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()

a. List Fragmentation

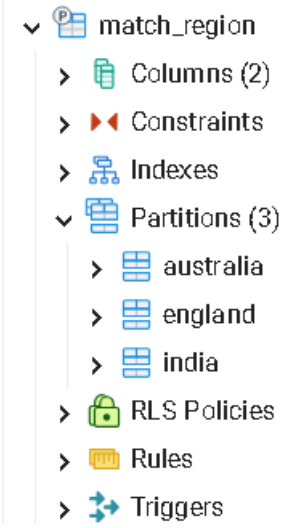
- Match_Region table is partitioned based on countries of venues. We selected India, England, and Australia as our 3 partitions among various other choices.
- These 3 regions were selected as most of the matches will be played in these 3 countries.
- Code Snippet of implementing list partitioning is as follows:

```
def list_partition(curs):
    create_partition_table= "CREATE TABLE IF NOT EXISTS match_region (venue_id serial , country text) PARTITION BY LIST (country);"

    curs.execute(create_partition_table)
    conn.commit()

    partition_queries = [
                "CREATE TABLE IF NOT EXISTS INDIA PARTITION OF match_region FOR VALUES IN ('India');",
                "CREATE TABLE IF NOT EXISTS ENGLAND PARTITION OF match_region FOR VALUES IN ('England');",
                "CREATE TABLE IF NOT EXISTS AUSTRALIA PARTITION OF match_region FOR VALUES IN ('Australia');"
            ]
```

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



b. 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:

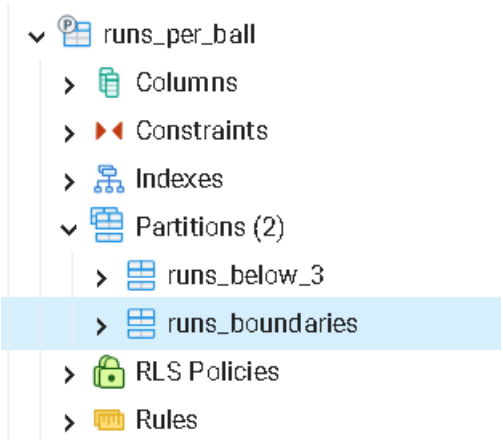
```
def range_partition(curs):

    table = f"CREATE TABLE IF NOT EXISTS runs_per_ball (batsman_id int, bowl_no int, runs_by_batter int) partition by range(runs_by_batter);"
    curs.execute(table)
    conn.commit()

    curs.execute(f"CREATE TABLE IF NOT EXISTS runs_below_3 PARTITION OF runs_per_ball FOR VALUES FROM ('1') TO ('3'); ")
    conn.commit()

    curs.execute(f"CREATE TABLE IF NOT EXISTS runs_boundaries PARTITION OF runs_per_ball FOR VALUES FROM ('4') TO ('6'); ")
    conn.commit()
```

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



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:

```yaml
services:
  postgres-master:
    image: postgres:latest
    container_name: postgres-master
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: Happyplace11*
      POSTGRES_DB: cassandrians
    ports:
      - "5432:5432"
    volumes:
      - ./master_data:/var/lib/postgresql/data
    networks:
      - postgres-network

  postgres-slave:
    image: postgres:latest
    container_name: postgres-slave
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: Happyplace11*
      POSTGRES_DB: postgres
      POSTGRES_MASTER_HOST: postgres-master
      POSTGRES_MASTER_PORT: 5432
      POSTGRES_MASTER_USER: postgres
      POSTGRES_MASTER_PASSWORD: Happyplace11*
    volumes:
      - ./slave_data:/var/lib/postgresql/data
    networks:
      - postgres-network

networks:
  postgres-network:
    driver: bridge
```

- ‘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.

```
(base) PS C:\Users\shreyanshraj\Desktop\CSE 512- DDS\Group project files\Cassandrians\Part_2> python master_slave_rep.py
 [+] Building 0.0s (0/0)                                                                                    docker:default
 [+] Running 3/3
  ✓ Network part_2_postgres-network  Created                                                                        0.1s
  ✓ Container postgres-master        Started                                                                        0.3s
  ✓ Container postgres-slave         Started                                                                        0.3s
PostgreSQL master-slave replication set up successfully!
 [+] Running 3/3
  ✓ Container postgres-slave         Removed                                                                        0.7s
  ✓ Container postgres-master        Removed                                                                        0.9s
  ✓ Network part_2_postgres-network  Removed                                                                        0.3s
(base) PS C:\Users\shreyanshraj\Desktop\CSE 512- DDS\Group project files\Cassandrians\Part_2>
```

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.

```
(base) PS C:\Users\shreyanshraj\Desktop\CSE 512- DDS\Group project files\Cassandrians\Part_1\part2> docker network create postgres1
77afa7efbc4bf4e8dff540fc629e55aace35bc3435edc0d70ff9e866fb779ceb
```

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

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:

postgres-1 > config > pg_hba.conf

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

host     replication     replicationUser         0.0.0.0/0        md5

# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
# IPv6 local connections:
host    all             all             ::1/128                 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust

host all all all scram-sha-256
```

- 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:

```
data_directory = '/data'
hba_file = 'part2/postgres-1/config/pg_hba.conf'
ident_file = 'part2/postgres-1/config/pg_ident.conf'

port = 5432
listen_addresses = '*'
max_connections = 100
shared_buffers = 128MB
dynamic_shared_memory_type = posix
max_wal_size = 1GB
min_wal_size = 80MB
log_timezone = 'Etc/UTC'
datestyle = 'iso, mdy'
timezone = 'Etc/UTC'

