

EJERCICIOS TEMA 7 – Procedural PostgreSQL

7.1.- Crea una funció anomenada **DEU_Q**, que traga els **números del 1 al 10** i els seus **quadrats**. (Utilitza **RAISE NOTICE**).

```
create or replace function deu_q() returns void as $cos$
begin
    for i in 1..10 loop
        raise notice 'Número: %. Quadrat: %', i, i*i;
    end loop;
end; $cos$
language 'plpgsql';
```

```
select deu_q();
```

7.2.- Fes una altra funció, **IMP**, que traga per pantalla els números **imparells** del 1 al 50. (Utilitza **RAISE NOTICE**).

```
create or replace function IMP() returns void as $cos$
begin
    for i in 1..50 loop
        if i % 2 != 0 then
            raise notice '%', i;
        end if;
    end loop;
end; $cos$
language 'plpgsql';
```

```
select imp();
```

7.3.- Fes una funció anomenada **TAULA_MULT**, per a que traga la taula de multiplicar del **paràmetre** que se li ha de passar. (Utilitza **RAISE NOTICE**). Aquest podria ser el seu aspecte, en executar-la:

```
create or replace function TAULA_MULT(n numeric) returns void as $cos$
begin
    raise notice 'TAULA DE MULTIPLICAR DEL %', $1;
    raise notice '=====';
    for i in 1..10 loop
        raise notice '%', $1 || ' x ' || i || ' = ' || n * i;
    end loop;
end; $cos$
language 'plpgsql';
```

```
select taula_mult(7);
```

7.4.- Fes una funció, anomenada **MAX2**, que tinga dos paràmetres **numèrics** i que **torne el màxim** entre aquests dos. (Ara ja **no** s'ha d'utilitzar **RAISE NOTICE**). Podeu comprovar el seu funcionament fent per exemple **SELECT MAX(5,7);** **SELECT MAX(15,7)**. Sempre ha de mostrar el màxim dels dos.

```
create or replace function MAX2(n1 numeric, n2 numeric) returns
numeric as $cos$
declare
    max numeric;
```

```

begin
    if n1 >= n2 then
        max := n1;
    else
        max := n2;
    end if;
return max;
end; $cos$
language 'plpgsql';

```

```

select max2(15,7);

```

7.5.- Utilitza l'anterior per a crear **MAX3**. Has d'utilitzar obligatòriament la funció **MAX2**

```

create or replace function MAX3(n1 numeric, n2 numeric, n3 numeric)
returns numeric as $cos$
declare
    max numeric;
begin
    if n1 >= n2 then
        max := max2(n1, n3);
    else
        max := max2(n2, n3);
    end if;
return max;
end; $cos$
language 'plpgsql';

```

```

select MAX3(4, 6, 1);

```

7.6.- (Voluntari) Fes la funció **LAT_A_TEXT**, tenint en compte que ha de quedar com en la taula **POBLACIONS**. Segurament la dificultat més gran serà aconseguir que apareguen les cometes després dels minuts i dels segons.

```

create or replace function LAT_A_TEXT(lat) returns text as $cos$
declare
    aux text;
begin
    if $1 is not null then
        aux := $1.g || '°' || $1.m || ''' || $1.s || ''' || $1.h;
    else aux := (null);
    end if;
return aux;
end; $cos$
language 'plpgsql';

```

```

select nom, latitud, LAT_A_TEXT(latitud), comarca from poblacions3;

```

7.7.- Fes una funció anomenada **POBLACIONS_ALTES** que accepti 2 paràmetres, el primer de tipus text que serà una comarca, i el segon numèric que serà una altura. Ha de traure les poblacions de la comarca del primer paràmetre que són més altes que el segon

paràmetre. Mostrarem el nom de la població i l'altura. Aquest podria ser el resultat en executar-se:

```
create or replace function POBLACIONS_ALTES(n text, h numeric) returns
void as $cos$
declare
    v_nom VARCHAR;
    v_alt numeric;
    cur cursor for select nom, altura from poblacions
                    where nom_c = n and altura > h
                    order by altura;
begin
    open cur;
    fetch cur into v_nom, v_alt;
    while v_nom is not null loop
        raise notice 'Nom: %. Altura: %', v_nom, v_alt;
        fetch cur into v_nom, v_alt;
    end loop;
    close cur;
end; $cos$
language 'plpgsql';
```

```
select poblacions_altes('Plana Alta', 700);
```

