Hewlett Packard
Enterprise

2021 年 9 月 1 日

Citus 10 検証結果

日本ヒューレット・パッカード合同会社
篠田典良

Hewlett Packard
Enterprise

目次

Hewlett Packard
Enterprise

1. 本文書について

1.1 本文書の概要

本文書は PostgreSQL データベースにスケールアウト機能を提供する Citus 10 について検証した資料です。

1.2 本文書の対象読者

本文書は、既にある程度 PostgreSQL に関する知識を持っているエンジニア向けに記述しています。インストール、基本的な管理等は実施できることを前提としています。

1.3 本文書の範囲

本文書は PostgreSQL 上で利用できる Citus 10 を使って、PostgreSQL をスケールアウトする構成を検証しています。すべての機能について記載および検証しているわけではありません。特に障害発生時の高可用性構成については検証していません。

1.4 本文書の対応バージョン

本文書は以下のバージョンとプラットフォームを対象として検証を行っています。

表 1　対象バージョン

種別	バージョン
PostgreSQL	13.2
Citus	Community Edition 10.2.1（2021 年 9 月 1 日現在の開発中バージョン）
オペレーティング・システム	Red Hat Enterprise Linux 7 Update 8 (x86-64)

1.5 本文書に対する質問・意見および責任

本文書の内容は日本ヒューレット・パッカード合同会社の公式見解ではありません。また内容の間違いにより生じた問題について作成者および所属企業は責任を負いません。本文書に対するご意見等ありましたら作成者 篠田典良（Mail: noriyoshi.shinoda@hpe.com）までお知らせください。

1.6 表記

本文書内にはコマンドや SQL 文の実行例および構文の説明が含まれます。実行例は以下のルールで記載しています。

表 2 例の表記ルール

表記	説明
#	Linux root ユーザーのプロンプト
$	Linux 一般ユーザーのプロンプト
太字	ユーザーが入力する文字列
postgres=#	PostgreSQL 管理者が利用する psql コマンド・プロンプト
postgres=>	PostgreSQL 一般ユーザーが利用する psql コマンド・プロンプト
下線部	特に注目すべき項目
<<以下省略>>	より多くの情報が出力されるが文書内では省略していることを示す
<<途中省略>>	より多くの情報が出力されるが文書内では省略していることを示す
{PostgreSQL}	PostgreSQL をインストールしたパス（/usr/local/pgsql を指す）

構文は以下のルールで記載しています。

表 3 構文の表記ルール

表記	説明
斜体	ユーザーが利用するオブジェクトの名前やその他の構文に置換
[ ]	省略できる構文であることを示す
{ A | B }	A または B を選択できることを示す
…	PostgreSQL の一般的な構文

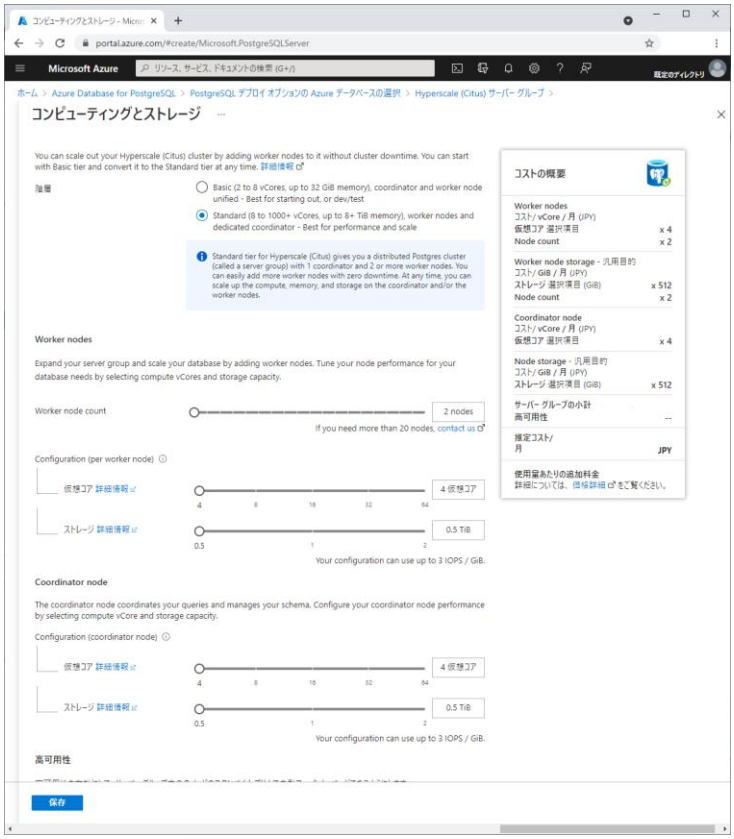
Hewlett Packard
Enterprise

2. 利用方法

2.1 Citus とは

Citus は Citus Data（https://www.citusdata.com/）が提供する PostgreSQL の拡張機能です。PostgreSQL 上のデータを複数ノードに分散し、スケールアップによるスループットの向上を可能にします。Citus は PostgreSQL のエクステンションとして提供されるため、PostgreSQL のソースコードを変更することなく、既存の PostgreSQL データベースにスケールアップ機能を追加することができます。また分散の単位はテーブルであるため、スケールアップ対象を特定の範囲に絞ることもできます。Citus Data は 2019 年に Microsoft に買収されており、Citus は Azure Database for PostgreSQL - Hyperscale (Citus)の中核的な技術として使用されています。Citus 10 を使用する Azure Database for PostgreSQL - Hyperscale (Citus) は 2021 年 8 月 18 日に GA 状態となりましたが、提供されるリージョンは限られています（https://azure.microsoft.com/ja-jp/updates/azure-database-for-postgresql-hyperscale-citus-columnar-compression-now-generally-available/）。

図 1 Azure Database for PostgreSQL - Hyperscale (Citus) 構成画面



2.1.1 Citus の構成

Citus は複数の PostgreSQL インスタンスと PostgreSQL エクステンションから構成されます。

□ コーディネーター・ノード

クライアントからの接続を受け付ける単一の PostgreSQL インスタンスです。処理を分散するためのメタデータを保持します。クライアントからの SQL 文を受け付けると、ワーカー・ノードに処理を依頼し、結果を受け取ります。分散を行わないテーブルを保持することもできます。

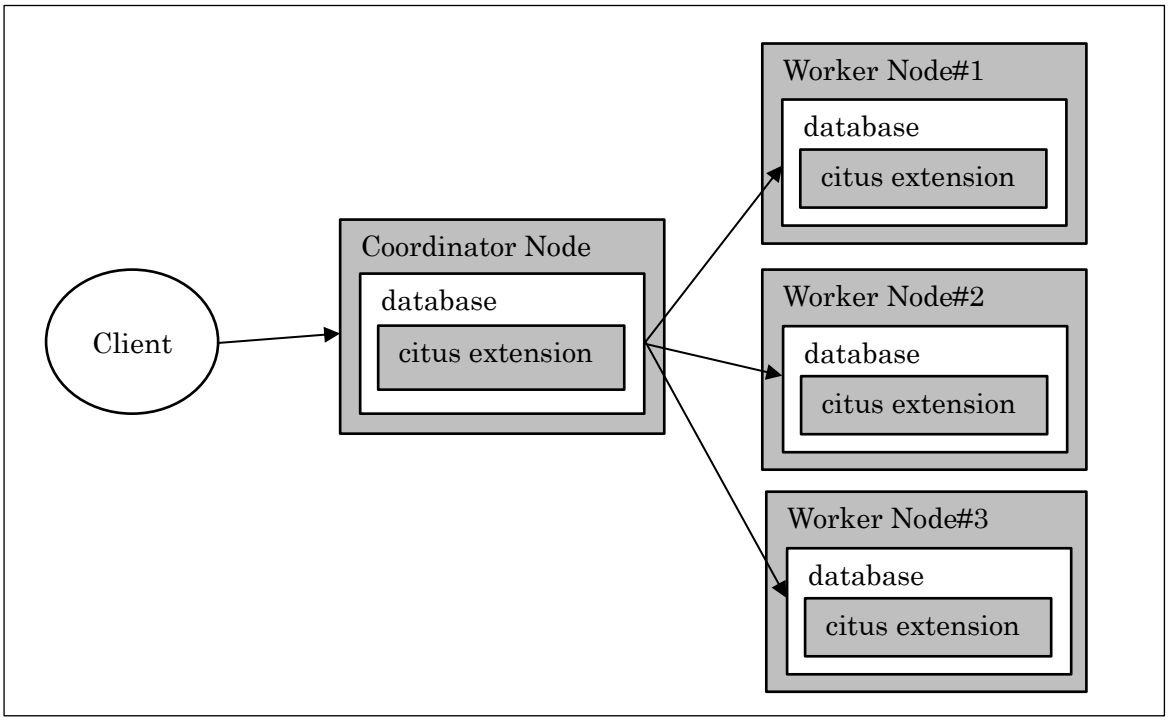
□ ワーカー・ノード

コーディネーター・ノードから SQL を受け取り、結果を返す PostgreSQL インスタンスです。データはワーカー・ノードに保存されます。コーディネーター・ノードが複数のワーカー・ノードを管理するクラスターを構成して処理を分散します。

□ citus エクステンション

Citus の全機能を提供する PostgreSQL エクステンションです。コーディネーター・ノードとすべてのワーカー・ノードにインストールします。ノードの役割は異なりますが、インストールするエクステンションは同一です。

図 2 citus の構成



各ワーカー・ノードはプライマリ・ノードとセカンダリ・ノードから構成されるノード・グループを構成することもできます。

□ プロセス構成

Citus は複数のバックグラウンド・プロセスから構成されています。Citus Maintenance Daemon プロセスは最初のトランザクションが実行されると起動します。

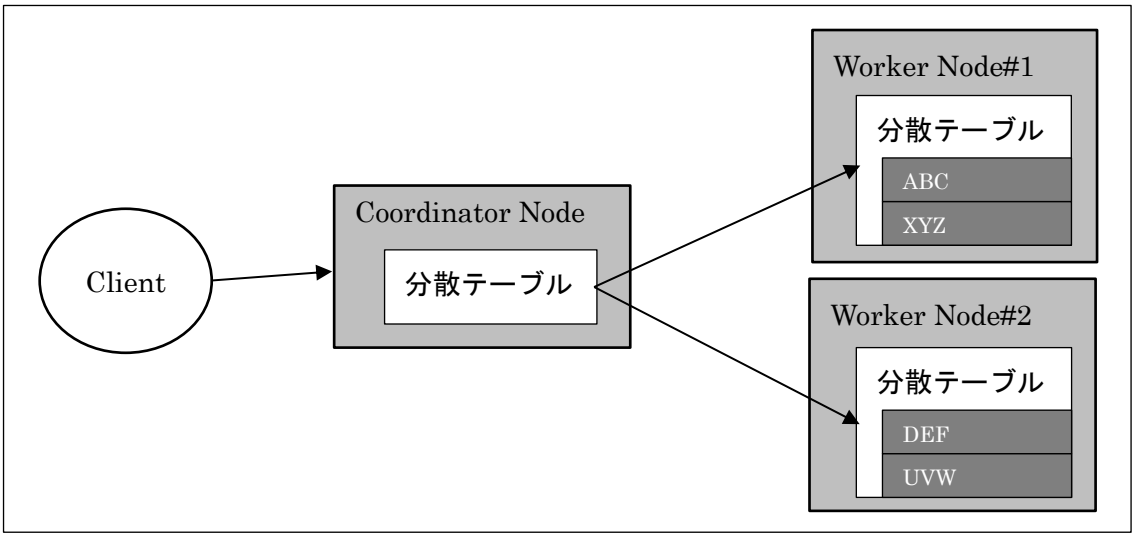
表 4 プロセス構成

プロセス名	起動インスタンス
postgres: Citus Maintenance Daemon	コーディネーター・ノード ワーカー・ノード
postgres	ワーカー・ノードで SQL 文を実行するプロセス（バックエンド）

