**What is HammerDB?**

- It is a database benchmarking tool. It creates a proxy schema to test, loads it with data and simulates the workload of multiple virtual users against the database.
- Documentation Link
- Link to other releases

**Installing on AWS EC2 t3.small (ubuntu Linux distro 20.04LTS)**

- HammerDB Installation

```
$ wget
https://github.com/TPC-Council/HammerDB/releases/download/v4.1/HammerDB-4.1
-Linux.tar.gz
$ tar -zxvf HammerDB-4.1-Linux.tar.gz
$ cd HammerDB-4.1
$ ./hammerdbcli
```

- Postgresql Installation

```
$ sudo apt install postgresql
$ sudo apt install libpq-dev
```

NOTE

- xyzz

**Installing on WSL 2 (LINK)**

- HammerDB Installation on WSL: (installing 4.1 release)

```
$ wget
https://github.com/TPC-Council/HammerDB/releases/download/v4.1/HammerDB-4.1
-Linux.tar.gz
$ tar -zxvf HammerDB-4.1-Linux.tar.gz
```

- Installing Postgresql on WSL:

```
$ sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt
$(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'

$ wget --quiet -O -
https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add
```

```
-

$ sudo apt-get update

$ sudo apt-get -y install postgresql postgresql-contrib

$ psql --version
psql (PostgreSQL) 14.0 (Ubuntu 14.0-1.pgdg20.04+1)

$ sudo service postgresql status
14/main (port 5432): down

$ sudo service postgresql start
 * Starting PostgreSQL 14 database server                    [OK]
```

NOTE

- Take a note of port no your server is running, default value is 5432 but some times it might run on port 5433. This will be used later to connect to the server.

**Running HammerDB**

After installing and configuring hammerdb and PostgreSQL. Now we test the database.
Step 1: setting up superuser

```
# login as postgres user
aayush@LAPTOP-D1NVAIOH:~$ sudo su postgres
# now login to postgresql cli
postgres@LAPTOP-D1NVAIOH:/home/aayush$ psql
psql (14.5 (Ubuntu 14.5-1.pgdg20.04+1))
Type "help" for help.

postgres=# create database db_test;
CREATE DATABASE
postgres=# create user user_name with encrypted password
'user_password';
CREATE ROLE
postgres=# alter role user_name with superuser createdb createrole
replication bypassrls login;
```

```
ALTER ROLE
```

Step 2: create build file

```
$ cd HammerDB-4.1/
$ touch pgbuild.tcl
$ vim pgbuild.tcl
```

```
dbset db pg
diset connection pg_host localhost
diset connection pg_port 5432
diset tpcc pg_count_ware 5
diset tpcc pg_num_vu 5
diset tpcc pg_superuser aayush
diset tpcc pg_superuserpass pass11
diset tpcc pg_storedprocs false
vuset logtotemp 1
vuset unique 1
Buildschema
~
~
:wq
```

```
$
```

Step 3: creating test file

```
$ cd HammerDB-4.1/
$ touch pgtest.tcl
$ vim pgtest.tcl
```

```
puts "setting config"
dbset db pg
diset connection pg_host localhost
diset connection pg_port 5432
diset tpcc pg_superuser aayush
```

```
diset tpcc pg_superuserpass pass11
diset tpcc pg_storedprocs false
diset tpcc pg_vacuum true
diset tpcc pg_driver timed
diset tpcc pg_rampup 2
diset tpcc pg_duration 2
vuset logtotemp 1
vuset unique 1
tcset logtotemp 1
tcset timestamps 1
loadscript
puts "TEST STARTED"
vuset vu 5
vucreate
tcstart
tcstatus
vurun
puts "RUNTIMER STARTED"
runtimer 300
print dict
vudestroy
tcstop
puts "TEST COMPLETE"
~
~
~
:wq
```

```
$
```

