Hewlett Packard
Enterprise

2018 年 7 月 2 日

Citus 検証結果

日本ヒューレット・パッカード株式会社
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 7.5 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 7.5 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。特に以下の機能の検証は含みません。

- コーディネータ・ノードのフェイルオーバー
- ワーカー・ノード障害後のリバランス

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	10.4（2018 年 7 月現在の最新バージョン）
Citus	Community Edition 7.5-4（2018 年 7 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 (x86-64)

Hewlett Packard
Enterprise

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・パッカード株式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

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 のスループットを向上させる目的でデータを複数ノードに分散し、スケールアップを可能にします。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。

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・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。

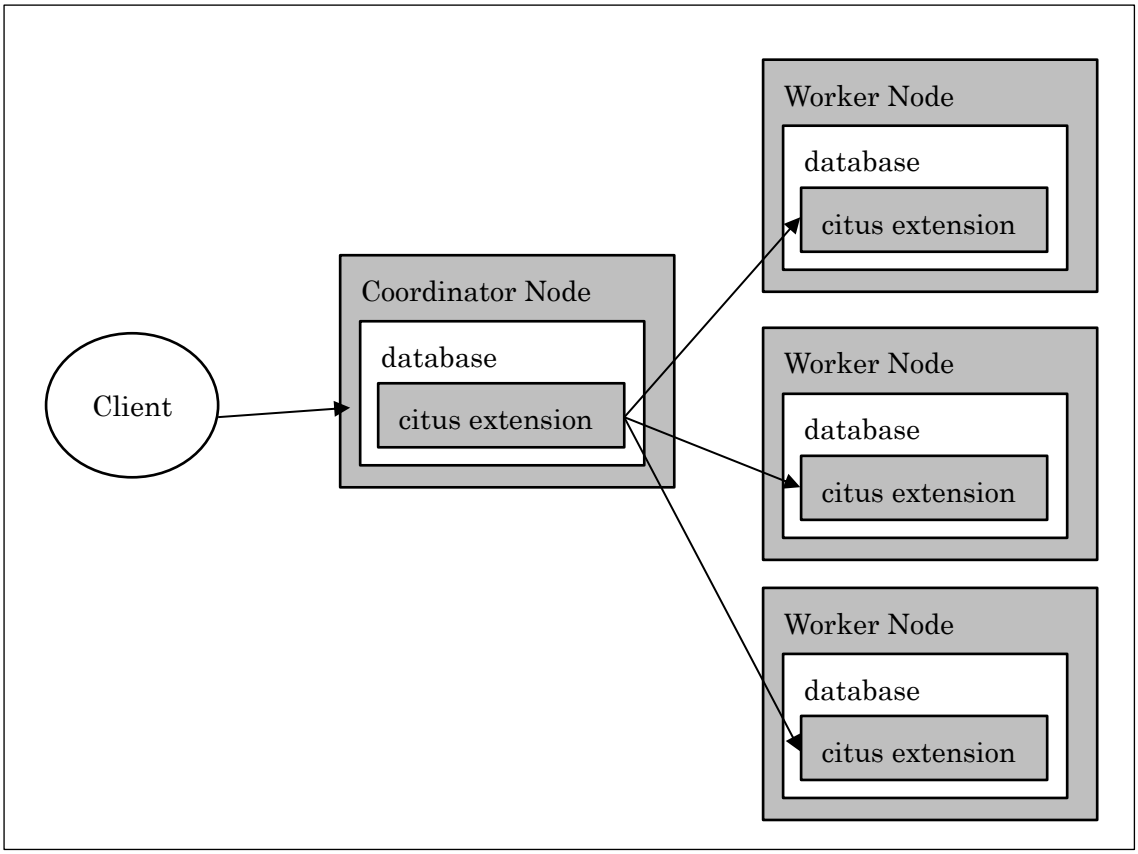
□ ワーカー・ノード

コーディネータ・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・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・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 1 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・ワーカー・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
bgworker: task tracker	コーディネータ・ノード ワーカー・ノード
bgworker: Citus Maintenance Daemon	コーディネータ・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 検証環境

検証は単一ノード内（ホスト名 rel74-1）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 5 インスタンス構成

インスタンス	ポート番号	役割	備考
1	5000	コーディネータ・ノード	
2	5010	コーディネータ・ノード	コーディネータ・ノードのスタンバイ
3	5001	ワーカー・ノード#1	
4	5002	ワーカー・ノード#2	
5	5003	ワーカー・ノード#3	
6	5011	ワーカー・ノード#4	ワーカー･ノード#1 のスタンバイ（セカンダリ・ノード）
7	5012	ワーカー・ノード#5	ワーカー･ノード#2 のスタンバイ（セカンダリ・ノード）
8	5013	ワーカー・ノード#6	ワーカー･ノード#3 のスタンバイ（セカンダリ・ノード）

表 6 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 7 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	5000～5013	各インスタンスで異なる値を設定

2.2 インストール

2.2.1 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「https://docs.citusdata.com/en/v7.4/installation/single_machine_rhel.html」で紹介されています。

2.2.2 ソース・コードからのインストール

検証環境はソース・コードからのインストールにより構築しました。ソース・コードは GitHub（https://github.com/citusdata/citus）から取得できます。

□ 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 Linux の libcurl-devel パッケージ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 からコピーした zip ファイルを展開します。configure コマンドに PostgreSQL をインストールしたパスを指定します。