#locale settings
lc_messages = 'en_US.utf8'          # locale for system error message
lc_monetary = 'en_US.utf8'          # locale for monetary formatting
lc_numeric = 'en_US.utf8'           # locale for number formatting
lc_time = 'en_US.utf8'              # locale for time formatting

default_text_search_config = 'pg_catalog.english'

#replication
wal_level = replica
archive_mode = on
archive_command = 'test ! -f /mnt/server/archive/%f && cp %p /mnt/server/archive/%f'
max_wal_senders = 3
```

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

```
(base) PS C:\Users\shreyanshraj\Desktop\CSE 512- DDS\Group project files\Cassandrians\Part_2> docker run -it --rm --name cassandrians `
>> --net postgres `
>> -e POSTGRES_USER=postgresadmin `
>> -e POSTGRES_PASSW ORD=admin123 `
>> -e POSTGRES_DB=postgresdb `
>> -e PGDATA="/data" `
>> -v ${PW D}/postgres-1/pgdata:/data `
>> -v ${PW D}/postgres-1/config:/config `
>> -v ${PW D}/postgres-1/archive:/mnt/server/archive `
>> -p 5000:5432 `
>> postgres:15.0 -c 'config_file=postgres-1\config\postgresql.conf'
```

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

Server starting:

```
(base) PS C:\Users\shreyanshraj\Desktop\CSE 512- DDS\Group project files\Cassandrians\Part_1> docker run -it --rm --name cassandrians `
>>   -e POSTGRES_PASSWORD=admin123 `
>>   -v ${PWD}/pgdata:/var/lib/postgresql/data `
>>   postgres:15.0
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.utf8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory /var/lib/postgresql/data ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Etc/UTC
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
initdb: hint: You can change this by editing pg_hba.conf or using the option -A, or --auth-local and --auth-host, the next time you ru
n initdb.

Success. You can now start the database server using:

    pg_ctl -D /var/lib/postgresql/data -l logfile start

waiting for server to start....2023-11-24 09:28:38.149 UTC [49] LOG:  starting PostgreSQL 15.0 (Debian 15.0-1.pgdg110+1) on x86_64-pc-
linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
2023-11-24 09:28:38.157 UTC [49] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2023-11-24 09:28:38.226 UTC [52] LOG:  database system was shut down at 2023-11-24 09:28:35 UTC
2023-11-24 09:28:38.278 UTC [49] LOG:  database system is ready to accept connections
 done
server started

/usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*

waiting for server to shut down...2023-11-24 09:28:38.425 UTC [49] LOG:  received fast shutdown request
.2023-11-24 09:28:38.433 UTC [49] LOG:  aborting any active transactions
2023-11-24 09:28:38.436 UTC [49] LOG:  background worker "logical replication launcher" (PID 55) exited with exit code 1
```

```
waiting for server to start....2023-11-24 09:28:38.149 UTC [49] LOG:  starting PostgreSQL 15.0 (Debian 15.0-1.pgdg110+1) on x86_64-pc-
linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
2023-11-24 09:28:38.157 UTC [49] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2023-11-24 09:28:38.226 UTC [52] LOG:  database system was shut down at 2023-11-24 09:28:35 UTC
2023-11-24 09:28:38.278 UTC [49] LOG:  database system is ready to accept connections
 done
server started

/usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*

waiting for server to shut down...2023-11-24 09:28:38.425 UTC [49] LOG:  received fast shutdown request
.2023-11-24 09:28:38.433 UTC [49] LOG:  aborting any active transactions
2023-11-24 09:28:38.436 UTC [49] LOG:  background worker "logical replication launcher" (PID 55) exited with exit code 1
2023-11-24 09:28:38.443 UTC [50] LOG:  shutting down
2023-11-24 09:28:38.448 UTC [50] LOG:  checkpoint starting: shutdown immediate
2023-11-24 09:28:38.548 UTC [50] LOG:  checkpoint complete: wrote 3 buffers (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=
0.031 s, sync=0.007 s, total=0.105 s; sync files=2, longest=0.004 s, average=0.004 s; distance=0 kB, estimate=0 kB
2023-11-24 09:28:38.563 UTC [49] LOG:  database system is shut down
 done
server stopped

PostgreSQL init process complete; ready for start up.

2023-11-24 09:28:38.701 UTC [1] LOG:  starting PostgreSQL 15.0 (Debian 15.0-1.pgdg110+1) on x86_64-pc-linux-gnu, compiled by gcc (Debi
an 10.2.1-6) 10.2.1 20210110, 64-bit
2023-11-24 09:28:38.702 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2023-11-24 09:28:38.702 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2023-11-24 09:28:38.716 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2023-11-24 09:28:38.780 UTC [62] LOG:  database system was shut down at 2023-11-24 09:28:38 UTC
2023-11-24 09:28:38.862 UTC [1] LOG:  database system is ready to accept connections
```

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

```
(base) PS C:\Users\shreyanshraj\Desktop\CSE 512- DDS\Group project files\Cassandrians\part_1\part2> docker run -it --rm --name postgres `
>>   -e POSTGRES_PASSWORD=admin123 `
>>   -v ${PWD}/pgdata:/var/lib/postgresql/data `
>>   postgres:15.0

PostgreSQL Database directory appears to contain a database; Skipping initialization

2023-11-24 09:18:58.861 UTC [1] LOG:  starting PostgreSQL 15.0 (Debian 15.0-1.pgdg110+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10
.2.1 20210110, 64-bit
2023-11-24 09:18:58.861 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2023-11-24 09:18:58.861 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2023-11-24 09:18:58.884 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2023-11-24 09:18:58.940 UTC [28] LOG:  database system was shut down at 2023-11-24 09:16:43 UTC
2023-11-24 09:18:58.993 UTC [1] LOG:  database system is ready to accept connections
```

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: