Tabele wykorzystywane w przykładach

create table departamenty(dep_id serial primary key, nazwa_departamentu text); create table pracownicy (emp_id serial primary key, imie text, nazwisko text, pensja numeric, data_zatrudnienia date, dep_id integer references departamenty(dep_id)); insert into departamenty values(1,'IT'); insert into departamenty values(2,'Dział sprzedaży'); insert into pracownicy values (1,'Jan','Kowalski',5500,current_date-1000,1); insert into pracownicy values (2,'Krzysztof','Nowak',4750,current_date-100,1); insert into pracownicy values (3,'Marek','Bobek',3200,current_date-22,1); insert into pracownicy values (4,'Grażyna','Typowa',1750,current_date-1500,2); insert into pracownicy values (5,'Anna','Zgrabna',1980,current_date-170,2); insert into pracownicy values (6,'Mirek','Typowy',1300,current_date-1100,2); insert into pracownicy values (7,'Halina','Przybyła',2100,current_date-760,2);	Tabele używane w wybranych przkładach

Bloki anonimowe

DO $$ begin end; $$	Najprostszy blok anonimowy
DO $$ begin end; $$ **LANGUAGE plpgsql;**	Alternatywa z dodaniem LANGUAGE – nie jest to wymagane
do $$ begin end; $$ **language 'plpgsql';**	Jak wyżej ale inna forma
DO $$ begin **raise notice 'hello world';** end; $$ LANGUAGE plpgsql;	Prosty komunikat wypisywany na ekran
Do $$ declare **x integer:=60;** begin raise notice 'x jest równe %',x; end; $$	Używanie zmiennych – deklaracja i przypisanie wartości

do $$ declare **x pracownicy.emp_id%type;** begin end; $$	Deklaracja zmiennej typu opartego o kolumnę tabeli
do $$ declare x integer:=90; begin **declare** **x integer:=60;** **begin** **raise notice 'x jest równe %',x;** **end;** end; $$ language plpgsql;	Blok zagnieżdżony

do
$$
declare
x integer:=9;
y integer:=8;
z integer;
begin
z=x+y;
raise notice 'z=%',z;
end;
$$
language plpgsql;
	Operacje arytmetyczne

do $$ declare **x integer:=2;** **y integer:=3;** **z numeric;** begin z:=x/y; raise notice '%',z; end; $$ do $$ declare **x numeric:=2;** **y numeric:=3;** **z numeric;** begin z:=x/y; raise notice '%',z; end; $$	Automatyczna konwersja w tle. Porównaj wyniki z obu bloków. Różnica w typie danych wejściowych.
do $$ declare wynik numeric; begin **select avg(pole1) into wynik from tabelka;** raise notice 'wynik=%',wynik; end; $$	Wczytywanie wartości z bazy do zmiennej typu prostego
do $$ declare wynik numeric; begin select avg(pole1) into wynik from tabelka; **raise notice 'wynik=%',trunc(wynik);** end; $$	Stosowanie funkcji SQL na zmiennych w pl/pgSQL

Typy złożone

do $$ declare **w pracownicy%rowtype;** begin end; $$	Zmienna typu wierszowego
do $$ declare **w pracownicy;** begin end; $$	Alternatywa do powyższego (nie trzeba podawać %rowtype)
create or replace function przyjmewiersza(**w pracownicy**) returns void as $$ begin raise notice 'ok! '; end; $$ language plpgsql;	Przyjmowanie zmiennej typu złożonego przez parametr. wersja (w pracownicy%rowtype) już nie przechodzi
do $$ declare **generyczny record;** begin select * into generyczny from pracownicy where emp_id=1; raise notice '%',generyczny; end; $$	Typ złożony generyczny – odpowiednik typu RECORD w Oracle.
create table tabelka (pole1 integer, pole2 numeric, pole3 text); insert into tabelka values (1,1/3,'lorem ipsum'); select * from tabelka;	„Ręczne” przypisywanie wartości do pól typu złożonego. Typ złożony oparty o budowę tabeli