例 1 ビルドとインストール

```
$ cd citus-master
$ ./autogen.sh
$ ./configure --prefix=/usr/local/pgsql
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
make -C src/backend/distributed/ all
make[1]: Entering directory `/home/postgres/citus-master/src/backend/distributed'
<<途中省略>>
cat citus--7.5-2.sql citus--7.5-2--7.5-3.sql > citus--7.5-3.sql
make[1]: Leaving directory `/home/postgres/citus-master/src/backend/distributed'
$
# make install
make -C src/backend/distributed/ all
make[1]: Entering directory `/home/postgres/citus-master/src/backend/distributed'
<<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-master/./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   | 7.5-4
installed_version |
comment           | Citus distributed database
```

2.2.3 エクステンションの導入

Citus を利用するには、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・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
```

インスタンス起動時に PostgreSQL のパラメーターmax_prepared_transactions（デフォルト 0）が設定されていない場合、自動的に 200 に設定されます。

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 8 作成されるテーブル（pg_catalog スキーマ）

テーブル名	用途
pg_dist_authinfo	用途不明
pg_dist_colocation	テーブル分散状況の取得
pg_dist_local_group	用途不明
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	用途不明
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

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 9 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_jobid_seq	ジョブ番号
pg_dist_node_nodeid_seq	ノード番号
pg_dist_placement_placementid_seq	不明
pg_dist_shardid_seq	ShardID 番号

2.3 利用方法

2.3.1 ノードの登録

コーディネータ・ノード上でワーカー・ノードを登録します。master_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。

例 4 ワーカー・ノードの追加

```
postgres=# SELECT * FROM master_add_node('rel74-1', 5001) ;
 nodeid | groupid | nodename | nodeport | noderack | hasmetadata | isactive | noderole |
nodecluster
--------+---------+----------+----------+----------+-------------+----------+----------+--
-----------
      1 |       1 | rel74-1  |     5001 | default  | f           | t        | primary  |
default
(1 row)

postgres=# SELECT * FROM master_add_node('rel74-1', 5002) ;
 nodeid | groupid | nodename | nodeport | noderack | hasmetadata | isactive | noderole |
nodecluster
--------+---------+----------+----------+----------+-------------+----------+----------+--
-----------
      2 |       2 | rel74-1  |     5002 | default  | f           | t        | primary  |
default
(1 row)

postgres=# SELECT * FROM master_add_node('rel74-1', 5003) ;
 nodeid | groupid | nodename | nodeport | noderack | hasmetadata | isactive | noderole |
nodecluster
--------+---------+----------+----------+----------+-------------+----------+----------+--
-----------
      3 |       3 | rel74-1  |     5003 | default  | f           | t        | primary  |
default
(1 row)
```

ワーカー・ノードの登録操作は SUPERUSER 権限が必要です。追加したノードの情報は pg_dist_node カタログまたは master_get_active_worker_nodes 関数で確認することができます。

例 5 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
--------+---------+----------+----------+----------+-------------+----------+----------+--
-----------
      1 |       1 | rel74-1  |     5001 | default  | f           | t        | primary  |
default
      2 |       2 | rel74-1  |     5002 | default  | f           | t        | primary  |
default
      3 |       3 | rel74-1  |     5003 | default  | f           | t        | primary  |
default
(3 rows)

postgres=> SELECT * FROM master_get_active_worker_nodes() ;
 node_name | node_port
-----------+-----------
 rel74-1   |      5003
 rel74-1   |      5002
 rel74-1   |      5001
(3 rows)
```

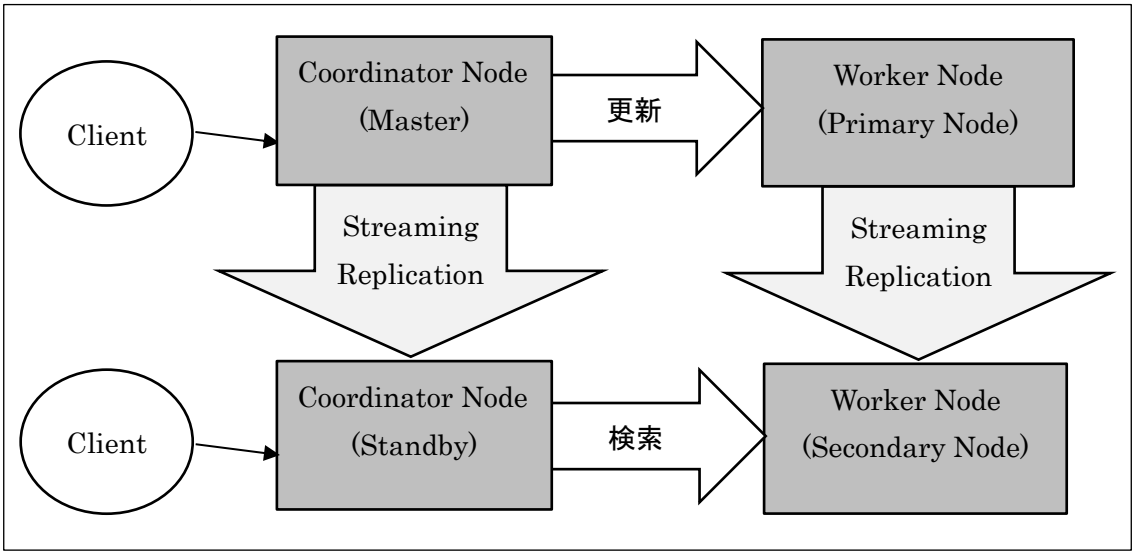
□ セキュリティ

コーディネータ・ノードとワーカー・ノード間の認証は pg_hba.conf ファイルの設定で行います。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、自動接続を行うためには pg_hba.conf ファイルに trust 設定をおこなうか、パスワード・ファイルによる認証を行います。

□ セカンダリ・ノード

ワーカー・ノードはプライマリ・ノードとセカンダリ・ノードでグループ化することができます。セカンダリ・ノードはプライマリ・ノードのレプリケーション・スタンバイとして構成します。スタンバイ・インスタンスでパラメーターcitus.use_secondary_nodes をalways（デフォルト never）に構成すると、検索処理をセカンダリ・ノードに対して実行します。