7.8.- Fes una funció anomenada **COMARQUES_NUMPOBLES** sense paràmetres que traga per pantalla les comarques ordenades alfabèticament amb el número de pobles de cadascuna

```
create or replace function COMARQUES_NUMPOBLES() returns void as $cos$
declare
    r record;
begin
    for r in select nom_c, count(nom) from poblacions
            group by nom_c
            order by nom_c
    loop
        raise notice 'Nom comarca: %. Número de pobles: %',
r.nom_c, r.count;
    end loop;
end; $cos$
language 'plpgsql';
```

```
select COMARQUES_NUMPOBLES();
```

7.9.- Fes una funció anomenada **COMARQUES_NUMPOBLES_NUMINSTITUTS** sense paràmetres que traga per pantalla les comarques ordenades alfabèticament amb el número de pobles de cadascuna i el número d'instituts. En la consulta tindrem dos dificultats:

- Hem d'agafar totes les poblacions, fins i tot les que no tenen institut
- Com que hem d'accedir als instituts, per a comptar els pobles haurem de comptar els pobles **distints**, i així si un poble té més d'un institut, no comptar-lo més d'una vegada

```

create or replace function COMARQUES_NUMPOBLES_NUMINSTITUTS() returns
void as $cos$
declare
    v_com varchar;
    v_pobles numeric;
    v_inst numeric;
    cur cursor for select poblacions.nom_c, count(distinct
poblacions.nom), count(instituts.cod_m)
        from poblacions left join instituts on poblacions.cod_m =
instituts.cod_m
        group by nom_c
        order by nom_c;
begin
    open cur;
    fetch cur into v_com, v_pobles, v_inst;
    while v_pobles is not null loop
        raise notice 'Nom comarca: %. Número de pobles: %. Número
instituts: %', v_com, v_pobles, v_inst;
        fetch cur into v_com, v_pobles, v_inst;
    end loop;
    close cur;
end; $cos$
language 'plpgsql';

```

```
select COMARQUES_NUMPOBLES_NUMINSTITUTS();
```

7.10.- Fes una funció anomenada **NUM_HABITANTS_COMARCA** que accepti un paràmetre de tipus text, i torne el número d'habitants d'eixa comarca

```

create or replace function NUM_HABITANTS_COMARCA(c text) returns
numeric as $cos$
declare
    habs numeric;
begin
    select SUM(poblacio) into habs from poblacions
        where c = nom_c;
    return habs;
end; $cos$
language 'plpgsql';

```

```
select NUM_HABITANTS_COMARCA('Plana Alta');
```

7.11.- Fer la funció **COMARQUES_NUMHABITANTS** sense paràmetres per a traure per pantalla totes les comarques i el número d'habitants. En la consulta has d'utilitzar obligatòriament la funció anterior

```

create or replace function COMARQUES_NUMHABITANTS() returns void as
$cos$
declare
    r record;
    aux numeric;
begin
    for r in select distinct nom_c from poblacions
        order by nom_c
        loop
            aux := NUM_HABITANTS_COMARCA(r.nom_c);
            raise notice 'Nom comarca: %. Número d'habitants: %',
r.nom_c, aux;
        end loop;
end;

```

```
end; $cos$  
language 'plpgsql';
```

```
select COMARQUES_NUMHABITANTS();
```

