PERCONA

Databases run better with Percona

Horizontally Scaling PostgreSQL

Working with the CitusDB Extension

Percona is hiring!

- Senior Software Engineer (PostgreSQL)
- Support Engineer (PostgreSQL)
- PostgreSQL Evangelist

… and more!

Before You Start

Who's this Workshop for ...

- Linux (Ubuntu)
    - CLI
        - ssh (login using public keys)
        - netstat
        - su
        - sudo
    - bash scripting (basic stuff)

- DBMS knowledge
    - general administration i.e. create tables etc

- PostgreSQL
    - Fully familiar with administrating a postgres cluster
        - User Account Creation
        - Creating
            - datacluster
            - database
            - tables
            - extensions
            - user accounts, includes assigning passwords
        - Configuration
            - authentication
            - permissions
            - basic tuning

Citus Workshop

Overview

- Introduction:
- About CitusDB
- Installation
- Configuration
- Exploring the Extension

- Scenarios
    - column-wise tables
    - data redundancy
    - a primer on horizontal scaling
- Conclusion:
    - The good
    - The bad

REFERENCES

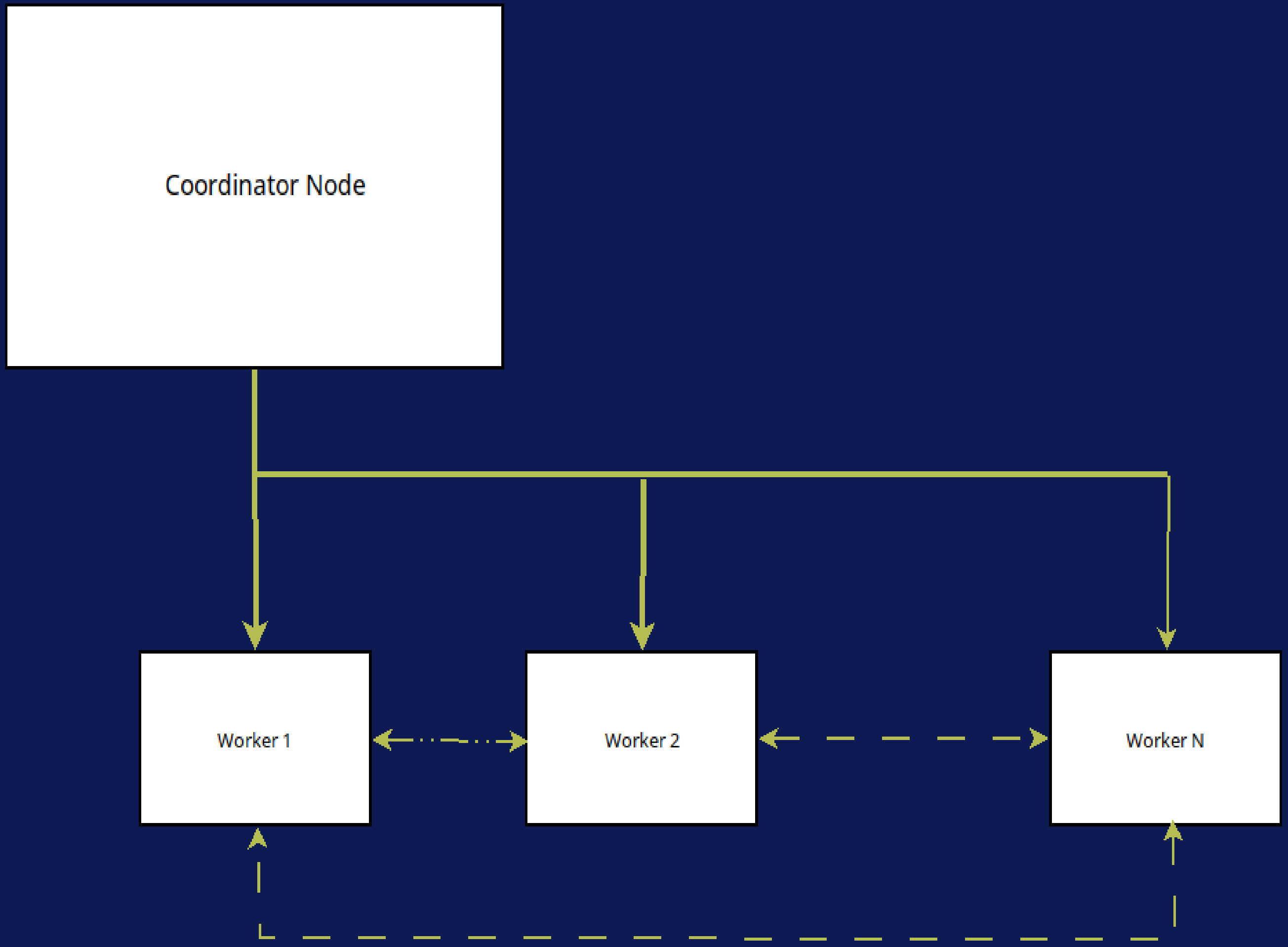
https://www.citusdata.com/faq
https://www.citusdata.com/faq
https://docs.citusdata.com/en/stable/

What is CitusDB

```
An extension that:
- horizontally scales PostgreSQL
- uses sharding and replication.
- Parallelizes SQL queries
- Can create column wise tables
```

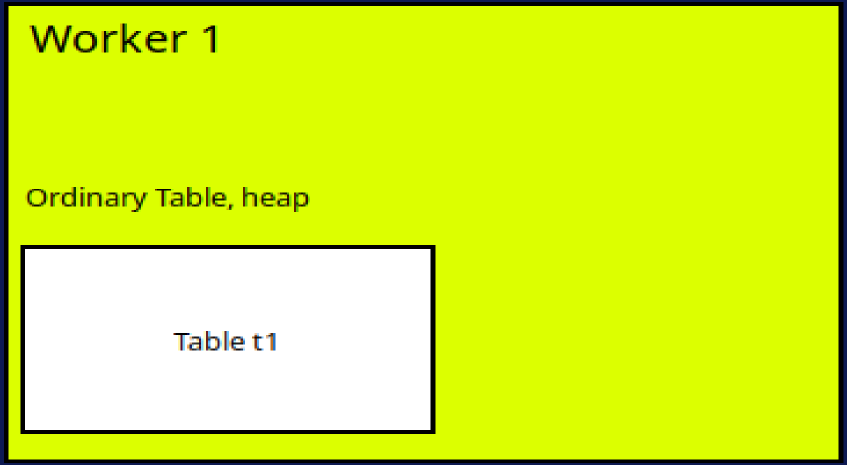
Some Citus Concepts

```
NODES

- coordinator
- worker
```



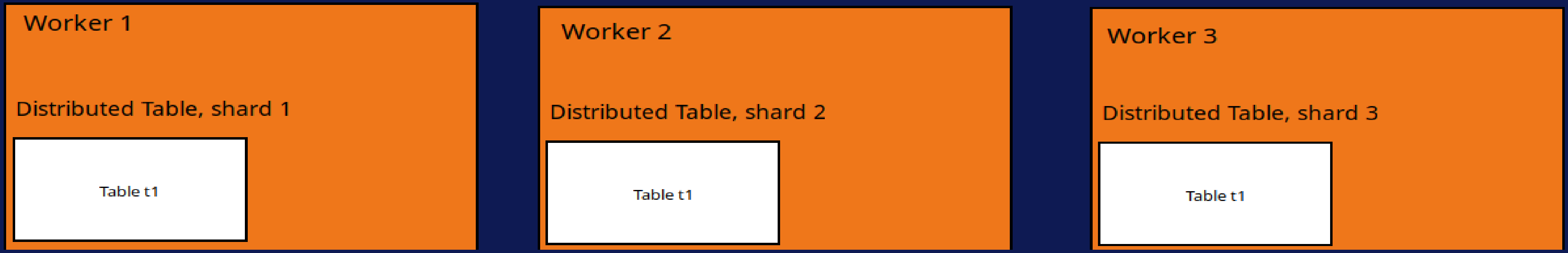
Some Citus Concepts

```
Ordinary Table (heap)
```