do $$ declare wiersz tabelka%rowtype; begin **wiersz.pole1:=1;** **wiersz.pole2:=4/5;** **wiersz.pole3:='Copacabana';** raise notice '% % %',wiersz.pole1,wiersz.pole2,wiersz.pole3; end; $$	

do
$$
declare
wiersz tabelka%rowtype;
begin
wiersz.pole1:=1;
wiersz.pole2:=4/5;
wiersz.pole3:='Copacabana';
raise notice '%',wiersz;
end;
$$
	Wypisywanie zawartości całego wiersza. Zauważ że nie trzeba wypisywać wszystkich pól jak w Oracle.

Instrukcje warunkowe

do
$$
declare
wzrost numeric:=1.76;
masa numeric:=84;
bmi numeric;
begin
bmi:=trunc(masa/power(wzrost,2),2);
raise notice 'bmi=%',bmi;
if(bmi<18.5) then raise notice 'niedowaga';
elsif (bmi<25) then raise notice 'OK';
else raise notice 'grubo';
end if;
end;
$$
	Instrukcja warunkowa IF. Jeśli warunek nie jest spełniony, sprawdzany jest kolejny.

do $$ declare x integer:=2; begin **case x** **when 1 then raise notice 'x równe 1';** **when 2 then raise notice 'x równe 2';** **when 3 then raise notice 'x równe 3';** **end case;** end; $$	Instrukcja warunkowa CASE. Warunki oparte o wartości pojawiające się w zmiennej X.

do
$$
declare
x integer:=2;
y integer:=10;
begin
case
when x=1 then raise notice 'x równe 1';
when x=1 and y<10 then raise notice 'x równe 1 i y mniejsze od 10';
else raise notice 'coś innego';
end case;
end;
$$
	Instrukcja CASE. Różne warunki

do $$ declare x integer:=2; begin **case x** **when 1,2,50 then** raise notice 'x równe 1... albo 2... albo nie wiem...'; when 3 then raise notice 'x równe 3'; when 4 then raise notice 'x równe 4'; else raise notice 'coś innego'; end case; end; $$	W PostgreSQL możemy rozdzielając przecinkiem wymienić kilka wartości dla zmiennej w warunku.

Pętle

do
$$
declare
x integer:=1;
begin
loop
raise notice '%',x;
exit when x=10;
x:=x+1;
end loop;
end;
$$
	Pętla z użyciem klauzuli exit when

do
$$
declare
x integer:=1;
begin
loop
raise notice '%',x;
IF(X=10) THEN EXIT;
END IF;
x:=x+1;
end loop;
end;
$$
	Warunkowe wywołanie EXIT

do
$$
declare
x integer:=1;
begin
while x<=10 loop
raise notice '%',x;
x:=x+1;
end loop;
end;
$$
	Pętla while. Pętla będzie się wykonywać tak długo jak długo prawdziwy jest warunek określony w klauzuli WHILE

do $$ begin **for x in 1..10 loop** raise notice '%',x; **end loop;** end; $$	Pętla for. Wykona się 10 razy.

do
$$
declare
z1 integer:=1;
z2 integer:=10;
begin
for x in z1..z2 loop
raise notice '%',x;
end loop;
end;
$$
	Pętla for z użyciem zmiennych jako punktów granicznych

do $$ begin **for x in reverse 10..1 loop** raise notice '%',x; **end loop;** end; $$	Pętla for z malejącym iteratorem. Zauważ że podobnie jak w Oracle wprowadzamy klauzulę REVERSE, jednak różnica polega na tym że tutaj jako pierwszą wartość graniczą podajemy tę większą.
do $$ begin **for x in 2..10 by 2 loop** raise notice '%',x; **end loop;** end; $$	Pętla for z przeskakiwaniem o dwie wartości :)
DO $$ DECLARE X INTEGER:=1; Y INTEGER:=1; BEGIN **LOOP** **LOOP** RAISE NOTICE 'Y=% X=%',Y,X; EXIT WHEN X=10; X:=X+1; **END LOOP**; X:=1; EXIT WHEN Y=10; Y:=Y+1; **END LOOP;** END; $$	Pętle zagnieżdżone