7.12.- Crear un trigger anomenat **TR_ALT_POS** que controle que l'altura d'una nova població siga estrictament positiva. La funció en la qual es basa es pot anomenar **ALT_POS**.

```
create or replace function ALT_POS() returns trigger as $cos$  
begin  
    if new.altura < 0 then  
        raise exception 'L'altura ha de ser positiva!';  
    end if;  
    return new;  
end; $cos$  
language 'plpgsql';
```

```
create trigger TR_ALT_POS before insert on poblacions  
for each row  
execute procedure alt_pos();
```

```
insert into poblacions values  
(1000, 'Grau de Castelló', 1500, 100, -50, '0°28''26"W',  
'39°45''00"N', 'C', 'Plana Alta');
```

7.13.- Modificar l'anterior per a que ho controle també quan es tracta d'una modificació.

```
drop trigger tr_alt_pos on poblacions;
```

```
create trigger TR_ALT_POS before insert or update on poblacions  
for each row  
execute procedure alt_pos();
```

```
update poblacions  
set altura = altura * -1  
where cod_m = '46001'
```

7.14.- Crear un trigger anomenat **TR_EXT_0_1000** que controle que l'extensió d'un municipi (població) estiga obligatòriament entre 0 i 1000, i a de ser sempre, tant si s'insereix una nova població com si es modifica. Però en aquesta ocasió, en compte de traure un error, el que farem serà modificar aquest valor: si és major que 1000, li

donarem el valor 1000, i si és negatiu li posarem 0. Ho aconseguirem modificant NEW.extensio, i com la funció del trigger torna sempre NEW, doncs agafarà el nou valor. Anomenem a la funció **EXT_0_1000**.

```
create or replace function EXT_0_1000() returns trigger as $cos$
begin
    if new.extensio < 0 then
        new.extensio = 0;
    elsif new.extensio > 1000 then
        new.extensio = 1000;
    end if;
    return new;
end; $cos$
language 'plpgsql';
```

```
create trigger TR_EXT_0_1000 before insert or update on poblacions
for each row
execute procedure EXT_0_1000();
```

```
insert into poblacions values
(1000, 'Grau de Castelló', 1500, -5, 5, '0°28''26"W',
'39°45''00"N', 'C', 'Plana Alta');
```

	149 cod_m ↑ ▾	ABC nom ▾	123 poblacio ▾	123 extensio ▾	123 altura ▾
1	1.000	Grau de Castelló	1.500	0	5
2	3.001	Atzúvia, l'	627	14,7	102
3	3.002	Agost	4.752	66,6	376
4	3.003	Agres	583	25,8	722

```
update poblacions
set extensio = 1200
where cod_m = '1000'
```

	149 cod_m ↑ ▾	ABC nom ▾	123 poblacio ▾	123 extensio ▾	123 altura ▾
1	1.000	Grau de Castelló	1.500	1.000	5
2	3.001	Atzúvia, l'	627	14,7	102
3	3.002	Agost	4.752	66,6	376
4	3.003	Agres	583	25,8	722

```
delete from poblacions
where cod_m = '1000'
```

7.15.- VOLUNTARI. En la taula POBLACIONS3 tenim controlat que la latitud introduïda siga correcta per mig del tipus lat, però no en la taula POBLACIONS, on és de tipus VARCHAR(50) i per tant es podria introduir una latitud incorrecta molt fàcilment. Crea un trigger que controle que quan s'introdueix o es modifica la **latitud** de **POBLACIONS** siga correcta. Per a això