図 2 セカンダリ・ノード



セカンダリ・ノードの登録は master_add_secondary_node 関数を実行します。プライマリ・ノードとなるワーカーのホスト名／ポート番号とセカンダリ・ノードとなるホスト名／ポート番号を指定します。

例 6 セカンダリ・ノードの登録

```
postgres=# SELECT * FROM master_add_secondary_node('rel74-1', 5011, 'rel74-
1', 5001) ;
 nodeid | groupid | nodename | nodeport | noderack | hasmetadata | isactive |
noderole  | nodecluster
--------+---------+----------+----------+----------+-------------+----------
+----------+-------------
      4 |       1 | rel74-1  |     5011 | default  | f           | t        |
secondary | default
(1 row)
```

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 7 設定エラー

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

2.3.2 テーブルの作成

Citus により分散化を行うテーブルは、コーディネータ・ノード上で PostgreSQL 標準の CREATE TABLE 文を使って作成します。

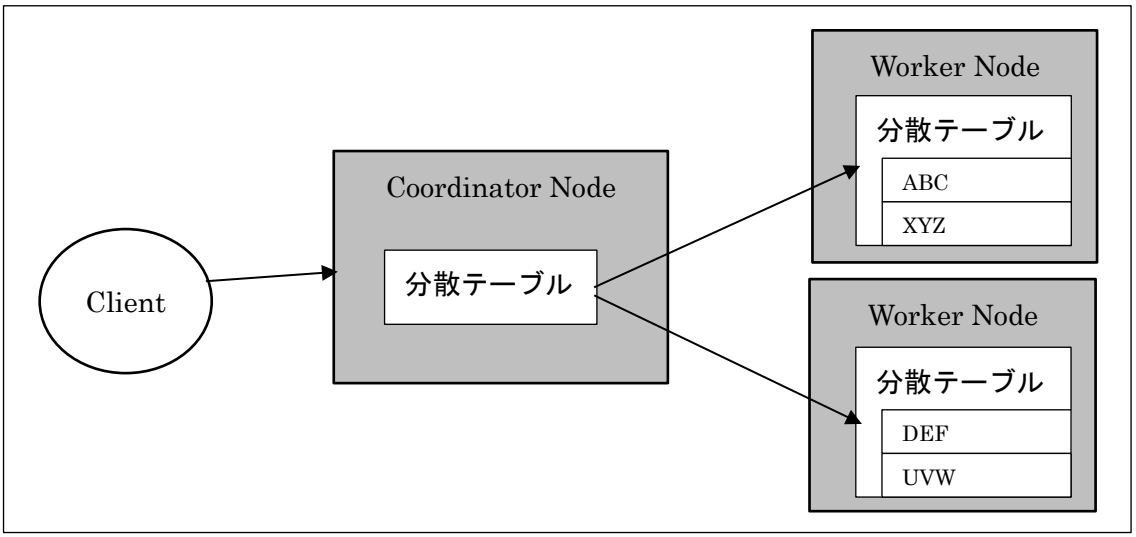
例 8 テーブルの作成

```
postgres=> CREATE TABLE dist1(c1 INT PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
```

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

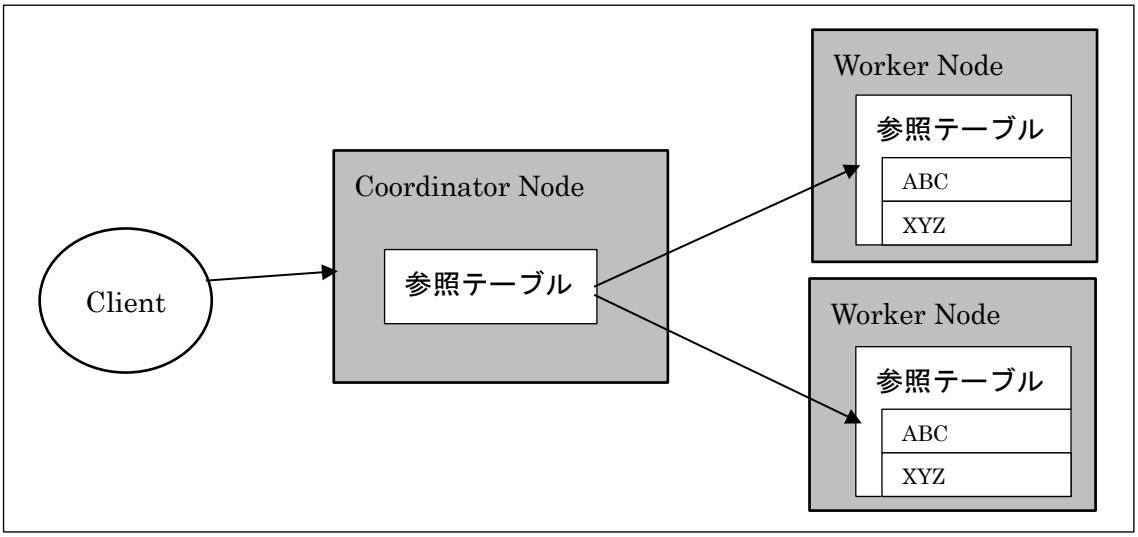
図 3 分散テーブル



2.3.4 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ファクト・テーブルとの結合を高速に行うことができます。CREATE TABLE 文でテーブルを作成後、create_reference_table 関数にテーブル名を指定して実行します。UNLOGGED テーブルは参照テーブルとして作成できますが、TEMP テーブルは参照テーブルとして作成できません。

図 4 参照テーブル



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・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が複数作成されます。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 10 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	
citus.shard_replication_factor	1	

