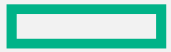


Hewlett Packard
Enterprise

Citus を使って
分散列指向データベースを作ってみよう

Noriyoshi Shinoda

November 12, 2021

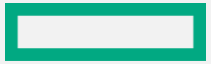
SPEAKER
篠田典良（しのだのりよし）

- ✓所属
  - ✓日本ヒューレット・パッカード合同会社
- ✓現在の業務
  - ✓PostgreSQL をはじめ、Oracle Database, Microsoft SQL Server, Vertica 等 RDBMS 全般に関するシステムの設計、移行、チューニング、コンサルティング
  - ✓PostgreSQL パッチ提供
  - ✓オープンソース製品に関する調査、検証
- ✓資格など
  - ✓Oracle ACE
  - ✓Oracle Database 関連書籍15冊の執筆
- ✓関連する URL
  - ✓「PostgreSQL 篠田の虎の巻」シリーズ
    http://h30507.www3.hp.com/t5/user/viewprofilepage/user-id/838802
  - ✓Oracle ACE ってどんな人？
    http://www.oracle.com/technetwork/jp/database/articles/vivadeveloper/index-1838335-ja.html

AGENDA

- ✓Citus とは
- ✓準備
- ✓テーブル構成
- ✓列指向テーブル
- ✓列指向テーブルと分散テーブル
- ✓その他

Citus とは？



Citus とは？
Citus とは？

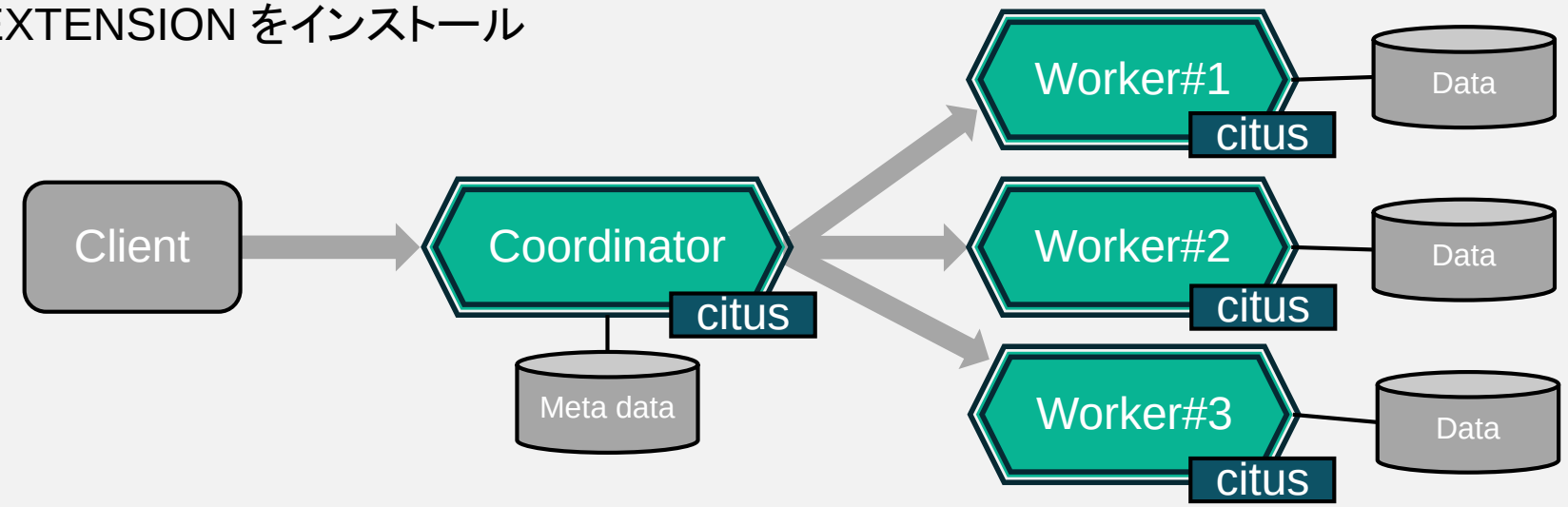
- ✓PostgreSQL でスケールアウト環境を実現
  - ✓複数ノードにまたがったパラレル・クエリーとパーティショニング機能
  - ✓スループット拡大を目指す
- ✓PostgreSQL の拡張（EXTENSION）として実装
  - ✓PostgreSQL 本体の変更なし
- ✓Citus Data（https://www.citusdata.com/）が開発
  - ✓2019年1月 Microsoft による買収
  - ✓Azure Database for PostgreSQL - Hyperscale (Citus) の中核技術となっている
  - ✓オープンソース版も提供（https://github.com/citusdata/citus）
- ✓以下の機能は含まない
  - ✓自動フェイルオーバー
  - ✓自動データ・リバランス
  - ✓バックアップ等の運用機能

