This tutorial shows how to install pl/proxy.

If you don’t have postgresql install it:

**sudo apt-get update**

**sudo apt-get install postgresql-9.1**

PL/Proxy also needs *postgresql-server-dev*

**sudo apt-get install postgresql-server-dev-9.1**

Once installed, clone pl/proxy to any folder chosen by you:

**git clone git://git.postgresql.org/git/plproxy.git**

You may need to install:

**sudo apt-get install flex**

**sudo apt-get install bison**

Once installed, execute:

**make**

**sudo make install**

That’s all, you’ve just installed PL/Proxy. Enter the sql folder inside plproxy. Now let's create a new proxy database by doing:

**sudo -u postgres psql**

**$postgres: create DATABASE testdb;**

**$postgres: create DATABASE part00;**

**$postgres: create DATABASE part01;**

**$postgres: \q**

**sudo -u postgres psql -f plproxy.sql testdb**

Now testdb is configured to use pl/proxy. The next step is to create two new db as shown above named *part00* and *part01*. It’s preferable to create a user for all the databases with a password and grant all the privileges:

**sudo -u postgres psql**

**create user testos with password 'testos';**

**grant all privileges on database testdb to testos;**

**grant all privileges on database part00 to testos;**

**grant all privileges on database part01 to testos;**

Let’s create a table called users in part0X:

**CREATE TABLE users (  
 username text,  
 email text  
);**

The next step is to create our plcluster in the testdb used as proxy:

**CREATE FOREIGN DATA WRAPPER plproxy;**

**\c testdb**

**CREATE SERVER usercluster FOREIGN DATA WRAPPER plproxy  
OPTIONS (connection\_lifetime '1800',  
 p0 'dbname=part00 host=127.0.0.1',  
 p1 'dbname=part01 host=127.0.0.1' );**

**CREATE USER MAPPING FOR PUBLIC SERVER usercluster;**

**GRANT ALL PRIVILEGES ON FOREIGN SERVER usercluster TO testos;**

**--GRANT USAGE ON SERVER usercluster TO testos;**

(grant all privileges on FOREIGN server usercluster to testos; #if necessary...)

Finally we will define two functions to insert and get users from the cluster:

Get users:

**CREATE OR REPLACE FUNCTION get\_user\_email(i\_username text)  
RETURNS SETOF text AS $$  
 CLUSTER 'usercluster';  
 RUN ON hashtext(i\_username) ;  
 SELECT email FROM users WHERE username = i\_username;  
$$ LANGUAGE plproxy;**

Insert users:

We will create a new function in part0X:

**CREATE OR REPLACE FUNCTION insert\_user(i\_username text, i\_emailaddress text)  
RETURNS integer AS $$  
 INSERT INTO users (username, email) VALUES ($1,$2);  
 SELECT 1;  
$$ LANGUAGE SQL;**

The next step is to create a proxy function:

**CREATE OR REPLACE FUNCTION insert\_user(i\_username text, i\_emailaddress text)  
RETURNS integer AS $$  
 CLUSTER 'usercluster';  
 RUN ON hashtext(i\_username);  
$$ LANGUAGE plproxy;**

Finally, in order to test our proxy we can execute:

SELECT insert\_user('Hello','hello@world.com');

**About PL/Proxy Language**

It contains only 4 statements: CONNECT, CLUSTER, RUN and SELECT.

Each function needs to have either CONNECT or pair of CLUSTER + RUN statements to specify where to run the function.

The **SELECT** statement is optional, if it is missing, **there will be default query generated based on proxy function signature.**

The RUN statement is also optional, it defaults to RUN ON ANY which means the query will be run random partition.

**EXAMPLE 1: GET\_USER**

There are several ways to do the same call by using plpgsql or just sql. This example is intended to show this.

Using SQL:

In the proxy side we will write:

**CREATE OR REPLACE FUNCTION get\_user(i\_username text)**

**RETURNS SETOF record AS $$**

**CLUSTER 'usercluster';**

**RUN ON hashtext(i\_username);**

**$$ LANGUAGE plproxy;**

While in the partition we’ll put:

**CREATE OR REPLACE FUNCTION get\_user(i\_username text)**

**RETURNS record AS $$**

**SELECT \* FROM users WHERE username=$1;**

**$$ LANGUAGE SQL;**

As you can see, both functions return a RECORD type. Due to this fact, we must specify the columns used by the record as follows:

**testdb=# SELECT \* FROM get\_user('Sven') AS (username text, email text);**

On the other hand, we can use plpgsql instead of sql:

**CREATE OR REPLACE FUNCTION get\_user(i\_username text)**

**RETURNS TABLE(username text, email text) AS $$**

**CLUSTER 'usercluster';**

**RUN ON hashtext(i\_username);**

**$$ LANGUAGE plproxy;**

Note the use of TABLE instead of a record this is because in the client we will execute:

**CREATE OR REPLACE FUNCTION get\_user(i\_username text)**

**RETURNS TABLE(username text, email text) AS $$**

**BEGIN**

**RAISE NOTICE 'I am server part01';**

**RETURN QUERY SELECT \* FROM users AS u WHERE u.username=i\_username; -- username could refer to either a PL/pgSQL variable or a table column. So use u.username instead**

**END;**

**$$ LANGUAGE plpgsql;**

In this case I wanted to return a table to show another return type because in this case it’s not necessary to define AS (username text, email text):

**CREATE OR REPLACE FUNCTION get\_user(i\_username text)**

**RETURNS SETOF record AS $$**

**BEGIN**

**RAISE NOTICE 'I am server part01';**

**RETURN QUERY SELECT \* FROM users AS u WHERE u.username=i\_username; -- username could refer to either a PL/pgSQL variable or a table column. So use u.username instead**

**END;**

**$$ LANGUAGE plpgsql;**

(PS: note the difference between the returns declaration from setof to just table).

**EXAMPLE 2: EXECUTE\_QUERY**

This example wants to show how to execute any kind of query:

**CREATE OR REPLACE FUNCTION execute\_query(q text, i\_username text)**

**RETURNS SETOF RECORD AS $$**

**CLUSTER 'usercluster';**

**RUN ON hashtext(i\_username);**

**$$ LANGUAGE plproxy;**

**CREATE OR REPLACE FUNCTION execute\_query(sql text, i\_username text)**

**RETURNS SETOF RECORD AS $$**

**BEGIN**

**sql := sql || ''''|| $2 || '''';**

**RAISE NOTICE '%', sql;**

**RETURN QUERY EXECUTE sql ;**

**END;**

**$$ LANGUAGE plpgsql;**

Note that both functions have the same number of arguments so it’s not necessary to use TARGET or SELECT and run another function. If instead of concatenating the sql string we put nothing, we will have to do:

**SELECT \* FROM execute\_query('SELECT \* FROM users WHERE username=' || '''1234''', '1234') AS (username text, email text);**

Here we have an example of how this query would be called in Java:

**private static void executeQuery(Connection connection, String username) throws SQLException {**

**String sql = "SELECT \* FROM users WHERE username=' || '''" + username + "''";**

**String query = "SELECT \* FROM execute\_query('" + sql + "', '" + username + "') AS (username text, email text);";**

**System.out.println(query);**

**PreparedStatement ps = connection.prepareStatement(query);**

**ResultSet resultSet = ps.executeQuery();**

**while (resultSet.next()) {**

**String s = resultSet.getString("email");**

**System.out.println(s);**

**}**

**ps.close();**

**}**