P12: Acquiring Postgre Procedural Language superpowers

INTRODUCTION

PL/pgSQL (Procedural Language/PostgreSQL) is a procedural programming language supported by the PostgreSQL ORDBMS. It closely resembles Oracle's PL/SQL language.

PL/pgSQL, as a fully featured programming language, allows much more procedural control than SQL, including the ability to use loops and other control structures. SQL statements and triggers can call functions created in the PL/pgSQL language.

You have to solve all the proposed problems and document it in a professional way.

PART A. Preparing the development environment

In order to solve this practise you can choose any client-server architecture and choose any of the database clients seen in class.

Download the following file (dellstore2-normal-1.0.tar.gz) and import it into a new database 'dell' and into that database into the schema 'data' (owner of everything 'alumne'). Revoke all permission from public to that database and to that schema.

Clue:

- **Using pgcli you can get errors importing the database, use psql...**

```
FigaGris:Temp sergi$ pgcli -h 192.168.56.101 -p 5432 -U alumne dell
Password for alumne:
Server: PostgreSQL 11.5 (Debian 11.5-1+deb10u1)
Version: 2.2.0
Chat: https://gitter.im/dbcli/pgcli
Home: http://pgcli.com
alumne@192:dell> \i /Users/sergi/Downloads/dellstore2-normal-1.0/dellstore2-normal-1.0.sql
'utf-8' codec can't decode byte 0x92 in position 6175578: invalid start byte
```

```
postgres=# \c dell
You are now connected to database "dell" as user "postgres".
dell=# \i /home/alumne/dellstore2-normal-1.0.sql
```

The result database should be something like this:

```
FigaGris:dellstore2-normal-1.0 sergi$ pgcli -h 192.168.56.101 -p 5432 -U alumne dell
Password for alumne:
Server: PostgreSQL 11.5 (Debian 11.5-1+deb10u1)
Version: 2.2.0
Chat: https://gitter.im/dbcli/pgcli
Home: http://pgcli.com
alumne@192:dell> \dt
+----------+------------+--------+---------+
| Schema   | Name       | Type   | Owner   |
|----------+------------+--------+---------|
| data     | categories | table  | alumne  |
| data     | cust_hist  | table  | alumne  |
| data     | customers  | table  | alumne  |
| data     | inventory  | table  | alumne  |
| data     | orderlines | table  | alumne  |
| data     | orders     | table  | alumne  |
| data     | products   | table  | alumne  |
| data     | reorder    | table  | alumne  |
+----------+------------+--------+---------+
SELECT 8
Time: 0.023s
```

We acces our virtual machine

```
C:\Users\asix>ssh aperellop@52.174.0.123
aperellop@52.174.0.123's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1047-azure x86_64)
```

We see our dockers and start postgres

```
aperellop@aperellop-ubuntu:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS                          NAMES
51911f1be3f5        dpage/pgadmin4      "/entrypoint.sh"         23 hours ago        Up 2 minutes                0.0.0.0:80->80/tcp, 443/tcp    peaceful_shirley
161cb0c9c70a        postgres            "docker-entrypoint.s…"   23 hours ago        Up About a minute           5432/tcp                       postgresSGBD
63b1b918c983        mariadb:latest      "docker-entrypoint.s…"   2 months ago        Exited (0) 3 weeks ago                             tustampa
5578f83a199e        mariadb:latest      "docker-entrypoint.s…"   3 months ago        Exited (0) 3 weeks ago                             some-mariadb
aperellop@aperellop-ubuntu:~$ sudo docker start 161cb0c9c70a
161cb0c9c70a
```

We enter tot the bash of our virtual machine and from it we access postgres

```
aperellop@aperellop-ubuntu:~$ sudo docker exec -it 161cb0c9c70a bash
root@161cb0c9c70a:/# psql -h localhost -p 5432 -U postgres -W
Password:
psql (13.2 (Debian 13.2-1.pgdg100+1))
Type "help" for help.
```