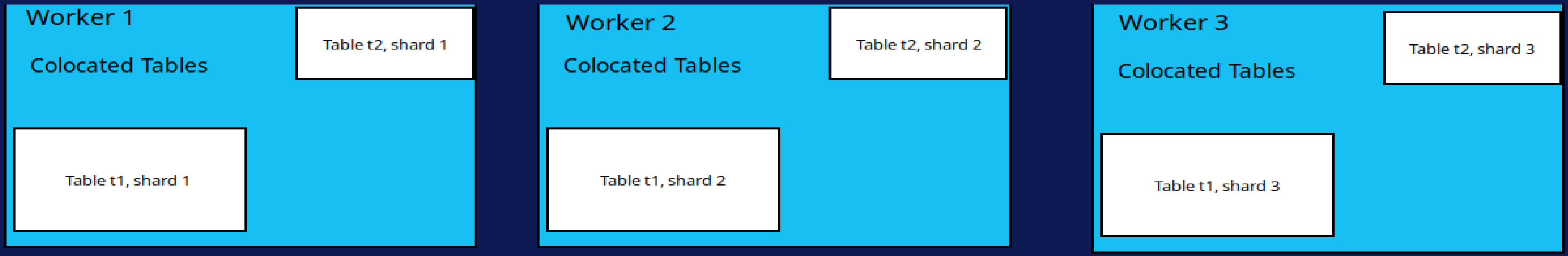
Some Citus Concepts

```
Distributed Table
```


Worker 1
Distributed Table, shard 1
Table t1
Worker 2
Distributed Table, shard 2
Table t1
Worker 3
Distributed Table, shard 3
Table t1

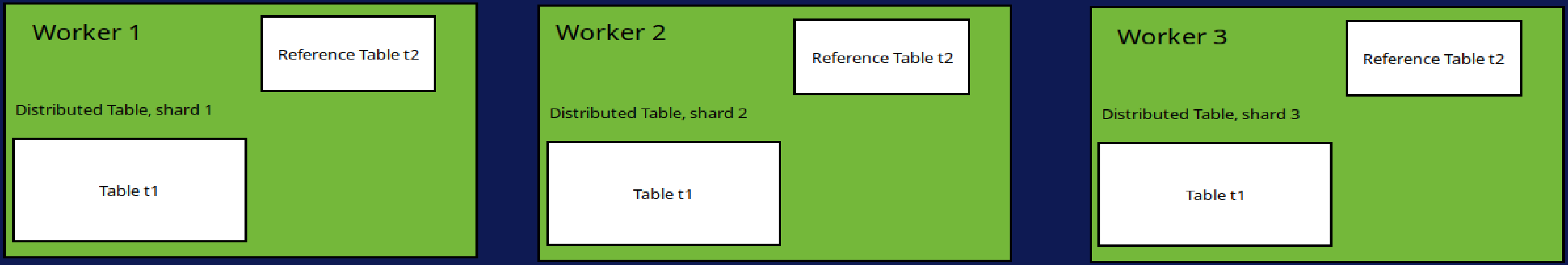
Some Citus Concepts

`Distributed table, Colocated (foreign keys)`



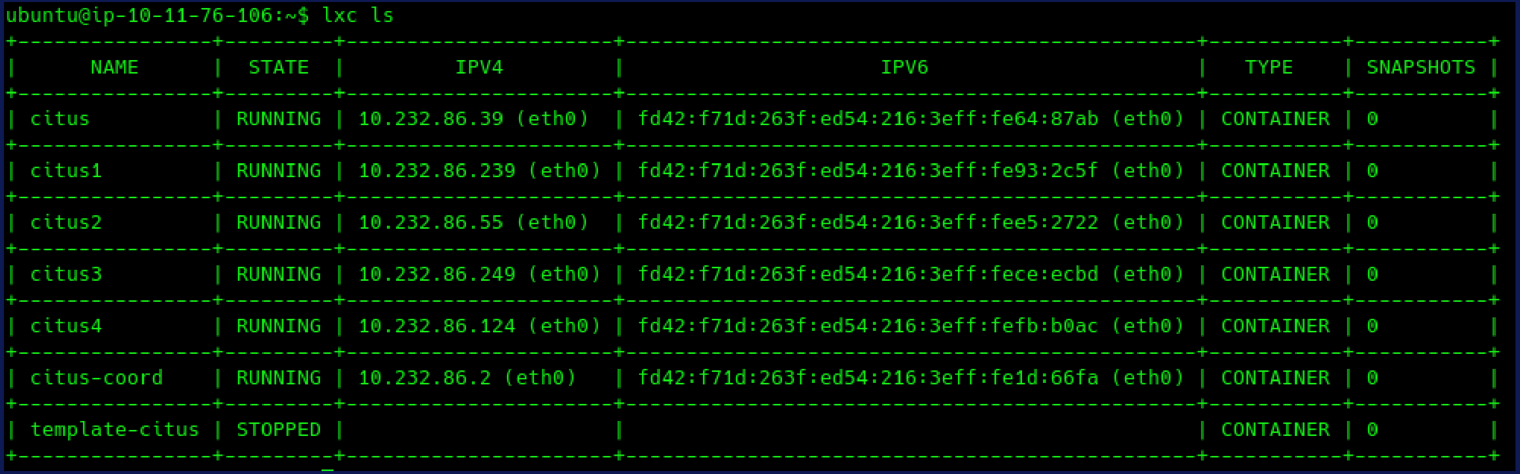
Some Citus Concepts

Reference Table (data redundancy)



About The Workshop Environment

```
About the EC2 instance (Ubuntu 22.04 LTS, jammy)

- INSTALL: The EC2 instance container: “citus”
  - method 1 (via citus portal)
  - method 2 (github clone source-code)
  - method 3 (install via DEBIAN package, based upon github)

- SCALE: The EC2 instance containers
    coordinator node
       citus-coord
    worker nodes
       citus1
       citus2
       citus3
```



```
ubuntu@ip-10-11-76-106:~$ lxc ls
+----------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
|      NAME      |  STATE  |         IPV4         |                     IPV6                      |   TYPE    | SNAPSHOTS |
+----------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
| citus          | RUNNING | 10.232.86.39 (eth0)  | fd42:f71d:263f:ed54:216:3eff:fe64:87ab (eth0) | CONTAINER | 0         |
+----------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
| citus1         | RUNNING | 10.232.86.239 (eth0) | fd42:f71d:263f:ed54:216:3eff:fe93:2c5f (eth0) | CONTAINER | 0         |
+----------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
| citus2         | RUNNING | 10.232.86.55 (eth0)  | fd42:f71d:263f:ed54:216:3eff:fee5:2722 (eth0) | CONTAINER | 0         |
+----------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
| citus3         | RUNNING | 10.232.86.249 (eth0) | fd42:f71d:263f:ed54:216:3eff:fece:ecbd (eth0) | CONTAINER | 0         |
+----------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
| citus4         | RUNNING | 10.232.86.124 (eth0) | fd42:f71d:263f:ed54:216:3eff:fefb:b0ac (eth0) | CONTAINER | 0         |
+----------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
| citus-coord    | RUNNING | 10.232.86.2 (eth0)   | fd42:f71d:263f:ed54:216:3eff:fe1d:66fa (eth0) | CONTAINER | 0         |
+----------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
| template-citus | STOPPED |                      |                                               | CONTAINER | 0         |
+----------------+---------+----------------------+-----------------------------------------------+-----------+-----------+
```

About The Workshop Environment, cont’d

STEPS: Creating an EC2 instance for the workshop

1. Navigate the to the AWS Management Console for Percona Consultants for the “Oregon Region”

2. Navigate to EC2 running instances and on the left Panel click, under IMAGES the AMI.

3. Under AMI Name, select AMI named “TEMPLATE-Citus-Webinar-v#”

4. Using the button on the top right side click “Launch instance from AMI”

6. Among the options you will choose select:
- Instance type t5.2xlarge
- Select/generate key pair (login)
- Under network settings: make the IP address for IPv4 public (ssh login is required)