2.1.2 分散テーブル

タプルを複数のワーカー・ノードに分散させて保存するテーブルを分散テーブル（Distributed Table）と呼びます。

図 3 分散テーブル



分散テーブルを作成する場合は通常のテーブルを作成後、create_distributed_table 関数にテーブル名と分散キーに使用する列名を指定して実行します。この操作はコーディネーター・ノードで行い、一般ユーザーでも実行できます。第３パラメーターとして分散方法も指定できます（append, hash, range）。デフォルトは hash です。コーディネーター・ノー

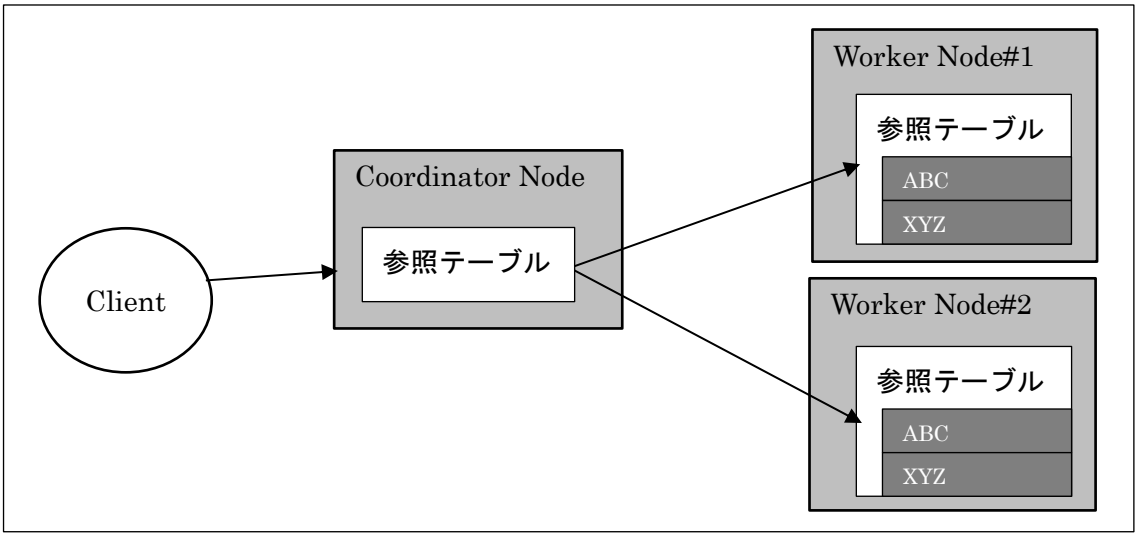
ドにのみデータが保存されるテーブル（従来のテーブル）はローカル・テーブルと呼ばれます。

2.1.3 参照テーブル

タプルを全ワーカー・ノードにミラー化するテーブルを参照テーブル（Reference Table）と呼びます。小規模なディメンジョン・テーブルは、参照テーブルとして作成すると分散テーブルとの結合を高速に行うことができます。

参照テーブルを作成するには CREATE TABLE 文で通常のテーブルを作成後、create_reference_table 関数にテーブル名を指定して実行します。

図 4 参照テーブル



2.1.4 列指向テーブル

Citus 10 の新機能として列指向テーブル（Columnar Table）が利用できるようになりました。一般的な RDBMS はデータをタプル（レコード）単位で保存します。これに対して列指向テーブルでは列の単位で圧縮保存します。列指向テーブルは必要なストレージ領域を小さくすることで全件検索時の I/O を削減することを目指しています。

Citus の列指向テーブルは前述の分散テーブルと組み合わせることにより、複数ノードによる分散列指向検索システムを構築できます。Citus Data 社では従来は FOREIGN DATA WRAPPER である cstore_fdw を提供してきましたが、Columnar Table は cstore_fdw に代わるソリューションとなります。

表 5 テーブルの構成と用語

テーブル種別	保存フォーマット	備考
分散テーブル	Heap テーブル	LOGGED / UNLOGGED
	列指向テーブル	
参照テーブル	Heap テーブル	LOGGED / UNLOGGED
	列指向テーブル	
ローカル・テーブル	Heap テーブル	LOGGED / UNLOGGED / TEMPORARY
	列指向テーブル	

2.2 インストールと準備

2.2.1 検証環境

検証は単一ノード内（ホスト名 rel78-5）で、複数のインスタンスを起動することで疑似的に複数ノードに分散する環境を構築しました。

表 6 インスタンス構成

インスタンス	ポート番号	役割	備考
1	5432	コーディネーター・ノード	
2	5442	ワーカー・ノード#1	
3	5443	ワーカー・ノード#2	
4	5444	ワーカー・ノード#3	
5	5445	ワーカー・ノード#4	後から追加／削除
6	5452	ワーカー・ノード#1	セカンダリ・ノード
7	5453	ワーカー・ノード#2	セカンダリ・ノード
8	5454	ワーカー・ノード#3	セカンダリ・ノード

表 7 共通設定（データベース）

設定項目	設定値	備考
データベース	postgres	citus エクステンションをインストール
接続ユーザー	demo	

表 8 検証環境の共通設定（PostgreSQL パラメーター）

パラメーター	設定値	備考
shared_preload_libraries	citus	必須
logging_collector	on	
listen_addresses	*	
port	5432～5445	各インスタンスで異なる値を設定
max_prepared_transactions	200	自動設定される
ssl	on	自動設定される
ssl_ciphers	ECDHE-ECDSA-AES128-GCM-SHA256:<<略>>	自動設定される

2.2.2 インストール方法

citus のインストールは RPM パッケージで行う方法と、ソースコードからインストールする方法があります。パッケージを使ってインストールする方法は Citus Data のホームページ「http://docs.citusdata.com/en/v10.1/installation/single_node_rhel.html」で紹介されています。Docker による利用方法は「https://github.com/citusdata/docker」を参照してください。

2.2.3 ソースコードからのインストール

検証環境はソースコードからインストールして構築しました。ソースコードは GitHub（https://github.com/citusdata/citus）から取得できます。

□ PostgreSQL の設定

Citus クラスターは複数の PostgreSQL インスタンスから構成されています。インスタンス間は SSL を使った通信が必須です（パラメーターcitus.node_conninfo のデフォルト値が sslmode=require であるため）。このため Citus を利用する PostgreSQL には SSL を有効化したバイナリを使用する必要があります。現状でサポートされている PostgreSQL のバージョンは 12 または 13 です。

□ パッケージの入手

Citus エクステンションのインストールには事前に libcurl-devel パッケージ、lz4-devel パッケージ、libzstd-devel パッケージ（Red Hat Enterprise Linux の場合）のインストールが必要です。またインストールされている PostgreSQL の情報入手のため、pg_config コマンドが環境変数 PATH 内にインストールされている必要があります。

□ ソースの展開とビルド

GitHub から git clone コマンドを使うかダウンロードした zip ファイルを展開します。

例 1 ビルドとインストール

```
$ cd citus
$ ./autogen.sh
$ ./configure
checking for a sed that does not truncate output... /bin/sed
checking for gawk... gawk
checking for flex... /bin/flex
<<途中省略>>
config.status: creating src/include/citus_config.h
config.status: creating src/include/citus_version.h

$ make
Makefile:51: warning: overriding recipe for target `check'
/usr/local/pgsql/lib/pgxs/src/makefiles/pgxs.mk:433: warning: ignoring old
recipe for target `check'
make -C src/backend/distributed/ clean
<<途中省略>>
  -L/usr/local/pgsql/lib -lpq  -lssl -lcrypto
make[1]: Leaving directory `/home/postgres/citus/src/backend/distributed'
$ su
# make install
Makefile:51: warning: overriding recipe for target `check'
/usr/local/pgsql/lib/pgxs/src/makefiles/pgxs.mk:433: warning: ignoring old
recipe for target `check'
<<途中省略>>
/bin/install -c -m 644 ./src/include/citus_version.h
'/usr/local/pgsql/include/server/'
/bin/install -c -m 644 /home/postgres/citus/./src/include/distributed/*.h
'/usr/local/pgsql/include/server/distributed/'
$
```

インストールが完了すると、{PostgreSQL}/share/extensions ディレクトリに citus エクステンションと追加ファイルが保存されます。pg_available_extensions カタログから確認できます。

例 2 インストールの確認

```
postgres=# SELECT * FROM pg_available_extensions WHERE name = 'citus' ;
-[ RECORD 1 ]-----+--------------------------
name              | citus
default_version   | 10.2-1
installed_version | 10.2-1
comment           | Citus distributed database
```

2.2.4 エクステンションの導入

Citus を利用するインスタンスには citus エクステンションを導入します。下記の例では postgres データベースに導入しています。postgres 以外のデータベースを使用する場合にはコーディネーター・ノードとワーカー・ノードにあらかじめ同一名称のデータベースを作成しておき、利用するデータベース上で CREATE EXTENSION 文を実行します。

コーディネーター・ノードとすべてのワーカー・ノードに同様にインストールを行います。事前にパラメーターshared_preload_libraries に citus が設定されている必要があります。

例 3 エクステンションの導入

```
postgres=# SHOW shared_preload_libraries ;
 shared_preload_libraries
--------------------------
 citus
(1 row)
postgres=# CREATE EXTENSION citus ;
CREATE EXTENSION
```

エクステンションが導入されると以下のパラメーターが自動的に更新されます。

表 9 自動設定されるパラメーター

パラメーター名	デフォルト値	設定値
ssl	off	on
ssl_ciphers	HIGH:MEDIUM:+3DES:!aNULL	ECDHE-ECDSA-AES128-GCM-SHA256:<<略>>
max_prepared_transactions	0	200

エクステンションを導入すると各データベースに以下のテーブルが作成されます。

表 10 作成されるテーブルまたはビュー（pg_catalog スキーマ）

テーブル名	用途
pg_dist_authinfo	ノード間通信の認証パラメーター
pg_dist_colocation	テーブル分散状況の取得
pg_dist_local_group	用途不明（おそらく現在は使われていない）
pg_dist_node	ワーカー・ノードの一覧
pg_dist_node_metadata	サーバーID を保持
pg_dist_partition	分散テーブル、参照テーブルの一覧
pg_dist_placement	ワーカー・ノードのシャードの場所を追跡
pg_dist_poolinfo	コネクション・プール情報
pg_dist_rebalance_strategy	リバランス戦略情報
pg_dist_shard	テーブルの分散方法の取得
pg_dist_shard_placement	テーブルの分散状況の取得（テーブルとノードの対応）
pg_dist_transaction	用途不明（おそらく現在は使われていない）
citus_dist_stat_activity	全ワーカー・ノードで実行される SQL 文情報
citus_lock_waits	全ワーカー・ノードで実行される SQL 文の待機情報
citus_shard_indexes_on_worker	用途不明（おそらく現在は使われていない）
citus_shards	シャードの一覧
citus_shards_on_worker	用途不明（おそらく現在は使われていない）
citus_stat_statements	クエリーの実行統計
citus_worker_stat_activity	ワーカー上のクエリー一覧

表 11 作成されるテーブル（citus スキーマ）

テーブル名	用途
pg_dist_object	分散オブジェクトの一覧

表 12 作成されるテーブル（columnar スキーマ）

テーブル名	用途
chunk	チャンクの一覧
chunk_group	チャンク・グループの一覧
options	列指向テーブルの構成オプション
stripe	ストライプの一覧

表 13 作成されるビュー（public スキーマ）

テーブル名	用途
citus_tables	分散テーブルと参照テーブルの一覧

エクステンションを導入すると各データベースに以下のシーケンスが作成されます。

表 14 作成されるシーケンス（columnar スキーマ）

シーケンス名	用途
storageid_seq	ストレージ ID 番号

表 15 作成されるシーケンス（pg_catalog スキーマ）

シーケンス名	用途
pg_dist_colocationid_seq	Co-Location 番号
pg_dist_groupid_seq	グループ番号
pg_dist_node_nodeid_seq	ノード番号
pg_dist_placement_placementid_seq	不明
pg_dist_shardid_seq	ShardID 番号

public スキーマには citus_tables ビューが追加されます。このビューからは分散テーブルと参照テーブルの一覧を確認することができます。

例 4 citus_tables ビューの検索

```
postgres=> SELECT table_name, citus_table_type, shard_count
                FROM citus_tables ;
table_name | citus_table_type | shard_count
------------+------------------+-------------
 dist1      | distributed      |           6
 dist2      | distributed      |          12
 ref1       | reference        |           1
(3 rows)
```