do
$$
declare
x integer:=1;
begin
loop
raise notice '%',x;
exit when x=10;
x:=x+1;
continue when x<=5;
raise notice 'hurra!';
end loop;
end;
$$
	Instrukcja CONTINUE z warunkiem od którego spełnienia zależy wykonanie następnych instrukcji

Wczytywanie danych z bazy i kursory

do $$ declare **wiersz tabelka%rowtype;** begin **select * into wiersz from tabelka;** raise notice '%',wiersz; end; $$	Wczytanie do typu wierszowego danych z tabeli – oczywiście musi to być jeden wiersz :)
do $$ declare x integer:=3; wiersz pracownicy; begin **select * into wiersz from pracownicy where emp_id=x;** end; $$	Wczytanie jednego wiersza z użyciem zmiennej w zapytaniu.
do $$ declare x integer:=30; wiersz pracownicy; begin select * into wiersz from pracownicy where emp_id=x; **if not found then** **raise exception 'nie ma pracownika o id=%',x;** **end if;** raise notice '%', wiersz; end; $$	Próba odczytania nieistniejącego wiersza. Wyjatku zadnego nie rzuca mimo ze nie ma takiego pracownika,ale mozna taki brak wyłapać i rzucić własnym wyjątkiem
do $$ declare **k1 refcursor;** begin null; end; $$	Deklaracja kursora dynamicznego – takiego którego treść zapytania podajemy w czasie wykonania.

do $$ declare **k2 cursor is select * from pracownicy;** begin null; end; $$	Deklaracja kursora ze z góry zdefiniowanym zapytaniem.
do $$ declare **k3 cursor(x integer) is select * from pracownicy where dep_id=x;** begin null; end; $$	Deklaracja kursora ze z góry zdefiniowanym zapytaniem używającym parametru
do $$ declare **k refcursor;** begin **open k for select * from pracownicy;** end; $$	Otwarcie kursora dynamicznego
do $$ declare **k refcursor;** x integer:=2; begin **open k for select * from pracownicy where dep_id=x;** end; $$	Otwarcie kursora dynamicznego z użyciem zapytania z parametrem.
do $$ declare **k refcursor;** x integer:=2; begin **open k for execute 'select * from pracownicy where dep_id='\|\|x;** end; $$	Otwarcie kursora dynamicznego z dynamicznie tworzonym zapytaniem (kursor w SQL dynamicznym)

do $$ declare **k refcursor;** x integer:=2; begin **open k for execute 'select * from pracownicy where dep_id=$1' using x;** end; $$	Otwarcie kursora dynamicznego z dynamicznie tworzonym zapytaniem z użyciem klauzuli using
do $$ declare **k2 cursor is select * from pracownicy;** begin **open k2;** end; $$	Otwarcie zwykłego kursora
do $$ declare **k2 cursor(x integer) is select * from pracownicy where dep_id=x;** begin **open k2(2);** end; $$	Otwarcie kursora sparametryzowanego
do $$ declare k1 cursor is select * from pracownicy; **w pracownicy%rowtype;** begin open k1; **fetch k1 into w;** raise notice '%',w; close k1; end; $$	Wczytanie jednego wiersza z kursora i wypisanie go na konsoli. Dla przyzwyczajonych do PL/SQL w Oracle - nie działa taka konstrukcja jak k%rowtype. Alternatywnie można zmienną przeznaczoną na wiersz zdeklarować jako : w record;

do $$ declare k1 cursor is select * from pracownicy; **w record;** begin open k1; **fetch k1 into w;** raise notice '%',w; close k1; end; $$	Zmienna przeznaczona do przechowywania wiersza zdeklarowana jako niezdefiowany typ złożony. Odczyt do takiej zmiennej.
do $$ declare eid integer; imie text; nazwisko text; k1 cursor is select * From pracownicy; begin **open k1;** **fetch k1 into eid,imie,nazwisko;** **raise notice '% % %',eid,imie,nazwisko;** **close k1;** end; $$	Wczytanie zawartości wiersza do kolejnych zmiennych . Ciekawostka – ilość podanych po into zmiennych nie musi odpowiadać ilości kolumn w zapytaniu – wartości podawane są wg kolejności. Jeśli zmiennych będzie za mało to po prostu dostaniemy pierwsze X kolumn.
do $$ declare k1 cursor is select * from pracownicy; w pracownicy%rowtype; begin open k1; **move k1;** fetch k1 into w; raise notice '%',w; close k1; end; $$	Przesunięcie o jeden wiersz w kursorze – bez jego odczytywania.
do $$ declare k1 cursor is select * from pracownicy; w pracownicy%rowtype; begin open k1; **move forward 2 from k1;** fetch k1 into w; raise notice '%',w;	Przesunięcie o wskazaną liczbę wierszy w kursorze bez ich odczytu.

close k1; end; $$	
do $$ declare k1 cursor is select * from pracownicy; w pracownicy%rowtype; begin open k1; **move last from k1;** fetch k1 into w; raise notice '%',w; close k1; end; $$	Przejście za ostatni wiersz kursora – fetch zwraca pusty wiersz.
do $$ declare k cursor is select * From pracownicy; w pracownicy%rowtype; begin open k; **move first from k;** fetch k into w; raise notice '%',w; close k; end; $$	Przesunięcie się do pierwszego wiersza w kursorze.
do $$ declare k1 cursor is select * from pracownicy; w pracownicy%rowtype; begin open k1; **move forward 4 from k1;** **move forward -2 from k1;** fetch k1 into w; raise notice '%',w; close k1; end; $$	Przesunięcie się najpierw o 4 wiersze do przodu, a następnie o 2 do tyłu w kursorze.

Wyjątki

do $$ declare x numeric; begin x:=1/0; **exception** **when division_by_zero then** raise notice 'nie można dzielić przez zero'; end; $$	Wyłapywanie wystąpień wyjątków

Funkcje

create or replace function nowa() returns numeric as $$ begin **return 2*7;** end; **$$ language plpgsql;**	Najprostsza funkcja zwracająca liczbę (bez parametrów wejściowych). Końcówka „language plpgsql” musi się tutaj znajdować.
select nowa();	Wywołanie funkcji bezparametrowej
do $$ declare wynik integer; begin **wynik:=dawaj10();** raise notice '%', wynik; end; $$	Wywołanie funkcji bezparametrowej z kodu plpgsql
create or replace function pirazydrzwi() returns numeric as **'select 3.14'** **language 'sql';** select pirazydrzwi();	Funkcja będąca po prostu opakowaniem zapytania w SQL i jej wywołanie. Zwróć uwagę na language.
create or replace function funkcjasql() returns numeric as **'select x from tujestliczba'** **language 'sql';** select funkcjasql();	Przykład zbliżony do powyższego, wraz z wywołaniem. Tym razem dane pochodzą jednak z tabeli
create or replace function pomnoz(x numeric,y numeric) returns numeric as **'select x*y'** **language 'sql';** select pomnoz(3,2);	Funkcja będąca opakowaniem do SQLowego wywołania będącego opakowaniem do zwykłej operacji arytmetycznej :)
create or replace function podziel(**numeric,numeric**) returns numeric as **'select $1/$2'** **language 'sql';** select podziel(1,2);	Można też nie podawać nazw parametrów funkcji – ale potem odnosimy się do nich po kolejności – patrz $1 i $2

create or replace function dodajDoX(y integer) returns integer as ' **update domanipulacji set x=x+y;** **select x from domanipulacji;** ' **language 'sql';** select dodajdox(4);	Funkcja może też zwracać wartość bezpośrednio z zapytania.
create or replace function zrobDDL() returns integer as ' **create table hello (x integer);** **select 1'** **language 'sql';** select zrobddl();	Można też zaszyć operację DDL.
create or replace function nic() **returns void** as '' language 'sql'; select nic();	Funkcja może też tak jak w C zwracać NIC. Deklarujemy wtedy typ zwracany jako VOID
create or replace function nicniezwracam() **returns void** as $$ begin raise notice 'a kuku'; end; **$$ language plpgsql;** select nicniezwracam();	Nic nie zwracająca funkcja innym sposobem (zwróć uwagę na language).
create or replace function srednie_zarobki() returns numeric as $$ declare wynik numeric; begin **select trunc(avg(pensja),2) into wynik from pracownicy;** raise notice 'srednie zarobki w całej firmie = %',wynik; **return wynik;** end; $$ language plpgsql; select srednie_zarobki();	Funkcja bezparametrowa zwracająca średnią zarobków w calej firmie.

create or replace function **srednie_zarobki(did integer)** returns numeric
as
$$
declare
wynik numeric;
begin
select trunc(avg(pensja),2) into wynik from pracownicy where dep_id=did;
raise notice 'srednie zarobki w departamencie % = %',did,wynik;
return wynik;
end;
$$ language plpgsql;
 	
select srednie_zarobki();
select srednie_zarobki(1); --przeciążanie jak widać jest możliwe
select srednie_zarobki(2);
select srednie_zarobki(3); -- null dla nieistniejącego departamentu
	Funkcja z parametrem o nazwie takiej jak już istniejąca bezparametrowa. W PostgreSQL jest możliwe przeciążanie funkcji bez użycia pakietów (pakietów swoją drogą tutaj nie ma).

create or replace function dawajwiersza(x integer) **returns pracownicy** as $$ declare **wiersz pracownicy%rowtype;** begin select * into wiersz from pracownicy where emp_id=x; **return wiersz;** end; $$ language plpgsql; select dawajwiersza(1);	Funkcja zwracająca typ złożony – wersja z returns pracownicy %rowtype nie działa.
create function wynikzapytania() **returns table (imie_nazwisko text,roczne_zarobki numeric)** as $$ begin **return query select imie\|\|' '\|\|nazwisko,pensja*12 roczne from pracownicy;** end; $$ language plpgsql; select wynikzapytania();	Funkcja zwracająca tabelę (format nieco zbliżony do jsona).

create or replace function parametr_zewnetrzny(x numeric,**y out numeric**) as $$ begin y:=x*2; end; $$ language plpgsql;	Parametr typu OUT w funkcji. Jeśli jest parametr out to nie musi być użyta klauzula returns w nagłówku deklaracji.
create or replace function dawajpracownikow(did integer) **returns setof pracownic**y as $$ declare w pracownicy%rowtype; begin **for w in select** * from pracownicy where dep_id=did order by pensja desc loop raise notice '%', w; **return next w;** end loop; **return;** end; $$ language plpgsql; select dawajpracownikow(1);	Fukcja zwracająca zestaw wierszy – działanie zbliżone do funkcji strumieniowych w Oracle. Zwracane dane są w formacie zbliżonym do JSONa

Dynamiczny SQL

$$ declare x integer; tab text:='pracownicy'; begin **execute 'select count(*) from '\|\|tab into x;** raise notice 'x=%',x; end; $$	Najprostszy przykład z konkatenacją i wczytaniem wyniku zapytania do zmiennej.

do
$$
declare
x integer;
tab text:='pracownicy';
did integer:=2;
begin
execute 'select count(*) from '\|\|tab\|\|' where dep_id=$1' into x using did;
raise notice 'x=%',x;
end;
$$
	Użycie klauzuli using – podanie zmiennej.

do $$ declare x integer; begin update pracownicy set pensja=pensja+100; **get diagnostics x:=ROW_COUNT;** raise notice 'x=%',x; end; $$	coś jak k%rowcount w Oracle

Wyzwalacze

CREATE or replace FUNCTION podniesienie_pensji() **RETURNS trigger** AS $$ BEGIN raise notice 'jeśli zmienia się departament to podnosimy pensję'; if(new.dep_id<>old.dep_id) then NEW.pensja:=old.pensja+100; raise notice '%',new; end if; **RETURN NEW;** END; $$ LANGUAGE plpgsql;	Deklaracja funkcji będącej podstawą dla wyzwalacza. Funkcja taka musi zwracać element typu wierszowego odpowiadającego budową wierszowi tabeli na którą zakładamy wyzwalacz. Jeśli wyzwalacz wierszowy typu BEFORE zwróci null zamiast wiersza, operacja która uruchomiła wyzwalacz nie zostaje wykonana. Zmieniając cokowiek w zmiennej NEW zmieniamy w aktualizowanym lub dodawanym wierszu.
CREATE TRIGGER tzd **BEFORE UPDATE ON pracownicy** **FOR EACH ROW** EXECUTE PROCEDURE podniesienie_pensji();	Założenie wyzwalacza wierszowego. Nie ma tutaj konstrukcji CREATE OR REPLACE (jak w Oracle). Wyzwalacze zarówno wierszowe jak i obiektowe mogą być zarówno BEFORE jak i AFTER.
CREATE TRIGGER tzd2 **BEFORE UPDATE ON pracownicy** **EXECUTE PROCEDURE** podniesienie_pensji();	Założenie wyzwalacza obiektowego.
drop trigger tzd on pracownicy;	Kasowanie wyzwalacza

INNE

do $$ begin **perform * from pracownicy;** end; $$	wykonanie SQLa ale bez zwracania wyników - słowo PERFORM zamiast SELECT
do $$ begin **perform wywolaj_mnie();** end; $$	Wywołanie funkcji bez odbierania tego co zwraca (nawet jeśli coś zwraca, a nie musi)
DO $$ DECLARE X INTEGER:=1; BEGIN RAISE NOTICE 'przed blokiem'; **<<caly_blok>>** begin RAISE NOTICE 'przed pętlą'; loop if(x=10) then **exit caly_blok;** end if; x:=x+1; end loop; RAISE NOTICE 'po pętli'; end; RAISE NOTICE 'po bloku'; END; $$	Wyskoczenie z zagnieżdżonego bloku
copy (select * from pracownicy order by nazwisko) to 'c:\temp\pracownicy.csv';	Pisanie do pliku
copy (select * from pracownicy order by nazwisko) to 'c:\temp\pracownicy.csv' with csv delimiter ';';	Pisanie do pliku ze zmianą rozdzielacza kolumn

do $$ begin copy (select * from pracownicy order by nazwisko) to 'c:\temp\pracownicy2.csv'; end; $$	Przykład pisania do pliku z bloku kodu
DO $$ DECLARE X INTEGER:=60; BEGIN RAISE NOTICE 'HELLO WORLD'; --ZWYKŁY KOMUNIKAT RAISE NOTICE 'HELLO WORLD ZMIENNA=%',X; -- ZWYKŁY KOMUNIKAT ZE ZMIENNĄ RAISE 'OMG!'; -- RZUCENIE BŁĘDEM O WSKAZANYM KOMUNIKACIE. PRZERYWA WYKONYWANIE PROGRAMU --RAISE EXCEPTION 'NIE MA TAKIEGO ID --> %', X USING HINT = 'SPRAWDZ PODANE ID'; -- WYJĄTEK Z PODPOWIEDZIĄ. PRZERYWA WYKONYWANIE PROGRAMU RAISE NOTICE 'COSTAM DALEJ'; --ZWYKŁY KOMUNIKAT RAISE division_by_zero; RAISE 'Duplicate user ID: %', x USING ERRCODE = '23505'; END; $$	Różne rodzaje komunikatów