NOTE

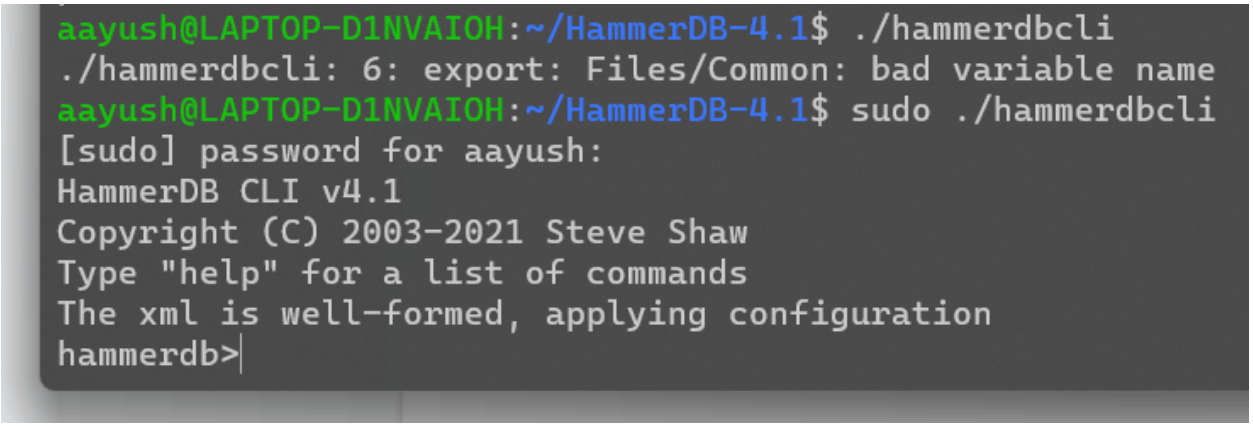
- Think of number of warehouses as size of your database.
- Number of virtual users denote number of concurrent users interacting with the database.

Step 4: creating tpcc schema (running your build file)

```
$ cd HammerDB-4.1/
$ ./hammerdbcli
```

NOTE

- executing ./hammerdbcli may sometimes throw an error saying bad variable name if so then run it as sudo ./hammerdbcli. (high possibility for it to happen on wsl.)