2.2.5 ワーカー・ノードの登録

コーディネーター・ノード上でワーカー・ノードを登録します。citus_add_node 関数にワーカー・ノードのホスト名（または TCP/IP アドレス）とポート番号を指定します。

citus_add_node 関数実行時点でワーカー・ノードは起動している必要があります。接続できない場合、ワーカー・ノードは登録できません。

例 5 ワーカー・ノードの登録

```
postgres=# SELECT citus_add_node('localhost', 5442) ;
 citus_add_node
----------------
              2
(1 row)
postgres=# SELECT citus_add_node('localhost', 5443) ;
 citus_add_node
----------------
              3
(1 row)
postgres=# SELECT citus_add_node('localhost', 5444) ;
 citus_add_node
----------------
              4
(1 row)
```

ワーカー・ノードの登録を行うユーザーは SUPERUSER 属性が必要です。追加したノードの情報は pg_dist_node カタログまたは citus_get_active_worker_nodes 関数で確認することができます。

例 6 ワーカー・ノードの確認

```
postgres=> SELECT * FROM pg_dist_node ;
nodeid | groupid | nodename  | nodeport | noderack | hasmetadata | isactive | noderole |
nodecluster | metadatasynced | shouldhaveshards
--------+---------+-----------+----------+----------+-------------+----------+----------+-
------------+----------------+------------------
      2 |       2 | localhost |     5442 | default  | f           | t        | primary  |
default     | f              | t
      3 |       3 | localhost |     5443 | default  | f           | t        | primary  |
default     | f              | t
      4 |       4 | localhost |     5444 | default  | f           | t        | primary  |
default     | f              | t
(3 rows)
postgres=> SELECT * FROM citus_get_active_worker_nodes() ;
node_name | node_port
-----------+-----------
 localhost |      5443
 localhost |      5442
 localhost |      5444
(3 rows)
```

□ セキュリティ

コーディネーター・ノードとワーカー・ノード間の認証は接続ユーザー名およびパスワードの入力設定が存在しないため、自動接続できる設定が必要です。

3. 検証結果

3.1 テーブルの作成

3.1.1 分散テーブル

コーディネーター・ノードで create_distributed_table 関数を実行すると分散テーブルを作成できます。ワーカー・ノード上には元のテーブルと同一構造のテーブル（シャードと呼びます）が作成されます。テーブル名は「{元のテーブル名}_{ShardID}」になります。{ShardID} 部分は６桁の数字から構成されます。テーブルの所有者は元のテーブルと同一になります。作成されるテーブル数は create_distributed_table 関数実行時のセッション・パラメーターcitus.shard_count と citus.shard_replication_factor に依存します。

セッション・パラメーターcitus.shard_count は分散テーブル数を指定します。セッション・パラメーターcitus.shard_replication_factor にはデータのミラー数を指定します。このためワーカー・ノード全体で作成される全テーブル数は「citus.shard_count × citus.shard_replication_factor」になります。

表 16 関連するセッション・パラメーター

パラメーター	デフォルト値	備考
citus.shard_count	32	create_distributed_table 関数のパラメーターでも指定可能
citus.shard_replication_factor	1	デフォルトではミラー無し

例 7 コーディネーター・ノードでテーブルの登録

```
postgres=> SET citus.shard_count = 6 ;
SET
postgres=> SET citus.shard_replication_factor = 2 ;
SET
postgres=> CREATE TABLE dist1(c1 INT PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT create_distributed_table('dist1', 'c1') ;
 create_distributed_table
--------------------------


(1 row)
```

ワーカー・ノードでは複数のテーブルが作成されます。上記の例では分散の個数が６で、ミラー個数が２であるため、全体で１２テーブル作成されます。３ノードで分散されるため、各ワーカー・ノードでテーブルが４個ずつ作成されます。

例 8 ワーカー・ノードでテーブルの自動作成

```
postgres=> ¥d
            List of relations
 Schema |     Name      | Type  |  Owner
--------+---------------+-------+----------
 public | citus_tables | view  | postgres
 public | dist1_102137 | table | demo
 public | dist1_102139 | table | demo
 public | dist1_102140 | table | demo
 public | dist1_102142 | table | demo
(5 rows)
```

テーブルのレプリカは異なるワーカー・ノード上に作成されます。同じテーブル名は同一のデータが格納されます。

表 17 テーブルの分散と名前

コーディネーター	ワーカー#1	ワーカー#2	ワーカー#3	備考
dist1	dist1_102137	dist1_102137		
		dist1_102138	dist1_102138	
	dist1_102139		dist1_102139	
	dist1_102140	dist1_102140		
		dist1_102141	dist1_102141	
	dist1_102142		dist1_102142	

シャードを構成するファイルの最大値を指定するcitus.shard_max_size(デフォルト1G)がありますが、ハッシュによる分散テーブルの場合、有効性が確認できませんでした。

データの分散方法は Co-Location と呼ばれる概念です。本資料では Co-Location については検証していません（マニュアル：https://docs.citusdata.com/en/stable/sharding/data_modeling.html#table-co-location）。元のテーブルと ShardID の関係は pg_dist_shard カタログで参照できます。

□ タプルの分散方法

create_distributed_table 関数の第３パラメータにはタプルの分散方法を指定します。デフォルト値は hash で、ハッシュ値の範囲によって保存先のシャードを決定します。このパラメーターにはその他に range と append を指定できます。時系列データの保存等には append が有効ですが検証は行っていません。

マニュアル： https://docs.citusdata.com/en/stable/develop/append.html

□ 分散テーブルの削除

コーディネーター・ノードで PostgreSQL 標準の DROP TABLE 文を実行すると、分散テーブルを削除することができます。自動的にワーカー・ノードのテーブルも削除されます。ワーカー・ノードからシャードを削除し、元のテーブルに戻すには undistribute_table 関数を実行します。シャード上のデータはコーディネーター・ノードのテーブルにコピーされます。

例 9 分散テーブルの解消

```
postgres=> SELECT undistribute_table('dist2') ;
NOTICE:  creating a new table for public.dist2
NOTICE:  moving the data of public.dist2
NOTICE:  dropping the old public.dist2
NOTICE:  renaming the new table to public.dist2
 undistribute_table
--------------------


(1 row)
```

□ 主キー制約と分散キー

create_distributed_table 関数にはテーブル名と分散に使用する列を指定します。テーブルに主キー／一意キー制約が存在する場合は、分散キーとして制約に含まれる列を指定する必要があります。

例 10 主キー以外を分散キーに指定したエラー

```
postgres=> CREATE TABLE const1(c1 INT PRIMARY KEY, c2 INT, c3 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT create_distributed_table('const1', 'c2') ;
ERROR:  cannot create constraint on "const1"
DETAIL:  Distributed relations cannot have UNIQUE, EXCLUDE, or PRIMARY KEY
constraints that do not include the partition column (with an equality operator
if EXCLUDE).
```

□ パーティション・テーブルと分散

分散テーブルはパーティション・テーブルもサポートしています。

例 11 パーティション・テーブルの分散実行

```
postgres=> CREATE TABLE part1(c1 INT, c2 VARCHAR(10)) PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> SELECT create_distributed_table('part1', 'c1') ;
create_distributed_table
--------------------------


(1 row)
```

パーティション・テーブルにパーティションを追加した場合、ワーカー・ノードにも自動的にパーティション用の分散テーブルが作成されます。

□ 分散テーブルの一覧

分散テーブルと参照テーブルの一覧は pg_dist_partition カタログまたは citus_tables ビューを検索することで取得できます。

例 12 分散テーブルの一覧

```
postgres=> SELECT logicalrelid, partmethod, repmodel FROM pg_dist_partition ;
 logicalrelid | partmethod | repmodel
--------------+------------+----------
 dist1        | h          | c
 part1        | h          | c
 part1v1      | h          | c
 ref1         | n          | t
(4 rows)
```

ワーカー・ノードに作成されるテーブル数やレプリカ数を検索するには、pg_dist_colocation カタログと pg_dist_partition カタログを結合します。

例 13 分散テーブルの属性取得

```
postgres=> SELECT logicalrelid, partmethod, repmodel, shardcount,
replicationfactor FROM pg_dist_partition p INNER JOIN pg_dist_colocation c ON
p.colocationid = c. colocationid ;
logicalrelid | partmethod | repmodel | shardcount | replicationfactor
--------------+------------+----------+------------+-------------------
 dist1        | h          | c        |          6 |                 2
 part1        | h          | c        |          6 |                 2
 part1v1      | h          | c        |          6 |                 2
 ref1         | n          | t        |          1 |                -1
(4 rows)
```

□ データ格納済テーブルの分散化

既にタプルが格納されているコーディネーター・ノード上のテーブルを分散化することができます。既存のタプルはワーカー・ノードにコピーされます。ただし既存データは削除されません。既存タプルを削除するには truncate_local_data_after_distributing_table 関数を実行します。

例 14 既存テーブルの分散化

```
postgres=> CREATE TABLE dist3(c1 INT PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> INSERT INTO dist3 SELECT * FROM data1 ;
INSERT 0 1000000
postgres=> SELECT create_distributed_table('dist3', 'c1') ;
NOTICE:  Copying data from local table...
NOTICE:  copying the data has completed
DETAIL:  The local data in the table is no longer visible, but is still on
disk.
HINT:          To      remove      the      local      data,      run:      SELECT
truncate_local_data_after_distributing_table($$public.dist3$$)
 create_distributed_table
--------------------------


(1 row)
postgres=> SELECT pg_size_pretty(pg_relation_size('dist3')) ;
 pg_size_pretty
----------------
 422 MB
(1 row)

postgres=> SELECT truncate_local_data_after_distributing_table('dist3') ;
 truncate_local_data_after_distributing_table
----------------------------------------------


(1 row)

postgres=> SELECT pg_size_pretty(pg_relation_size('dist3')) ;
 pg_size_pretty
----------------
 0 bytes
(1 row)
```

□ 列の制限

GENERATED AS IDENTITY 列はサポートされません。

例 15 GENERATED 列を持つテーブル

```
postgres=> CREATE TABLE dist4(c1 INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
                c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT create_distributed_table('dist4', 'c1') ;
ERROR:  cannot distribute relation: dist4
DETAIL:  Distributed relations must not use GENERATED ... AS IDENTITY.
```

serial 型の列を分散キーに指定すると、シャード・テーブルでは単純な integer 型に変換されます。このためシーケンスの操作はマスター・ノードで行われると思われます。

例 16 マスター・ノードで SERIAL 型列を持つテーブル作成

```
postgres=> CREATE TABLE serial1(c1 SERIAL, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT create_distributed_table('serial1', 'c1') ;
 create_distributed_table
--------------------------


(1 row)
```

例 17 ワーカー・ノードのテーブル構成

```
postgres=> ¥d serial1_102103
                  Table "public.serial1_102103"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | integer               |           | not null |
 c2     | character varying(10) |           |          |
```

serial 型に限らず列のデフォルトにシーケンスを指定している場合はシャード・テーブルのテーブル定義からはシーケンス情報は削除されます。

□ 実行計画

分散テーブルに対する SQL 文の実行計画を確認します。分散テーブルに対する実行計画は Custom Scan (Citus Adaptive) から始まるノードが表示されます。Task Count: にはアクセスしたワーカーのシャード数が表示されます。Task Shown: には EXPLAIN 文で表示されている実行計画が全体の中の何個かを示します。デフォルトでは一部のみ表示されます。citus.explain_all_tasks パラメーターを on にすることで全タスクの実行計画が表示されます。