Citus とは？
インスタンス構成

✓Coordinator Node
- ✓クライアントからの接続を受け付ける PostgreSQL インスタンス
- ✓メタデータを管理

✓Worker Node
- ✓実際にデータを保存する PostgreSQL インスタンス
- ✓Worker Node 間は通信を行わない

✓すべてのノードに citus EXTENSION をインストール



Citus とは？
インスタンス構成

- ✓Coordinator Node で実行される処理
  - ✓クライアントからの接続受付
  - ✓最終的なソート（ORDER BY）
  - ✓シーケンスの処理（SERIAL 列、GENERATED AS IDENTITY 列含む）
  - ✓テーブルとインデックス以外のオブジェクトの処理
- ✓Worker Node で実行される処理
  - ✓Coordinator Node から依頼される SQL 文の処理
  - ✓ANALYZE 文の実行
  - ✓VACUUM 文の実行
  - ✓テーブル・データの保持

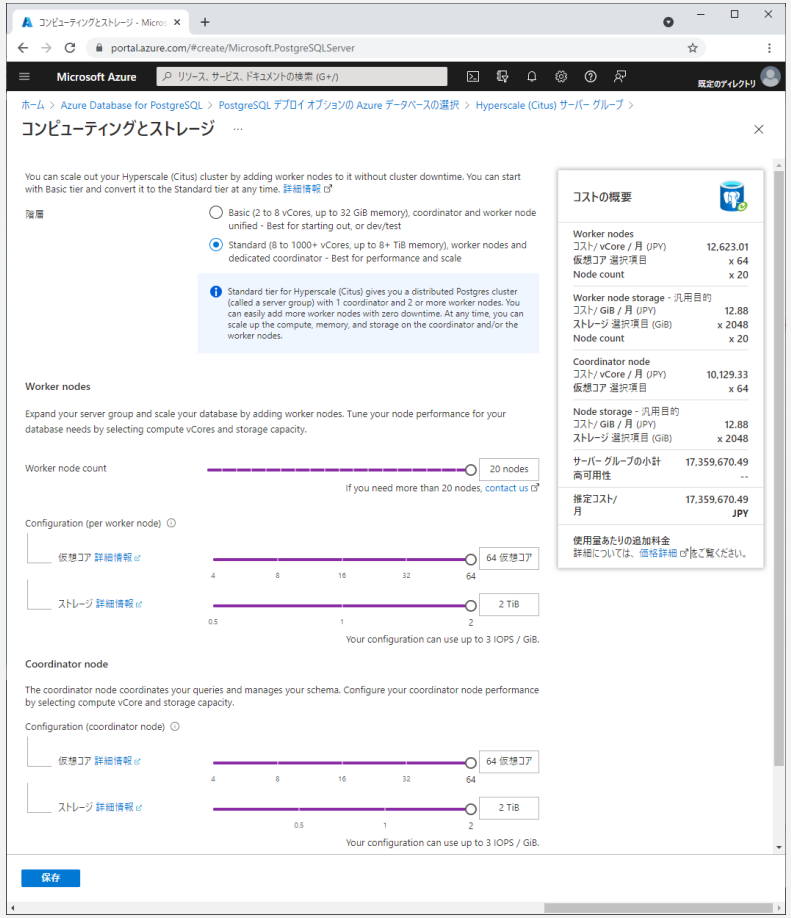
Citus とは？
Azure Database for PostgreSQL – Hyperscale (Citus)

✓構成可能なオプション

Node / Option	Resource	Min – Max	Note
Coordinator Node	# of vCores	2 – 64	
	Storage Size (TiB)	0.125 – 2.0	
	Memory (GiB)	16 – 256	
Worker Node	# of Nodes	0 – 20	
	# of vCores	4 – 64	Per node
	Storage Size (TiB)	0.5 – 2.0	Per node
	Memory Size (GiB)	32 – 512	Per node
Option	High Availability	Off or On	
PostgreSQL	Version	11 – 14	一部リージョンのみ

Citus とは？
Azure Database for PostgreSQL – Hyperscale (Citus)

✓構成可能なオプション



準備

準備
citus EXTENSION

✓インストール
- ✓Coordinator Node と Worker Node にインストール
- ✓テーブルを作成する全データベースで CREATE EXTENSION 文を実行
- ✓インストール・バイナリは全ノードで同一
- ✓アプリケーションに必要なエクステンション（pgcrypto など）も全ノードにインストール

✓設定な設定
- ✓Coordinator Node と Worker Node 共通

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

準備
Worker Node の登録

- ✓Worker Node の登録
  - ✓パスワード無しの接続許可が必要
  - ✓SSL による通信を行う(citus.node_conninfo)
  - ✓citus_add_node 関数でホスト名(または TCP/IP アドレス)とポート番号を指定

```
postgres=# SELECT citus_add_node('cituswk1', 5432) ;
 citus_add_node
----------------
              2
(1 row)
postgres=# SELECT citus_add_node('cituswk2', 5433) ;
 citus_add_node
----------------
              3
(1 row)
・・・
```

