Sekwencje, widoki i PL/pgSQL

Sekwencje

Sekwencja jest to specjalna tabele, dla której dostępne są funkcje umożliwiające automatyczne zwiększanie/zmniejszanie (zgodnie z ustaloną regułą) przechowywanej wartości. W ten sposób generowany jest ciąg (arytmetyczny), który można używać w innych tabelach.

Przykładową sekwencję zaczynającą się od wartości 1 i zwiększającą się o 10 można zdefiniować następująco:

```
CREATE SEQUENCE foo START 1 INCREMENT BY 10;
```

Więcej możliwości opisanych jest na stronie dokumentacji. Po utworzeniu sekwencji, za pomocą funkcji *nextval*, *currval* i *setval* można czytać i modyfikować wartości sekwencji. Funkcje te opisane są tutaj.

```
SELECT nextval('foo');   -- zwiększa wartość sekwencji i zwraca nową wartość
SELECT currval('foo');   -- tylko zwraca aktualną wartość
SELECT setval('foo',13); -- ustawia nową wartość dla sekwencji
```

W Postgresie istnieje możliwość użycia typu serial w celu auto generowania wartości za pomocą odpowiedniej sekwencji.

Widoki (perspektywy)

Często zdarza się, iż specyficzne zapytanie jest często uruchamiane jako podzapytanie. Można wtedy wprowadzić na nie skrót, który z zewnątrz będzie wyglądał jak tabela (fizycznie tabelą nie jest). Taką możliwość dają widoki.

```
CREATE VIEW widok AS SELECT ...;
```

Programowanie w języku PLSQL

Poza zapytaniami SQL jest też możliwość napisania bardziej skomplikowanego kodu. Służy do tego język PL/SQL.

Podgląd istniejących języków (w Postgres) można dokonać następująco:

```
SELECT * FROM pg_language;
```

Poniżej jest przykład prostej funkcji napisanej w języku PL/SQL:

```
create or replace function  my_fun(a integer)
    returns integer as
$$
declare
    b       integer;
begin
    b = a + 1;
    return b;
end;
$$
language plpgsql;
```

Listę wszystkich funkcji otrzymujemy po wykonaniu komendy \df (lub \df+).

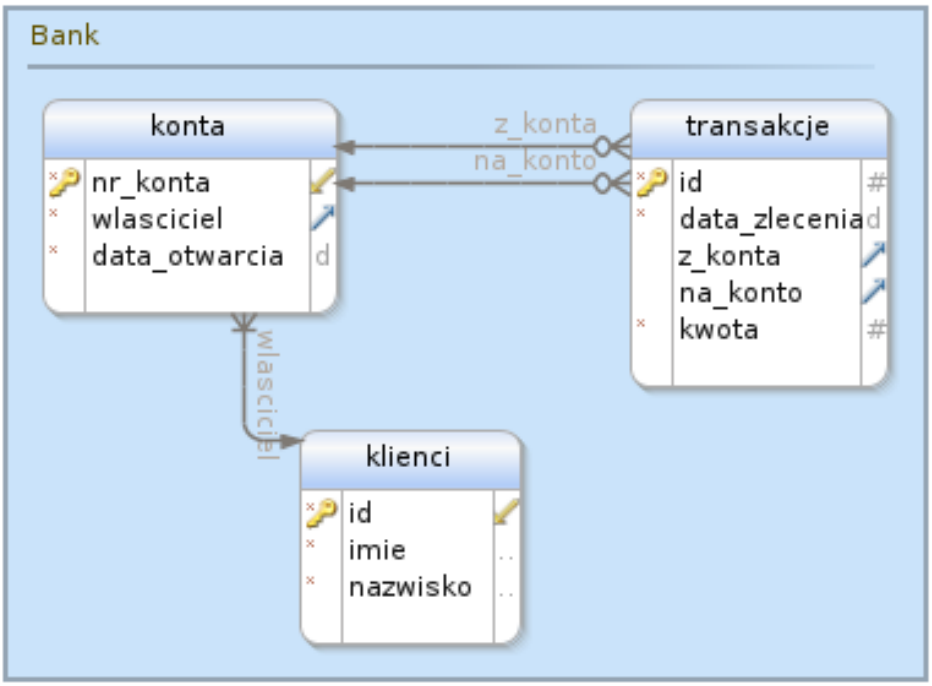
W przypadku gdy nasza funkcja pracuje na zadanej tabeli i używa zmiennych o typach zgodnych z typami kolumn tej tabeli, to możemy (bez ręcznego sprawdzania i przepisywania) odwołać się do odpowiednich typów za pomocą konstrukcji %TYPE i %ROWTYPE.

```
create or replace function  my_fun()
returns pracownicy.placa_pod%TYPE as
$$
declare
    v_prac pracownicy%ROWTYPE;
    p_prac pracownicy.placa_pod%TYPE;
begin
    SELECT * INTO v_prac FROM pracownicy
    WHERE etat = 'PROFESOR';
    SELECT MIN(placa_pod) INTO p_prac FROM pracownicy;
    return p_prac + v_prac.placa_pod ;
end;
$$
language plpgsql;
```

W języku PL/SQL można korzystać z pętli FOR, konstrukcji warunkowych typu IF-THEN-ELSE, zwracać wyniki całych zapytań SQL, zwracać też wiersz po wierszu jako zbiór danych lub też rzucać i łapać wyjątki. Wszystkie te rzeczy opisane są w dokumentacji.