例 18 分散テーブルに対する実行計画

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT COUNT(*) FROM dist1 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=250.00..250.02 rows=1 width=8) (actual time=289.538..289…
   Output: COALESCE((pg_catalog.sum(remote_scan.count))::bigint, '0'::bigint)
   -> Custom Scan (Citus Adaptive)  (cost=0.00..0.00 rows=100000 width…
         Output: remote_scan.count
         Task Count: 6
         Tuple data received from nodes: 42 bytes
         Tasks Shown: One of 6
         ->  Task
               Query: SELECT count(*) AS count FROM public.dist1_102139 dist1
WHERE true
               Tuple data received from node: 7 bytes
               Node: host=localhost port=5444 dbname=postgres
               ->  Finalize Aggregate  (cost=18695.05..18695.06 row…
                     Output: count(*)
                     ->  Gather  (cost=18694.83..18695.04 rows=2 …
                           Output: (PARTIAL count(*))
       <<途中省略>>
                  Planning Time: 53.419 ms
                   Execution Time: 111.194 ms
 Planning Time: 0.286 ms
 Execution Time: 289.570 ms
(29 rows)
```

□ テーブル構造の制限

UNLOGGED テーブルは分散テーブルとして作成できますが、TEMPORARY テーブルは分散テーブルとして作成できません。

□ トリガー

トリガーが指定されたテーブルに対する create_distributed_table 関数の実行はエラーになります。

例 19 トリガーを持つテーブルの分散テーブル化

```
postgres=> SELECT create_distributed_table('dist2', 'c1') ;
ERROR:  cannot distribute relation "dist2" because it has triggers
DETAIL:  Citus does not support distributing tables with triggers.
HINT:  Drop all the triggers on "dist2" and retry.
```

□ テーブルのサイズ

ワーカーに分散されたテーブル全体のサイズは citus_total_relation_size 関数で取得できます。参照テーブルに指定するとミラー化されたテーブルの合計サイズになります。

例 20 トータル・サイズ

```
postgres=> CREATE TABLE ref2(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT create_reference_table('ref2') ;
 create_reference_table
------------------------


(1 row)

postgres=> SELECT citus_total_relation_size('ref2') ;
 citus_total_relation_size
---------------------------
                     49152
(1 row)
```

□ ワーカー・ノードで実行される SQL

分散テーブル作成時にはワーカー・ノードでは以下の SQL 文が実行されます。

例 21 ワーカー・ノードの SQL（抜粋）

```
-  BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;SELECT
   assign_distributed_transaction_id(0, 3, '2021-07-30 17:24:41.802404+09');
-  SELECT worker_apply_shard_ddl_command (102279, 'CREATE TABLE public.dist1
   (c1 numeric, c2 character varying(10)) USING heap')
-  CREATE TABLE public.dist1 (c1 numeric, c2 character varying(10)) USING heap
-  ALTER TABLE public.dist1 OWNER TO demo
   上記 3 SQL をシャード分繰り返し
-  PREPARE TRANSACTION 'citus_0_51998_3_3'
-  COMMIT PREPARED 'citus_0_51998_3_3'
```

□ TABLESPACE の制限

デフォルト以外の TABLESPACE に作成されたテーブルを分散化すると、シャードには TABLESPACE 句が伝播しません。

例 22 TABLESPACE の指定（コーディネーター・ノード）

```
postgres=> CREATE TABLE dist1(c1 NUMERIC, c2 VARCHAR(10)) TABLESPACE ts1 ;
CREATE TABLE
postgres=> SELECT create_distributed_table('dist1', 'c1') ;
 create_distributed_table
--------------------------


(1 row)
```

例 23 TABLESPACE の指定（ワーカー・ノード）

```
postgres=> ¥d+ dist1_102461
                          Table "public.dist1_102461"
 Column |         Type          | Collation | Nullable | Default | Storage…
--------+-----------------------+-----------+----------+---------+--------…
 c1     | numeric               |           |          |         | main   …
 c2     | character varying(10) |           |          |         | extende…
Access method: heap
```

3.1.2 参照テーブル

参照テーブルを作成するにはコーディネーター・ノードで create_reference_table 関数を実行します。この関数を実行すると全ワーカー・ノード上に同一のテーブルが作成されます。テーブル名は「{元のテーブル名}_{ShardID}」になります。{ShardID}部分は６桁の数字から構成されます。この操作は一般ユーザーでも実行できます。

例 24 コーディネーター・ノードでテーブルの登録

```
postgres=> CREATE TABLE ref1(c1 INT PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT create_reference_table('ref1') ;
 create_reference_table
------------------------


(1 row)
```

ワーカー・ノードでは同一構成のテーブルが作成されます。

例 25 ワーカー・ノードでテーブルの自動作成

```
postgres=> ¥d ref1_102432
                  Table "public.ref1_102432"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | integer               |           | not null |
 c2     | character varying(10) |           |          |
Indexes:
    "ref1_pkey_102432" PRIMARY KEY, btree (c1)
```

□ 実行計画

参照テーブルの検索は分散テーブルと同じように「Custom Scan (Adaptive)」と呼ばれるノードで表示されます。

例 26 参照テーブルの検索実行計画

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT COUNT(*) FROM ref1 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Custom Scan (Citus Adaptive)  (cost=0.00..0.00 rows=0 width=0) (act…
   Output: remote_scan.count
   Task Count: 1
   Tuple data received from nodes: 1 bytes
   Tasks Shown: All
   ->  Task
         Query: SELECT count(*) AS count FROM public.ref1_102220 ref1
         Tuple data received from node: 1 bytes
         Node: host=localhost port=5442 dbname=postgres
         ->  Aggregate  (cost=24.50..24.51 rows=1 width=8) (actual time=0 …
               Output: count(*)
               ->  Seq Scan on public.ref1_102220 ref1  (cost=0.00..21.60 …
                     Output: c1, c2
       <<途中省略>>
 Planning Time: 0.037 ms
 Execution Time: 1.383 ms
(17 rows)
```

□ トリガー

参照テーブルにはトリガーは追加できません。トリガー付のテーブルを参照テーブルに変換するとサポートされないと警告が出力されますが、トリガー自体は有効になっています。

例 27 トリガー付テーブルを参照テーブルに変換

```
postgres=> CREATE TABLE ref3 (c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE TRIGGER trig1_ref3 BEFORE TRUNCATE ON ref3 EXECUTE FUNCTION
               trig_func1() ;
CREATE TRIGGER
postgres=> SELECT create_reference_table('ref3') ;
ERROR:  cannot distribute relation "ref3" because it has triggers
DETAIL:  Citus does not support distributing tables with triggers.
HINT:  Drop all the triggers on "ref3" and retry.
postgres=> ¥d ref3
                     Table "public.ref3"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | numeric               |           | not null |
 c2     | character varying(10) |           |          |
<<途中省略>>
Triggers:
    trig1_ref3 BEFORE TRUNCATE ON ref3 FOR EACH STATEMENT EXECUTE FUNCTION
trig_func1()
```

□ テーブルの削除

テーブルの削除は DROP TABLE 文を実行します。ワーカー・ノードからシャードを削除し、元のテーブルに戻すには undistribute_table 関数を実行します。

□ テーブル構造の制限

UNLOGGED テーブルは参照テーブルとして作成できますが、TEMPORARY テーブルは参照テーブルとして作成できません。GENERATED AS IDENTITY 列はサポートされません。

3.1.3 列指向テーブル

列指向テーブル（Columnar Table）は Citus 10 の新機能です。列指向テーブルは PostgreSQL 12 の Pluggable Storage Engine の機能を利用し、アクセス・メソッドとして実装されています。citus エクステンションを導入すると columnar アクセス・メソッドが利用できるようになります。

例 28 テーブル・アクセス・メソッド

```
postgres=> SELECT * FROM pg_am ;
  oid  |  amname  |         amhandler          | amtype
-------+----------+----------------------------+--------
     2 | heap     | heap_tableam_handler       | t
   403 | btree    | bthandler                  | i
   405 | hash     | hashhandler                | i
   783 | gist     | gisthandler                | i
  2742 | gin      | ginhandler                 | i
  4000 | spgist   | spghandler                 | i
  3580 | brin     | brinhandler                | i
 16870 | columnar | columnar.columnar_handler  | t
(8 rows)
```

□ 列指向テーブルの作成

列指向テーブルを作成するには CREATE TABLE 文に USING columnar 句を指定します。

例 29 列指向テーブルの作成

```
postgres=> CREATE TABLE column1(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10))
                USING columnar ;
CREATE TABLE
postgres=> ¥d+ column1
                                        Table "public.column1"
 Column |         Type          | Collation | Nullable | Default | Storage…
--------+-----------------------+-----------+----------+---------+---------…
 c1     | numeric               |           | not null |         | main    …
 c2     | character varying(10) |           |          |         | extended…
Indexes:
    "column1_pkey" PRIMARY KEY, btree (c1)
Access method: columnar
```

テーブル作成時にはいくつかのパラメーターが影響します。列指向テーブルの作成に関するパラメーターを表にまとめています。

表 18 テーブル作成時のパラメーター

パラメーター名	デフォルト	説明
columnar.chunk_group_row_limit	10000	チャンクに含まれる新規挿入データの数
columnar.compression	zstd	圧縮アルゴリズム（none, pglz, lz4, zstd）
columnar.compression_level	3	圧縮レベル（1～19）
columnar.stripe_row_limit	150000	ストライプに含まれる新規挿入データの最大数

マニュアルにはパラメーターcolumnar.compression には lz4hc が指定できるように書かれていますが設定はエラーになります。各テーブルのオプション値を確認する場合は columnar.options テーブルを検索します。作成済のテーブルに対してオプションを変更するには alter_columnar_table_set 関数を実行します。

例 30 列指向テーブルの作成オプション

```
postgres=> SELECT * FROM columnar.options ;
 regclass | chunk_group_row_limit | stripe_row_limit | compression_level | compression
----------+-----------------------+------------------+-------------------+-------------
 column1  |                 10000 |           150000 |                 3 | zstd
 column2  |                 10000 |           150000 |                 3 | zstd
(2 rows)
```

マニュアル：http://docs.citusdata.com/en/v10.1/develop/api_udf.html#alter-columnar-table-set

□ 用語

列指向テーブル独自に以下の用語を使用します。

・ ストライプ

データの格納単位をストライプと呼び、圧縮の単位になります。ストライプは単一トランザクション内で挿入されるタプル数またはcolumnar.stripe_row_limitパラメーターで指定されるタプル数のいずれか小さい方でまとめられます。ストライプの内容は columnar.stripe カタログを参照します。columnar.stripe カタログはストレージ ID（storage_id 列）で識別されます。列指向テーブルとストレージ ID の対応を確認するため

にはテーブルに対して VACUUM VERBOSE 文を実行します。

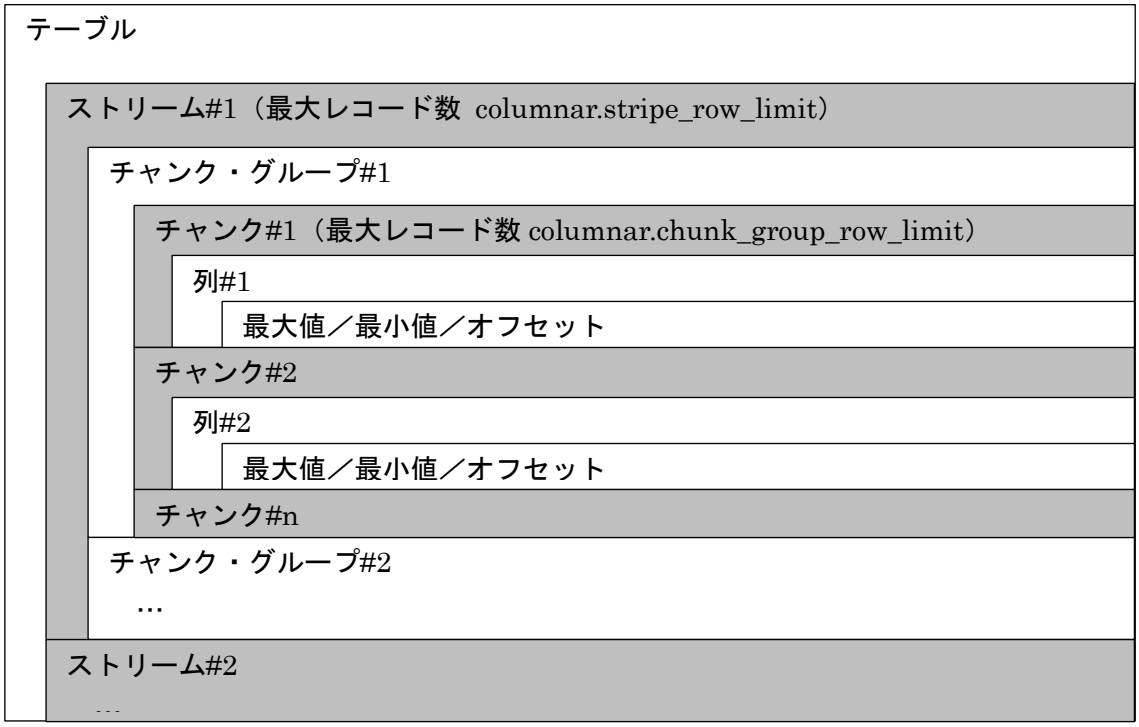
・ チャンク

ストライプ内の列データを一定数単位でまとめたブロックをチャンクと呼びます。チャンクには列値に加えて最大値、最小値、ストリーム内のオフセット等が記録されています。チャンクの内容は columnar.chunk カタログを参照します。

・ チャンク・グループ

複数列のデータをまとめたグループをチャンク・グループと呼びます。チャンク・グループの内容は columnar.chunk_group カタログを参照します。

図 5 列指向テーブルのストレージ構造



□ 圧縮効果

以下の例では 2 列（1 列はシーケンス値、2 列目は単一文字列）、1,000 万タプルのテーブルを作成し、圧縮効率を確認しています。INSERT SELECT 文による一括ロードによる圧縮効果が確認できます。逆に小規模な挿入を繰り返すとテーブル・サイズが拡大する傾向にあります。

表 19 圧縮効率（1,000 万タプルを lz4 圧縮）

テーブルの状態	サイズ（MB）	説明
Heap テーブル	422	
列指向テーブル（一括ロード）	13	
列指向テーブル（10 タプル単位 COMMIT）	7,813	

圧縮方法による格納サイズを比較しています。テーブルの構造とデータは前述の表と同じです。検索 SQL は「SELECT * FROM テーブル」を実行しています。格納効率は zstd が高くなっていますが、格納時間にばらつきがあります。検索時間は有意な差が確認できませんでした。

表 20 圧縮効率（1,000 万タプルを持つテーブルで検証）

圧縮方法	圧縮レベル	サイズ（MB）	格納時間（秒）	検索時間（ms）	備考
lz4	1	42	4.334	1258.162	
	19	42	4.492	832.505	
zstd	1	13	3.327	863.642	
	19	13	31.566	854.601	
pglz	1	35	12.494	842.261	
	19	35	12.814	748.730	
none	-	232	27.636	786.118	

□ 既存のテーブルからの変換

通常のテーブルから列指向テーブルに変換する場合は alter_table_set_access_method 関数を実行してアクセス・メソッドを変更します。この関数はタプルが格納されているテーブルに対しても実行できます。この関数は列指向テーブルから通常の Heap テーブルに変換することもできます。alter_table_set_access_method 関数を実行すると、テーブル定義から主キー制約、一意制約、インデックス定義が削除されます。

例 31 列指向テーブルへの変換

```
postgres=> CREATE TABLE column2(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> INSERT INTO column2 SELECT * FROM data1 ;
INSERT 0 10000000
postgres=> SELECT alter_table_set_access_method('column2', 'columnar') ;
NOTICE:  creating a new table for public.column2
NOTICE:  moving the data of public.column2
NOTICE:  dropping the old public.column2
NOTICE:  renaming the new table to public.column2
 alter_table_set_access_method
-------------------------------


(1 row)
```

□ 実行計画

列指向テーブルへの検索実行計画では Custom Scan (Columnar Scan) ノードが表示されます。Columnar Chunk Groups Removed by Filter ノードが追加されています。Chunk は列内のデータ範囲を示します。Removed by Filter は Chunk に必要なデータが無いため Chunk 全体をスキップしたことを示します。

例 32 列指向テーブルに対する実行計画

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT COUNT(*) FROM column1 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=25000.00..25000.01 rows=1 width=8) (actual time=642.396…
   Output: count(*)
   ->  Custom Scan (ColumnarScan) on public.column1  (cost=0.00..0.00 rows …
         Columnar Chunk Groups Removed by Filter: 0
 Planning Time: 0.147 ms
 Execution Time: 642.418 ms
(6 rows)
```

□ 物理ファイル

列指向テーブルの物理ファイル・フォーマットは通常の Heap テーブルとは異なりますが、1GB までは単一ファイル、その後は 1GB 単位のファイルが複数作成される点では変化がありません。

例 33 列指向テーブルのファイル

```
postgres=> SELECT pg_size_pretty(pg_relation_size('column4')) ;
 pg_size_pretty
----------------
 16 GB
(1 row)
postgres=> SELECT pg_relation_filepath('column4') ;
 pg_relation_filepath
----------------------
 base/13580/17265
(1 row)
Postgres=> ¥! ls -l data.1/base/13580/17265*
-rw-------. 1 postgres 1073741824 Jul 30 13:56 data/base/13580/17265
-rw-------. 1 postgres 1073741824 Jul 30 13:54 data/base/13580/17265.1
-rw-------. 1 postgres 1073741824 Jul 30 13:55 data/base/13580/17265.10
<<以下省略>>
```

□ 分散テーブル

列指向テーブルは分散テーブルとして作成することができます。作成方法は通常の分散テーブルと同じです。下記の例では「Task Count: 6」となっており、６分割されたテーブルから結果を受け取っていることがわかります。

例 34 列指向テーブルの分散化

```
postgres=> CREATE TABLE column3(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10))
                USING columnar ;
CREATE TABLE
postgres=> SELECT create_distributed_table('column3', 'c1') ;
create_distributed_table
--------------------------


(1 row)
postgres=> INSERT INTO column3 SELECT * FROM data1 ;
INSERT 0 10000000
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM column3 ;
QUERY PLAN
------------------------------------------------------------------------------
 Aggregate  (cost=250.00..250.02 rows=1 width=8) (actual time=213.067..213…
   ->  Custom Scan (Citus Adaptive)  (cost=0.00..0.00 rows=100000 …
         Task Count: 6
         Tuple data received from nodes: 42 bytes
         Tasks Shown: One of 6
         ->  Task
               Tuple data received from node: 7 bytes
               Node: host=localhost port=5443 dbname=postgres
               ->  Aggregate  (cost=4170.91..4170.92 rows=1 width=8)…
                     ->  Custom Scan (ColumnarScan) on column3_102344 colu…
                           Columnar Chunk Groups Removed by Filter: 0
                 Planning Time: 0.105 ms
                 Execution Time: 177.422 ms
 Planning Time: 0.189 ms
 Execution Time: 213.086 ms
(15 rows)
```

□ パーティション

パーティション・テーブルと組み合わせて特定のパーティションのみ列指向テーブルとすることもできます。下記の例では特定のパーティションを列指向テーブルとして、かつ分散テーブル化しています。

例 35 パーティションとの組み合わせ

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION
                BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES FROM (1)
                TO (1000000) ;
CREATE TABLE
postgres=> CREATE TABLE part1v2 PARTITION OF part1 FOR VALUES FROM (1000000)
                TO (2000000) USING columnar ;
CREATE TABLE
postgres=> SELECT create_distributed_table('part1', 'c1') ;
 create_distributed_table
--------------------------


(1 row)
```

3.2 テーブルのメンテナンス

3.2.1 インデックスの作成

Citus 環境のテーブルに対してインデックスを作成します。

□ 分散テーブル／参照テーブル

分散テーブル、参照テーブルのコーディネーター・ノードのテーブルに対してインデックスを作成すると、ワーカー・ノードのテーブルにも同一構成のインデックスが追加されます。この動作はパラメーターcitus.enable_ddl_propagation を off に設定することで無効にできます（デフォルト値 on）。

例 36 インデックスの作成（コーディネーター・ノード）

```
postgres=> CREATE INDEX idx1_dist1 ON dist1(c2) ;
CREATE INDEX
```

例 37 インデックスの定義（ワーカー・ノード）

```
postgres=> ¥d dist1_102137
                  Table "public.dist1_102079"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | integer               |           | not null |
 c2     | character varying(10) |           |          |
Indexes:
    "dist1_pkey_102137" PRIMARY KEY, btree (c1)
    "idx1_dist1_102137" btree (c2)
```

□ 列指向テーブル

列指向テーブルはインデックスをサポートしていません。マニュアル（http://docs.citusdata.com/en/v10.1/admin_guide/table_management.html#usage）には以下の記述があります。

```
● No index support, index scans, or bitmap index scans
```

しかし CREATE TABLE 文の PRIMARY KEY 制約は有効に動作します。列指向テーブルに対する CREATE INDEX 文は警告が出力されますが成功します。マニュアルの問題か、本来 CREATE INDEX 文が動作してはいけないのかは不明です。

例 38 主キーとインデックスの作成

```
postgres=> CREATE TABLE column4 (c1 INT PRIMARY KEY, c2 VARCHAR(10))
                USING columnar ;
CREATE TABLE
postgres=> INSERT INTO column4 SELECT * FROM data1 ;
INSERT 0 10000000
postgres=> CREATE INDEX idx1_column4 ON column4 (c2) ;
NOTICE:  falling back to serial index build since parallel scan on columnar
tables is not supported
CREATE INDEX
postgres=> ¥d+ column4
                                        Table "public.column4"
 Column |         Type          | Collation | Nullable | Default | Storage…
--------+-----------------------+-----------+----------+---------+---------…
 c1     | integer               |           | not null |         | plain  …
 c2     | character varying(10) |           |          |         | extended…
Indexes:
    "column4_pkey" PRIMARY KEY, btree (c1)
    "idx1_column4" btree (c2)
Access method: columnar
```

実際に列指向テーブルに対する主キー検索を行うことができます。

例 39 列指向テーブルに対するインデックス検索

```
postgres=> EXPLAIN ANALYZE SELECT * FROM column1 WHERE c1=10000 ;
                                                        QUERY PLAN
------------------------------------------------------------------------------------------ …
 Index Scan using column1_pkey on column1  (cost=23.64..31.66 rows=1 width=…
   Index Cond: (c1 = '10000'::numeric)
 Planning Time: 0.189 ms
 Execution Time: 0.386 ms
(4 rows)
```

3.2.2 列の追加

分散テーブル、参照テーブル共にコーディネーター・ノードのテーブルに対して列を追加すると、ワーカー・ノードのテーブルにも同一構成の列が追加されます。この動作はパラメーターcitus.enable_ddl_propagation を off に設定することで無効にできます（デフォルト値 on）。

例 40 列の追加（コーディネーター・ノード）

```
postgres=> ALTER TABLE dist1 ADD COLUMN c3 VARCHAR(10) ;
ALTER TABLE
```

例 41 列の追加状況確認（ワーカー・ノード）

```
postgres=> ¥d dist1_102137
                  Table "public.dist1_102079"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | integer               |           | not null |
 c2     | character varying(10) |           |          |
 c3     | character varying(10) |           |          |
Indexes:
    "dist1_pkey_102137" PRIMARY KEY, btree (c1)
```

□ その他の変更

ALTER TABLE 文はすべて実行できるわけではありません。一部の構文の実行は制限されています。制限された構文を実行するとエラー・メッセージが出力されます。

例 42 実行できない ALTER TABLE 構文

```
postgres=> ALTER TABLE dist1 SET TABLESPACE ts1 ;
ERROR:  alter table command is currently unsupported
DETAIL:  Only ADD|DROP COLUMN, SET|DROP NOT NULL, SET|DROP DEFAULT,
ADD|DROP|VALIDATE CONSTRAINT, SET (), RESET (), ATTACH|DETACH PARTITION and
TYPE subcommands are supported.
```

3.2.3 バックアップ

pg_dump コマンドによる論理バックアップ・ファイルには分散テーブル、参照テーブル共にデータが格納されます。しかしテーブル作成時の DDL には create_distributed_table 関数や create_reference_table 関数は含まれません。テーブル・アクセス・メソッド（columnar 等）は一時的に SET default_table_access_method 文が実行されてアクセス・メソッド単位にテーブルが作成されるため、リストア時に列指向テーブルが作成されます。

3.2.4 参照テーブルへの変換

分散テーブルから参照テーブルに変換する関数として upgrade_to_reference_table 関数がマニュアル上に記載がありますが、Citus 9.5 から Citus 10.0-4 にアップグレードする際に削除されています。

3.3 SQL 文の実行

ここでは分散テーブルと参照テーブルに対する SQL 文の実行計画とワーカー・ノードで再実行される SQL 文を検証しています。

3.3.1 SELECT 文

SELECT 文が実行された場合の実行計画と、ワーカー・ノードで実行される SQL 文を検証しました。EXPLAIN 文で実行計画を確認する際には citus.explain_all_tasks パラメーターを on にすることで全タスクの実行計画が表示されます。デフォルトでは一部の実行計画のみが出力されます。また citus.log_remote_commands を on（デフォルト off）に設定すると、コーディネーター・ノードで実行した SQL 文からワーカーに投入した SQL 文をログに出力できます。

□ 単純検索

WHERE 句に分散キーを指定して、データが格納されているノードが特定できる場合は該当ノードのみで同一の SQL 文が実行されます。

例 43 検索の実行（単一シャード）

```
postgres=> SET citus.log_remote_commands = on ;
SET
postgres=> SELECT * FROM dist1 WHERE c1 = 1000 ;
NOTICE:  issuing SELECT c1, c2 FROM public.dist1_102461 dist1 WHERE (c1
OPERATOR(pg_catalog.=) (1000)::numeric)
DETAIL:  on server demo@localhost:5442 connectionId: 1
  c1  |  c2
------+-------
 1000 | data1
(1 row)
```

□ 集計

WHERE 句を指定しない場合や分散キーが指定されていない場合は全ワーカー・ノードで同一の SQL 文が実行されます。集計関数もワーカー・ノードにプッシュダウンされます。

例 44 集計関数の実行（コーディネーター・ノード）

```
postgres=> SELECT COUNT(*) FROM dist1 ;
NOTICE:  issuing SELECT count(*) AS count FROM public.dist1_102461 dist1 WHERE
true
DETAIL:  on server demo@localhost:5442 connectionId: 1
<<途中省略>>
NOTICE:  issuing SELECT count(*) AS count FROM public.dist1_102464 dist1 WHERE
true
DETAIL:  on server demo@localhost:5444 connectionId: 4
  count
----------
 10000000
(1 row)
```

□ 結合（１）

分散テーブルと参照テーブルの結合を行った場合、ワーカーノード内で結合を行い、最終的にコーディネーター・ノードで集計しています。

例 45 検索の実行（コーディネーター・ノード）

```
postgres=> SELECT COUNT(*) FROM dist1 INNER JOIN ref1 ON dist1.c1 = ref1.c1
                WHERE dist1.c1 = 1000 ;
NOTICE:  issuing SELECT count(*) AS count FROM (public.dist1_102461 dist1 JOIN
public.ref1_102220      ref1      ON      ((dist1.c1      OPERATOR(pg_catalog.=)
(ref1.c1)::numeric))) WHERE (dist1.c1 OPERATOR(pg_catalog.=) (1000)::numeric)
DETAIL:  on server demo@localhost:5442 connectionId: 1
 count
-------
     1
(1 row)
```

□ 結合（２）

分散テーブル間の結合はエラーになります。解消するには citus.enable_repartition_joins を on にします。

例 46 分散テーブル間の結合

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM dist1 INNER JOIN dist2
                ON dist1.c1 = dist2.c1 WHERE dist1.c1 < 1000 ;
ERROR:  the query contains a join that requires repartitioning
HINT:  Set citus.enable_repartition_joins to on to enable repartitioning
```

□ 関数の実行

SELECT 文に CURRENT_DATE 関数を指定した場合に関数を実行するノードを検証しました。関数はワーカー・ノードで実行されていることがわかります。

例 47 関数を含む SELECT 文の実行（コーディネーター・ノード）

```
postgres=> SELECT current_date, c1 FROM dist1 WHERE c1 = 1000 ;
NOTICE:     issuing  SELECT  CURRENT_DATE  AS  "current_date",  c1  FROM
public.dist1_102461 dist1 WHERE (c1 OPERATOR(pg_catalog.=) (1000)::numeric)
DETAIL:  on server demo@localhost:5442 connectionId: 1
 current_date |  c1
--------------+------
 2021-08-03   | 1000
(1 row)
```

3.3.2 INSERT 文 / UPDATE 文 / DELETE 文

データ更新 DML の実行 SQL 文を確認します。

□ INSERT 文

分散テーブルに対する単純な INSERT 文はいずれかのワーカー・ノードでそのまま実行されます。

コーディネーター・ノード上で開始したトランザクション内の最初の DML が発行された直後にワーカー・ノード上でもトランザクションが開始されます。ワーカー・ノード上ではトランザクションの開始と同時に分散トランザクションを管理する assign_distributed_transaction_id 関数が実行されます。この関数にはコーディネーター・ノードの共有メモリー上で取得されたトランザクション ID が指定されます。

その後 INSERT 文と COMMIT 文が実行されます。下記の例では dist1 テーブルは citus.shard_replication_factor を 2 に設定しているため、INSERT 文が２回実行されています。

例 48 INSERT 文の実行（コーディネーター・ノード）

```
postgres=> INSERT INTO dist1 VALUES (0, 'zero') ;
NOTICE:  issuing BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;SELECT
assign_distributed_transaction_id(0, 8, '2021-08-03 16:32:16.167673+09');
DETAIL:  on server demo@localhost:5442 connectionId: 7
NOTICE:  issuing INSERT INTO public.dist1_102464 (c1, c2) VALUES (0,
'zero'::character varying)
DETAIL:  on server demo@localhost:5442 connectionId: 7
NOTICE:  issuing BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;SELECT
assign_distributed_transaction_id(0, 8, '2021-08-03 16:32:16.167673+09');
DETAIL:  on server demo@localhost:5443 connectionId: 9
NOTICE:  issuing INSERT INTO public.dist1_102464 (c1, c2) VALUES (0,
'zero'::character varying)
DETAIL:  on server demo@localhost:5443 connectionId: 9
NOTICE:  issuing COMMIT
DETAIL:  on server demo@localhost:5442 connectionId: 7
NOTICE:  issuing COMMIT
DETAIL:  on server demo@localhost:5443 connectionId: 9
INSERT 0 1
```

□ INSERT SELECT 文

INSERT SELECT 文は COPY 文に変換されて実行されます。以下の例では分散テーブル dist1 から読み込んで分散テーブル dist2 テーブルにインサートしています。ワーカー・ノード内で INSERT SELECT 文が完結していることがわかります。

例 49 INSERT SELECT 文の実行（分散テーブル⇒分散テーブル）

```
postgres=> INSERT INTO dist2 SELECT * FROM dist1 ;
<<途中省略>>
NOTICE:  issuing INSERT INTO public.dist2_102499 AS citus_table_alias (c1, c2)
SELECT c1, c2 FROM public.dist1_102461 dist1 WHERE (c1 IS NOT NULL)
<<途中省略>>
NOTICE:  issuing COMMIT PREPARED 'citus_0_109305_38_25'
DETAIL:  on server demo@localhost:5445 connectionId: 16
INSERT 0 10000000
```

INSERT SELECT 文を使ってローカル・テーブル data1 から分散テーブル dist2 にデータを挿入すると、COPY 文に変換されて実行されていることがわかります。

例 50 INSERT SELECT 文の実行（ローカル・テーブル⇒分散テーブル）

```
postgres=> INSERT INTO dist2 SELECT * FROM data1 ;
<<途中省略>>
NOTICE:  issuing COPY public.dist2_102502 (c1, c2) FROM STDIN WITH (format
'binary')
DETAIL:  on server demo@localhost:5442 connectionId: 1
<<途中省略>>
NOTICE:  issuing COMMIT PREPARED 'citus_0_109305_40_41'
DETAIL:  on server demo@localhost:5445 connectionId: 24
INSERT 0 10000000
```

□ UPDATE 文 / DELETE 文

UPDATE 文や DELETE 文はほぼそのままワーカー・ノードで実行されます。

例 51 DELETE 文の実行（コーディネーター・ノード）

```
postgres=> DELETE FROM dist1 WHERE c1 < 2000 ;
<<途中省略>>
NOTICE:  issuing DELETE FROM public.dist1_102462 dist1 WHERE (c1
OPERATOR(pg_catalog.<) (2000)::numeric)
DETAIL:  on server demo@localhost:5444 connectionId: 10
<<途中省略>>
NOTICE:  issuing COMMIT PREPARED 'citus_0_101348_13_26'
DETAIL:  on server demo@localhost:5444 connectionId: 10
DELETE 2000
```

□ TRUNCATE 文

TRUNCATE 文はほぼそのままワーカー・ノードで実行されます。

例 52 TRUNCATE 文の実行（コーディネーター・ノード）

```
postgres=> TRUNCATE TABLE dist1 ;
<<途中省略>>
NOTICE:  issuing TRUNCATE TABLE public.dist1_102461 CASCADE
DETAIL:  on server demo@localhost:5442 connectionId: 20
<<途中省略>>
NOTICE:  issuing COMMIT PREPARED 'citus_0_101348_16_29'
DETAIL:  on server demo@localhost:5444 connectionId: 22
TRUNCATE TABLE
```

3.3.3 ANALYZE 文 / VACUUM 文

コーディネーター・ノードのテーブルに対して ANALYZE 文を実行すると、ワーカー・ノードのテーブルに対しても ANALYZE 文が実行されます。VACUUM 文も同様の動きになります。VACUUM VERBOSE 文の実行結果にワーカー・ノードの情報は含みません。

例 53 ANALYZE 文の実行（コーディネーター・ノード）

```
postgres=> ANALYZE VERBOSE dist1 ;
INFO:  analyzing "public.dist1"
INFO:  "dist1": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows;
0 rows in sample, 0 estimated total rows
<<途中省略>>
NOTICE:  issuing ANALYZE VERBOSE public.dist1_102461
DETAIL:  on server demo@localhost:5442 connectionId: 20
<<途中省略>>
NOTICE:  issuing COMMIT PREPARED 'citus_0_101348_18_51'
DETAIL:  on server demo@localhost:5444 connectionId: 37
ANALYZE
```

□ 列指向テーブルの VACUUM

列指向テーブルに対する VACUUM VERBOSE 文にはストレージ ID、データのサイズ、圧縮率の情報等が出力されます。

例 54 ANALYZE 文の実行（コーディネーター・ノード）

```
postgres=> VACUUM VERBOSE column1 ;
INFO:  statistics for "column1":
storage id: 10000000017
total file size: 13123584, total data size: 12734988
compression rate: 19.04x
total row count: 10000000, stripe count: 67, average rows per stripe: 149253
chunk count: 2000, containing data for dropped columns: 0, zstd compressed:
2000

VACUUM
```

3.3.4 SET 文

SET 文はワーカー・ノードには伝播しません。citus.propagate_set_commands パラメーターの設定値を local（デフォルト none）に指定することで、SET LOCAL 文のみワーカー・ノードに伝播させることができます。

3.3.5 実行できない DML

Citus により作成されたテーブルに対しては基本的に全 SQL が実行できますが、いくつか例外があります。

□ UPDATE 文／DELETE 文

分散テーブルでは分散キーに指定された列は更新できません。

例 55 分散キーの更新

```
postgres=> UPDATE dist1 SET c1=0 WHERE c1=1 ;
ERROR:  modifying the partition value of rows is not allowed
```

列指向テーブルは UPDATE 文／DELETE 文自体がサポートされていません。

例 56 列指向テーブルの更新エラー

```
postgres=> UPDATE column1 SET c2='update' WHERE c1=1 ;
ERROR:  UPDATE and CTID scans not supported for ColumnarScan
postgres=> DELETE FROM column1 WHERE c1=100 ;
ERROR:  UPDATE and CTID scans not supported for ColumnarScan
```

□ TABLESAMPLE 句

分散テーブル、列指向テーブルに対しては TABLESAMPLE 句が使用できません。参照テーブルには実行可能です。

例 57 TABLESAMPLE 句

