単一ノードの Citus を表しています。

ひとたびパーティション化されたテーブルを 1 つの Citus ノードにシャーディングすれば、Citus クラスターをスケールアウトしてテーブルを複数のノードに分散するのは簡単にできるようになります。

Citus を使用しシャードパーティションを複数のノードに分散する

より多くのデータを管理するためにアプリケーションを拡張する必要がある場合、ノードのリソース（CPU、メモリ、ディスク）がボトルネックになる可能性があります。この場合、Postgres をシャードして複数のノードに分散したくなるでしょう。

単一ノードの Citus でパーティションを既にシャーディングしている場合は、Citus ノードを追加してからクラスター全体でテーブルのバランスを再調整す

ることで、パーティションを簡単に分散できます。これを行う方法を以下で探ってみましょう。

クラスターに新しいノードを追加するには、まず UDF `citus_add_node` を使用して、DNS 名（またはそのノードの IP アドレス）とポートを `pg_dist_node` カタログテーブルに追加する必要あります。(Citus を Azure 上のマネージド サービスとして利用している場合、Azure ポータルでワーカーノード数スライダーを移動するだけで、クラスターにノードを追加できます。

```
-- shard partitioned table to have sharded partitioned table
SELECT create_distributed_table('time_series_events', 'user_id');
```

次にテーブルのバランスを変更して、既存のシャードを新しく追加されたノードに移動する必要があります。`rebalance_table_shards` を使用すると、ノード間でシャードのバランスを均等に再調整できます。

```
-- rebalance shards evenly among the nodes
SELECT rebalance_table_shards();
```

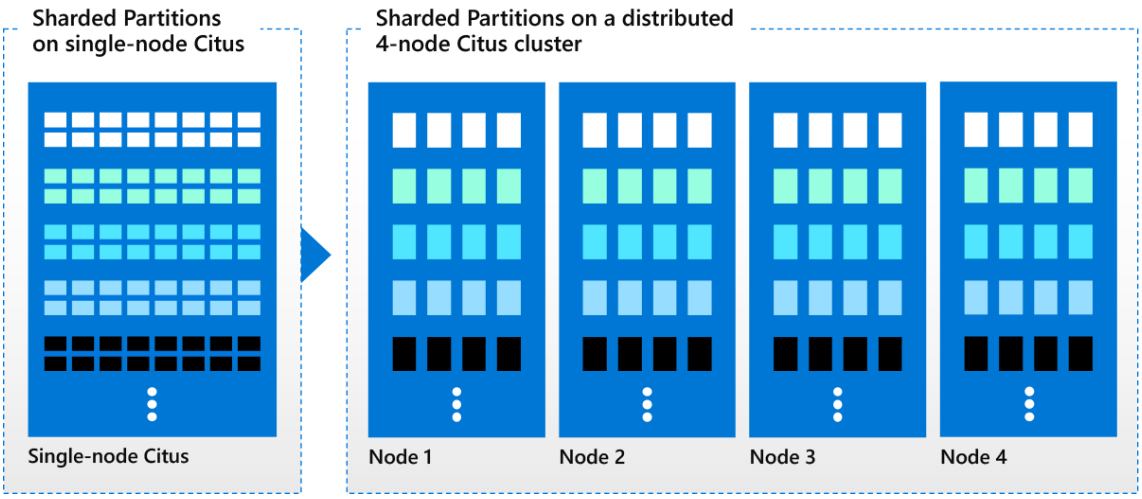

図 2: シャードパーティションを Citus データベースクラスターの複数のノードに分散する

- あるいは、マルチノードの Citus クラスターが既にあり、クラスター内のノード間でパーティション化されたテーブルをシャーディングする場合は、`create_distributed_table` を使用する必要があります。パーティションは、指定したディストリビューション列によってノード間にシャーディングされます。

Citus のドキュメントで、クラスター管理の詳細な説明を確認できます。

PostgreSQL のための他の時間パーティション拡張

時間パーティション UDF を導入する前は、Citus で時間パーティションへの分割を行う一般的なアプローチは、pg_partman 拡張を使用することでした。一部の高度なシナリオでは、`pg_partman` を利用することは依然として有益です。特に、`pg_partman` は「テンプレートテーブル」の作成をサポートしています。「テンプレートテーブル」を使用して、異なるパーティションに異なるインデックスを持つことができます。「テンプレートテーブル」でパーティション列をカバーしていない一意のインデックスを作成することもできます。他の全ての機能については、pg_partman のドキュメントを確認して下さい。

PostgreSQL のもう 1 つの時間パーティション拡張は TimescaleDB です。残念ながら、Citus と TimescaleDB は現在互換性がありません。重要なこととし

て、Citus には単一ノードからペタバイト規模の時系列ワークロードまで拡張できる、非常に成熟した分散クエリエンジンがあります。Citus は PostgreSQL 14 とも完全に互換性があり、date_bin 関数などの時系列データを処理するための新機能を備えています。

分散リレーショナル時系列データベースとしての Citus

この記事では、スケーラブルな方法で時系列データを管理する方法について説明します。Postgres のパーティション機能、Citus の分散データベース機能、`pg_cron` の自動化を組み合わせることで、分散リレーショナル時系列データベースが得られます。パーティションの作成と削除のための新しい Citus ユーザー定義関数は、物事をはるかに簡単にします。

機能	**Postgres と Citus の 時系列データベース機能の説明**
パーティション	Postgres のネイティブ範囲パーティション機能を使用して、期間を「範囲」として使用して、大きなテーブルを小さなパーティションに分割します
パーティション管理	Citus の新機能を使用して時間パーティションの管理を簡素化する
圧縮	Citus Columnar を使用して古いパーティションを圧縮し、ストレージを節約し、クエリのパフォーマンスを向上させる
自動化	Postgres の `pg_cron` 拡張を使用して、パーティションの作成、削除、圧縮をスケジュールします
シャーディング	Citus を使用して Postgres のテーブルをシャーディングし、単一ノードの Citus またはク

機能	Postgres と Citus の 時系列データベース機能の説明
	ラスター全体でクエリのパフォーマンスを向上させる
ノード間での分散	Citus を使用して、シャードを複数のノードに分散し、並列分散クエリを有効にし、 複数のノードのメモリ、CPU、およびストレージを使用する

Citus 拡張機能を Postgres で使用したい場合は、Citus パッケージをダウンロードするか、マネージドデータベースサービスとしてクラウドで Citus をプロビジョニングできます。最新の Citus リリースについては、Onder の 10.2 ブログ記事で学習するか、ドキュメントの時系列ユースケースガイドをご覧ください。

また、時系列ワークロードのスケーリングや Citus 全般についてご質問がある場合は、パブリック Slack チャネルからお気軽にお問い合わせください。さらに掘り下げて Citus を試してみるには、スタートページが出発点として便利です。

脚注

1. Citus 10.2 以降、**ハッシュ**および **btree** インデックス型が Citus Columnar でサポートされるようになりました。

原文：https://www.citusdata.com/blog/2021/10/22/how-to-scale-postgres-for-time-series-data-with-citus/