テーブル構成

テーブル構成
Distributed Table（分散テーブル）

✓データを分散して保存するテーブル

✓ファクトテーブル向き

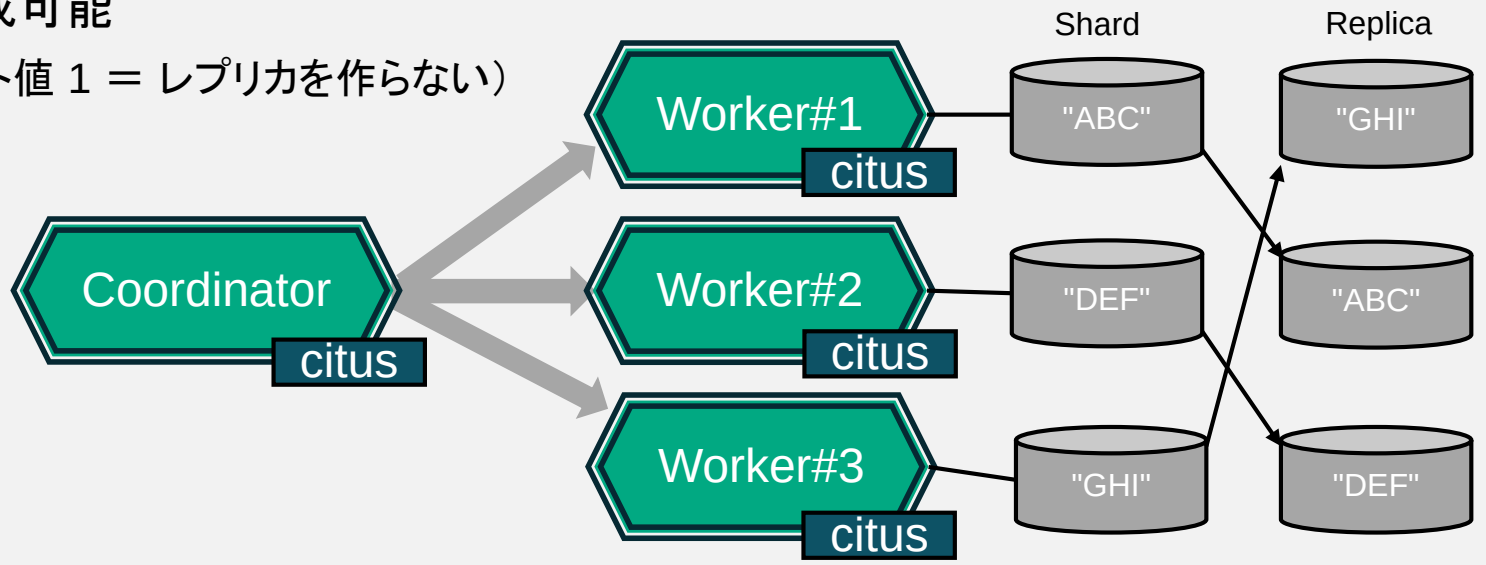
✓分散キーとして列を指定（デフォルトではハッシュ値の範囲によって分散先テーブルを決定）

✓分割数を指定可能

citus.shard_count（デフォルト値 32）

✓異なる Worker Node にレプリカを作成可能

citus.shard_replication_factor（デフォルト値 1 ＝ レプリカを作らない）



テーブル構成
Distributed Table（分散テーブル）

✓分散テーブル数とレプリカ数を指定

```
postgres=> SET citus.shard_count = 6 ;
SET
postgres=> SET citus.shard_replication_factor = 2 ;
SET
```

✓テーブルの作成例

```
postgres=> CREATE TABLE dist1(key1 NUMERIC PRIMARY KEY, val1 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT create_distributed_table('dist1', 'key1') ;
 create_distributed_table
--------------------------

(1 row)
```

テーブル構成
Distributed Table（分散テーブル）

✓Worker Node に作成されるテーブル
- ✓テーブル名は元のテーブル名に ShardID が追加される
- ✓レプリカを指定すると同一名称のテーブルが異なる Worker Node に作成
- ✓名前が同じテーブルには同一データが格納
- ✓Worker Node で作成されるテーブル名の確認には citus_shards カタログを検索

Coordinator	Worker#1	Worker#2	Worker#3
dist1	dist1_102046	dist1_102046	
		dist1_102047	dist1_102047
	dist1_102048		dist1_102048
	dist1_102049	dist1_102049	
		dist1_102050	dist1_102050
	dist1_102051		dist1_102051

テーブル構成
Distributed Table（分散テーブル）

- ✓Worker Node に対する DML
  - ✓WHERE 句に分散列が含まれる場合は、該当する Worker Node にのみアクセスする
  - ✓Coordinator Node からそれぞれの Worker Node に PQsendQuery, PQsendQueryParams を使って非同期実行される
  - ✓Worker Node 間は通信を行わない
  - ✓Worker Node をまたいだ更新処理は Coordinator Node から複数の Worker に DML を実行し、2PC で同期をとる
  - ✓トランザクション開始時に Coordinator Node で採番されたトランザクションIDを Worker Node に assign_distributed_transaction_id 関数で伝達

テーブル構成
Distributed Table（分散テーブル）

✓既存データが格納されたテーブル

- ✓既にデータが格納されているテーブルも Distributed Table に変換できる
- ✓既存テーブルのデータ削除は truncate_local_data_after_distriburing_table 関数を実行

```
postgres=> SELECT create_distributed_table('data2', '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($$public.data2$$)
 create_distributed_table
--------------------------

(1 row)
postgres=> SELECT truncate_local_data_after_distributing_table('data2') ;
 truncate_local_data_after_distributing_table
----------------------------------------------

(1 row)
```

テーブル構成
Distributed Table（分散テーブル）

✓Distributed Table を Local Table に戻すこともできる

```
postgres=> SELECT undistribute_table('data2') ;
NOTICE:  creating a new table for public.data2
NOTICE:  moving the data of public.data2
NOTICE:  dropping the old public.data2
NOTICE:  renaming the new table to public.data2
 undistribute_table
--------------------


(1 row)
```

テーブル構成
Distributed Table（分散テーブル）

✓実行計画

✓citus.explain_all_tasks は Worker Node へ投入する実行計画を出力するかを決める

```
postgres=> SET citus.explain_all_tasks = on ;
SET
postgres=> EXPLAIN SELECT SUM(c1) FROM dist1 WHERE c1=1000 AND c2='data1' ORDER BY 1 ;

                                  QUERY PLAN
------------------------------------------------------------------------------------------
 Custom Scan (Citus Adaptive)  (cost=0.00..0.00 rows=0 width=0)
   Task Count: 1
   Tasks Shown: All
   ->  Task
         Node: host=cituswk1 port=5432 dbname=postgres
         ->  Index Scan using dist1_pkey_102040 on dist1_102040 dist1  (cost=0.43..8.45 ro …
               Index Cond: (c1 = '1000'::numeric)
               Filter: ((c2)::text = 'data1'::text)
(8 rows)
```

テーブル構成
Distributed Table（分散テーブル）

✓実行計画

✓citus.log_remote_commands は Worker Node へ投入する SQL 文のログ出力を行う設定

```
postgres=> SET citus.log_remote_commands = on ;
SET
postgres=> SELECT SUM(c1) FROM dist1 WHERE c1=1000 AND c2='data1' ORDER BY 1 ;

NOTICE:  issuing SELECT sum(c1) AS sum FROM public.dist1_102040 dist1 WHERE
((c1 OPERATOR(pg_catalog.=) (1000)::numeric) AND ((c2 COLLATE "default")
OPERATOR(pg_catalog.=) 'data1'::text)) ORDER BY (sum(c1))
DETAIL:  on server demo@cituswk1:5432 connectionId: 1
 sum
------
 1000
(1 row)
```

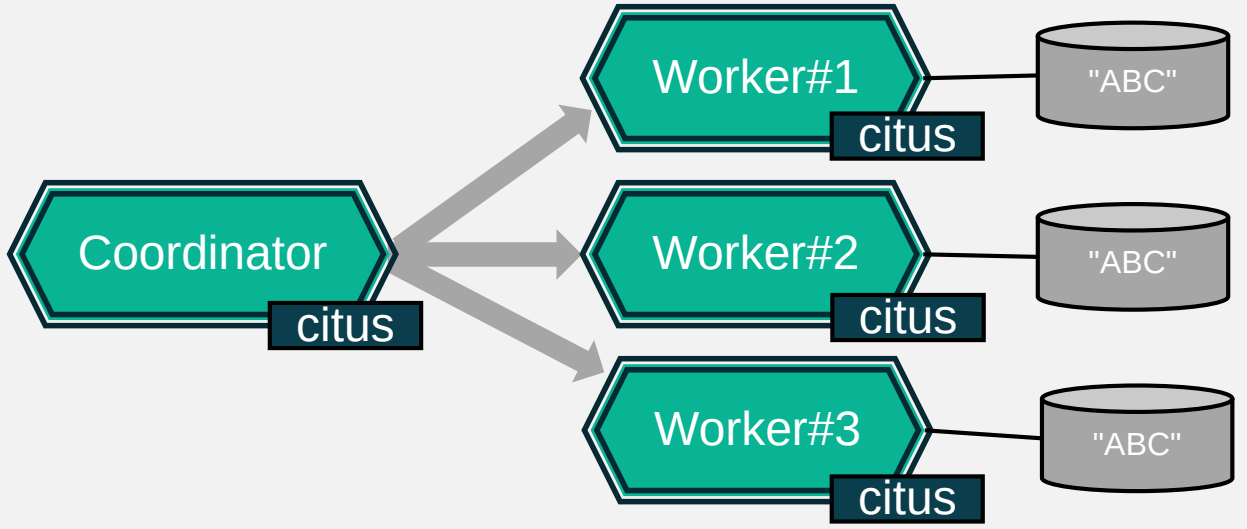
テーブル構成
Distributed Table（分散テーブル）

✓テーブル・メンテナンス用の SQL 文は、テーブル名を変更してそのまま Worker Node で実行
- ✓VACUUM / VACUUM FULL 文
- ✓ANALYZE 文
- ✓ALTER TABLE 文
- ✓CREATE INDEX 文／DROP INDEX 文

テーブル構成
Reference Table（参照テーブル）

✓全ノードに同一データを保存するテーブル

✓ディメンジョンテーブル向き



テーブル構成
Reference Table（参照テーブル）

✓テーブルの作成例

```
postgres=> CREATE TABLE ref1(key1 NUMERIC PRIMARY KEY, val1 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT create_reference_table('ref1') ;
 create_reference_table
------------------------


(1 row)
```

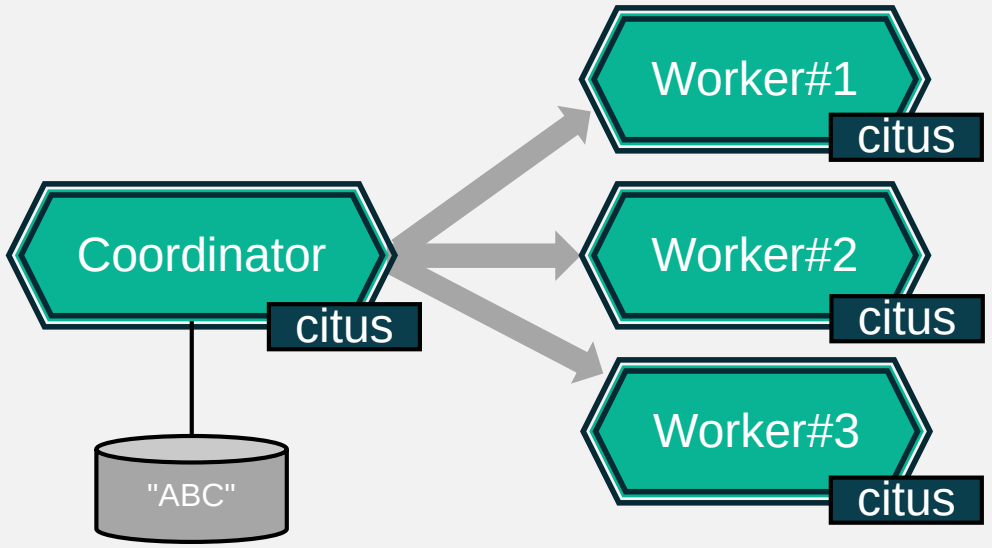
✓Worker Node のテーブル確認例

```
postgres=> ¥d
           List of relations
 Schema |     Name      | Type  | Owner
--------+---------------+-------+-------
 public | ref1_102084   | table | demo
(1 row)
```

テーブル構成
Local Table

✓Coordinator Node 上にデータを保持するテーブル

✓Worker Node とは通信を行わない



テーブル構成
制約

✓分散テーブルに対して実行できないSQL
- ✓分散キー列の更新（UPDATE / INSERT ON CONFLICT 文）
- ✓SELECT FOR UPDATE / SHARE 文（レプリカを作成している場合）
- ✓TABLESAMPLE 句
- ✓INSERT VALUES 文に対する generate_series 関数等
- ✓SERIALIZABLE トランザクション分離レベル（マニュアルに記載無し）

✓制約がある構文
- ✓相関サブクエリー
- ✓GROUPING SETS 句
- ✓PARTITION BY 句
- ✓Local Table と Distributed Table の結合
- ✓Coordinator Node 上のテーブルに作成されたトリガー

https://docs.citusdata.com/en/v10.2/develop/reference_workarounds.html
https://docs.microsoft.com/ja-jp/azure/postgresql/concepts-hyperscale-limits
（インデックスのサポート等、Citus 10.2 で利用できる項目がアップデートされていない）

列指向テーブル

列指向テーブル
概要

✓英語マニュアル上は Columnar Table
✓Citus 10 の新機能
✓列単位に圧縮されて保存されたテーブル
✓圧縮による全件検索の高速化
✓Table Access Method (Pluggable Table Storage Interface) として実装

```
postgres=> SELECT * FROM pg_am WHERE amtype='t' ;
  oid  |  amname  |          amhandler          | amtype
-------+----------+-----------------------------+--------
     2 | heap     | heap_tableam_handler        | t
 16849 | columnar | columnar.columnar_handler | t
(2 rows)
```

列指向テーブル
テーブルの作成

✓CREATE TABLE 文に USING columnar 句を指定して作成

```
postgres=> CREATE TABLE column1(c1 NUMERIC, c2 VARCHAR(10)) USING columnar ;
postgres=> ¥d+
                                    List of relations
 Schema |     Name      | Type  |  Owner   | Persistence | Access method |  Size   | …
--------+---------------+-------+----------+-------------+---------------+---------+-…
 public | citus_tables  | view  | postgres | permanent   |               | 0 bytes |
 public | column1       | table | demo     | permanent   | columnar      | 16 kB   |
(2 rows)
```

列指向テーブル
テーブルの作成

✓既存の Heap テーブルからの変更も可能(alter_table_set_access_method 関数)

```
postgres=> SELECT alter_table_set_access_method('data2', 'columnar') ;
NOTICE:  creating a new table for public.data2
NOTICE:  moving the data of public.data2
NOTICE:  dropping the old public.data2
NOTICE:  renaming the new table to public.data2
 alter_table_set_access_method
-------------------------------

(1 row)
```

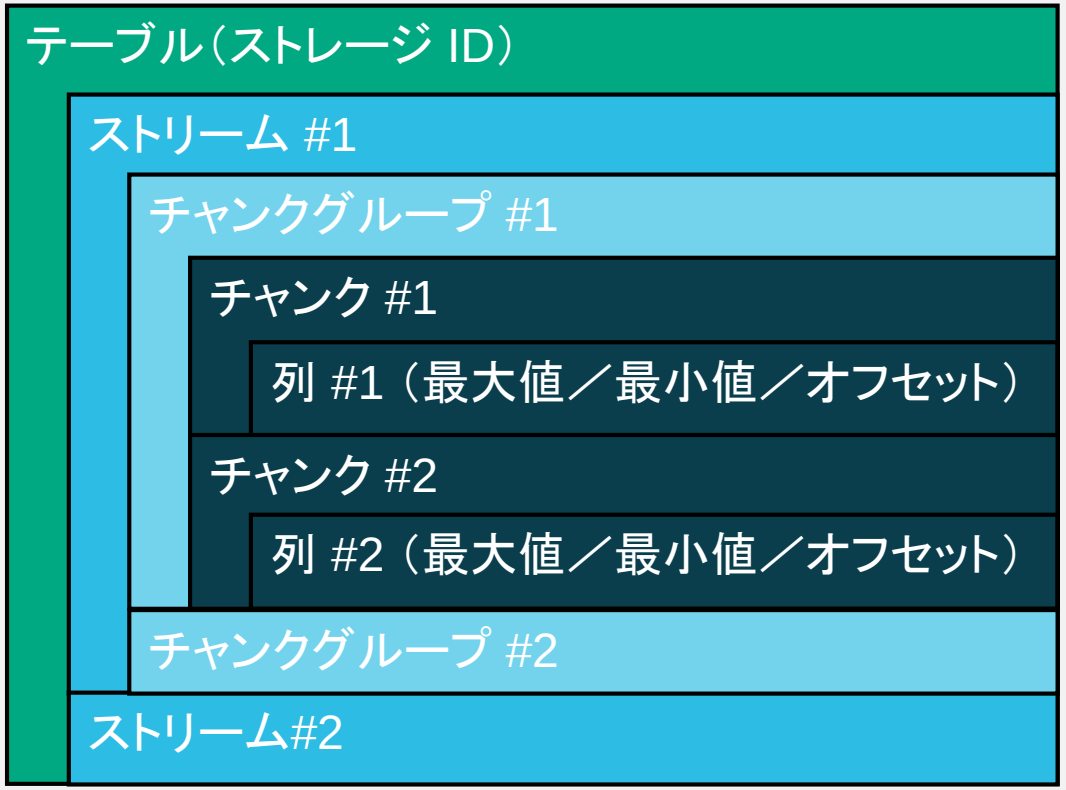
✓PostgreSQL 15 では ALTER TABLE SET ACCESS METHOD 文が利用可能になる予定

https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=b0483263dda0824cc49e3f8a022dab07e1cdf9a7

Add support for SET ACCESS METHOD in ALTER TABLE

列指向テーブル
用語

✓Heap テーブルと異なる構造を持つ



- ✓テーブル
  - ✓ストレージ ID を持つ
- ✓ストリーム
  - ✓圧縮単位
  - ✓トランザクションまたは columnar.stripe_row_limit の単位
- ✓チャンクグループ
  - ✓チャンクの集合
  - ✓複数列をまとめて管理
  - ✓columnar.chunk_group_row_limit の単位で作成
- ✓チャンク
  - ✓列値、最大値／最小値／オフセット

列指向テーブル
カタログ

✓列指向テーブルに関係するカタログ（columnar スキーマ）

カタログ名	説明	主な情報
columnar.stripe	ストライプ一覧	ファイル内オフセット、レコード数、データ長
columnar.chunk_group	チャンク・グループ一覧	レコード数
columnar.chunk	チャンク一覧	列情報、最小値、最大値、圧縮情報
columnar.options	オプション一覧	テーブル毎のオプション設定値

列指向テーブル
ストレージ ID

✓テーブルと 1 対 1
✓列指向テーブル作成時や TRUNCATE 文の実行時に採番
✓VACUUM VERBOSE 文の実行で確認

```
postgres=> VACUUM VERBOSE column1 ;
INFO:  statistics for "column1":
storage id: 10000000003
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
postgres=> SELECT * FROM columnar.stripe WHERE storage_id = 10000000003 ;
 storage_id  | stripe_num | file_offset | data_length | column_count | chunk_row_count | …
-------------+------------+-------------+-------------+--------------+-----------------+ …
 10000000003 |          1 |       16336 |      191013 |            2 |           10000 | …
…
```

列指向テーブル
オプション

✓オプションの指定

パラメーター	説明	デフォルト値	値の範囲
columnar.compression	圧縮方法	zstd	lz4, none, pglz も使用可
columnar.compression_level	圧縮レベル	3	1 ～ 19
columnar.stripe_row_limit	ストライプ行数の最大	150000	1000 ～ 10000000
columnar.chunk_group_row_limit	チャンクグループ行数の最大	10000	1000 ～ 100000

```
postgres=> SELECT * FROM columnar.options ;

 regclass | chunk_group_row_limit | stripe_row_limit | compression_level | compression
----------+-----------------------+------------------+-------------------+-------------
 column1  |                 10000 |           150000 |                 3 | zstd
(1 row)
```

列指向テーブル
オプション

✓オプションの変更
- ✓alter_columnar_table_set 関数で変更可能
- ✓既存のタプルには変更なし

```
postgres=> SELECT alter_columnar_table_set
        (table_name => 'column1', compression => 'lz4', compression_level => 19) ;
 alter_columnar_table_set
--------------------------


(1 row)
postgres=> SELECT * FROM columnar.options ;

 regclass | chunk_group_row_limit | stripe_row_limit | compression_level | compression
----------+-----------------------+------------------+-------------------+-------------
 column1  |                 10000 |           150000 |                19 | lz4
(1 row)
```

列指向テーブル
圧縮効果

✓圧縮効果(2 列／1,000 万タプル)
- ✓column1 : 列指向テーブル／一括コミット
- ✓column2 : 列指向テーブル／10 タプル単位コミット
- ✓data1 : Heap テーブル

```
postgres=> ¥d+
                                        List of relations
 Schema |     Name      | Type  |  Owner   | Persistence | Access method |  Size   |…
--------+---------------+-------+----------+-------------+---------------+---------+…
 public | citus_tables  | view  | postgres | permanent   |               | 0 bytes |
 public | column1       | table | demo     | permanent   | columnar      | 13 MB   |
 public | column2       | table | demo     | permanent   | columnar      | 7813 MB |
 public | data1         | table | demo     | permanent   | heap          | 422 MB  |
(4 rows)
```

列指向テーブル
実行計画

✓実行計画の確認

```
postgres=> EXPLAIN ANALYZE SELECT SUM(c1) FROM column1 WHERE c1 = 10000 ;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------...
 Aggregate  (cost=11.61..11.62 rows=1 width=32) (actual time=3.864..3.865 rows=1 loops=1)
   ->  Custom Scan (ColumnarScan) on column1  (cost=0.00..11.60 rows=1 width=6) (actual tim…
         Filter: (c1 = '10000'::numeric)
         Rows Removed by Filter: 9999
         Columnar Projected Columns: c1
         Columnar Chunk Group Filters: (c1 = '10000'::numeric)
         Columnar Chunk Groups Removed by Filter: 999
 Planning Time: 0.284 ms
 Execution Time: 3.893 ms
(9 rows)
```

列指向テーブル
インデックス

✓Citus 10.2 からサポート(Citus 10.1 でも動作する)
- ✓BTREE, HASH のみサポート

```
postgres=> CREATE INDEX idx1_column1 ON column1(c2) ;
CREATE INDEX
postgres=> EXPLAIN ANALYZE SELECT * FROM column1 WHERE c2='value0'  ;

                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------…
 Index Scan using idx1_column1 on column1  (cost=0.43..27.31 rows=1 width=12) (actual time…
   Index Cond: ((c2)::text = 'value0'::text)
 Planning Time: 0.148 ms
 Execution Time: 0.048 ms
(4 rows)
```

列指向テーブル
制約

✓実行できない SQL や制約がある機能
- ✓ UPDATE / DELETE 文
- ✓ TABLESAMPLE 句
- ✓ SELECT FOR UPDATE 文
- ✓ UNLOGGED / TEMPORARY テーブル
- ✓ SERIALIZABLE トランザクション分離レベル（マニュアル上の記述）

```
postgres=> UPDATE column1 SET c2='update' WHERE c1=1000 ;
ERROR:  UPDATE and CTID scans not supported for ColumnarScan

postgres=> SELECT * FROM column1 FOR UPDATE ;
ERROR:  UPDATE and CTID scans not supported for ColumnarScan

postgres=> SELECT * FROM column1 TABLESAMPLE SYSTEM (1) ;
ERROR:  sample scans not supported on columnar tables
```

https://docs.citusdata.com/en/v10.2/admin_guide/table_management.html#limitations