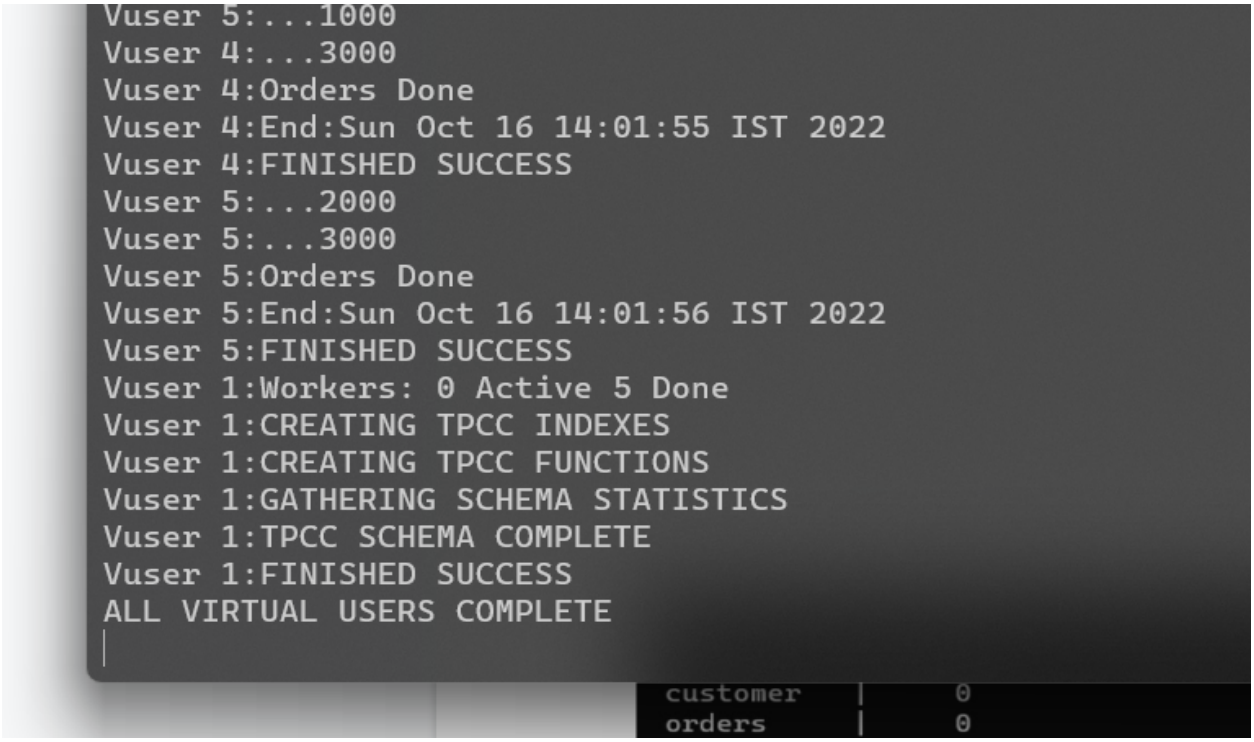

Once you are in hammerdb cli continue with the execution of pgbuld.tcl file.

hammer> source ‘build file location’. I’ve created the pgbuil.tcl file in the same directory.

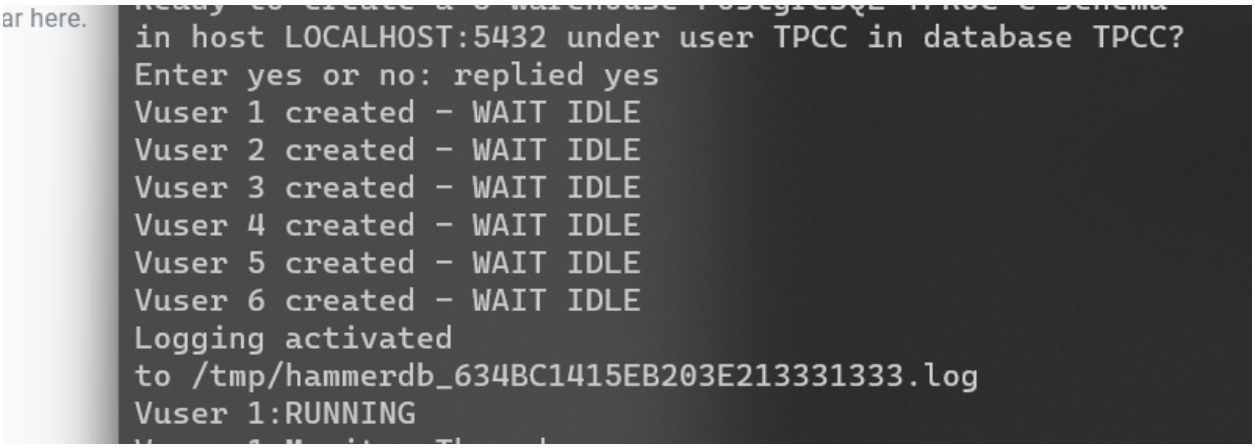
```
hammer> source pgbuild.tcl
```

This will start building the tpcc schema as per your tcl file instructions.

Give it some time to finish execution. Once done it’ll promt something like this.




You can press enter and exit from the CLI. we can check the log file and proxy table that hammerdb created.

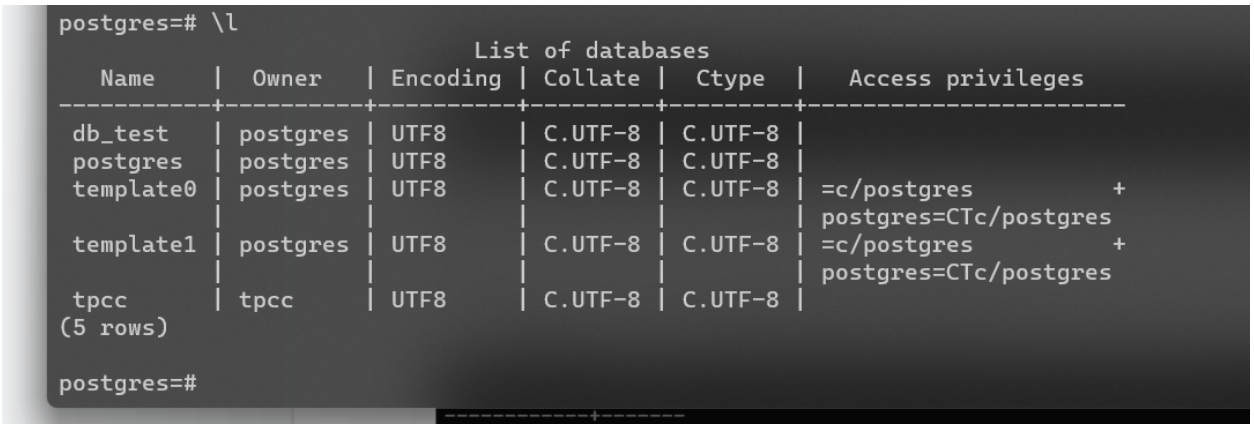



We can see as the test starts it creates a log file and it is in /tmp folder with name hammerdb_xxxx.log.

Let’s check the schema built by hammerdb.
From your home directory

```
$ sudo su postgres
$ psql
postgres=# \l
```

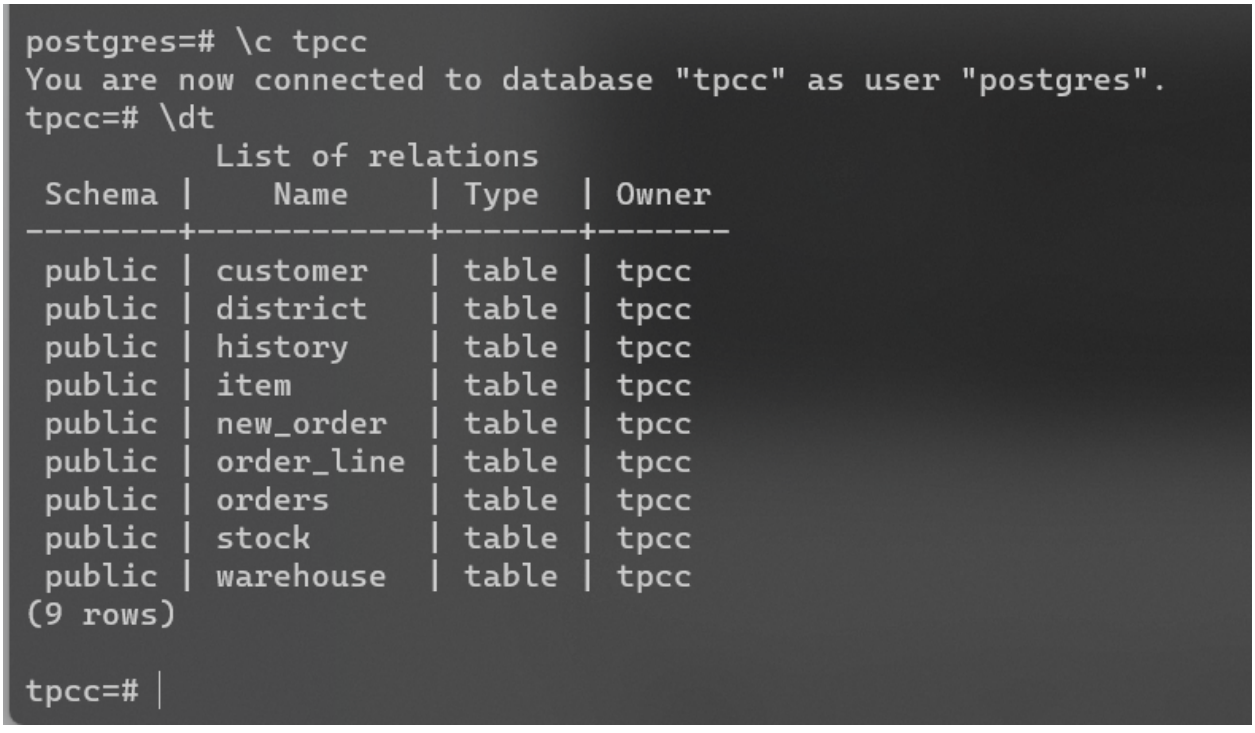
Above command will list the databases.



```
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding | Collate |  Ctype  |   Access privileges
-----------+----------+----------+---------+---------+-----------------------
 db_test   | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
 postgres  | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
 template0 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
           |          |          |         |         | postgres=CTc/postgres
 template1 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
           |          |          |         |         | postgres=CTc/postgres
 tpcc      | tpcc     | UTF8     | C.UTF-8 | C.UTF-8 |
(5 rows)

postgres=#
```

We can see a ‘tpcc’ database is created. Let’s connect to tpcc and check the relations in it.



```
postgres=# \c tpcc
You are now connected to database "tpcc" as user "postgres".
tpcc=# \dt
          List of relations
 Schema |    Name    | Type  | Owner
--------+------------+-------+-------
 public | customer   | table | tpcc
 public | district   | table | tpcc
 public | history    | table | tpcc
 public | item       | table | tpcc
 public | new_order  | table | tpcc
 public | order_line | table | tpcc
 public | orders     | table | tpcc
 public | stock      | table | tpcc
 public | warehouse  | table | tpcc
(9 rows)

tpcc=#
```

We will study the relation later in upcoming section.

NOTE

- Remember **'sudo ./hammerdbcli auto pgbuild.tcl'** can result in incomplete schema.

```
tpcc=# select relname, n_tup_ins as row from pg_stat_user_tables;
  relname   |  row
------------+-------
 customer   |     0
 orders     |     0
 warehouse  |     1
 stock      |  3882
 order_line |     0
 district   |     0
 history    |     0
 new_order  |     0
 item       | 30015
(9 rows)
```

Step 5: Now let's test the schema.
Login to hammerdbcli

```
$ cd HammerDB-4.1/
$ ./hammerdbcli
```

```
hammer> source pgtest.tcl
```

Once the test is complete you can scroll up to check the NOPM(number of orders per minute) and TPM(Transaction per minute) for your database these parameter help check how well your database perform.

```
Vuser 1:2 ...,
Vuser 1:Test complete, Taking end Transaction Count.
Vuser 1:5 Active Virtual Users configured
Vuser 1:TEST RESULT : System achieved 12439 NOPM from 28625 PostgreSQL TPM
Vuser 1:Checkpoint and Vacuum
Vuser 5:FINISHED SUCCESS
35340 PostgreSQL tpm
Timer: 4 minutes elapsed
```

This can be further tuned to get better performance. You can further check the logs and count log in /tmp directory. The name of the file is hdbtcount.log.

```
Error: Virtual Users exist, destroy with vudestroy before creating
Transaction Counter logging activated to /tmp/hdbtcount.log
Transaction Counter Started
```

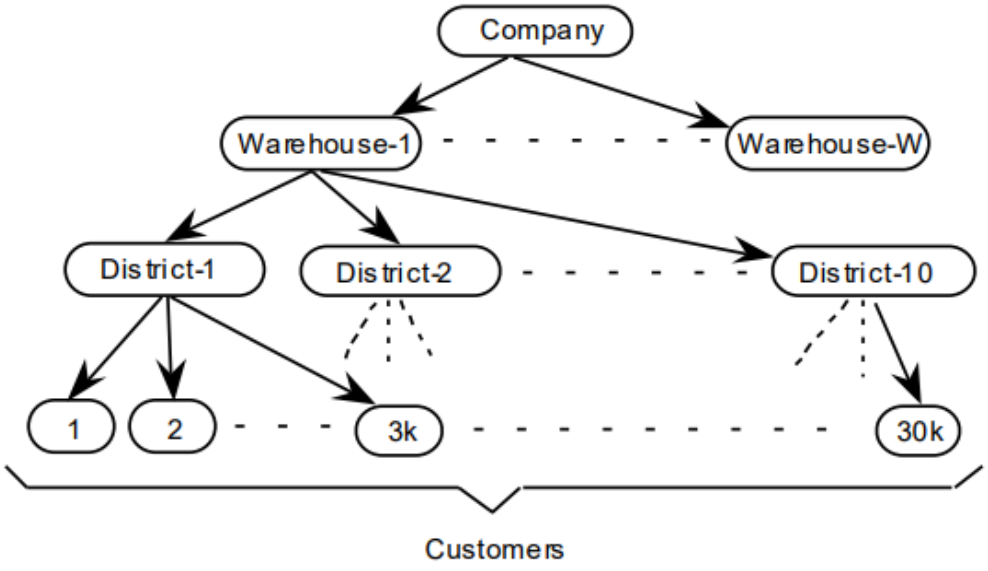
NOTE

- You can use 'sudo ./hammerdbcli auto pgtest.tcl' to run the test file without loggin to hammerdb CLI.

**HammerDB schema**

Let's try to understand the schema that hammerdb builds.

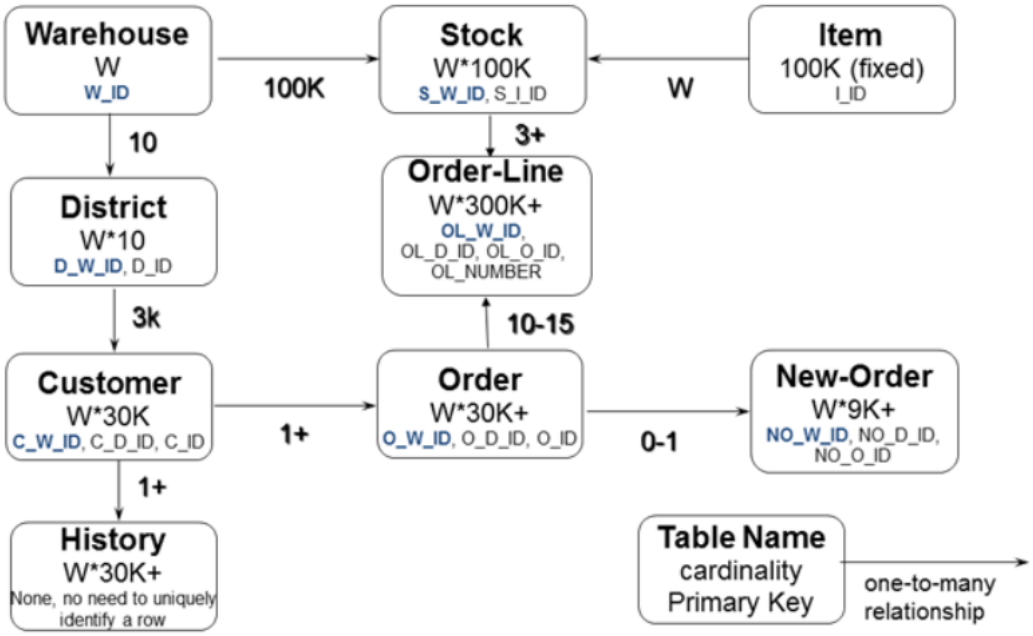
The Company portrayed by the benchmark is a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. As the Company's business expands, new warehouses and associated sales districts are created. Each regional warehouse covers 10 districts. Each district serves 3,000 customers. All warehouses maintain stocks for the 100,000 items sold by the Company. The following diagram illustrates the warehouse, district, and customer hierarchy of TPC-C's business environment.




