Funkcje PostgreSQL

Schemat funkcji

CREATE OR REPLACE FUNCTION *func_name(*

    arg1 arg1_datatype)

RETURNS *some_type | setof sometype | TABLE (..) AS*

$$

BODY of function

$$

LANGUAGE *language_of_function*

BODY dla plpgsql:

declare deklaracje

begin

    instrukcje

end;

Poziom zmienności funkcji

- VOLATILE - funkcja może wykonywać wszelkie modyfikacje na bazie danych. Jej zwracana wartość może się zmieniać w sposób dowolny
- STABLE - funkcja nie może modyfikować bazy danych, dla tych samych argumentów, oraz nie zmodyfikowanej odpytywanej tabeli zwraca te same wyniki.
- IMMUTABLE - funkcja nie może modyfikować bazy danych, dla tych samych argumentów, niezależnie od stanu bazy danych zwraca te same wyniki.

Funkcje SQL

```
--Funkcja zwracająca tabelę
CREATE FUNCTION blok_2.geom_info(param_name varchar)
RETURNS TABLE (id_b int, typ_b varchar(50), description text) AS
$$
SELECT id_b, typ_b, st_astext(geom_b) FROM blok_2.budynki WHERE typ_b = $1;
$$
LANGUAGE 'sql' STABLE;

--$1 – pierwszy argument
select * from blok_2.geom_info('budynek mieszkalny')


--Funkcja zwracająca record
CREATE FUNCTION blok_2.geom_info_rec(param_name varchar, OUT id_b int
, OUT typ_b varchar, OUT description text)
RETURNS SETOF record AS
$$
SELECT id_b, typ_b, st_astext(geom_b) FROM blok_2.budynki WHERE typ_b = $1;
$$
LANGUAGE 'sql' STABLE;

select * from blok_2.geom_info_rec('budynek mieszkalny')
```

Funkcja plpgsql zwracająca obiekt

```
CREATE OR REPLACE FUNCTION gis.hexagon(
   width integer,
   x double precision,
   y double precision
)
RETURNS geometry AS
$BODY$
DECLARE
      height float := width*2/sqrt(3);
      polygon_string varchar:='polygon((' ||        x ||' '|| y                ||' , ' ||
                                          x+width/2 ||' '|| y+height/4     ||' , ' ||
                                          x+width/2 ||' '|| y+height*3/4  ||' , ' ||
                                                x ||' '|| y+height         ||' , ' ||
                                           x-width/2 ||' '|| y+height*3/4  ||' , ' ||
                                          x-width/2  ||' '|| y+height/4     ||' , ' ||
                                                   x  ||' '|| y                ||   '))';
BEGIN
      RETURN st_geomfromtext(polygon_string,2180);
END;
$BODY$
 LANGUAGE plpgsql STABLE
```

Funkcja zwracająca tablicę grid

```
CREATE OR REPLACE FUNCTION gis.rect_grid(
	in gridres integer,
	in xmin float,
	in ymin float,
	in xmax float,
	in ymax float,
	in srid integer
)
RETURNS TABLE(gid bigint, geom geometry) AS
$$
DECLARE
xcount float :=ceil(abs(xmax-xmin)/gridres+1);
ycount float :=ceil(abs(ymax-ymin)/gridres+1);
BEGIN
RETURN QUERY
	select row_number() OVER () as gid,
		ST_SetSRID(ST_MakeBox2d(
				ST_Point(xmin+(x-1)*gridres, ymin+(y-1)*gridres),
				ST_Point(xmin+x*gridres, ymin+y*gridres)),srid) as geom
	from
	generate_series(1,xcount::int) as x(x) cross join
	generate_series(1,ycount::int) as y(y);
END;
$$
LANGUAGE 'plpgsql' STABLE;
select * from gis.rect_grid(100,360664,491265,363854,493674,2180)
```

Funkcja zwracająca rekordy dla grid

```
CREATE OR REPLACE FUNCTION blok_2.grid(
	in resolution integer,
	in geom_name varchar,
	in sche varchar,
	in tbl varchar,
	in srid integer,
	out grid_x integer,
	out grid_y integer,
	out geom geometry
)
RETURNS SETOF record AS
$$
DECLARE
	rec record;
	sql text;
BEGIN
sql :='
with zasieg as
--wyznaczenie zasięgu mapy na podstawie zasięgu warstwy
(select ST_SetSRID(CAST(ST_Extent('||quote_ident(geom_name)||')as geometry),'||srid||')as geom,
'||resolution||' as gridres
from '||quote_ident(sche)||'.'||quote_ident(tbl)||'
),'
```

```
--wyznaczenie rozmiaru grid
'grid_dim as
(select '
--ilość oczek o wielkości resolution
'CAST((ST_XMax(geom)- ST_XMin(geom))as integer)/gridres+1 as xcount,
CAST((ST_YMax(geom)- ST_YMin(geom))as integer)/gridres+1 as ycount,'
--wyznaczenie współrzędnych zaięgu
'ST_XMin(geom)as xmin,
ST_XMax(geom)as xmax,
ST_YMin(geom)as ymin,
ST_YMax(geom)as ymax
from  zasieg
),'
--generowanie gridu
'grid as (
select x, y,
ST_SetSRID(ST_MakeBox2d(
	ST_Point(xmin+(x-1)*gridres, ymin+(y-1)*gridres),
	ST_Point(xmin+x*gridres, ymin+y*gridres)
	),2180) as grid_geom
from
(select generate_series(1,xcount)from grid_dim) as x(x)
cross join
(select generate_series(1,ycount)from grid_dim) as y(y)
cross join grid_dim
cross join zasieg
)
```

```
select grid.x as x, grid.y as y, ST_Intersection(zasieg.geom, grid_geom)as geom
from
zasieg
inner join
grid
on (ST_Intersects(zasieg.geom, grid_geom))';


for rec in execute sql
LOOP
	grid_x := rec.x;
	grid_y := rec.y;
	geom := rec.geom;
	RETURN NEXT;
END LOOP;


RETURN;
END;
$$
LANGUAGE 'plpgsql' STABLE;
```

Funkcja plpython: geokodowanie

```
copy adresy(user_id,ulica,numer,kod,miejscowosc,data_wyg) from 'C:\adresy.csv' with delimiter as ';' CSV HEADER;

--przy pomocy application stack bulidera dodać rozszerzenie pythona edb language pack w add-ons, tools and utilites
--następnie wykonać
create extension plpython3u;

--dodać funkcję w wewnątrz pytona może być potrzebna biblioteka geopy
CREATE OR REPLACE FUNCTION google_geocode( in addr text, out lon double precision,out lat double precision)
RETURNS record
AS
$$
from geopy.geocoders import GoogleV3
geoc = GoogleV3()
address, (latitude, longitude) = geoc.geocode(addr)
return (longitude, latitude)
$$
language 'plpython3u';
--wstawianie gemetrii na podstawie adresów - gotowe polecenie:
update adresy
set
geom=(select geom from
(
select id as idb, ST_Transform(ST_setsrid(ST_point((g).lon,(g).lat),4326),2180) as geom
from (select id, google_geocode(concat_ws(' ',ulica,numer,kod,miejscowosc))as g from adresy
)as foo)as b where id=idb)
```