例 9 コーディネータ・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 10 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 | dist1_102046 | table | demo
 public | dist1_102048 | table | demo
 public | dist1_102049 | table | demo
 public | dist1_102051 | table | demo
(4 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 11 テーブルの分散

ノード	ワーカー#1	ワーカー#2	ワーカー#3	備考
テーブル名	dist1_102046	dist1_102046		
		dist1_102047	dist1_102047	
	dist1_102048		dist1_102048	
	dist1_102049	dist1_102049		
		dist1_102050	dist1_102050	
	dist1_102051		dist1_102051	

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。

□ 分散テーブルの削除

コーディネータ・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。

□ 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 11 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。ただしパラメーター citus.shard_replication_factor を 1 に指定する必要があります。

例 12 パーティション・テーブルの分散実行エラー

```
postgres=> CREATE TABLE part1(c1 INT, c2 VARCHAR(10)) PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> SHOW citus.shard_replication_factor ;
citus.shard_replication_factor
--------------------------------
 2
(1 row)
postgres=> SELECT create_distributed_table('part1', 'c1') ;
ERROR:  distributing partitioned tables with replication factor greater than 1
is not supported
```

例 13 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=> SET citus.shard_replication_factor = 1 ;
SET
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 カタログを検索することで取得できます。

例 14 分散テーブルの一覧

```
postgres=> SELECT logicalrelid, partmethod, repmodel FROM pg_dist_partition ;
 logicalrelid | partmethod | repmodel
--------------+------------+----------
 dist1        | h          | c
 dist2        | h          | c
 ref1         | n          | t
(3 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 15 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
 dist2       | h          | c        |          6 |                 2
 ref1        | n          | t        |          1 |                 3
(3 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。

例 16 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 temp1 ;
INSERT 0 1000000
postgres=> SELECT create_distributed_table('dist3', 'c1') ;
NOTICE:  Copying data from local table...
 create_distributed_table
--------------------------


(1 row)
postgres=> SELECT pg_relation_filepath('dist3') ;
 pg_relation_filepath
----------------------
 base/13212/17040
(1 row)

postgres=> ¥! ls -l data/base/13212/17040
-rw------- 1 postgres postgres 44285952 Jul  2 22:34 data/base/13212/17040
```

□ トリガー

コーディネータ・ノード上で実行する CREATE TRIGGER 文はエラーになりませんが、ワーカー・ノードにはトリガーの定義は伝搬しません。また実行されるトリガーを検証したところ、TRUNCATE 文のトリガー以外は発行されませんでした。

3.1.2 参照テーブル

create_reference_table 関数を実行すると、ワーカー・ノード上に単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 17 コーディネータ・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 18 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)
```

□ トリガー

コーディネータ・ノード上で実行する CREATE TRIGGER 文はエラーになりませんが、ワーカー・ノードにはトリガーの定義は伝搬しません。また実行されるトリガーを検証した

ところ、TRUNCATE 文のトリガー以外は発行されませんでした。

3.1.3 SERIAL 列

serial 型の列を分散キーに指定すると、ワーカー・ノードに作成される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 19 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 20 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) |           |          |
```

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

コーディネータ・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 21 インデックスの作成（コーディネータ・ノード）

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

例 22 インデックスの作成（ワーカー・ノード）

```
postgres=> ¥d dist1_102079
                  Table "public.dist1_102079"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | integer               |           | not null |
 c2     | character varying(10) |           |          |
Indexes:
    "dist1_pkey_102079" PRIMARY KEY, btree (c1)
    "idx1_dist1_102079" btree (c2)
```

3.2.2 列の追加

コーディネータ・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 23 列の追加（コーディネータ・ノード）

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

例 24 列の追加（ワーカー・ノード）

```
postgres=> ¥d dist1_102079
                  Table "public.dist1_102079"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | integer               |           | not null |
 c2     | character varying(10) |           |          |
 c3     | character varying(10) |           |          |
Indexes:
    "dist1_pkey_102079" 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 25 実行できない ALTER TABLE 構文

```
postgres=> ALTER TABLE dist1 ENABLE TRIGGER dist1_before_insert_row_trig ;
ERROR:  alter table command is currently unsupported
DETAIL:  Only ADD|DROP COLUMN, SET|DROP NOT NULL, SET|DROP DEFAULT, ADD|DROP
CONSTRAINT, SET (), RESET (), ATTACH|DETACH PARTITION and TYPE subcommands are
supported.
```

3.2.3 分散テーブルから参照テーブルへの変換

分散テーブル数（citus.shard_count）が 1 である分散テーブルは、参照テーブルに変換することができます。upgrade_to_reference_table 関数にテーブル名を指定して実行します。この操作はデータが格納されているテーブルでも実行できます。

例 26 テーブルの変換

```
postgres=> INSERT INTO dist3 SELECT * FROM temp1 ;
INSERT 0 1000000
postgres=> SELECT upgrade_to_reference_table('dist3') ;
NOTICE:  Replicating reference table "dist3" to the node rel74-1:5002
NOTICE:  Replicating reference table "dist3" to the node rel74-1:5003
 upgrade_to_reference_table
----------------------------


(1 row)
```

3.3 SQL 文の実行

ここでは 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。

□ 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 27 検索の実行（コーディネータ・ノード）

```
postgres=> SELECT * FROM dist1 WHERE c1 = 100 ;
 c1  |  c2
-----+-------
 100 | data1
(1 row)
```

例 28 コーディネータ・ノードの実行計画