列指向テーブルと分散テーブル

列指向テーブルと分散テーブル
概要

- ✓テーブルの種類を組み合わせることができる
  - ✓列指向テーブルを分散テーブル（Distributed Table）や参照テーブル（Reference Table）に
  - ✓列指向テーブルを特定のパーティションに
  - ✓列指向テーブルを含むパーティション・テーブルを分散テーブルに

列指向テーブルと分散テーブル
列指向テーブルを分散化

✓列指向テーブルを Distributed Table 化／ Reference Table 化

```
postgres=> CREATE TABLE coldist1(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10)) USING columnar ;
CREATE TABLE
postgres=> SELECT create_distributed_table('coldist1', 'c1') ;
 create_distributed_table
--------------------------


(1 row)
postgres=> CREATE TABLE coldist2(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10)) USING columnar ;
CREATE TABLE
postgres=> SELECT create_reference_table('coldist2') ;
 create_reference_table
------------------------


(1 row)
```

列指向テーブルと分散テーブル
列指向テーブルをパーティション化

✓列指向テーブルのパーティション化

```
postgres=> CREATE TABLE part1(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10)) PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES FROM (1000000) TO (2000000)
        USING heap ;
CREATE TABLE
postgres=> CREATE TABLE part1v2 PARTITION OF part1 FOR VALUES FROM (2000000) TO (3000000)
        USING columnar ;
CREATE TABLE
```

✓パーティション・プルーニングにより列値によって制約が変わるので注意が必要

```
postgres=> UPDATE part1 SET c2='update' WHERE c1=100000 ;
UPDATE 1
postgres=> UPDATE part1 SET c2='update' WHERE c1=200000 ;
ERROR:  UPDATE and CTID scans not supported for ColumnarScan
```

列指向テーブルと分散テーブル
列指向テーブルを含むパーティション・テーブルの分散化

✓列指向テーブルを含むパーティション・テーブルの分散化

```
postgres=> CREATE TABLE part1(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10)) PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES FROM (1000000) TO (2000000)
        USING heap ;
CREATE TABLE
postgres=> CREATE TABLE part1v2 PARTITION OF part1 FOR VALUES FROM (2000000) TO (3000000)
        USING columnar ;
CREATE TABLE
postgres=> SELECT create_distributed_table('part1', 'c1');
 create_distributed_table
--------------------------

(1 row)
```

その他

その他
リバランサー

✓Worker Node の増減時にオンライン・リバランスを実行可能
  ✓全テーブル、特定のテーブルを選択可能

```
postgres=# SELECT rebalance_table_shards() ;

NOTICE:  Moving shard 102525 from cituswk3:5434 to cituswk4:5435 …
NOTICE:  Moving shard 102524 from cituswk2:5433 to cituswk4:5435 ...
NOTICE:  Moving shard 102523 from cituswk1:5432 to cituswk4:5435 ...
NOTICE:  Moving shard 102536 from cituswk3:5434 to cituswk4:5435 ...
NOTICE:  Moving shard 102535 from cituswk2:5433 to cituswk4:5435 ...
NOTICE:  Moving shard 102537 from cituswk1:5432 to cituswk4:5435 ...
 rebalance_table_shards
------------------------


(1 row)
```

その他
概算

✓HyperLogLog を使って COUNT (DISTINCT) 文を高速に計算
✓citus.count_distinct_error_rage に 0 より大きい値を設定(0.0～1.0)

```
postgres=# CREATE EXTENSION hll ;
CREATE EXTENSION

postgres=> SET citus.count_distinct_error_rate = 0.05 ;
SET
postgres=> SELECT COUNT(DISTINCT c1) FROM dist1 ;

NOTICE:  issuing SELECT public.hll_add_agg(public.hll_hash_any(c1), 9) AS count FROM
        public.dist1_102040 dist1 WHERE true
DETAIL:  on server demo@cituswk1:5432 connectionId: 1
NOTICE:  issuing SELECT public.hll_add_agg(public.hll_hash_any(c1), 9) AS count FROM
        public.dist1_102041 dist1 WHERE true
DETAIL:  on server demo@cituswk2:5433 connectionId: 2
...
```

まとめ

まとめ

制約と運用に注意すれば気軽にスケールアウトが可能

✓比較的簡単にテーブル単位でスケールアウト環境を構築
✓ノードをまたいだパラレル・クエリー＋パーティショニングによる性能改善の可能性
✓列指向テーブルは全件検索の高速化を見込める
✓障害対策やバックアップは独自に実装が必要
✓SQL 文の実行制約があるので事前のアプリケーション検証を推奨

THANK YOU

Mail: noriyoshi.shinoda@hpe.com
Twitter: @nori_shinoda