We list all databases

```
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
 dell      | chriskl  | UTF8     | en_US.utf8 | en_US.utf8 |
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
(4 rows)
```

We check wich database we are conected to

```
postgres=# \c
Password:
You are now connected to database "postgres" as user "postgres".
```

We create the dell database, connect to it and enter all the content

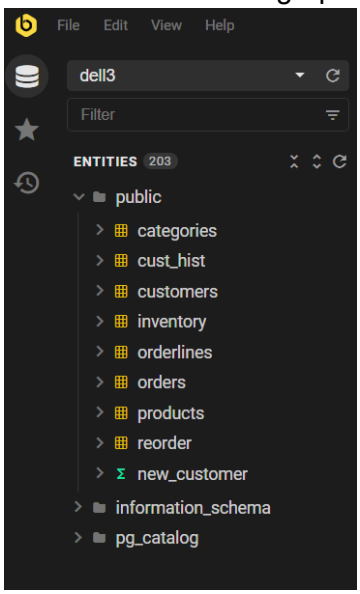
```
postgres=# create database dell2;
CREATE DATABASE
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
 dell      | chriskl  | UTF8     | en_US.utf8 | en_US.utf8 |
 dell2     | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
(5 rows)
```

```
postgres=# \c dell2
Password:
You are now connected to database "dell2" as user "postgres".
```

We list the tables of database to check that the import has gone well

```
dell2=# \dt
          List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | categories | table | postgres
 public | cust_hist  | table | postgres
 public | customers  | table | postgres
 public | inventory  | table | postgres
 public | orderlines | table | postgres
 public | orders     | table | postgres
 public | products   | table | postgres
 public | reorder    | table | postgres
(8 rows)
```

And we link it to our graphics tool



PART B. QUESTIONS

1.- Create a function ‘del_under18’ to delete the customers under 18 years old. The function must return the deleted customer’s ids. To test if the function works insert:

```
INSERT INTO data.customers VALUES (20001, 'Sergi', 'González', '6224597470 Dell Way', NULL,
'DVCINXG', NULL, 0, 'Australia', 2, 'sg@dell.com', '6224597470', 3, '1869697669055999',
'2010/07', 'user20001', 'password', 17, 40000, 'F');
INSERT INTO data.customers VALUES (20002, 'Pep', 'López', '6224597470 Dell Way', NULL,
'DVCINXG', NULL, 0, 'Australia', 2, 'sg@dell.com', '6224597470', 3, '1869697669055999',
'2010/07', 'user20002', 'password', 17, 40000, 'F');
```

```
alumne@192:dell> select del_under18();
+---------------+
| del_under18   |
|---------------|
| 20001         |
| 20002         |
+---------------+
SELECT 2
Time: 0.030s
```

```
1 CREATE OR REPLACE FUNCTION del_under_18()
2 RETURNS SETOF INTEGER AS $$
3 DELETE FROM customers WHERE age <18
4 RETURNING customerid;
5 $$LANGUAGE  sql;
```

```
1 select del_under_18();
```