Customers call the Company to place a new order or request the status of an existing order. Orders are composed of an average of 10 order lines (i.e., line items). One percent of all order lines are for items not in-stock at the regional warehouse and must be supplied by another warehouse.

The best way to illustrate this is in an ER diagram which is depicted as follows with cardinality and relation between the tables mentioned.

Figure 3.2. TPROC-C Schema




## SQL queries

Let's run some queries to better understand the schema relation.

1. **Customers with more than 100 orders.**
    - select c_first, c_middle, c_last from customer where c_id in (select o_c_id from orders group by o_c_id having count(o_c_id)>100) limit 10;
2. **Which warehouse has the highest number of new orders.**
    - select w_id, w_name from warehouse where w_id = (select no_w_id from new_order group by no_w_id order by count(no_w_id) desc limit 1);
3. **Which district has the highest demand under the warehouse with the highest number of new orders?**
    -
4. **Warehouses that have item prices higher than x.**
    - select w_id, w_name from warehouse where w_id in (select s_w_id from stock join item on s_i_id=i_id where i_price>99);

5. **List the item stock in increasing order of quantity.**
    - 
6. **customer with the highest demand in each district.**
    - 
7. **List Items that are out of stock in order of increasing demand.**
    - 
8. **Which city has the highest customer-to-order ratio?**
    - 
9. 

**Tuning**

- Read the following articles to understand what tuning is and why we do it.
  - Medium article 1 (memory parameters)
  - Medium article 2 (kernel parameters)
  - Medium article 3 (autovaccum)
  - Intel documentation
- check the excel file to learn about the kernel and postgres parameters.

Following are the steps to change postgres.conf file. Locate the postgres.conf file and edit the value of the parameters.

```
$ cd /etc/postgresql/14/main
$ sudo vim postgresql.conf
```

I am using rd5.2xlarge (8 CPUs 64GB memory) on AWS to run this test.

```
autovaccum=on
effective_cache_data=16GB
wal_buffers=-1 (-1 sets wal_buffers based on shared_buffers)
maintenance_work_mem=128MB
work_mem=2000MB
temp_buffer=2000MB
huge_pages=try
shared_buffers=8000MB
max_connections=256
synchronous_commit=off
```

I've changed the above parameters and following are the results for 5 virtual users and 5 warehouses.

| | NOPM | TPM |
|---|---|---|
| Without tuning | 63525 | 146028 |
| After tuning postgres.conf | 154447 | 355599 |

From the above table we can check the database performed approx 2.5 times better.

To further improve the performance we can set kernel parameters.

Uses the following command to set the values of different kernel parameters.

```
$ sudo sysctl -w parameter=value
```

```
Hugepages = 1GB
vm.dirty_background_ratio=5
vm.dirty_ratio =20
vm.dirty_background_bytes=0
vm.swappiness=1
vm.overcommit_memory=1
```

NOTE

- Remember to reboot once you have made all the changes.

**Python Execution**

We will use flask so let's download it.

```
$ pip install flask
```

We would also need to download psycopg2 to connect to your server. If you wanna know more about it visit here.

```
$ pip install psycopg2-binary
```

You can download it globally or in an environment.

Let's make an app directory.

```
$ mkdir flaskapp
$ cd flaskapp
$ mkdir templates
```

NOTE

- Make sure to add the templates directory cuz when you run the app it looks into the templates folder.

We will add all the index.html in this folder.