```
postgres=> SELECT * FROM dist1 TABLESAMPLE SYSTEM(1) ;
ERROR:  could not run distributed query which use TABLESAMPLE
HINT:  Consider using an equality filter on the distributed table's partition
column.
postgres=> SELECT * FROM column1 TABLESAMPLE SYSTEM(1) ;
ERROR:  sample scans not supported on columnar tables
```

□ 再帰 CTE

WITH RECURSIVE 句は分散テーブルでは使用できません。参照テーブル、列指向テーブルに対しては実行できます。

例 58 WITH RECURSIVE 句

```
postgres=> WITH RECURSIVE r AS (
               SELECT * FROM dist1 WHERE c1 = 1
               UNION ALL
               SELECT dist1.* FROM dist1, r  WHERE dist1.c1 = r.c1
           )
           SELECT * FROM r ORDER BY c1 ;
ERROR:  recursive CTEs are not supported in distributed queries
```

☐ SELECT FOR UPDATE

分散テーブル、列指向テーブルに対しては SELECT FOR UPDATE/FOR SHARE 文が使用できません。

例 59 SELECT FOR UPDATE 文

```
postgres=> SELECT * FROM dist1 WHERE c1 = 100 FOR UPDATE ;
ERROR:  could not run distributed query with FOR UPDATE/SHARE commands
HINT:  Consider using an equality filter on the distributed table's partition
column.
```

☐ ローカル・テーブルとの結合

コーディネーター・ノードに作成されたローカル・テーブルと、分散テーブルは結合できます。citus.local_table_join_policy パラメーターで制御できます。デフォルト値は auto です。設定できる値は以下の通りです。

表 21 citus.local_table_join_policy

設定値	説明	備考
auto	変換先を自動決定	デフォルト
never	ローカル・テーブルと分散テーブルの結合を禁止	
prefer-local	ローカル・テーブルの変換を優先	
prefer-distributed	分散テーブルの変換を優先	

☐ GROUPING SETS 句

分散テーブルに対しては GROUPING SETS 句、CUBE 句、ROLLUP 句は実行できません。参照テーブル、列指向テーブルには実行可能です。

例 60 GROUPING SETS 句

```
postgres=> SELECT c1, c2, SUM(c3) FROM dist1 GROUP BY GROUPING
                SETS ((c1), (c2), ()) ;
ERROR:  could not run distributed query with GROUPING SETS, CUBE, or ROLLUP
HINT:  Consider using an equality filter on the distributed table's partition
column.
```

□ INSERT ON CONFLICT 文

分散テーブルでは INSERT ON CONFLICT 文はサポートされていますが、分散キーを更新することはできません。参照テーブルではこの制限はありません。

例 61 INSERT ON CONFLICT 文

```
postgres=> INSERT INTO dist1 VALUES (100, 'conflict') ON CONFLICT
                ON CONSTRAINT dist1_pkey DO UPDATE SET c2='update' ;
INSERT 0 1
postgres=> INSERT INTO dist1 VALUES (100, 'conflict') ON CONFLICT
                ON CONSTRAINT dist1_pkey DO UPDATE SET c1=0 ;
ERROR:  modifying the partition value of rows is not allowed
```

□ generate_series 関数による一括 INSERT

分散テーブル generate_series 関数等による一括 INSERT はサポートされません。参照テーブルにはこの制限はありません。

例 62 INSERT 文の実行（コーディネーター・ノード）

```
postgres=> INSERT INTO dist1 VALUES (generate_series(1, 100), 'generate') ;
ERROR:  set-valued function called in context that cannot accept a set
LINE 1: INSERT INTO dist1 VALUES (generate_series(1, 100), 'generate...
                                  ^
```

□ 統計情報

pg_stat_all_tables ビューや pg_statio_all_tables ビューは一部データのみ更新されます。

□ その他の制約

その他の制約は以下の URL で参照できます。

URL: http://docs.citusdata.com/en/v10.1/admin_guide/table_management.html#usage

3.3.6 セッション

コーディネーター・ノードとワーカー・ノード間のセッションについて検証しました。デフォルト設定では SQL 文がアクセスするワーカー・ノードのテーブル単位にセッションを新規に作成し、トランザクションが完了すると切断しています。citus.max_cached_conns_per_worker パラメーター（デフォルト 1）を拡大することで、コネクションをプールすることができますが、どのような単位で接続／切断するのかまでは確認できませんでした。

以下はコーディネーター・ノードで「SELECT COUNT(*) FROM dist1」文を実行した場合のログです（パラメーターlog_connections、log_disconnections を on に指定しています）。複数のセッションが動的に作成され、コミット後にクローズされていることがわかります。

例 63 ワーカー・ノードのセッション

```
LOG:  connection received: host=::1 port=46302
LOG:  connection authorized: user=demo database=postgres application_name=…
LOG:  statement: BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;…
LOG:  statement: SELECT count(*) AS count FROM public.dist1_102461 dist1 …
LOG:  connection received: host=::1 port=46308
LOG:  connection authorized: user=demo database=postgres application_name=…
LOG:  statement: BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;…
LOG:  statement: SELECT count(*) AS count FROM public.dist1_102464 dist1…
LOG:  statement: COMMIT
LOG:  statement: COMMIT
LOG:  disconnection: session time: 0:00:02.127 user=demo database=postgres…
```

3.3.7 その他

テーブル定義に関する SQL 文以外はワーカー・ノードに伝播しません。コーディネーター・ノードで CREATE USER 文や CREATE DATABASE 文を実行すると、ワーカー・ノードでも同様の操作が必要であるというメッセージが出力されます。

例 64 ユーザーの作成（コーディネーター・ノード）

```
postgres=# CREATE USER sample PASSWORD '*****' ;
NOTICE:  not propagating CREATE ROLE/USER commands to worker nodes
HINT:  Connect to worker nodes directly to manually create all necessary users
and roles.
CREATE ROLE
```

3.4 ワーカー・ノードの増減

3.4.1 ワーカー・ノード追加

ワーカー・ノードを追加するにはコーディネーター・ノードで citus_add_node 関数を実行します。citus エクステンション導入済のインスタンスを指定します。この関数実行時点で新規追加したインスタンスに参照ノードのコピーが作成されます。分散テーブルのデータは移動されません。

例 65 ワーカー・ノードの追加

```
postgres=# SELECT citus_add_node('localhost', 5445) ;
NOTICE:  Replicating reference table "ref1" to the node localhost:5445
 citus_add_node
----------------
              7
(1 row)
```

□ リバランス

ワーカー・ノードが追加されるとデータの偏りが発生するためリバランスが必要になります。まず get_rebalance_table_shards_plan 関数を実行すると、リバランスが必要なテーブルと移動元、移動先の情報を表示できます。

例 66 リバランスの必要性確認

```
postgres=> SELECT * FROM get_rebalance_table_shards_plan() ;
table_name | shardid | shard_size | sourcename | sourceport | targetname | targetport
------------+---------+------------+------------+------------+------------+------------
 dist2      |  102525 |          0 | localhost  |       5444 | localhost  |       5446
 dist2      |  102524 |          0 | localhost  |       5443 | localhost  |       5446
 dist2      |  102523 |          0 | localhost  |       5442 | localhost  |       5446
 dist1      |  102536 |          0 | localhost  |       5444 | localhost  |       5446
 dist1      |  102535 |          0 | localhost  |       5443 | localhost  |       5446
 dist1      |  102537 |          0 | localhost  |       5442 | localhost  |       5446
(6 rows)
```

分散テーブルを新規追加したワーカー・ノードにリバランスするには、コーディネーター・ノードで rebalance_table_shards 関数を実行します。rebalance_table_shards 関数は内部で citus_move_shard_placement 関数を実行しています。既存ワーカー・ノード上からは COPY TO STDOUT 文を使ってデータを取得しています。新規のワーカー・ノードでは worker_append_table_to_shard 関数で新しいシャードを作成しているようです。

例 67 リバランスの実行（テーブル全体）

```
postgres=# SELECT rebalance_table_shards() ;
NOTICE:  Moving shard 102525 from localhost:5444 to localhost:5446 ...
NOTICE:  Moving shard 102524 from localhost:5443 to localhost:5446 ...
NOTICE:  Moving shard 102523 from localhost:5442 to localhost:5446 ...
NOTICE:  Moving shard 102536 from localhost:5444 to localhost:5446 ...
NOTICE:  Moving shard 102535 from localhost:5443 to localhost:5446 ...
NOTICE:  Moving shard 102537 from localhost:5442 to localhost:5446 ...
 rebalance_table_shards
------------------------


(1 row)
```

テーブル単位に実行することもできます。

例 68 リバランスの実行（テーブル単位）

```
postgres=> SELECT rebalance_table_shards('dist1') ;
NOTICE:  Moving shard 102462 from localhost:5444 to localhost:5445 ...
NOTICE:  Moving shard 102461 from localhost:5443 to localhost:5445 ...
NOTICE:  Moving shard 102463 from localhost:5442 to localhost:5445 ...
 rebalance_table_shards
------------------------


(1 row)
```

この関数にはテーブル名以外に多くのパラメーターを指定できます。

表 22 rebalance_table_shards 関数のパラメーター

パラメーター名	データ型	説明	デフォルト
relation	regclass	リバランス対象テーブル	NULL
threshold	real	ノードの平均使用率	NULL
max_shard_moves	integer	移動するシャードの最大数	1000000
excluded_shard_list	bigint[]	例外シャードのリスト	{}
shard_transfer_mode	shard_transfer_mode	レプリケーション方法（auto, force_logical, block_writes）	auto
drain_only	boolean	true の場合、pg_dist_node の shouldhaveshards=false のワーカーからシャードを移動	false
rebalance_strategy	name	リバランス方法の名前	NULL

3.4.2 ワーカー・ノード停止時の動作

４個のワーカー・ノードのうち、１個を停止して動作を検証しました。以下のテーブルの操作を行いました。

表 23 検証対象テーブル

テーブル名	種類	ミラー（citus.shard_replication_factor）
dist1	分散テーブル	2
dist2	分散テーブル	1
ref1	参照テーブル	-

□ 再起動後の SELECT 文

ワーカー・ノードが異常終了し、再起動した場合、再起動したノードにアクセスする SELECT 文は一度失敗します。再実行することで正常にアクセスできます。

例 69 再起動後の SELECT 文

```
postgres=> SELECT * FROM dist2 WHERE c1=13 ;
ERROR:  terminating connection due to unexpected postmaster exit
CONTEXT:  while executing command on localhost:5445
postgres=> SELECT * FROM dist2 WHERE c1=13 ;
 c1 |  c2
----+-------
 13 | data1
(1 row)
```

□ SELECT 文の実行

単一のワーカー・ノードが終了した直後は、ミラーが存在する分散テーブル dist1 の検索でもエラーが発生する可能性があります。再実行することで他のワーカーノードで正常にアクセスできますが警告が出力される場合があります。

例 70 分散テーブルの検索（ミラーあり）

```
postgres=> SELECT * FROM dist1 WHERE c1 = 3 ;
WARNING:  connection to the remote node localhost:5442 failed with the
following error: could not connect to server: Connection refused
        Is the server running on host "localhost" (::1) and accepting
        TCP/IP connections on port 5442?
could not connect to server: Connection refused
        Is the server running on host "localhost" (127.0.0.1) and accepting
        TCP/IP connections on port 5442?  c1  |  c2
------+-------
    3 | data1
(1 row)
```

ミラーが存在しない dist2 テーブルは WHERE 句の指定により、停止ノードを参照するとエラーになります。

例 71 分散テーブルの検索（ミラーなし）

```
postgres=> SELECT * FROM dist2 WHERE c1=8 ;
c1 |  c2
----+-------
  8 | data1
(1 row)

postgres=> SELECT * FROM dist2 WHERE c1=9 ;
ERROR:  connection to the remote node localhost:5442 failed with the following
error: could not connect to server: Connection refused
        Is the server running on host "localhost" (::1) and accepting
        TCP/IP connections on port 5442?
could not connect to server: Connection refused
        Is the server running on host "localhost" (127.0.0.1) and accepting
        TCP/IP connections on port 5442?
```

参照テーブルの検索は警告が出力される場合もありますが、正常に行われます。

例 72 参照テーブルの検索