7. Launch the *EC2 instance.*

8. *Login EC2 instance:* `ssh -i <your private key.pem> ubuntu@<EC2host>`

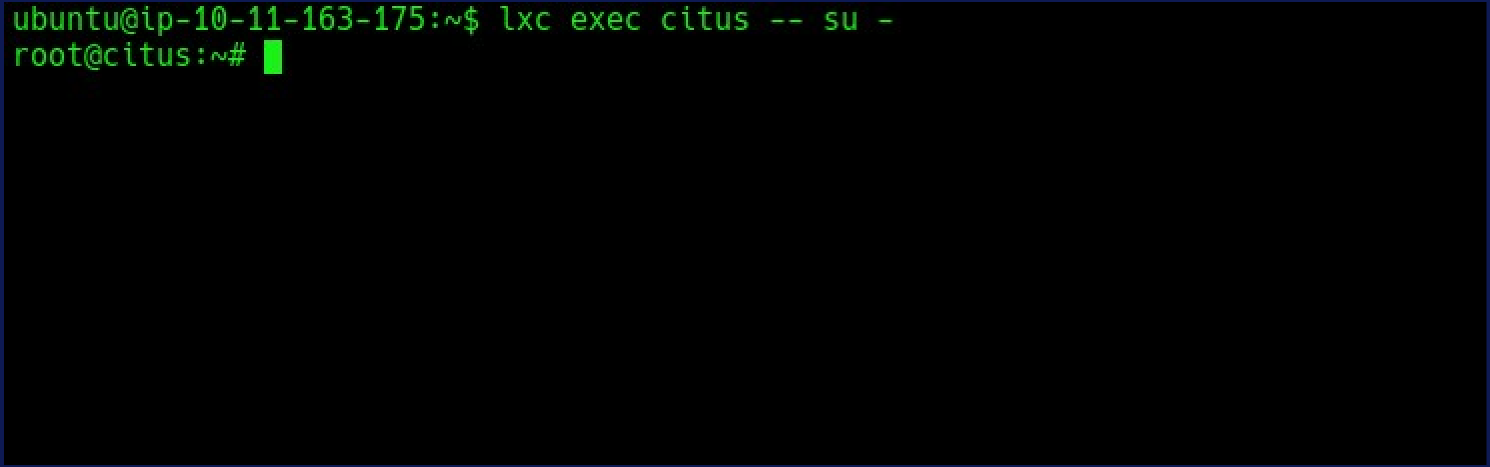
9. List *the containers on the EC2 host: here is the list of running containers you will use*
- *citus*
- *citus1*
- *citus2*
- *citus3*
- *citus4*
- *cius-coord*

10. *Login the containers on the EC2 host: start with container “citus”*

```
      lxc exec <container> -- su -
```

LOGIN

ACCESSING LXD CONTAINER “citus”



```
ubuntu@ip-10-11-163-175:~$ lxc exec citus -- su -
root@citus:~# 
```

Getting It

CitusDB Portal

```
# CONFIGURE REPOSITORY
curl https://install.citusdata.com/community/deb.sh | sudo bash
apt-get update && apt-get upgrade -y

# INSTALL THE SERVER
apt install -y postgresql-16-citus-12.1

# INSTALL CITUS MODULE
pg_conftool 16 main set shared_preload_libraries 'citus'

# CONFIGURE POSTGRES
pg_conftool 16 main set wal_level 'logical'
pg_conftool 16 main set listen_addresses '*'
systemctl restart postgresql@16-main
pg_lsclusters
```

Getting It

Method 2: GITHUB

```
### GET ALL YOUR PACKAGES ###
# execute the following as root
# It's understood you've followed either METHOD 1 above
# OR
# The standard install instructions for community postgres
#
echo "deb-src http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" >> /etc/apt/sources.list.d/pgdg.list

apt-get update && apt-get upgrade -y
apt-get install -y  git gcc make liblz4-dev libzstd-dev \
                    postgresql-server-dev-16 libkrb5-dev autoconf

git clone https://github.com/citusdata/citus.git

cd citus
git branch -a | grep remotes/origin/release- | less
git checkout remotes/origin/release-12.1
git branch

### COMPILE AND INSTALL ###
export PATH=/usr/lib/postgresql/16/bin:/usr/sbin:/usr/bin:/bin:/snap/bin

cd $HOME/citus

# remove telemetry
./configure --without-libcurl
make install DESTDIR=$HOME/installdir-citus12.1/pg16

# confirm it compiled correctly: an old hacker trick
ldd root/installdir-citus12.1/pg16/usr/lib/postgresql/16/lib/citus.so

# install into existing pg 15 binary path
# make install

# confirm it's seen by postgres 16
Select * from pg_available_extension_versions() where name~'^citus' order by 1;
```

Getting It

Method 3: Package Install, DEB

```
# NOTE: it's understood that the community repository is already registered
#       and the citus package is already present in your root directory

cd $HOME
apt install ./postgresql-16-citusdb_12.1-1_amd64.deb


# configuration

pg_conftool 16 main set shared_preload_libraries 'citus'
pg_conftool 16 main set listen_addresses '*'
systemctl restart postgresql@16-main
```

Exploring the CitusDB Extension

```
-- login psql

create database db01
\c db01

select *
    from pg_available_extension_versions()
    where name~'^citus'
    order by 1,2;

show shared_preload_libraries;

create extension citus;

select * from pg_extension;
```

Exploring the CitusDB Extension
About CitusDB Functions

```
select proname, prosrc
    from pg_proc
    where probin ~ 'citus'
    order by 1;

set search_path=citus,citus_internal,columnar,columnar_internal;

db01=# \df
```

```
                                           List of functions
      Schema       |                    Name                     | Result data type | Argument data types | Type
-------------------+---------------------------------------------+------------------+---------------------+------
 citus_internal    | find_groupid_for_node                       | integer          | text, integer       | func
 citus_internal    | pg_dist_node_trigger_func                   | trigger          |                     | func
 citus_internal    | pg_dist_rebalance_strategy_trigger_func     | trigger          |                     | func
 citus_internal    | pg_dist_shard_placement_trigger_func        | trigger          |                     | func
 citus_internal    | refresh_isolation_tester_prepared_statement | void             |                     | func
 citus_internal    | replace_isolation_tester_func               | void             |                     | func
 citus_internal    | restore_isolation_tester_func               | void             |                     | func
 columnar          | get_storage_id                              | bigint           | regclass            | func
 columnar_internal | columnar_ensure_am_depends_catalog          | void             |                     | func
 columnar_internal | columnar_handler                            | table_am_handler | internal            | func
 columnar_internal | downgrade_columnar_storage                  | void             | rel regclass        | func
 columnar_internal | upgrade_columnar_storage                    | void             | rel regclass        | func


ATTENTION:
- many of the functions have been added to pg catalogs
- many of the functions are not meant to be used directly
```

Exploring the CitusDB Extension
About CitusDB Schemas & Tables

```
db01=# \dn
            List of schemas
       Name        |       Owner
-------------------+-------------------
 citus             | postgres
 citus_internal    | postgres
 columnar          | postgres
 columnar_internal | postgres
 public            | pg_database_owner

set search_path=citus,citus_internal,columnar,columnar_internal;

db01=# \d
                    List of relations
      Schema       |     Name      |   Type   |  Owner
-------------------+---------------+----------+----------
 columnar          | chunk         | view     | postgres
 columnar          | chunk_group   | view     | postgres
 columnar          | options       | view     | postgres
 columnar          | storage       | view     | postgres
 columnar          | stripe        | view     | postgres
 columnar_internal | storageid_seq | sequence | postgres



TIP: use '\d+' to get detailed information about each relation
```

Exploring the CitusDB Extension
About CitusDB Runtime Options

```
select *
    from pg_settings
    where name~'citus'
    order by 1,2;

                        name                        |     setting
----------------------------------------------------+-----------------
 citus.all_modifications_commutative                | off
 citus.background_task_queue_interval               | 5000
 citus.cluster_name                                 | default
 citus.coordinator_aggregation_strategy             | row-gather
 citus.count_distinct_error_rate                    | 0
 citus.cpu_priority                                 | 0
 citus.cpu_priority_for_logical_replication_senders | inherit
 citus.defer_drop_after_shard_move                  | on
 citus.defer_drop_after_shard_split                 | on
 citus.defer_shard_delete_interval                  | 15000
 citus.desired_percent_disk_available_after_move    | 10
 citus.distributed_deadlock_detection_factor        | 2
 citus.enable_binary_protocol                       | on
.
.
.
 citus.stat_tenants_limit                           | 100
 citus.stat_tenants_log_level                       | off
 citus.stat_tenants_period                          | 60
 citus.stat_tenants_track                           | none
 citus.stat_tenants_untracked_sample_rate           | 1
 citus.task_assignment_policy                       | greedy
 citus.task_executor_type                           | adaptive
 citus.use_citus_managed_tables                     | off
 citus.use_secondary_nodes                          | never
 citus.values_materialization_threshold             | 100
 citus.version                                      | 12.1.1
 citus.worker_min_messages                          | notice
 citus.writable_standby_coordinator                 | off
(65 rows)
```

Working With CitusDB

3 Scenarios

```
Scenario 1: Columnar Tables

     host: citus
```

```
Scenario 2: Data Redundancy

     hosts: citus-coord      (coordinator node)
            citus1           (worker node)
            citus2           (worker node)
            citus3           (worker node)
            citus4           (worker node)
```

```
Scenario 3: Horizontal Scaling

     hosts: citus-coord
            citus1
            citus2
            citus3
            citus4
```

Columnar Tables

CitusDB Columnar Tables

Row-wise vs Column-wise

About Columnar Tables

- Row-oriented systems:
- PostgreSQL (default row-wise configuration)
- A column-oriented DBMS (columnar DBMS):

CitusDB Columnar Tables

Cont’d

Introducing The Citus Columnar Extension

The CITUS columnar extension feature set includes:

- Highly compressed tables
- Projection Pushdown
- Chunk Group Filtering.

