CSE 512- Distributed Database Systems

Group Project: Cassandrian

Part-2: Data fragmentations and Replication

1. Data fragmentation:

All the fragmentation implementation is done in the ‘fragmentation.py’ file. It consists of two significant functions which are as follows:

* Range\_partition()
* List\_partition()
  1. List Fragmentation
* Match\_Region table is partitioned based on countries of venues. We selected India, England, and Australia as our 3 partitions among various other choices.
* These 3 regions were selected as most of the matches will be played in these 3 countries.
* Code Snippet of implementing list partitioning is as follows:

A black screen with white text

Description automatically generated

* Snippet of partition tables created (in pgAdmin tool):

A screenshot of a computer

Description automatically generated

* 1. Partition by range:

Bowl\_by\_bowl table is partitioned, and range partition is set on the ‘runs\_by\_batter’.

Two ranges are as follows:

* Runs\_below\_3: this partition indicates singles, doubles or 3 runs scored by batsman and does not include any extras.
* Runs\_boundaries: this partition included all the boundaries scored by the batsman and does not include any extras.

Code Snippet of implementing range partitioning is as follows:

A black background with white text

Description automatically generated

* Snippet of partition tables created (in pgAdmin tool):

A screenshot of a computer

Description automatically generated

Data Replication:

We set up data replication using master-slave operation, where ‘Cassandrian’ database is the master DB and Postgres is the slave database.

Master DB has both read/write access and Slava DB has only read access.

We will be using Docker container to set up the replication process.

All the configuration rules are implemented in the ‘docker-compose.yaml’ file.

* Snippet of the configuration in ‘docker-compose’ file:

A screenshot of a computer screen

Description automatically generated

* ‘master\_slave\_rep.py’ : we replicate data from master DB to slave DB using this python file. In this file, we setup 2 docker containers i.e. master container and slave container and provide configuration file consisting of user credentials, replication rules among other information.

A black screen with white text

Description automatically generated

Data replication (Approach 2: Peer to Peer network)

In this form of data replication, we created a common network over which different instances would interact with each other.

In this, both the instances will have read and write authority.

Peer to peer network minimizes the issue of single-point-of-failure, which is common with master-slave replication.

Following steps were done with implementing peer to peer replication:

1. A network was set up through which all instances can interact with each other.



1. We set up two postgres instances i.e., postgres-1 and postgres-2 as two instances in peer-to-peer replication model.

A screenshot of a computer

Description automatically generated

Both the instances will have their own configuration files and can be customized individually as per the requirements. These configuration files are as follows:

* Pd\_ident.config file- Function of this file is to map OS users to database user accounts. All the code is commented on this file as we did not used it in our replication.
* Pg\_hba.conf file- It stands host-based authentication file. This file will tell postgresql.config file which IP address and users can be trusted and also controls the way user logs into postgres
* Snippet of the pg\_hba config file is as follows:

A screenshot of a computer

Description automatically generated

* Postgresql.config file – this is our main file which contains all the configuration data. This file also includes hba and ident files. There are numerous other settings that we can customize based on the requirements.
  + Snippet of the postgresql.config files is as follows:

A screen shot of a computer

Description automatically generated

After setting up the configuration files, we start our first instance using the command:

A screen shot of a computer

Description automatically generated

Server can be started to enable communication between multiple instances as follows:

Server starting:

A screenshot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

2. To persist data to PostgreSQL, we simply mount a docker volume.

A black screen with white text

Description automatically generated

WAL(Write Ahead Logs)

It is one of the important aspects of data replication process.

PostgreSQL has a mechanism of writing transaction logs to file and does not accept the transaction until it’s been written to the transaction log and flushed to disk. This ensures that if there is a crash in the system, that the database can be recovered from the transaction log. Hence it is "writing ahead".

Overview of this model is as follows:

A screen shot of a computer screen

Description automatically generated