del_under_18
20001
20002

2.- Create a new function ‘del_under18_2’ returning the number of customers before and after deleting the ones under 18 y.o. Create a type to return both values in a row. To test if the function works insert:

```
INSERT INTO data.customers VALUES (20001, 'Sergi', 'González', '6224597470 Dell Way', NULL,
'DVCINXG', NULL, 0, 'Australia', 2, 'sg@dell.com', '6224597470', 3, '1869697669055999',
'2010/07', 'user20001', 'password', 17, 40000, 'F');
INSERT INTO data.customers VALUES (20002, 'Pep', 'López', '6224597470 Dell Way', NULL,
'DVCINXG', NULL, 0, 'Australia', 2, 'sg@dell.com', '6224597470', 3, '1869697669055999',
'2010/07', 'user20002', 'password', 17, 40000, 'F');
```

```
alumne@192:dell> select del_under18_2();
+------------------+
| del_under18_2    |
|------------------|
| (20002,20000)    |
+------------------+
SELECT 1
Time: 0.038s
```

```
alumne@192:dell> select * from del_under18_2();
+----------+---------+
| before   | after   |
|----------+---------|
| 20000    | 20000   |
+----------+---------+
SELECT 1
Time: 0.031s
```

```
1 CREATE TYPE return_before_after as (before int, after int);
2 CREATE OR REPLACE FUNCTION del_under18_2()
3 RETURNS return_before_after AS $$
4 DECLARE
5 before INTEGER;
6 after INTEGER;
7 BEGIN
8 SELECT COUNT(*) INTO before
9 FROM customers;
0 DELETE FROM customers WHERE age < 18;
1 SELECT COUNT(*) INTO after
2 FROM customers;
3 RETURN (before,after);
4 end;
5 $$ LANGUAGE plpgsql;
```

```
1 select * from del_under18_2()
```

before	after
20000	20000

3.- Create a function to insert new categories. Do four versions: the first one with named parameters, the second one with numbered parameters, the third one with parameters with the same name that the table fields (category, categoryname), and the last one with only a single parameter (data type 'categories'). Use language sql. Headers:

```
CREATE FUNCTION insert_category1_sql(pcategory integer, pname varchar) RETURNS void AS
CREATE FUNCTION insert_category2_sql(integer, varchar) RETURNS void AS
CREATE FUNCTION insert_category3_sql(category integer, categoryname varchar) RETURNS void AS
CREATE FUNCTION insert_category4_sql(pcategory categories) RETURNS void AS
```

```
dell=> select insert_category1_sql(21, 'category21');
 insert_category1_sql
----------------------

(1 fila)
```

```
dell=> select insert_category2_sql(22, 'category22');
 insert_category2_sql
----------------------

(1 fila)
```

```
dell=> select insert_category3_sql(23, 'category23');
 insert_category3_sql
----------------------

(1 fila)
```

```
dell=> select insert_category4_sql((24, 'category24'));
 insert_category4_sql
----------------------

(1 fila)
```

```sql
CREATE function insert_category1_sql (pcategory integer, pname varchar)
returns void as $$
begin
    insert into categories (category, categoryname) values (pcategory, pname);
end; $$ language plpgsql;
select insert_category1_sql(21, 'category21');
```

```sql
CREATE function insert_category2_sql (integer, varchar)
returns void as $$
begin
    insert into categories (category, categoryname) values ($1, $2);
end; $$ language plpgsql;
select insert_category2_sql(22, 'category22');
```

```sql
CREATE function insert_category3_sql (category integer, categoryname varchar)
returns void as $$
begin
    insert into categories (category, categoryname) values (category, categoryname);
end; $$ language plpgsql;
select insert_category3_sql(23, 'category23');
```

```sql
CREATE function insert_category4_sql (pcategory categories)
returns void as $$
begin
    insert into categories values (pcategory);
end; $$ language plpgsql;
select insert_category4_sql((24, 'category24'));
```

4.- Create the following functions to return the first name and the last name of a customer. Write two versions: one with language sql and another one with plpgsql. It's mandatory to use the following headers:

```
CREATE FUNCTION show_name_sql(id integer, OUT first varchar, OUT last varchar) AS
CREATE FUNCTION show_name_plpgsql(id integer, OUT first varchar, OUT last varchar) AS
```

```
dell=> select show_name_sql(2);
     show_name_sql
----------------------
 (HQNMZH,UNUKXHJVXB)
(1 fila)

dell=> select first, last from show_name_sql(2);
 first  |    last
--------+------------
 HQNMZH | UNUKXHJVXB
(1 fila)
```

```
dell=> select show_name_plpgsql(2);
  show_name_plpgsql
----------------------
 (HQNMZH,UNUKXHJVXB)
(1 fila)

dell=> select first, last from show_name_plpgsql(2);
 first  |    last
--------+------------
 HQNMZH | UNUKXHJVXB
(1 fila)
```

```
set search_path = "data";
```

```
create type fullname as (name varchar, lname varchar);
CREATE FUNCTION show_name_sql(int) RETURNS fullname AS $$
    select firstname, lastname from customers where customerid = $1;
$$ LANGUAGE SQL;

select show_name_sql(2);
select * from show_name_sql(2);
```

```
create type fullname as (name varchar, lname varchar);
create or replace function show_name_plsql(cid int)
returns fullname as $$
declare
   name varchar;
   lname varchar;
begin
   select firstname into name from customers where customerid = $1;
   select lastname into lname from customers where customerid = $1;
   return(name, lname);
end; $$ language plpgsql;

select show_name_plsql(2);
select * from show_name_plsql(2);
```