- Els caràcters 1 i 2 han de ser els **graus**, que han d'estar entre 00 i 90
- El caràcter 3 ha de ser °
- Els caràcters 4 i 5 formen els **minuts**, i han d'estar entre 00 i 59
- El caràcter 6 ha de ser '
- Els caràcters 7 i 8 formen els **segons**, i han d'estar entre 00 i 59
- El caràcter 9 ha de ser ''
- El caràcter 10 ha de ser N o S
- Si no s'acompleix alguna de les restriccions anteriors, ha d'eixir un error dient que la latitud ha d'estar entre 00°00'00"N i 90°00'00"N , o entre 00°00'00"S i 90°00'00"S

```
create or replace function control_lat() returns trigger as $cos$
declare
    g varchar(2);
    m varchar(2);
    s varchar(2);
    p varchar;
begin
    g := SUBSTR(new.latitud, 1, 2);
    m := SUBSTR(new.latitud, 4, 2);
    s := SUBSTR(new.latitud, 7, 2);
    p := SUBSTR(new.latitud, 10, 1);
    if to_number(g, '00') < 0 or
       to_number(g, '00') > 90 or
       SUBSTR(new.latitud, 3, 1) != '°' or
       to_number(m, '00') < 0 or
       to_number(m, '00') > 59 or
       SUBSTR(new.latitud, 6, 1) != "'" or
       to_number(s, '00') < 0 or
       to_number(s, '00') > 59 or
       SUBSTR(new.latitud, 9, 1) != '"' or
       (p != 'N' and p != 'S') or
       (g = '90' and (m != '00' or s != '00'))
    then
        raise exception 'La latitud ha d'estar entre 00°00'00"N i
90°00'00"N, o entre 00°00'00"S i 90°00'00"S.';
    end if;
    return new;
end; $cos$
language 'plpgsql';
```

```
create trigger tr_control_lat before insert or update on poblacions
for each row
execute procedure control_lat();
```

```
insert into poblacions values
(1000, 'Grau de Castelló', 1500, 5, 5, '0°28'26"W',
'90°01'00"S', 'C', 'Plana Alta');
```

```
delete from poblacions
where cod_m = '1000'
```

7.16.- Crear els dos **operadors de comparació** que quedaven per al tipus **lat**: < i <=.

Primer haureu de crear les 2 funcions que queden: **menor(lat,lat)** i **menor_igual(lat,lat)**

Menor <

```
CREATE OR REPLACE FUNCTION menor (lat,lat) RETURNS bool AS '
BEGIN
    RETURN NOT major($1,$2);
END; '
LANGUAGE plpgsql;
```

```
CREATE OPERATOR <
    (LEFTARG = lat,
    RIGHTARG = lat,
    PROCEDURE = menor,
    COMMUTATOR = > ,
    NEGATOR = >= );
```

```
SELECT * FROM POBLACIONS3
WHERE latitud < ' (N,40,0,0) '
ORDER BY NOM;
```

Menor igual <=

```
CREATE OR REPLACE FUNCTION menor_igual (lat,lat) RETURNS bool AS '
BEGIN
    RETURN NOT major($1,$2) or igual($1, $2);
END; '
LANGUAGE plpgsql;
```

```
CREATE OPERATOR <=
    (LEFTARG = lat,
    RIGHTARG = lat,
    PROCEDURE = menor_igual,
    COMMUTATOR = >= ,
    NEGATOR = > );
```

```
SELECT * FROM POBLACIONS3
WHERE latitud <= ' (N,39,59,10) '
ORDER BY NOM;
```

	Grilla		ABC nom	latitud				ABC comarca	
				ABC h	123 g	123 m	123 s		
1	Texto		Castelló de la Plana	N	39	59	10	Plana Alta	
2			Higueras	N	39	59	1	Alt Palància	
3			Toro, el	N	39	58	58	Alt Palància	
4			Pavías	N	39	58	23	Alt Palància	
5			Villamalur	N	39	57	52	Alt Millars	
6			Onda	N	39	57	41	Plana Baixa	

7.17.- Crear la funció d'agregat **MIN** per al tipus de dades **lat**.

```
CREATE OR REPLACE FUNCTION MIN2(lat1 lat,lat2 lat) RETURNS lat AS
$cos$
BEGIN
    IF lat2 < lat1 THEN RETURN lat2;
    ELSE RETURN lat1;
    END IF;
END; $cos$
LANGUAGE plpgsql;
```

```
CREATE AGGREGATE MIN (
    BASETYPE = lat,
    SFUNC = MIN2,
    STYPE = lat,
    INITCOND = ' (N,90,0,0) ' );
```

La població amb la latitud mínima:

```
SELECT MIN(latitud)
FROM POBLACIONS3;
```

La població amb menor latitud de cada comarca:

```
SELECT comarca,MIN(latitud)
FROM POBLACIONS3
GROUP BY comarca
ORDER BY MIN(latitud);
```