CitusDB Columnar Tables

A Working Example

```
-- Time: 8282.706 ms (00:08.283)
create table if not exists t1(id,qty)
using heap
as
select (random()*10)::int, (random()*1000)::int from generate_series(1,10e6);

-- Time:4078.320 ms (00:04.078)
create table if not exists t2(id,qty)
using columnar
as
select (random()*10)::int, (random()*1000)::int from generate_series(1,10e6);
```

```
-- \d+ t?
                                           Table "public.t1"
 Column |  Type   | Collation | Nullable | Default | Storage | Compression | Stats target | Description
--------+---------+-----------+----------+---------+---------+-------------+--------------+-------------
 id     | integer |           |          |         | plain   |             |              |
 qty    | integer |           |          |         | plain   |             |              |

Access method: heap

                                           Table "public.t2"
 Column |  Type   | Collation | Nullable | Default | Storage | Compression | Stats target | Description
--------+---------+-----------+----------+---------+---------+-------------+--------------+-------------
 id     | integer |           |          |         | plain   |             |              |
 qty    | integer |           |          |         | plain   |             |              |

Access method: columnar
```

CitusDB Columnar Tables

A Working Example Cont'd

```
-- \dt+ t?

                                  List of relations
 Schema | Name | Type  |  Owner   | Persistence | Access method |  Size  | Description
--------+------+-------+----------+-------------+---------------+--------+-------------
 public | t1   | table | postgres | permanent   | heap          | 342 MB |
 public | t2   | table | postgres | permanent   | columnar      | 26 MB  |
```

CitusDB Columnar Tables

SQL Statements, Preliminary

SQL	Timings
-- HEAP TABLE drop table if exists t1,t2; create table if not exists t1(id,qty) using heap as select (random()*10)::int, (random()*1000)::int from generate_series(1,10e6);	8.5s
-- COLUMNAR TABLE create table if not exists t2(id,qty) using columnar as select (random()*10)::int, (random()*1000)::int from generate_series(1,10e6);	4.8s
-- HEAP TABLE, adding 5 million records do $$ Begin for i in 5.1e6..10e6 loop insert into t1 values((random()*10)::int,(random()*1000)::int); end loop; end $$;	14.5s
-- COLUMNAR TABLE, adding 5 million records do $$ Begin for i in 5.1e6..10e6 loop insert into t2 values((random()*10)::int,(random()*1000)::int); end loop; end $$;	11.2s
-- HEAP TABLE create index on t1(id);	6.3s
-- COLUMNAR TABLE create index on t2(id);	9.8s

```
Relative performance can vary drastically depending upon
RAM, CPU, DISK ETC
```

CitusDB Columnar Tables

SQL Statement Query Plans, Part I

SQL	Timings
-- HEAP TABLE explain analyze select id,qty from t1;	**1204.1 ms**
-- COLUMNAR TABLE explain analyze select id,qty from t2;	**1448.036 ms**
-- HEAP TABLE explain analyze select id,qty from t1 order by random();	10265.894 ms
-- COLUMNAR TABLE explain analyze select id,qty from t2 order by random();	10789.348 ms
-- HEAP TABLE explain analyze select sum(qty) from t1;	**553.483 ms**
-- COLUMNAR TABLE explain analyze select sum(qty) from t2;	**1552.351 ms**
-- HEAP TABLE explain analyze select id,sum(qty) from t1 group by id;	962.594 ms
-- COLUMNAR TABLE explain analyze select id,sum(qty) from t2 group by id;	2593.904 ms

10 million records, 2 column tables

Using the aforementioned table definitions, the following metrics were generated with the runtime parameter

max_parallel_workers_per_gather = 2

It's quite evident that, at least for these two tables, there's no performance benefit of a columnar accessed table over a regular heap accessed one

CitusDB Columnar Tables

SQL Statement Query Plans, Part II

```
create table t3 (
c1 bigserial primary key
,c2 bigint
,c3 text default 'aowjfa fawjfawofjawofjawoifawevvaerarpfjkaofvaweawe[OJARGOIAJOAFWF'
,c4 text default 'aowjfa fawjfawofjawofjawoifawevvaerarpfjkaofvaweawe[OJARGOIAJOAFWF'
.
,c99 text default 'aowjfa fawjfawofjawofjawoifawevvaerarpfjkaofvaweawe[OJARGOIAJOAFWF'
,c100 text default 'aowjfa fawjfawofjawofjawoifawevvaerarpfjkaofvaweawe[OJARGOIAJOAFWF'
) using heap;
```

```
create table t4(
c1 bigserial primary key
,c2 bigint
,c3 text default 'aowjfa fawjfawofjawofjawoifawevvaerarpfjkaofvaweawe[OJARGOIAJOAFWF'
,c4 text default 'aowjfa fawjfawofjawofjawoifawevvaerarpfjkaofvaweawe[OJARGOIAJOAFWF'
.
,c99 text default 'aowjfa fawjfawofjawofjawoifawevvaerarpfjkaofvaweawe[OJARGOIAJOAFWF'
,c100 text default 'aowjfa fawjfawofjawofjawoifawevvaerarpfjkaofvaweawe[OJARGOIAJOAFWF'
) using columnar;
```

```
--
insert into t3(c2) select generate_series(1,5e6)*random()*10;
--
create index on t3(c2);

--
insert into t4(c2) select generate_series(1,5e6)*random()*10;
create index on t4(c2);
```

Although the number of records is halved, the number of columns is increased from two to one hundred.

CitusDB Columnar Tables

SQL Statement Query Plans, Part II

```
                                 List of relations
 Schema | Name | Type  |  Owner   | Persistence | Access method | Size  | Description
--------+------+-------+----------+-------------+---------------+-------+-------------
 public | t3   | table | postgres | permanent   | heap          | 67 GB |
 public | t4   | table | postgres | permanent   | columnar      | 89 MB |
```

```
                                          List of relations
 Schema |      Name       | Type  |  Owner   | Table | Persistence | Access method |  Size   | Description
--------+-----------------+-------+----------+-------+-------------+---------------+---------+-------------
 public | t3_c2_idx       | index | postgres | t3    | permanent   | btree         | 105 MB |
 public | t3_length_idx   | index | postgres | t3    | permanent   | btree         | 33 MB  |
 public | t3_pkey         | index | postgres | t3    | permanent   | btree         | 107 MB |
 public | t4_c2_idx       | index | postgres | t4    | permanent   | btree         | 105 MB |
 public | t4_length_idx   | index | postgres | t4    | permanent   | btree         | 33 MB  |
 public | t4_pkey         | index | postgres | t4    | permanent   | btree         | 107 MB |
```

The columnar table’s resultant compression is remarkable as the default size is reduced by a factor of 752X.

CitusDB Columnar Tables

SQL Statement Query Plans, Part II

SQL	Timings Workers Per Gather 4	 1
-- HEAP TABLE without index explain analyze select sum(c2) from t3;	**9.6s**	**8.7s**
-- COLUMNAR TABLE without index explain analyze select sum(c2) from t4;	**590.176ms**	**596.459ms**
-- HEAP TABLE explain analyze select count(c3) from t3;	10.4s	8.8s
-- COLUMNAR TABLE explain analyze select count(c3) from t4;	509.209ms	541.452ms
-- HEAP TABLE explain analyze select max(length(c25)) from t3;	**1m34s**	**1m17s**
-- COLUMNAR TABLE explain analyze select max(length(c25)) from t4;	**1.1s**	**1.2s**
-- HEAP TABLE explain analyze select sum(length(c50)) from t3;	1m33s	1m18s
-- COLUMNAR TABLE explain analyze select sum(length(c50)) from t4;	1.2s	1.2s

5 million records, 100 column tables

In order to get a better idea of performance differences a second set of tables, at a greater scale were created.

However this time, while the number of records was halved, the number of columns was increased from two to one hundred.

Even if most of the columns are simply copies of one another, the columnar table's resultant compression is remarkable as the default size is reduced by a factor of **752X!**

Unlike the first set of query plans, these ones clearly demonstrate a significant performance improvement.

Curious to see what would change in the way of performance, the varying the **max_parallel_workers_per_gather** doesn't appear to have changed much.

CitusDB Columnar Tables

Working with Indexes

SQL	Timings max parallel workers 4	1
-- HEAP TABLE using BTREE index explain analyze select sum(c2) from t3;	467.789ms	748.939ms
-- COLUMNAR TABLE using BTREE index explain analyze select sum(c2) from t4;	561.522ms	599.629ms
-- HEAP TABLE using EXPRESSION index explain analyze select max(length(c90)) from t3;	1.614ms	2.346ms
-- COLUMNAR TABLE using EXPRESSION index explain analyze select max(length(c90)) from t4;	31.980ms	38.766ms

Regarding The Relevance of Indexes

Because btree indexes of the same size are applied to both HEAP and columnar table's field, ie "c2" the resultant performance characteristics are basically the same but with the columnar table's index is slightly slower due to the extra processing required to uncompress the table's values.

The differences in performance were distinct when the less commonly used EXPRESSION index was used. Evidently these indexes are less efficient on columnar tables than the more mature HEAP access tables:

```
-- Expression indexes are created
create index on t3(length(c90));
create index on t4(length(c90));

-- example
select max(length(c90)) from t[34];
```

Regarding The Relevance Of Runtime Parameters

Unlike the results shown in the previous table, there were some changes querying a column with an index.

The btree index performance on the columnar table was more consistent than the HEAP table's

```
select sum(c2) from t[34];
```

CitusDB Columnar Tables

About DML Operations & Table Constraints

```
-- RECALL, create a new database and execute the following:

create table if not exists t1(id,qty)
using heap as select (random()*10)::int, (random()*1000)::int from generate_series(1,10e6);

create table if not exists t2(id,qty)
using columnar as select (random()*10)::int, (random()*1000)::int from generate_series(1,10e6);
```

```
-- fails
delete from t2 where id=5;
ERROR: UPDATE and CTID scans not supported for ColumnarScan

-- fails
update t2 set id=5;
ERROR: UPDATE and CTID scans not supported for ColumnarScan
```

```
-- Creating indexes on a columnar table is restricted
--  to btree indexes

-- works
create index on t2 using btree (id);

-- fails
create index on t2 using columnar (id);
ERROR: unsupported access method for the index on columnar table t2
```

Creating foreign key constraints aren't implemented

```
-- create table "t3"
select generate_series as id into t3 from generate_series(0,15);
alter table t3 add primary key(id);

-- works for our standard table t1
alter table t1 add foreign key(id) references t3(id);

-- fails with the columnar table t2
alter table t2 add foreign key(id) references t3(id);
ERROR: Foreign keys and AFTER ROW triggers are not supported for columnar tables
HINT: Consider an AFTER STATEMENT trigger instead.

--works after converting table t1 from COLUMNAR to HEAP
alter table t2 set access method heap;
alter table t2 add foreign key(id) references t3(id);
alter table t2 set access method columnar;
```

CitusDB Columnar Tables

Partitioning

Columnar tables can be used as partitions; a partitioned table can be made up of any combination of row and columnar partitions.

An excellent use case are INSERT once and READ only table partitions where one can leverage both its compression and better performing OLAP type queries for very large tables.

CitusDB Columnar Tables

Caveat, Reasons to use it

CITUS Columnar Extensions used when:

- There are “many” columns
- Space is at a premium
- The typical row is byte “heavy”
- OLAP is a major component of overall activity i.e. lots of different kinds of SELECT statements.
- INSERT performance is not a priority.

TIP, In regards to Partitioning:

- A partitioned table can be made up of any combination of row and columnar partitions.
- An excellent use case are INSERT once and READ only table partitions.
- Leverages its compression over very large tables.

CitusDB Columnar Tables

Caveat, as of version 11.1

Beware

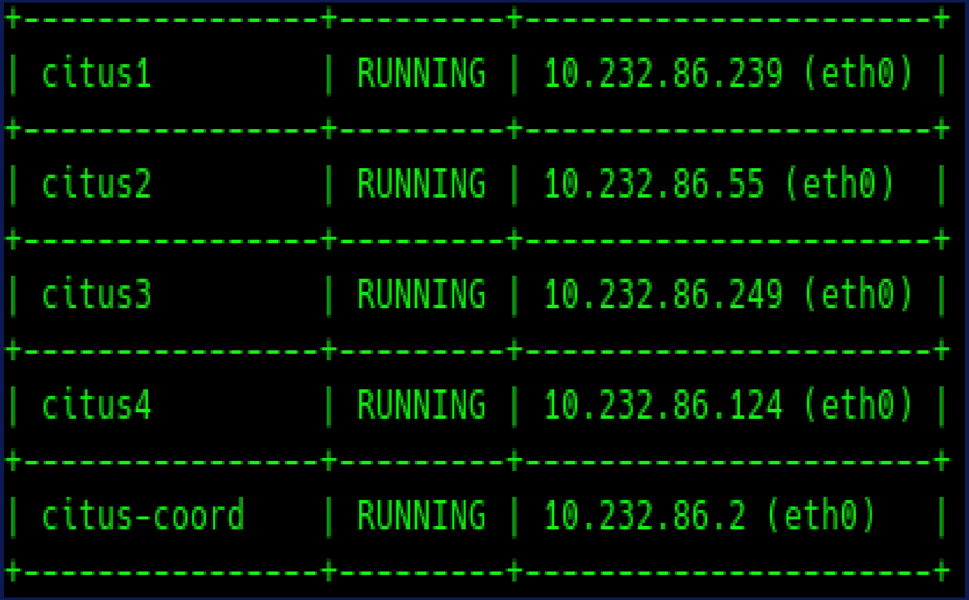
- It can take more time to create the table than standard heap access based tables.
- The query performance is equal or slower with smallish tables compared to heap based tables.
- There is no update/delete possible in a Citus columnar table.
- The indexes are limited to btree.
- There is no logical replication.

REFERENCE:
https://github.com/citusdata/citus/blob/main/src/backend/columnar/README.md

Scaling A CitusDB Cluster

About The Cluster

```
SETUP
- postgres version 16
- citus version 12
- postgresql.conf configured
- pg_hba.conf configured
- postgres password: postgres
- .pgpass already configured for remote logins
```



```
+----------------+---------+----------------------+
| citus1         | RUNNING | 10.232.86.239 (eth0) |
+----------------+---------+----------------------+
| citus2         | RUNNING | 10.232.86.55 (eth0)  |
+----------------+---------+----------------------+
| citus3         | RUNNING | 10.232.86.249 (eth0) |
+----------------+---------+----------------------+
| citus4         | RUNNING | 10.232.86.124 (eth0) |
+----------------+---------+----------------------+
| citus-coord    | RUNNING | 10.232.86.2 (eth0)   |
+----------------+---------+----------------------+
```

Working With Data Redundancy

Working With Data Redundancy

Data Redundancy

- A condition created where the same piece of data is held in multiple locations
- Basically a database version of a RAID

Advantages

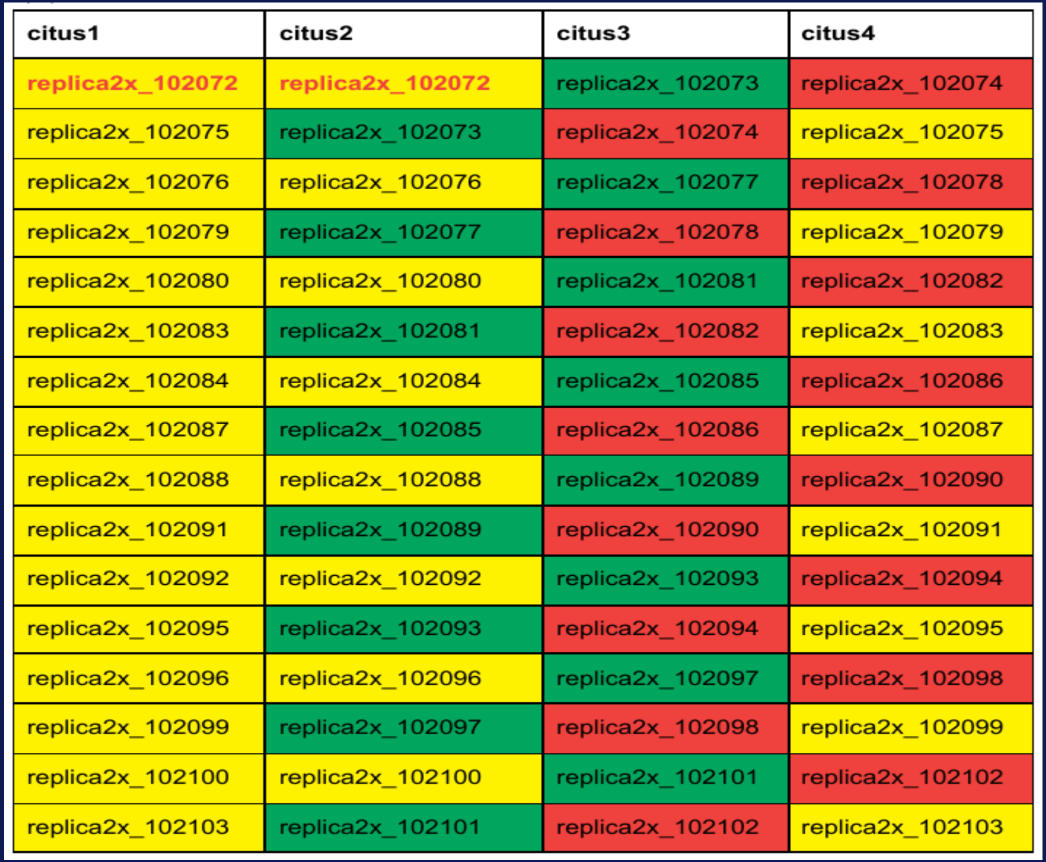
- No Standy hosts
- No fail-overs
- No downtime
- No backups