5.- Create a function to increase the price of the products by 5%. You return the new price (use RETURNING CLAUSE). Write two versions: one with language sql and another one with plpgsql. Headers:

```
CREATE FUNCTION increase_price_sql(prod products) RETURNS numeric AS
```

```
CREATE FUNCTION increase_price_sql(prod products)
RETURNS numeric AS $$
update products set price=price + 0.05 * price where prod.prod_id = prod_id returning price;
$$ language sql;
```

```
CREATE FUNCTION increase_price_plpgsql(prod products) RETURNS numeric AS
```

```
 9 CREATE FUNCTION increase_price_plpgsql(prod products) RETURNS numeric AS $$
10 begin
11     update products set price=price + 0.05 * price where prod.prod_id = prod_id;
12     return (select price from products where prod.prod_id = prod_id);
13 end; $$ language plpgsql;
```

```
dell=> SELECT common_prod_id, increase_price_sql(products.*)
dell-> FROM products
dell-> WHERE title='ACADEMY ADAPTATION';
 common_prod_id | increase_price_sql
----------------+--------------------
           7173 |              30.44
(1 fila)
```

```
dell=>  SELECT  common_
dell->  FROM  products
dell->  WHERE  title='A
 common_prod_id  |  inc
-----------------+-----
            7173  |
(1  fila)
```

```
6 select common_prod_id, increase_price_sql(products.*) from products where title='ACADEMY ADAPTATION';
```

common_prod_id	increase_price_sql
7173	31.96

```
15 select common_prod_id, increase_price_plpgsql(products.*) from products where title='ACADEMY ADAPTATION';
```

common_prod_id	increase_price_plpgsql
7173	33.56

6.- Create a function to return all data of a customer. Write two versions: one with language sql and another one with plpgsql. Headers:

```
CREATE FUNCTION show_cust_sql(id integer) RETURNS customers AS
CREATE or replace FUNCTION show_cust_plpgsql(id integer) RETURNS customers AS
```

```
dell=> select * from show_cust_sql(2);
dell=> select (show_cust_sql(2)).firstname;
 firstname
-----------
 HQNMZH
(1 fila)
```

```
dell=> select * from show_cust_plpgsql(2);
dell=> select (show_cust_plpgsql(2)).firstname;
 firstname
-----------
 HQNMZH
(1 fila)
```

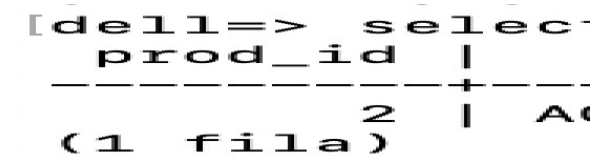
```sql
CREATE FUNCTION show_cust_sql(int) RETURNS customers AS $$
    SELECT * FROM customers WHERE customerid = $1;
$$ LANGUAGE SQL;


create or replace function show_cust_plpgsql(id INTEGER) returns customers as $$
declare customer record;
begin
    SELECT  * into customer FROM customers WHERE customerid = $1;
    return customer;
end; $$ language 'plpgsql';
```

```sql
select * from show_cust_plpgsql(2);
select (show_cust_plpgsql(2)).firstname;
```

customerid	firstname	lastname	address1	address2	city	state	zip	country	region	email	phone	creditcard
2	HQNMZH	UNUKXHJVXB	5119315633 Dell Way	(NULL)	YNCERXJ	AZ	11802	US	1	UNUKXHJVXB@dell.com	5119315633	1

