CÓDIGO EN POSTGRESQL ELABORADO POR AUREL PARA IMPLEMENTAR UNA BASE DE DATOS DISTRIBUIDA EN DOS NODOS

Fuente: https://dev.to/aurelmegn/setting-up-distributed-database-architecture-with-postgresql-261
Fecha del autor: 20/Oct/2019

Let assume that the library is split into two cities: (Paris, and Lagos). In the generated queries of our application, we notice that many queries are of this kind:

select booksample.state, booksample.lendable, booksample.location from booksample where booksample.location = 'lagos';

select booksample.state, booksample.lendable, booksample.location from booksample where booksample.location = 'paris';

On the master server

Let create the different table on the master database.

The master database is the one at the front of the user, we will create the tables as normal table to start. Here we would use the Paris's server as the master.

create the database:

create database dd_test;

create the tables student and book:

--table student
drop table if exists student cascade;
create table student (id serial primary key, name varchar);
--table book
drop table if exists book cascade;
create table book(id serial primary key, title varchar);

table booksample:

drop table if exists booksample cascade;
create table booksample(id serial primary key, state varchar, lendable varchar, location varchar, book_id int references book(id));

table lend:

drop table if exists lend cascade;
create table lend(student_id int references student(id), booksample_id int references booksample(id), at date, returned_at date);

On Paris's server

In this example, as we are using the Paris's site as master, this query should be run on the same postgres server as the previous one. Let create the tables booksample_paris and lend_paris

-- paris site
drop table if exists booksample_paris cascade;
create table booksample_paris(check(location='paris')) inherits(booksample);

drop table if exists lend_paris cascade;
create table if not exists lend_paris() inherits(lend);

The check(location='paris') will help postgres to fetch the row in a "smart" way.

I use the inherits keyword to tell postgres that the table booksample_paris is actually a part of the table booksample. Thus, when we issue a:

*explain select * from booksample;*

we get

```
dd_test=> explain select * from booksample;
                    QUERY PLAN
-------------------------------------------------------------------------
 Append  (cost=0.00..151.56 rows=1251 width=104)
   -> Seq Scan on booksample  (cost=0.00..0.00 rows=1 width=104)
   -> Seq Scan on booksample_paris  (cost=0.00..16.10 rows=610 width=104)
(3 rows)
```

It means that postgres will try to grab the rows of the table booksample_paris on a select.

On Lagos's server

Let's create the database

create database dd_test;

Here we are going to set up the postgres server to listen to the other network interfaces. This is done by modifying the configuration file located at (on most Linux os) /etc/postgresql/9.5/main/postgresql.conf

```
#------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------

# - Connection Settings -

listen_addresses = '*'     # what IP address(es) to listen on;
            # comma-separated list of addresses;
            # defaults to 'localhost'; use '*' for all
```

 # (change requires restart)

The next step is to allow a user to connect through the network interfaces by modifying /etc/postgresql/9.5/main/pg_hba.conf

 # TYPE   DATABASE   USER   ADDRESS   METHOD

 # IPv4 local connections:
host   all   test_user   all   md5

Let's create now the partitions tables:

-- booksample_lagos
drop table if exists booksample_lagos cascade;
create table booksample_lagos(id int, state varchar, lendable bool default false, location varchar, book_id int);

-- lend_lagos
drop table if exists lend_lagos cascade;
create table lend_lagos(student_id int, booksample_id int, at date, returned_at date);

On the master server

We will use the foreign table ** feature of postgres to be able to access the **Lagos's database tables remotely from the master server . To be able to do this, we should create the postgres_fdw extension in our database. This action should be done only by an administrator, so let's connect to the database as the administrator postgres user and do:

/!\ There are many alternatives to foreign table use, such as the use of materialized views + triggers or postgres partitioning feature.

create extension postgres_fdw;

create server master_server foreign data wrapper postgres_fdw options (host '{ip address of the lagos postgres server}', port '5432' , dbname 'dd_test');

create user mapping for master_user server master_server options (user '{our username on lagos server}', password '{our password on lagos server}');

alter server master_server owner to master_user;

First, we create the extension postgres_fdw and after a "foreign data server" on the master postgres server. We now create a user mapping to be able to query the Lagos server.

Now, let's create the foreign tables located on the master server which map to the shard on the Lagos servers.

drop foreign table if exists booksample_lagos cascade;
create foreign table booksample_lagos (check(location='lagos')) inherits(booksample) server master_server;

```
drop foreign table if exists lend_lagos cascade;
create foreign table lend_lagos () inherits(lend) server master_server;
```

In its actual state, the master server will fill the the table booksample and lend when a query like this is executed.
```
insert into booksample values(1, 'new','paris',1)
```

This is not a good behavior as the new partitions we created will not hold any data. To fix this situation, we will use "triggers" to redirect the row into their normal destination.

The trigger bellows is to redirect the booksample insertion into the correct partition: either booksample_lagos or booksample_paris based on the value of attribute location:

```
-- trigger on insert booksample
create or replace function booksample_trigger_fn() returns trigger as
$$
begin

    if new.location = 'paris' then
        insert into booksample_paris values(new.*);
    elsif new.location = 'lagos' then
        insert into booksample_lagos values(new.*);
    end if;

    return null;
end
$$
language plpgsql;
```

```
drop trigger if exists booksample_trigger on booksample;
create trigger booksample_trigger before insert on booksample for each row execute procedure booksample_trigger_fn();
```

Now, we would like to redirect the queries on the table lend to the correct partition. Here we store the row into the site where the booksample belongs to.

```
create or replace function lend_trigger_fn() returns trigger as
$$
declare
    vbooksample booksample%rowtype;
begin
    -- select the booksample referenced by the booksample_id
    select * into vbooksample from booksample where id=new.booksample_id;

    -- get the location to use and save the row
    if vbooksample.location = 'paris' then
        insert into lend_paris values(new.*);
    elsif vbooksample.location = 'lagos' then
        insert into lend_lagos values(new.*);
    end if;

    return null;
```

endtut
$$
language plpgsql;

drop trigger if exists lend_trigger on lend;
create trigger lend_trigger before insert on lend for each row execute procedure lend_trigger_fn();

Our database is now functional.

Test

Let's run some query to get an overview of our database capabilities:

insert into book(title) values('book#1');
insert into student(name) values('std#1'),('std#2'), ('std#3');

-- thanks to the trigger we wrote, the insertion will be executed in the right server
insert into booksample(state, lendable, location, book_id) values('old', true, 'lagos', 1), ('old', true, 'paris', 1), ('new', true 'lagos', 1);

insert into lend(student_id , booksample_id ,at) values(1,1, now()), (2,2, now()), (3,3, now());

Let see what happen when we try to select all the row.

*explain select * from booksample;*

-- thanks to the check(location='{}') constraint, the select query is executed in a "smart" way
*explain select * from booksample where location='lagos';*
                    QUERY PLAN

 Append (cost=0.00..118.08 rows=4 width=104)
   -> Seq Scan on booksample (cost=0.00..0.00 rows=1 width=104)
      Filter: ((location)::text = 'lagos'::text)
   -> Foreign Scan on booksample_lagos (cost=100.00..118.06 rows=3 width=104)
(4 rows)

*explain select * from booksample where location='paris';*
                    QUERY PLAN

 Append (cost=0.00..118.08 rows=4 width=104)
   -> Seq Scan on booksample (cost=0.00..0.00 rows=1 width=104)
      Filter: ((location)::text = 'paris'::text)
   -> Foreign Scan on booksample_paris (cost=100.00..118.06 rows=3 width=104)
(4 rows)

Thanks to the check(location='{}') constraint, the select query is executed in a "smart" way. The DBMS scan only the table on Paris's server or Lagos's server.

Thank you for your attention.