```
postgres=> EXPLAIN SELECT * FROM dist1 WHERE c1 = 100 ;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
--------------
 Custom Scan (Citus Router)  (cost=0.00..0.00 rows=0 width=0)
   Task Count: 1
   Tasks Shown: All
   ->  Task
         Node: host=rel74-1 port=5002 dbname=postgres
         ->  Index Scan using dist1_pkey_102082 on dist1_102082 dist1  (cost=0.42..8.44
rows=1 width=10)
               Index Cond: (c1 = 100)
(7 rows)
```

例 29 検索の実行ログ（ワーカー・ノード#1 で log_statement='all'指定のログ）

```
LOG:  statement: SELECT c1, c2 FROM public.dist1_102082 dist1 WHERE (c1
OPERATOR(pg_catalog.=) 100)
```

□ 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 30 集計関数の実行（コーディネータ・ノード）

```
postgres=> SELECT COUNT(*) FROM dist1 ;
  count
---------
 1000000
(1 row)
```

例 31 コーディネータ・ノードの実行計画

```
postgres=> EXPLAIN VERBOSE SELECT COUNT(*) FROM dist1 ;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------
---------------
 Aggregate  (cost=0.00..0.00 rows=0 width=0)
   Output: COALESCE((pg_catalog.sum(remote_scan.count))::bigint, '0'::bigint)
   ->  Custom Scan (Citus Real-Time)  (cost=0.00..0.00 rows=0 width=0)
         Output: remote_scan.count
         Task Count: 6
         Tasks Shown: One of 6
         ->  Task
               Node: host=rel74-1 port=5001 dbname=postgres
               ->  Aggregate  (cost=2992.59..2992.60 rows=1 width=8)
                     Output: count(*)
                     ->  Seq Scan on public.dist1_102079 dist1  (cost=0.00..2574.87
rows=167087 width=0)
(11 rows)
```

例 32 集計関数の実行ログ（ワーカー・ノード#1 で log_statement='all'指定のログ）

```
LOG:  statement: COPY (SELECT count(*) AS count FROM dist1_102080 dist1 WHERE
true) TO STDOUT
```

□ 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の実行計画は以下の通りです。

例 33 検索の実行（コーディネータ・ノード）

```
postgres=> SELECT COUNT(*) FROM dist1 INNER JOIN ref1 ON dist1.c1 = ref1.c1
        WHERE dist1.c1 = 1000 ;
 count
-------
     1
```

例 34 コーディネータ・ノードの実行計画

```
postgres=> EXPLAIN SELECT COUNT(*) FROM dist1 INNER JOIN ref1 ON dist1.c1 = ref1.c1
         WHERE dist1.c1 = 1000 ;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------
------------------------------
 Custom Scan (Citus Router)  (cost=0.00..0.00 rows=0 width=0)
   Task Count: 1
   Tasks Shown: All
   ->  Task
         Node: host=rel74-1 port=5003 dbname=postgres
         ->  Aggregate  (cost=16.74..16.75 rows=1 width=8)
               ->  Nested Loop  (cost=0.70..16.74 rows=1 width=0)
                     ->  Index Only Scan using dist1_pkey_102080 on dist1_102080 dist1
(cost=0.42..8.44 rows=1 width=4)
                           Index Cond: (c1 = 1000)
                     ->  Index Only Scan using ref1_pkey_102072 on ref1_102072 ref1
(cost=0.28..8.29 rows=1 width=4)
                           Index Cond: (c1 = 1000)
(11 rows)
```

例 35 検索の実行ログ（ワーカー・ノード#1 で log_statement='all'指定のログ）

```
LOG:  statement: SELECT count(*) AS count FROM (public.dist1_102080 dist1 JOIN
public.ref1_102072 ref1 ON ((dist1.c1 OPERATOR(pg_catalog.=) ref1.c1))) WHERE
(dist1.c1 OPERATOR(pg_catalog.=) 1000)
```

□ 結合（２）

分散テーブル間の結合を行った場合、デフォルト設定ではエラーになりました。パラメーターcitus.enable_repartition_joins を on に指定する必要がありました。

例 36 検索の実行（コーディネータ・ノード）

```
postgres=> 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
postgres=> SET citus.enable_repartition_joins = on ;
SET
postgres=> SELECT COUNT(*) FROM dist1 INNER JOIN dist2 ON dist1.c1 = dist2.c1
        WHERE dist1.c1 < 1000 ;
 count
-------
   999
(1 row)
```

例 37 コーディネータ・ノードの実行計画

```
postgres=> EXPLAIN SELECT COUNT(*) FROM dist1 INNER JOIN dist2
        ON dist1.c1 = dist2.c1 WHERE dist1.c1 < 1000 ;
                              QUERY PLAN
------------------------------------------------------------------------
 Aggregate  (cost=0.00..0.00 rows=0 width=0)
   ->  Custom Scan (Citus Task-Tracker)  (cost=0.00..0.00 rows=0 width=0)
         Task Count: 12
         Tasks Shown: None, not supported for re-partition queries
         ->  MapMergeJob
               Map Task Count: 6
               Merge Task Count: 12
         ->  MapMergeJob
               Map Task Count: 32
               Merge Task Count: 12
(10 rows)
```

例 38 検索の実行ログ（ワーカー・ノード#1 で log_statement='all'指定のログ）

```
LOG:  statement: SELECT worker_hash_partition_table  (180724170762, 1, 'SELECT
c1 FROM dist1_102079 dist1 WHERE (c1 OPERATOR(pg_catalog.<) 1000)', 'c1',
'integer'::regtype,        '{-2147483648,-1789569707,-1431655766,-1073741825,-
715827884,-357913943,-
2,357913939,715827880,1073741821,1431655762,1789569703}'::integer[])
LOG:  statement: SELECT worker_hash_partition_table  (180724170763, 28,
'SELECT c1 FROM dist2_102136 dist2 WHERE true', 'c1', 'integer'::regtype, '{-
2147483648,-1789569707,-1431655766,-1073741825,-715827884,-357913943,-
2,357913939,715827880,1073741821,1431655762,1789569703}'::integer[])
LOG:  statement: SELECT worker_fetch_partition_file  (180724170763, 15, 4,
165, 'rel74-1', 5003)
```

□ 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 39 関数を含む SELECT 文の実行（コーディネータ・ノード）

```
postgres=> SELECT current_date, c1 FROM dist1 WHERE c1 = 1000 ;
 current_date |  c1
--------------+------
 2018-06-19   | 1000
(1 row)
```

例 40 検索の実行ログ（ワーカー・ノード#1 で log_statement='all'指定のログ）

```
LOG:  statement:  SELECT  CURRENT_DATE  AS  "current_date",  c1  FROM
public.dist1_102080 dist1 WHERE (c1 OPERATOR(pg_catalog.=) 1000)
```

3.3.2 INSERT 文 / UPDATE 文 / DELETE 文

データ更新 DML の実行計画を確認します。

□ 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。

例 41 INSERT 文の実行（コーディネータ・ノード）

```
postgres=> INSERT INTO dist1 VALUES (0, 'zero') ;
INSERT 0 1
```

例 42 検索の実行ログ（ワーカー・ノード#1 で log_statement='all'指定のログ）

```
LOG:  statement: INSERT INTO dist1 VALUES (0, 'zero');
```

□ INSERT SELECT 文

INSERT SELECT 文は COPY 文に変換されて実行されます。

例 43 INSERT 文の実行（コーディネータ・ノード）

```
postgres=> INSERT INTO dist2 SELECT * FROM dist1 ;
INSERT 0 1000001
```

例 44 検索の実行ログ（ワーカー・ノード#1 で log_statement='all'指定のログ）

```
LOG:  statement: BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;SELECT
assign_distributed_transaction_id(0, 11, '2018-06-12 23:40:51.045614+09');
LOG:  statement: COPY public.dist2_102136 (c1, c2) FROM STDIN WITH (FORMAT
BINARY)
LOG:  statement: PREPARE TRANSACTION 'citus_0_10772_11_105'
LOG:  statement: COMMIT PREPARED 'citus_0_10772_11_105'
```

□ 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 45 INSERT 文の実行（コーディネータ・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...
                                  ^
```

□ 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 46 DELETE 文の実行（コーディネータ・ノード）

```
postgres=> DELETE FROM dist1 WHERE c1 < 1000 ;
DELETE 1000
```

例 47 検索の実行ログ（ワーカー・ノード#1 で log_statement='all'指定のログ）

```
LOG:  statement: DELETE FROM public.dist1_102079 dist1 WHERE (c1
OPERATOR(pg_catalog.<) 1000)
```

□ 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 48 TRUNCATE 文の実行（コーディネータ・ノード）

```
postgres=> TRUNCATE TABLE dist2 ;
TRUNCATE TABLE
```

例 49 検索の実行ログ（ワーカー・ノード#1 で log_statement='all'指定のログ）

```
LOG:  statement: TRUNCATE TABLE public.dist2_102111 CASCADE
```

3.3.3 ANALYZE 文 / VACUUM 文

コーディネータ・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。

例 50 ANALYZE 文の実行（コーディネータ・ノード）

```
postgres=> ANALYZE dist1 ;
ANALYZE
```

例 51 検索の実行ログ（ワーカー・ノード#1 で log_statement='all'指定のログ）

```
LOG:  statement: ANALYZE public.dist1_102081
```

3.3.4 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。

□ TABLESAMPLE 句

分散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 52 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.
```

□ 再帰 CTE

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

例 53 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

分散テーブルと参照テーブルに対しては SELECT FOR UPDATE 文が使用できません。

例 54 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.
```

ただし以下の条件がすべて合致している場合には SELECT FOR UPDATE 文を実行できます。

- 分散テーブルをパーティション・テーブルとして作成
- 分散キーとパーティション化列が一致
- 分散キーに対する等価検索（WHERE 分散キー=値）

□ ローカル・テーブルとの結合

コーディネータ・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せん。

例 55 ローカル・テーブルと分散テーブルの結合

```
postgres=> CREATE TABLE local1(c1 INT PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT d.* FROM dist1 d INNER JOIN local1 l ON d.c1 = l.c1 ;
ERROR:  relation local1 is not distributed
```

ただしローカル・テーブルをサブクエリー化することで実行できるようになります。

例 56 ローカル・テーブルと分散テーブルの結合（サブクエリー化）

```
postgres=> SELECT d.* FROM dist1 d INNER JOIN (SELECT * FROM local1) l ON d.c1
= l.c1 ;
   c1   |  c2   |   c1   |   c2
--------+-------+--------+--------
     31 | data1 |     31 | local1
     35 | data1 |     35 | local1
<<以下省略>>
```

□ 相関サブクエリー

分散テーブルでは、相関サブクエリーの記述に制限があります。GROUP BY 句を伴わない集計関数は記述できません。GROUP BY 句の記述がある場合でも一部制限があります。

例 57 相関サブクエリー

```
postgres=> SELECT c1, c2 FROM dist1 d1 WHERE
     d1.c1 = (SELECT max(c1) FROM dist1 d2 WHERE d1.c1 = d2.c1) ;
ERROR:  cannot push down this subquery
DETAIL:  Aggregates without group by are currently unsupported when a subquery
references a column from another query
```

□ 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。参照テーブルには実行可能です。

例 58 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.
```

□ PARTITION BY 句

分散テーブルに対して PARTITION BY 句に分散キーを含まないウィンドウ関数は実行できません。参照テーブルには実行可能です。

例 59 PARTITION BY 句

```
postgres=> SELECT c1, RANK() OVER (PARTITION BY c2) FROM dist1 ;
ERROR:  could not run distributed query because the window function that is used cannot be pushed down
HINT:  Window functions are supported in two ways. Either add an equality filter on the distributed tables' partition column or use the window functions with a PARTITION BY clause containing the distribution 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 60 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
```

□ EXPLAIN ANALYZE 文

バグか仕様かは不明ですが、主キー制約が指定された分散テーブルに対して EXPLAIN ANALYZE 文を実行するとエラーが表示されます。実際にはデータは格納されます。

例 61 EXPLAIN ANALYZE 文

```
postgres=> EXPLAIN ANALYZE INSERT INTO dist1 VALUES (0, 'data0') ;
WARNING:  duplicate key value violates unique constraint "dist1_pkey_102282"
DETAIL:  Key (c1)=(0) already exists.
WARNING:  duplicate key value violates unique constraint "dist1_pkey_102282"
DETAIL:  Key (c1)=(0) already exists.
WARNING:  could not commit transaction for shard 102282 on any active node
ERROR:  could not commit transaction on any active node
```

3.3.5 セッション

コーディネータ・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、SQL 文が完了すると切断しています。コネクション・プールの設定は行われていないようです。パラメーター等で実現できるかについては未検証です。

以下はコーディネータ・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）。３つのサーバー・プロセス（4861, 4856, 4858）が接続、SQL 文の実行、切断を実行していることがわかります。

例 62 セッション

```
[4861] LOG:  connection received: host=192.168.1.101 port=41708
[4861] LOG:  connection authorized: user=demo database=postgres
[4858] LOG:  connection received: host=192.168.1.101 port=41704
[4858] LOG:  connection authorized: user=demo database=postgres
[4861] LOG:  statement: COPY (SELECT count(*) AS count FROM dist1_102227 dist1
WHERE true) TO STDOUT
[4856] LOG:  connection received: host=192.168.1.101 port=41700
[4856] LOG:  connection authorized: user=demo database=postgres
[4856] LOG:  statement: COPY (SELECT count(*) AS count FROM dist1_102223 dist1
WHERE true) TO STDOUT
[4858] LOG:  statement: COPY (SELECT count(*) AS count FROM dist1_102225 dist1
WHERE true) TO STDOUT
[4861]  LOG:  disconnection:  session  time:  0:00:00.059  user=demo
database=postgres host=192.168.1.101 port=41708
[4856]  LOG:  disconnection:  session  time:  0:00:00.053  user=demo
database=postgres host=192.168.1.101 port=41700
[4858]  LOG:  disconnection:  session  time:  0:00:00.061  user=demo
database=postgres host=192.168.1.101 port=41704
```

3.3.6 ログ

コーディネータ・ノードでパラメーターcitus.log_remote_commands を on に設定すると、ワーカー・ノードで実行される SQL 文がログに出力されます。このパラメーターのデフォルト値は off です。

例 63 ログの出力設定

```
postgres=> SET citus.log_remote_commands = on ;
SET
postgres=> SELECT COUNT(*) FROM dist1 ;
  count
---------
 1000000
(1 row)
```

例 64 出力されるログ

```
LOG:  statement: SELECT COUNT(*) FROM dist1 ;
LOG:  issuing COPY (SELECT count(*) AS count FROM dist1_102280 dist1 WHERE
true) TO STDOUT
DETAIL:  on server rel74-1:5001
STATEMENT:  SELECT COUNT(*) FROM dist1 ;
LOG:  issuing COPY (SELECT count(*) AS count FROM dist1_102285 dist1 WHERE
true) TO STDOUT
DETAIL:  on server rel74-1:5003
STATEMENT:  SELECT COUNT(*) FROM dist1 ;
<<以下省略>>
```

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・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 65 ユーザーの作成（コーディネータ・ノード）

```
postgres=# c
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 障害発生時の動作

3.4.1 ワーカー・ノード停止時の動作

３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 12 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	-

□ SELECT 文の実行

単一ノードが停止しても、ミラーが存在する dist1 テーブルの検索は結果が常に返ります。ただし警告が出力される場合があります。

例 66 分散テーブル（ミラーあり）

```
postgres=> SELECT * FROM dist1 WHERE c1 = 1000 ;
WARNING:  connection error: rel74-1:5003
DETAIL:  could not connect to server: Connection refused
        Is the server running on host "rel74-1" (192.168.1.101) and accepting
        TCP/IP connections on port 5003?
  c1  |  c2
------+-------
 1000 | 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 67 分散テーブル（ミラーなし）

```
postgres=> SELECT * FROM dist2 WHERE c1 = 1001 ;
  c1  |  c2
------+-------
 1001 | data1
(1 row)

postgres=> SELECT * FROM dist2 WHERE c1 = 1002 ;
WARNING:  connection error: rel74-1:5003
DETAIL:  could not connect to server: Connection refused
        Is the server running on host "rel74-1" (192.168.1.101) and accepting
        TCP/IP connections on port 5003?
ERROR:  could not receive query results
```

参照テーブルの検索は正常に行われます。

例 68 参照テーブルの検索

```
postgres=> SELECT COUNT(*) FROM ref1 ;
count
-------
  1000
(1 row)
```

□ 更新 SQL

タプルを更新する SQL は停止するホストを参照する更新処理はエラーになります。

例 69 テーブルの更新

```
postgres=> DELETE FROM dist1 ;
ERROR:  connection error: rel74-1:5003
DETAIL:  could not connect to server: Connection refused
        Is the server running on host "rel74-1" (192.168.1.101) and accepting
        TCP/IP connections on port 5003?
postgres=>
postgres=> DELETE FROM dist2 ;
ERROR:  connection error: rel74-1:5003
DETAIL:  could not connect to server: Connection refused
        Is the server running on host "rel74-1" (192.168.1.101) and accepting
        TCP/IP connections on port 5003?
postgres=>
postgres=> DELETE FROM ref1 ;
ERROR:  connection error: rel74-1:5003
DETAIL:  could not connect to server: Connection refused
        Is the server running on host "rel74-1" (192.168.1.101) and accepting
        TCP/IP connections on port 5003?
```

3.4.2 ノード削除とリバランス

障害発生時のワーカー・ノードの削除方法とリバランス方法について検証しました。

□ ノード削除

障害が発生したノードの削除は master_remove_node 関数を実行します。しかし、障害ノードにオブジェクトが存在する場合にはエラーになります。

例 70 ノード削除エラー

```
postgres=# SELECT master_remove_node('rel74-1', 5003) ;
ERROR:  you cannot remove the primary node of a node group which has shard
placements
```

障害ノードに依存するオブジェクトの情報は、pg_dist_shard_placement カタログを検索します。ノードを切り離すために、該当するオブジェクトを DELETE 文で削除します。

例 71 依存オブジェクトの検索

```
postgres=# SELECT shardid, shardstate, nodename, nodeport FROM
pg_dist_shard_placement ;
 shardid | shardstate | nodename | nodeport
---------+------------+----------+----------
  102152 |          1 | rel74-1  |     5001
  102150 |          1 | rel74-1  |     5001
  102149 |          1 | rel74-1  |     5001
  102147 |          1 | rel74-1  |     5001
  102151 |          1 | rel74-1  |     5002
  102150 |          1 | rel74-1  |     5002
  102148 |          1 | rel74-1  |     5002
  102147 |          1 | rel74-1  |     5002
  102152 |          1 | rel74-1  |     5003
  102151 |          1 | rel74-1  |     5003
  102149 |          1 | rel74-1  |     5003
  102148 |          1 | rel74-1  |     5003
(12 rows)
```

例 72 依存オブジェクトの削除とノード削除

```
postgres=# DELETE FROM pg_dist_shard_placement WHERE nodeport = 5003 ;
DELETE 4
postgres=# SELECT master_remove_node('rel74-1', 5003) ;
 master_remove_node
--------------------


(1 row)
```

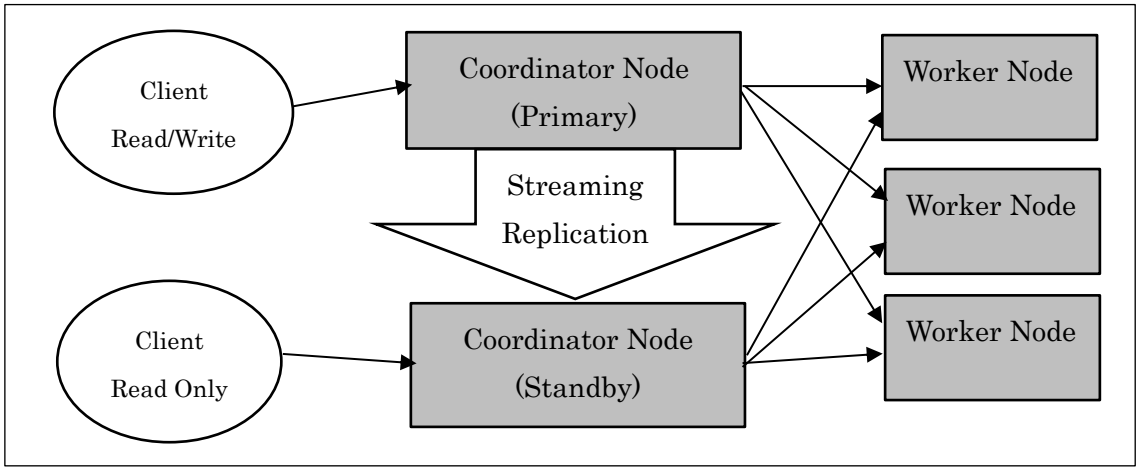
□ リバランス

ノード数が増減した場合にデータのリバランスを行う関数が rebalance_table_shards です。しかしこの関数は Citus Enterprise にのみ含まれるため検証できませんでした。

3.4.3 コーディネータ・ノードの可用性

コーディネータ･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 5 コーディネータ・ノードの可用性



例 73 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
---------
 1000000
(1 row)

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

3.5 外部との接続

Citus 7.5 インストール時と２４時間単位に稼働状況を Citus Data 社に転送します。データの転送は Citus Maintenance Daemon プロセスが実行し、cURL のライブラリ関数を使っています。転送内容は以下の情報です。マニュアルの「Checks For Updates and Cluster Statistics」セクションに記載されています。この動作はパラメーター citus.enable_statistics_collection を off に設定することで停止できます。

転送 URL: https://reports.citusdata.com/v1/usage_reports

表 13 転送情報（OS 関連）

内容	取得方法	例
オペレーティング・システムの種類	uname(3)	Linux
オペレーティング・システムのリリース	uname(3)	#1 SMP Wed Oct 19 11:24:13 EDT 2016
アーキテクチャ	uname(3)	x86_64

Microsoft Windows 環境では GetSystemInfo, GetVersionEx API を使っています。

表 14 転送情報（Citus 関連）

内容	備考
Citus バージョン情報	
テーブル数	
ワーカー・ノード数	
クラスター・サイズ	
ノード・メタデータ	pg_dist_node_metadata カタログの内容か

転送エラーが発生すると以下のログが出力されます。

例 74 転送エラー・ログ

```
WARNING:  Sending HTTP request failed.
HINT:  Error code: Timeout was reached.
CONTEXT:  Citus maintenance daemon for database 13212 user 10
```

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
  https://docs.citusdata.com/en/v7.4/index.html
- GitHub
  https://github.com/citusdata/citus

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

以上