7.- Create a function to return the id, name and price of a product passing its identifier. Write two versions: one with language sql and another one with plpgsql. Headers:

```
CREATE or replace FUNCTION show_prod_sql(INOUT prod_id integer, OUT title varchar(50), OUT price numeric) AS
CREATE or replace FUNCTION show_prod_plpgsql(INOUT prod_id integer, OUT title varchar(50), OUT price numeric) RETURNS RECORD AS
```



```
dell=> select * from show_prod_plpgsql(2);
 prod_id |    title    | price
---------+-------------+-------
       2 | ACADEMY ACE | 20.99
(1 fila)
```

```sql
CREATE or replace FUNCTION show_prod_sql(INOUT prod_id integer, OUT title varchar(50), OUT price numeric) AS $$
    SELECT products.prod_id, products.title, products.price FROM products WHERE products.prod_id = $1;
$$ LANGUAGE SQL;

select * from show_prod_sql(2);
```

```sql
CREATE or replace FUNCTION show_prod_plpgsql(INOUT prod_id integer, OUT title varchar(50), OUT price numeric) RETURNS
$$
begin
    SELECT products.title into title FROM products WHERE products.prod_id =  $1;
    SELECT products.price into price FROM products WHERE products.prod_id =  $1;
    return;
end;
$$ language 'plpgsql';

select * from show_prod_plpgsql(2);
```

8.- Write an average function (without parameters) that returns the average price of the products. Write two versions: one with language sql and another one with plpgsql. Make a third version not using avg. Headers:

```
CREATE or replace FUNCTION avg_price_sql() RETURNS numeric AS
```

```sql
CREATE or replace FUNCTION avg_price_sql() RETURNS numeric AS
$$
    SELECT avg(price) FROM products;
$$ LANGUAGE SQL;

select * from avg_price_sql();
```

```
CREATE or replace FUNCTION avg_price_plpgsql() RETURNS numeric AS
```

```sql
CREATE or replace FUNCTION avg_price_plpgsql() RETURNS numeric AS
$$
begin
    return(SELECT avg(price) FROM products);

end;
$$ language 'plpgsql';

select * from avg_price_plpgsql();
```

```
CREATE or replace FUNCTION avg_price_noavg() RETURNS numeric AS
```

```sql
CREATE or replace FUNCTION avg_price_noavg() RETURNS numeric AS
$$
    declare average products.price%type;
begin
    SELECT sum(price)/count(*) into average FROM products;
    return average;

end;
$$ language 'plpgsql';

select * from avg_price_noavg();
```

9.- Using the function ‘avg_price_sql’, do the following queries:

a. Average price of all the products.

```
select avg_price_sql() as average_of_all_products;
```

average_of_all_products
20.0155570000000000

b. Title and price of products that cost more than the average product price.

```
select title, price from products where price > avg_price_sql();
```

title	price
ACADEMY ACADEMY	25.99
ACADEMY ACE	20.99
ACADEMY AIRPLANE	25.99
ACADEMY ALAMO	26.99
ACADEMY ALASKA	23.99
ACADEMY ALI	24.99
ACADEMY ALIEN	27.99
ACADEMY ALLEY	27.99
ACADEMY ALONE	28.99
ACADEMY ALTER	27.99
ACADEMY AMERICAN	24.99
ACADEMY ANACONDA	24.99

c. Title and price of products that cost more than a 20% to the average price.

```
select title, price from products where price > avg_price_sql()*0.2
```