Now let's write some python scripts. In the flask app let's create main.py

main.py

```
from flask import Flask, render_template
import psycopg2

app = Flask(__name__)

def get_connection():
    connection = psycopg2.connect(
        host="localhost",
        port=5432,
        database="tpcc",
        user="aayush",
        password="pass11"
    )
    return connection

@app.route('/')
def index():
    connect = get_connection()
    cur = connect.cursor()
    cur.execute('select c_first from customer where c_id in (select o_c_id from
orders group by o_c_id having count(o_c_id)>100) limit 10;')
    orders = cur.fetchall()
    cur.close()
    connect.close()
    return render_template('index.html', orders=orders)

if __name__ == '__main__':
    app.run(debug=True)
```

Now let's code index.html and base.html. Base.html contains the basic html layout for the website and index.html inherits properties from base.html

base.html

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %} {% endblock %}</title>
</head>
<body>
    <div class="content">
        {% block content %} {% endblock %}
    </div>
</body>
</html>
```

index.html

```
{% extends 'base.html' %}
{% block title %} Orders {% endblock %}
{% block content %}
{%for order in orders%}
    <div>{{order[0]}}</div>
{% endfor %}
{% endblock %}
```

Now this is the index file that will be served as mentioned in our python script.

Now before we run the script let’s set the variables.

```
$ export FLASK_APP=main
```

Set this value to your script so flask knows which script to execute.

Next, let’s set the developer environment.

```
$ export FLASK_ENV=development
```

Now we are set to run it.

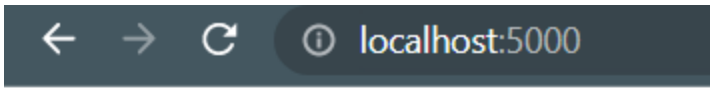
In the flask app directory do

```
$ flask run
```

Once you run it you'll see something like this.

```
aayush@LAPTOP-D1NVAIOH:~/Projects/flaskapp$ flask run
 * Serving Flask app 'routes'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Your flask server is running at localhost:5000 you can see it is serving index.html by visiting localhost:5000.




mLRHwVfAw9g
SRvPHE1YqDjV
JbbOnL7qOVo6ht1
QdiNbnraQNA5Bd3
DoZnI2J8GSjgSS
sDddhEpSWQ7bbn6
q8KvCVlBQWNf
Yg54AwQUlx44
3ahC6VkkSZ
ELzV0NfjyNdJKfC

You can see your results here.

NOTE

- You might come across an error something like this

```
Traceback (most recent call last):
  File "/home/aayush/.local/lib/python3.8/site-packages/flask/app.py", line 2525, in wsgi_app
    response = self.full_dispatch_request()
  File "/home/aayush/.local/lib/python3.8/site-packages/flask/app.py", line 1822, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/home/aayush/.local/lib/python3.8/site-packages/flask/app.py", line 1820, in full_dispatch_request
    rv = self.dispatch_request()
  File "/home/aayush/.local/lib/python3.8/site-packages/flask/app.py", line 1796, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
  File "/home/aayush/Projects/flaskapp/routes.py", line 19, in index
    connect = get_connection()
  File "/home/aayush/Projects/flaskapp/routes.py", line 8, in get_connection
    connection = psycopg2.connect(
  File "/home/aayush/.local/lib/python3.8/site-packages/psycopg2/__init__.py", line 122, in connect
    conn = _connect(dsn, connection_factory=connection_factory, **kwasync)
psycopg2.OperationalError: connection to server at "localhost" (127.0.0.1), port 5432 failed: Connection refused
        Is the server running on that host and accepting TCP/IP connections?

127.0.0.1 - - [17/Oct/2022 21:40:35] "GET / HTTP/1.1" 500 -
127.0.0.1 - - [17/Oct/2022 21:40:35] "GET /favicon.ico HTTP/1.1" 404 -
```

Make sure postgres is running at port 5432. You can start it using

```
$ sudo service postgresql start
```