```
postgres=> SELECT COUNT(*) FROM ref1 ;
 count
---------
 10000000
(1 row)
```

□ 更新 SQL

タプルを更新する SQL は停止しているホストを参照する更新処理はエラーになります。停止しているホストにアクセスせずに完結する更新処理は成功します。

例 73 テーブルの更新

```
postgres=> DELETE FROM dist1 ;
ERROR:  connection to the remote node localhost:5442 failed with the following
error: could not connect to server: Connection refused
        Is the server running on host "localhost" (::1) and accepting
        TCP/IP connections on port 5442?
could not connect to server: Connection refused
        Is the server running on host "localhost" (127.0.0.1) and accepting
        TCP/IP connections on port 5442?
postgres=>
postgres=> DELETE FROM dist2 ;
ERROR:  connection to the remote node localhost:5442 failed with the following
error: could not connect to server: Connection refused
        Is the server running on host "localhost" (::1) and accepting
        TCP/IP connections on port 5442?
could not connect to server: Connection refused
        Is the server running on host "localhost" (127.0.0.1) and accepting
        TCP/IP connections on port 5442?
postgres=>
postgres=> DELETE FROM ref1 ;
ERROR:  connection to the remote node localhost:5442 failed with the following
error: could not connect to server: Connection refused
        Is the server running on host "localhost" (::1) and accepting
        TCP/IP connections on port 5442?
could not connect to server: Connection refused
        Is the server running on host "localhost" (127.0.0.1) and accepting
        TCP/IP connections on port 5442?
postgres=>
postgres=> DELETE FROM dist2 WHERE c1=100 ;
DELETE 1
```

3.4.3 ワーカー・ノード削除

不要になったワーカー・ノードの削除方法ついて検証しました。

□ ノード削除

ワーカー・ノードを行う際にはまず citus_drain_node 関数を実行します。指定されたワーカー・ノードに保存されたテーブルが他のワーカー・ノードに移動されます。

例 74 指定されたワーカー・ノードからデータを移動

```
postgres=# SELECT citus_drain_node('localhost', 5445) ;
NOTICE:  Moving shard 102461 from localhost:5445 to localhost:5443 ...
NOTICE:  Moving shard 102462 from localhost:5445 to localhost:5442 ...
NOTICE:  Moving shard 102463 from localhost:5445 to localhost:5442 ...
NOTICE:  Moving shard 102526 from localhost:5445 to localhost:5442 ...
NOTICE:  Moving shard 102530 from localhost:5445 to localhost:5443 ...
NOTICE:  Moving shard 102534 from localhost:5445 to localhost:5444 ...
 citus_drain_node
------------------


(1 row)
```

データが移動されたワーカー・ノードには参照テーブルしか残りません。

例 75 データ移動されたワーカー・ノード

```
postgres=> ¥d
            List of relations
 Schema |     Name     | Type  |  Owner
--------+--------------+-------+----------
 public | citus_tables | view  | postgres
 public | ref1_102220  | table | demo
(2 rows)
```

データ移行が完了したら citus_disable_node 関数を実行してワーカー・ノードを無効にします。

例 76 ワーカー・ノードの無効化

```
postgres=# SELECT citus_disable_node('localhost', 5445) ;
 citus_disable_node
--------------------


(1 row)
```

ワーカー・ノードを無効化しても pg_dist_node カタログからは削除されません。無効化したノードは isactive 列が false になります。再度有効化するには citus_activate_node 関数を実行します。この関数を実行すると参照テーブルのコピーが行われます。

例 77 ワーカー・ノードの再有効化

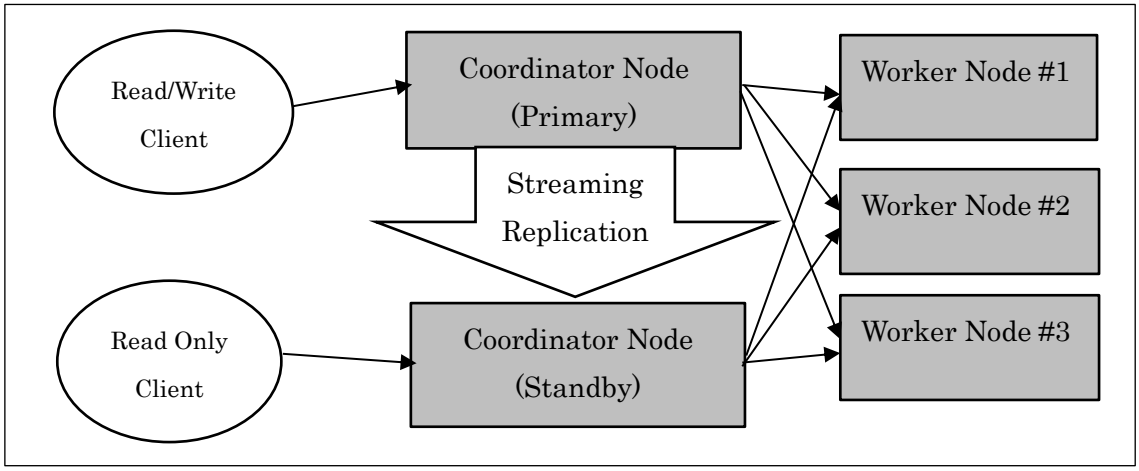
```
postgres=# SELECT citus_activate_node('localhost', 5445) ;
NOTICE:  Replicating reference table "ref1" to the node localhost:5445
 citus_activate_node
---------------------
                   7
(1 row)
```

特定のワーカー・ノードを完全に削除するには pg_dist_node カタログから対象ノードを DELETE 文で削除します。ただしこの方法はマニュアルには記載が無いため正しいかについて確認できませんでした。

3.4.4 コーディネーター・ノードの可用性

コーディネーター・ノードはユーザーからの SQL 文を受け付ける単一のインスタンスです。このため停止するとユーザー・アプリケーションが停止します。Citus Data ではストリーミング・レプリケーションとクラスタウェアを使って冗長化することを推奨しています。ストリーミング・レプリケーションのスタンバイ・インスタンスは検索用の SQL 文であれば、ワーカー・ノードのデータを利用することができます。

図 6 コーディネーター・ノードの可用性



例 78 ストリーミング・レプリケーション環境のスタンバイ・インスタンスから

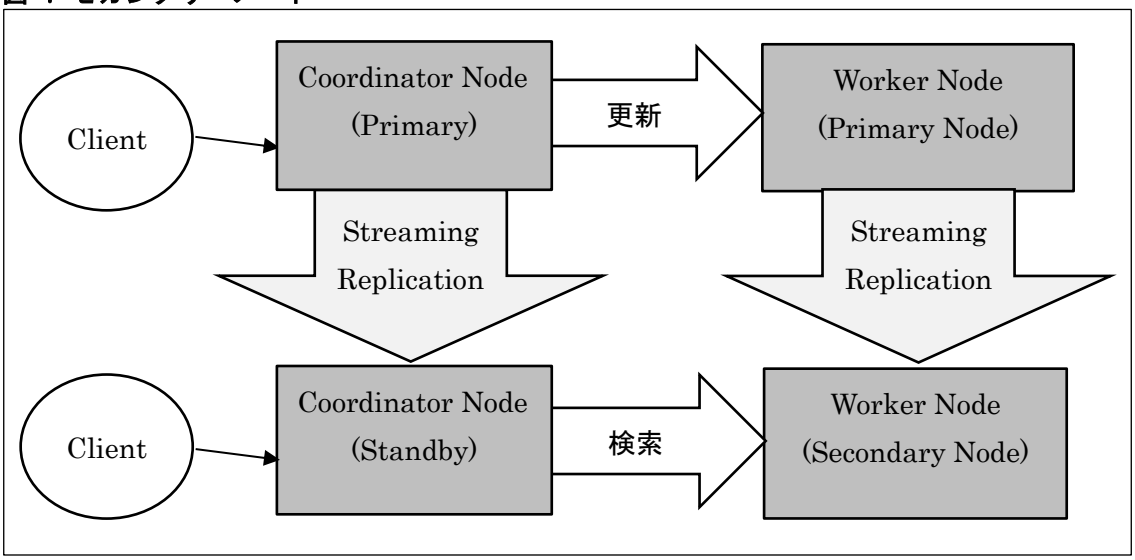
```
postgres=> SELECT COUNT(*) FROM dist1 ;
  count
----------
 10000000
(1 row)

postgres=> INSERT INTO dist1 VALUES (0, 'replica') ;
ERROR:  writing to worker nodes is not currently allowed
DETAIL:  the database is read-only
```

3.4.5 ワーカー・ノードの可用性

ワーカー・ノードはストリーミング・レプリケーションを構成してグループ化することができます。スタンバイ・インスタンスをセカンダリ・ノードと呼びます。セカンダリ・ノードを Citus クラスターに登録することで、読み取り専用のワーカー・ノードとして負荷分散することができます。

図 7 セカンダリ・ノード



セカンダリ・ノードの登録は citus_add_secondary_node 関数を実行します。セカンダリ・ノードとなるホスト名／ポート番号、プライマリ・ノードとなるワーカーのホスト名／ポート番号の順で指定します。pg_dist_node カタログを確認すると noderole 列が secondary となっているタプルを確認できます。

例 79 セカンダリ・ノードの登録

```
postgres=# SELECT citus_add_secondary_node('localhost', 5452, 'localhost',
5442) ;
 citus_add_secondary_node
--------------------------
                       14
(1 row)
postgres=# SELECT nodeid, nodename, nodeport, noderole FROM pg_dist_node ;
 nodeid | nodename  | nodeport | noderole
--------+-----------+----------+-----------
      2 | localhost |     5442 | primary
      3 | localhost |     5443 | primary
      4 | localhost |     5444 | primary
     14 | localhost |     5452 | secondary
     15 | localhost |     5453 | secondary
     16 | localhost |     5454 | secondary
(6 rows)
```

パラメーターcitus.use_secondary_nodes（デフォルト値 none）を always に変更すると、SQL 文の実行をセカンダリ・ノードに転送するようになります。残念ながら検索処理と更新処理を別々のノードに振り分ける機能は無いようです。

例 80 セカンダリ・ノードの利用

```
postgres=> SET citus.log_remote_commands = on ;
SET
postgres=> SHOW citus.use_secondary_nodes ;
 citus.use_secondary_nodes
---------------------------
 always
(1 row)

postgres=> SELECT COUNT(*) FROM dist1 ;
NOTICE:  issuing SELECT count(*) AS count FROM public.dist1_102535 dist1 WHERE
true
DETAIL:  on server demo@localhost:5452 connectionId: 1 ← セカンダリを参照
NOTICE:  issuing SELECT count(*) AS count FROM public.dist1_102537 dist1 WHERE
true
DETAIL:  on server demo@localhost:5454 connectionId: 3 ← セカンダリを参照
<<以下省略>>
```

パラメーターcitus.use_secondary_nodes は動的に変更することはできません。

例 81 設定エラー

```
postgres=# SET citus.use_secondary_nodes = always ;
ERROR:  parameter "citus.use_secondary_nodes" cannot be set after connection
start
```

Hewlett Packard
Enterprise

参考にした URL

本資料の作成には、以下の URL を参考にしました。

- Citus Data
  https://www.citusdata.com/
- Citus ドキュメント
  http://docs.citusdata.com/en/v10.1/
- GitHub
  https://github.com/citusdata/citus
- Azure Database for PostgreSQL - Hyperscale (Citus) ドキュメント
  https://docs.microsoft.com/ja-jp/azure/postgresql/hyperscale/
- Azure Database for PostgreSQL - Hyperscale (Citus): Columnar compression now generally available
  https://azure.microsoft.com/ja-jp/updates/azure-database-for-postgresql-hyperscale-citus-columnar-compression-now-generally-available/

Hewlett Packard
Enterprise

変更履歴

変更履歴

版	日付	作成者	説明
0.1	2018/06/12	篠田典良	社内レビュー版を作成
1.0	2018/06/15	篠田典良	社内公開版を作成
1.1	2018/07/02	篠田典良	社外公開版を作成
2.0	2021/09/01	篠田典良	Citus 10 を利用して再検証

以上