title	price
ACADEMY ACADEMY	25.99
ACADEMY ACE	20.99
ACADEMY AFFAIR	14.99
ACADEMY AFRICAN	11.99
ACADEMY AGENT	15.99
ACADEMY AIRPLANE	25.99
ACADEMY AIRPORT	16.99
ACADEMY ALABAMA	10.99
ACADEMY ALADDIN	9.99
ACADEMY ALAMO	26.99
ACADEMY ALASKA	23.99
ACADEMY ALI	24.99
ACADEMY ALICE	12.99
ACADEMY ALIEN	27.99
ACADEMY ALLEY	27.99

d. Title and price of products that theirs price is equal to the average price to 20% (ie those that theirs price is between 80% and 120% of the average price).

```
select title, price from products where price between avg_price_sql()*0.8 and avg_price_sql()*1.2
```

title	price
ACADEMY ACE	20.99
ACADEMY AIRPORT	16.99
ACADEMY ALASKA	23.99
ACADEMY AMELIE	17.99
ACADEMY AMISTAD	16.99
ACADEMY ANYTHING	17.99
ACADEMY APACHE	18.99
ACADEMY ARIZONA	22.99
ACADEMY ARK	20.99
ACADEMY ARMAGEDDON	18.99
ACADEMY ARMY	21.99
ACADEMY ARTIST	23.99
ACADEMY ATLANTIS	22.99
ACADEMY ATTACKS	23.99
ACADEMY AUTUMN	21.99
ACADEMY BACKLASH	18.99
ACADEMY BADMAN	23.99

10.- Create a function to return all products of a category. Write two versions: one with language sql and another one with plpgsql.

Clue 1: SETOF.
Clue 2: For the plpgsql version check the slide 28.

```
dell=> select count(*) from show_prod_cat_sql(1);
 count
-------
   627
(1 fila)
```

```
dell=> select count(*) from show_prod_cat_plpgsql(1);
 count
-------
   627
(1 fila)
```

```
CREATE or replace FUNCTION show_prod_cat_sql (x int)
RETURNS table(prod_id int, category int, title varchar(50), actor varchar(50), price numeric, special smallint, commo
   select * from products where category=$1;
$$ LANGUAGE SQL;

select count(*) from show_prod_cat_sql(1);
```

```
CREATE or replace FUNCTION show_prod_cat_plpgsql(in x int)
RETURNS setof products AS $$
begin
    return query (select * from products where products.category=x);
end;
$$ language 'plpgsql';

select count(*) from show_prod_cat_plpgsql(1);
```

11.- Repeat exercise 10 (only the plpgsql version) using TABLE to return the values. Clue: Slide 25. Header of the function:

```
CREATE or replace FUNCTION show_prod_cat2_plpgsql(catid integer)
RETURNS TABLE(
prod_id integer,
category integer,
title character varying(50),
actor character varying(50),
price numeric(12,2),
special smallint,
common_prod_id integer
) AS
```

```
CREATE or replace FUNCTION show_prod_cat2_plpgsql(x int)
RETURNS table(prod_id int, category int, title varchar(50), actor varchar(50), price numeric, special smallint, commo
begin

    return query (select * from products where products.category=$1);
end;
$$ language 'plpgsql';

select * from show_prod_cat2_plpgsql(1);
```

DANGER: 12.- Repeat the last query using a CURSOR (we'll see this topic later…). Use this header:

```
CREATE or replace FUNCTION show_prod_cat_plpgsql(catid integer)
RETURNS SETOF products AS
```

Clue: slide 108 (or searching on the Internet).

DANGER: 13.- Create a function "mySum" to add integers, decimals and strings. Note that if you define a function only for integers an error will appear… CLUE: slide 53.

```
mydb=> select mySum(1,1), mySum(1.1,1), mySum('1','1');
 mysum | mysum | mysum
-------+-------+-------
     2 |   2.1 | 11
```

PART C. CONTROL VERSION ON THE GENERATED CODE

Upload all the final code in sql format into your github repository in a correct order and structure and provide its URL.