Disadvantages

- X-times nodes
- Significant disk space requirements
- SQL statement limitations (KISS)

CitusDB Data Redundancy

Data Redundancy Across 4 Nodes



citus1	citus2	citus3	citus4
replica2x_102072	replica2x_102072	replica2x_102073	replica2x_102074
replica2x_102075	replica2x_102073	replica2x_102074	replica2x_102075
replica2x_102076	replica2x_102076	replica2x_102077	replica2x_102078
replica2x_102079	replica2x_102077	replica2x_102078	replica2x_102079
replica2x_102080	replica2x_102080	replica2x_102081	replica2x_102082
replica2x_102083	replica2x_102081	replica2x_102082	replica2x_102083
replica2x_102084	replica2x_102084	replica2x_102085	replica2x_102086
replica2x_102087	replica2x_102085	replica2x_102086	replica2x_102087
replica2x_102088	replica2x_102088	replica2x_102089	replica2x_102090
replica2x_102091	replica2x_102089	replica2x_102090	replica2x_102091
replica2x_102092	replica2x_102092	replica2x_102093	replica2x_102094
replica2x_102095	replica2x_102093	replica2x_102094	replica2x_102095
replica2x_102096	replica2x_102096	replica2x_102097	replica2x_102098
replica2x_102099	replica2x_102097	replica2x_102098	replica2x_102099
replica2x_102100	replica2x_102100	replica2x_102101	replica2x_102102
replica2x_102103	replica2x_102101	replica2x_102102	replica2x_102103

Data Redundancy, a database version of a RAID

Here is an example of a table named replica2x which has 2X redundancy across a cluster of nodes. The colors indicate specific shards of the table that are duplicated.

For example, if node citus1 goes offline, the sharded table it holds still has copies on nodes citus2 and citus4.

Likewise it can be said that if node citus2 goes offline the same data is still available on nodes 1,3,4.

CitusDB Data Redundancy

The CitusDB Cluster Setup

Method

- Step 1: set shard replication factor from 1X to 2X
- Step 2: create table myevents2x with 2X redundancy
- Step 3: identify and locate shards across citus1 and citus2
- Step 4: identify some records to query from shards known to be on nodes citus1 and citus2
- Step 5: test
  - shutdown citus1; perform afore identified query
  - startup citus1, shutdown citus2; perform afore identified query
  - restart citus2; perform afore identified query

You may have to edit your own queries as the resultant values may be different for your particular setup.

CitusDB Data Redundancy

Step 1

```
# Execute as "postgres", in container citus-coord
# lxc exec citus-coord -- su postgres -

for u in citus1 citus2 citus3 citus4 citus-coord
do
    echo "=== $u ==="
    dropdb -h $u -U postgres --if-exists db01
    createdb -h $u -U postgres db01
    psql -h $u -U postgres db01 -c 'create extension citus'
done
```

```
# Update shard replication factor from 1X to 2X

psql -h citus-coord db01 <<_eof_
    select citus_set_coordinator_host('citus-coord', 5432);

-- this INSERT is non-standard but it makes it easy to see the shards
    insert into pg_dist_node(nodename)
    values ('citus1')
          ,('citus2')
          ,('citus3')
          ,('citus4');

    alter system set citus.shard_replication_factor=2;
    select pg_reload_conf();
_eof_
```

```
# validate
psql -h citus-coord db01 <<_eof_
    show citus.shard_replication_factor;
    select nodeid,nodename,groupid,isactive from pg_dist_node order by 1;
_eof_
```

CitusDB Data Redundancy

Step 2

Create a new table with 2X redundancy

```
# execute on host:citus-coord, database db01

psql 'host=citus-coord dbname=db01 user=postgres' <<_eof_

create table myevents2x (
    device_id bigint,
    event_id bigserial,
    event_time timestamptz default now(),
    data jsonb not null,
    primary key (device_id, event_id)
);

-- distribute the events among the nodes
select create_distributed_table('myevents2x', 'device_id');

-- confirm table has been added across the cluster
select * from master_get_active_worker_nodes() order by 1;

-- populate the table
insert into myevents2x (device_id, data)
    select s % 100, ('{"measurement":'||random()||'}')::jsonb
    from generate_series(1,1000000) s;
_eof_
```

CitusDB Data Redundancy

Step 3

```
Locate a shard common on nodes citus1 and citus2
  ex: myevents2x_102020 (numbers can be different)

# execute the following on host citus-coord
for u in citus1 citus2 citus3 citus4
do
    echo "==== $u ===="
    psql -h $u -U postgres db01 -c "select tablename from pg_tables where tablename~'myevents2x' order by 1"
done | less -S
```

CitusDB Data Redundancy

Step 3, cont'd

Citus 1	Citus 2
myevents2x_102008	myevents2x_102008
myevents2x_102011	myevents2x_102009
myevents2x_102012	myevents2x_102012
myevents2x_102015	myevents2x_102013
myevents2x_102016	myevents2x_102016
myevents2x_102019	myevents2x_102017
myevents2x_102020	**myevents2x_102020**
myevents2x_102023	myevents2x_102021
myevents2x_102024	myevents2x_102024
myevents2x_102027	myevents2x_102025
myevents2x_102028	myevents2x_102028
myevents2x_102031	myevents2x_102029
myevents2x_102032	myevents2x_102032
myevents2x_102035	myevents2x_102033
myevents2x_102036	myevents2x_102036
myevents2x_102039	myevents2x_102037

CitusDB Data Redundancy

Step 4

```
Locate and return the first three records
  of a shard located on nodes citus1 and citus2

# execute the following on host citus-coord
for u in citus1 citus2
do
    SHARD="myevents2x_102020"
    SQL="select * from $SHARD order by 1,2 limit 3"
    echo "==== host:$u, shard:$SHARD ===="
    psql -h $u -U postgres db01 -c "$SQL"
done | less -S
```

```
==== host:citus1 shard:myevents2x_102020 ====
 device_id | event_id |          event_time           |                data
-----------+----------+-------------------------------+-----------------------------------
        36 |       36 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.7757022150136692}
        36 |      136 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.521251631712812}
        36 |      236 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.7271407128988034}
(3 rows)

==== host:citus2 shard:myevents2x_102020 ====
 device_id | event_id |          event_time           |                data
-----------+----------+-------------------------------+-----------------------------------
        36 |       36 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.7757022150136692}
        36 |      136 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.521251631712812}
        36 |      236 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.7271407128988034}
(3 rows)
```

CitusDB Data Redundancy

Step 5

```
Redundancy Test: shutdown node citus1

These next few steps demonstrate the ability to continuously query by returning those records found
in shard myevents2x_102020.

# execute on citus1
# lxc exec citus1 -- su -

systemctl stop postgresql@16-pg1


psql -h citus-coord -U postgres db01 <<_eof_
    select * from myevents2x where device_id=36 and event_id in (36,136,236) order by 1,2;
_eof_
```

```
WARNING:  connection to the remote node citus1:5432 failed with the following error: server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.



 device_id | event_id |          event_time           |                data
-----------+----------+-------------------------------+-------------------------------------
        36 |       36 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.7757022150136692}
        36 |      136 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.521251631712812}
        36 |      236 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.7271407128988034}
(3 rows)
```

CitusDB Data Redundancy

Step 5, cont’d

```
Redundancy Test: restart citus1 and shutdown citus2


# execute on citus1
systemctl start postgresql@16-main

# execute on citus2
systemctl stop postgresql@16-main


psql -h citus-coord -U postgres db01 <<_eof_
    select * from myevents2x where device_id=36 and event_id in (36,136,236) order by 1,2;
_eof_
```

```
 device_id | event_id |          event_time           |                data
-----------+----------+-------------------------------+-------------------------------------
        36 |       36 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.7757022150136692}
        36 |      136 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.521251631712812}
        36 |      236 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.7271407128988034}
(3 rows)
```

CitusDB Data Redundancy

Step 5, cont'd

```
Redundancy Test: restart citus2


# execute on citus2
systemctl start postgresql@16-main


psql -h citus-coord -U postgres db01 <<_eof_
    select * from myevents2x where device_id=36 and event_id in (36,136,236) order by 1,2;
_eof_
```

```
 device_id | event_id |          event_time           |                 data
-----------+----------+-------------------------------+-------------------------------------
        36 |       36 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.7757022150136692}
        36 |      136 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.521251631712812}
        36 |      236 | 2024-02-02 18:51:39.689077+00 | {"measurement": 0.7271407128988034}
(3 rows)
```

CitusDB Data Redundancy

Caveat, Issues to Beware

- Limited to performing simple SQL statements
- Current feature risks being deprecated in future releases ...
- Alternate architecture replicating Current feature risks being deprecated in future releases ...

CitusDB Horizontal Scaling

CitusDB Horizontal Scaling

The CitusDB Cluster Setup

Method

- Step 1: one worker node
    - without indexes
    - with indexes
- Step 2: two worker node
- Step 3: three worker node
    - add 3rd node
    - add Foreign Key constraints, ala reference tables

CitusDB Horizontal Scaling

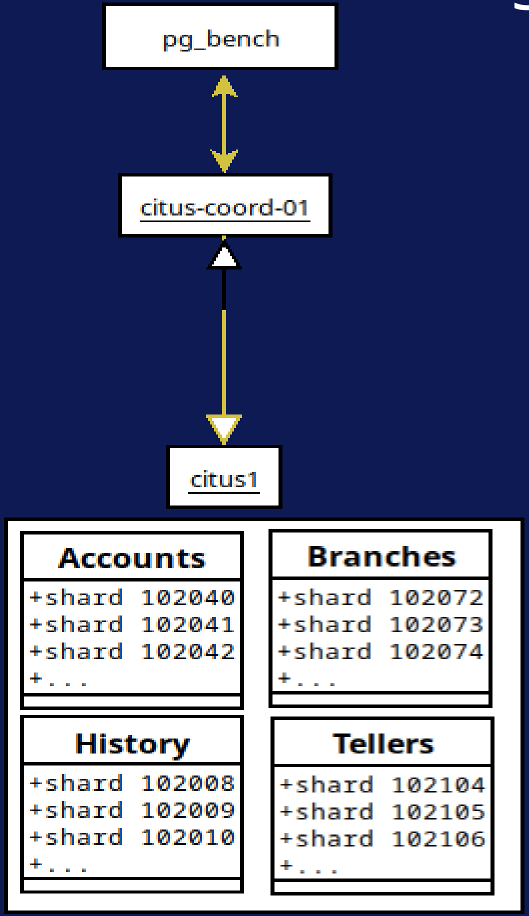
Setup, Clean up

```
# EXECUTE ON COORDINATOR NODE
# lxc exec citus-coord -- su postgres -

# purge the previous scenario
psql -h citus-coord db01 <<_eof_

-- remove the distributed table
    drop table if exists myevents2x;

-- remove all active worker nodes
    select * from citus_remove_node('citus1',5432);
    select * from citus_remove_node('citus2',5432);
    select * from citus_remove_node('citus3',5432);
    select * from citus_remove_node('citus4',5432);

-- validate;there shouldn't be any active worker nodes
    select * from master_get_active_worker_nodes() order by 1;

-- reset the replication factor from 2x to 1x
    alter system set citus.shard_replication_factor=1;
    select pg_reload_conf();
_eof_
```

```
# EXECUTE ON COORDINATOR NODE
# lxc exec citus-coord -- su postgres -

# drop db01
for u in citus-coord citus1 citus2 citus3 citus4
do
    echo "=== $u ===="
    psql "host=$u user=postgres password=postgres dbname=db01"<<_eof_
        drop extension if exists citus cascade;
       \c postgres
        drop database if exists db01;
_eof_
done

# create pgbench database
for u in citus-coord citus1 citus2 citus3 citus4
do
    echo "=== $u ===="
    psql "host=$u user=postgres password=postgres dbname=postgres" <<_eof_
        create database pgbench;
       \c pgbench
        drop extension if exists citus cascade;
        create extension citus;
_eof_
done
```

CitusDB Horizontal Scaling

STEP 1: One Worker Node



```
psql -h citus-coord pgbench <<_eof1_
      select citus_set_coordinator_host('citus-coord', 5432);
      select citus_add_node('citus1', 5432);
_eof1_

# create only the tables
pgbench -h citus-coord -iI t pgbench

psql -h citus-coord pgbench -c “\dt+”

                             List of relations
Schema |       Name        | Type  |  Owner    | Persistence | Access method |  Size   |
-------+-------------------+-------+-----------+-------------+---------------+---------+
public | citus_tables      | view  | postgres  | permanent   |               | 0 bytes |
public | pgbench_accounts  | table | postgres  | permanent   | heap          | 0 bytes |
public | pgbench_branches  | table | postgres  | permanent   | heap          | 0 bytes |
public | pgbench_history   | table | postgres  | permanent   | heap          | 0 bytes |
public | pgbench_tellers   | table | postgres  | permanent   | heap          | 0 bytes |

# distributes the pgbench tables across node citus1
psql -h citus-coord pgbench <<_eof1_
    BEGIN;
        select create_distributed_table('pgbench_history', 'aid');
        select create_distributed_table('pgbench_accounts', 'aid');
        select create_distributed_table('pgbench_branches', 'bid');
        select create_distributed_table('pgbench_tellers', 'tid');
    COMMIT;
_eof1_

# populate the tables
pgbench -h citus-coord -iI g -s 300 pgbench
```

CitusDB Horizontal Scaling

One Worker Node, Adding Indexes

```
# ADD INDEXES
pgbench -h citus-coord -iI p pgbench


# Inspect Tables
for u in citus-coord citus1
do
    echo "============= $u ============="
    psql -h $u -U postgres pgbench <<<"\d pgbench_*"
done | less -S
```

CitusDB Horizontal Scaling

One Worker Node, Benchmarking

A 60 Second benchmark, wo indexes

```
# execute the following on the coordinator node
pgbench -h citus-coord -c 20 -j 3 -T 60 -P 3 pgbench

------------------------------
scaling factor: 300
query mode: simple
number of clients: 20
number of threads: 3
maximum number of tries: 1
duration: 60 s
number of transactions actually processed: 1570
number of failed transactions: 0 (0.000%)
latency average = 764.754 ms
latency stddev = 147.887 ms
initial connection time = 148.497 ms
tps = 26.084725 (without initial connection time)
```

A 60 Second benchmark, with indexes

```
# execute the following on the coordinator node
pgbench -h citus-coord -c 20 -j 3 -T 60 -P 3 pgbench

------------------------------
scaling factor: 300
query mode: simple
number of clients: 20
number of threads: 3
maximum number of tries: 1
duration: 60 s
number of transactions actually processed: 82479
number of failed transactions: 0 (0.000%)
latency average = 14.505 ms
latency stddev = 13.796 ms
initial connection time = 154.991 ms
tps = 1377.323730 (without initial connection time)
```

CitusDB Horizontal Scaling

BEWARE of the Citus extension version

ATTENTION: Adding indexes to versions older than 12 requires a work-around

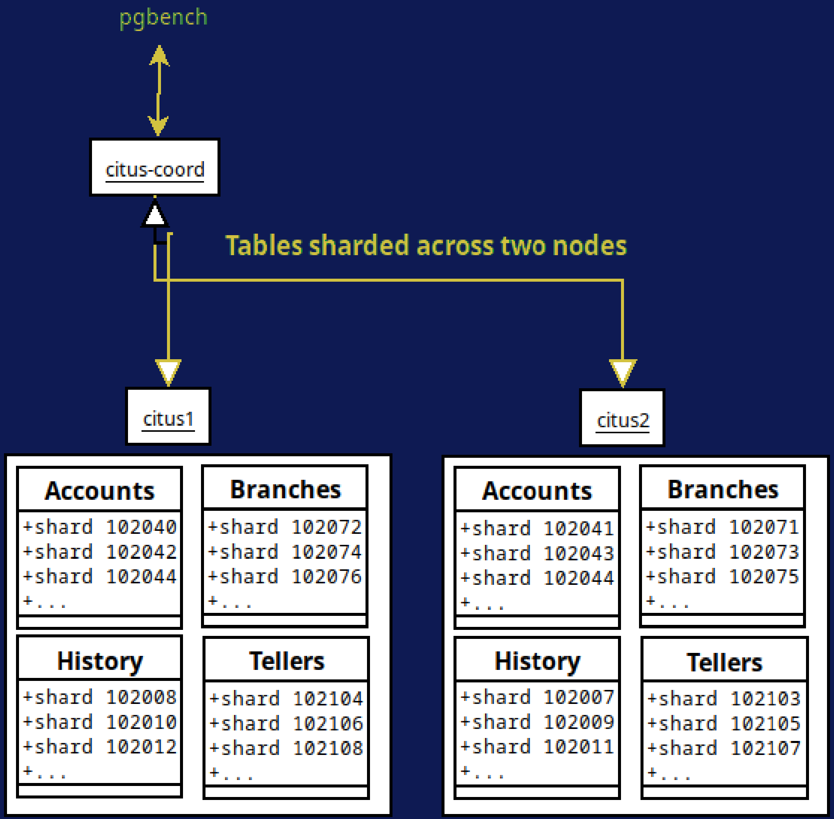
```
pgbench -h citus-coord -iI p pgbench
```

```
creating primary keys...
pgbench: error: query failed: ERROR: cannot create constraint
without a name on a distributed table
pgbench: detail: Query was: alter table pgbench_branches add
primary key (bid)
```

```
# THIS WORKS!
#
# indexes and constraints must be explicitly named
#

psql -h citus-coord pgbench <<_eof1_
BEGIN;

create unique index pgbench_accounts_pk on pgbench_accounts(aid);
create unique index pgbench_branches_pk on pgbench_branches(bid);
create unique index pgbench_tellers_pk on pgbench_tellers(tid);

alter table pgbench_accounts
add constraint pk_accounts primary key using index pgbench_accounts_pk;

alter table pgbench_branches
add constraint pk_branches primary key using index pgbench_branches_pk;

alter table pgbench_tellers
add constraint pk_tellers primary key using index pgbench_tellers_pk;

-- adding REPLICA IDENTITY (no PK present)
alter table pgbench_history replica identity full;

COMMIT;
_eof1_
```

CitusDB Horizontal Scaling

Two Worker Node



```
psql -h citus-coord pgbench <<_eof_

-- ADD REPLICA IDENTITY
alter table public.pgbench_history REPLICA IDENTITY FULL;

-- ADD Node
select citus_add_node('citus2', 5432);

-- REBALANCE shards across two nodes
select * from rebalance_table_shards();

_eof_
```

```
------------------------------------------------------------------
NOTICE:  Moving shard 102009 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102013 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102029 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102034 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102020 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102015 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102011 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102019 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102023 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102038 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102024 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102017 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102028 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102026 from citus1:5432 to citus2:5432 ...
NOTICE:  Moving shard 102022 from citus1:5432 to citus2:5432
------------------------------------------------------------------
```

CitusDB Horizontal Scaling

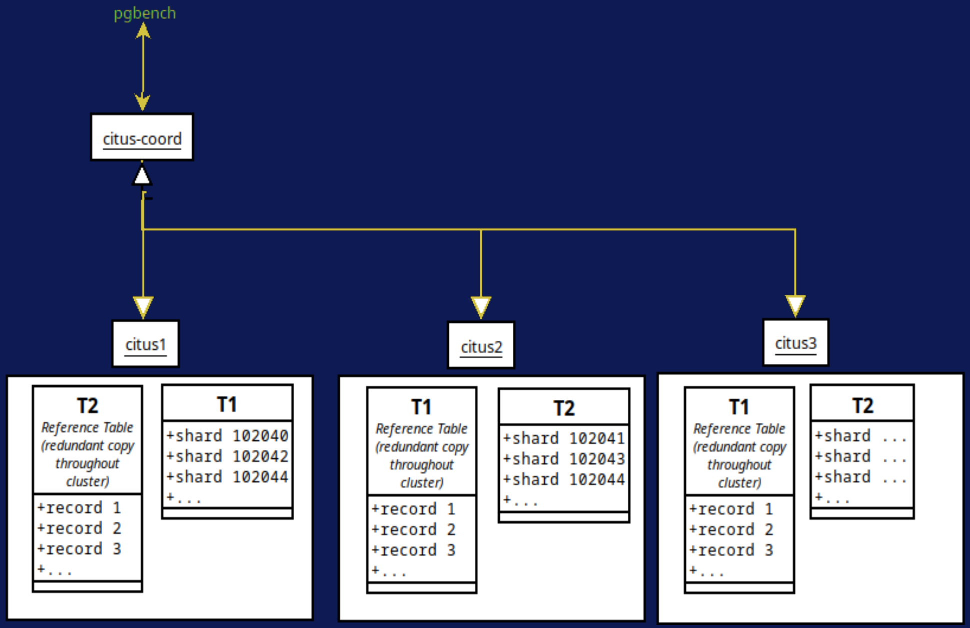
CAVEAT

```
# FAILS
# - If logical replication NOT ENABLED (wal_level=logical)
# - Tables must have PK, UNIQ constraints or REPLICA IDENTITY

psql -h citus-coord pgbench <<_eof_
select * from rebalance_table_shards();
_eof_
```

```
ERROR:  cannot use logical replication to transfer shards of the relation pgbench_history
since it doesn't have a REPLICA IDENTITY or PRIMARY KEY

DETAIL:  UPDATE and DELETE commands on the shard will error out during logical replication
unless there is a REPLICA IDENTITY or PRIMARY KEY.

HINT:  If you wish to continue without a replica identity set the shard_transfer_mode to
'force_logical' or 'block_writes'.
```

CitusDB Horizontal Scaling

Three Worker Node, reference tables



```
Create Reference & Distributed Tables Across Cluster

-- create new tables
create table t1(id serial primary key, comment text);

create table t2 (
    id serial primary key,
    t1_fk int references t1(id) default (random()*1000)::int%4
);

------------------------------------------
select create_reference_table('t1');
select create_distributed_table('t2', 'id');
```

```
Execute On Coordinator Node

-- drop tables
drop table if exists pgbench_accounts,pgbench_branches,
                     pgbench_history,pgbench_tellers;

--------------------------------
-- ADD node
select citus_add_node('citus3', 5432);
```

```
Data Population


insert into t1
values (0,'apple')
      ,(1,'oranges')
      ,(2,'pineapple')
      ,(3,'strawberry');

insert into t2 (select * from generate_series(1,1E6)t1);
```

CitusDB Horizontal Scaling

Foreign Key Constraints (reference tables) cont'd

Validation

```
select shard_name, citus_table_type,
       nodename, shard_size
from citus_shards;


 shard_name | citus_table_type |  nodename   | shard_size
------------+------------------+-------------+------------
 t1_102280  | reference        | citus1      |      32768
 t1_102280  | reference        | citus3      |      32768
 t1_102280  | reference        | citus-coord |      32768
 t1_102280  | reference        | citus2      |      32768
 t2_102282  | distributed      | citus-coord |    1941504
 t2_102283  | distributed      | citus1      |    1933312
 t2_102284  | distributed      | citus2      |    1933312
 t2_102285  | distributed      | citus3      |    1933312
 t2_102286  | distributed      | citus-coord |    1933312
 t2_102287  | distributed      | citus1      |    1933312
 t2_102288  | distributed      | citus2      |    1916928
 t2_102289  | distributed      | citus3      |    1933312
.
.
.
 t2_102306  | distributed      | citus-coord |    1925120
 t2_102307  | distributed      | citus1      |    1933312
 t2_102308  | distributed      | citus2      |    1933312
 t2_102309  | distributed      | citus3      |    1925120
 t2_102310  | distributed      | citus-coord |    1933312
 t2_102311  | distributed      | citus1      |    1925120
 t2_102312  | distributed      | citus2      |    1933312
 t2_102313  | distributed      | citus3      |    1933312
(36 rows)
```

```
select nodeid,nodename,groupid,isactive from
pg_dist_node order by 1;

 nodeid |    nodename     | groupid | isactive
--------+-----------------+---------+----------
      1 | citus-coord     |       0 | t
      2 | citus1          |       1 | t
      3 | citus2          |       2 | t
      4 | citus3          |       3 | t
```

Citus Workshop

```
Conclusion:
- The good: very flexible, grows to fit your needs
- The bad: it’s complicated, you still need REPLICA nodes




References:
    https://www.citusdata.com/faq
    https://docs.citusdata.com/en/stable/
```

THANK YOU!

QUESTIONS?

https://github.com/rbernierZulu/pg_conf_san-jose-2024