Zadania

Wszystkie poniższe zadania dotyczą bazy bank.sql o następującym schemacie:



Uwaga: Zadanie 10a oraz 10b należy wysyłać poprzez umieszczenie rozwiązania odpowiednio w bloku *–ZAD10a* oraz *–ZAD10b*.

1. Utwórz sekwencję *seq*, której wartości przebiegają od 10 do 99 w interwale wielkości 1.

2. Korzystając z wartości sekwencji *seq* (załącz rozwiązanie zadania 1) jako klucza głównego, wstaw do tabeli *klienci* dwa nowe rekordy:

   - Anna Nowak
   - Jan Kowalski

3. Utwórz unikalny indeks na kolumnach *(z_konta, na_konto, data_zlecenia)* tabeli *transakcje*.

4. Utwórz widok *wplaty_wyplaty*, który zwraca liczbę wpłat oraz wypłat z każdego konta znajdującego się w bazie banku (gdy dla danego konta nie było żadnej wpłaty/wypłaty należy wypisać 0). Widok powinien zawierać trzy kolumny: *Konto*, *Ilosc wyplat* oraz *Ilosc wplat*. W przypadku gdy widok *wplaty_wyplaty* istnieje już w bazie, należy go dynamicznie podmienić. W rozwiązaniu nie wolno korzystać ze słowa kluczowego *drop*.

5. Korzystając z widoku *wplaty_wyplaty* z poprzedniego zadania (załącz rozwiązanie zadania 4) sprawdź, czy widoki są automatycznie modyfikowane wraz ze zmianą zawartości bazy danych. W tym celu wypisz zawartość widoku *wplaty_wyplaty*, utwórz nową wpłatę w wysokości *500.00* na konto *1004* o kluczu głównym równym *200*, na następnie ponownie wypisz zawartość widoku *wplaty_wyplaty*. Na koniec usuń widok *wplaty_wyplaty*.

6. Utwórz funkcję *oblicz_koszt*, która przyjmuje jeden argument typu *numeric(11,2)* i zwraca wartość typu *numeric(11,2)*. Funkcja ta powinna zwracać koszt dokonania transakcji, tj. 2% wartości kwoty transakcji. Ostateczny wynik jest zaokrąglany do dwóch miejsc po przecinku.

7. Korzystając z funkcji *oblicz_koszt* z poprzedniego zadania (załącz rozwiązanie zadania 6), zwróć koszty wszystkich transakcji zapisanych w bazie danych.

8. Napisz funkcję *bilans_kont*, która zwraca tabelę o sygnaturze *(konto numeric(11), suma_wplat numeric(11,2), suma_wyplat numeric(11,2))*. W tabeli powinny się znaleźć sumaryczne kwoty wpłat oraz wypłat dla każdego konta zapisanego w bazie banku.

9. Na podstawie funkcji *bilans_kont* z poprzedniego zadania (załącz rozwiązanie zadania 8), utwórz zapytanie zestawiające aktualne bilanse wszystkich kont bankowych. Wynik powinien zawierać dwie kolumny *konto* oraz *bilans*.

10a. Napisz funkcję *silnia* obliczającą silnię z danej liczby typu numerycznego. Rozwiązanie musi korzystać z pętli.

10b. Podobnie jak w podpunkcie 10a, napisz funkcję *silnia* korzystając wprost z rekurencyjnej definicji silni.

11. Bank postanowił przyznać wszystkim swoim klientom bonus świąteczny. Na każde konto ma zostać przelana pewna frakcja p sumy wszystkich wypłat dokonanych z tego konta. Zaimplementuj funkcję *bonus_swiateczny*, która przyznaje wszystkim klientom bonus świąteczny. Funkcja powinna przyjmować jeden nieobowiązkowy argument p i dodawać do tabeli *transakcje* odpowiednią krotkę, w której jedną z wartości będzie *p*suma_wyplat_z_konta*. W przypadku, gdy p nie jest podany, należy użyć domyślnej wartości *0.01*.

    Każdy przyznany bonus jest transakcją, którą należy zarejestrować z pomocą sekwencji *seq*. W tym celu w rozwiązaniu utwórz sekwencję *seq*, której wartości przebiegają od 1000 do 5000 w interwale wielkości 10. Ponadto, użyj funkcji *bilans_kont* z zadania 8 (załącz to rozwiązanie).

12. Utwórz funkcję *stan_konta* przyjmującej dwa argumenty *konto* o typie *numeric(11)* oraz *czas* o typie *timestamp*. Funkcja powinna zwracać stan konta *konto* w chwili *czas*. W przypadku, gdy istnieje moment od chwili założenia konta do chwili *czas* (włącznie), kiedy stan konta spadł poniżej zera, należy zakończyć działanie funkcji poprzez rzucenie wyjątku o wiadomości *Wykryto ujemny bilans konta*.

13. Korzystając z rozwiązania poprzedniego zadania (załącz je), napisz funkcję *historia_konta*, która dla danego konta zwróci całą jego historię. Wynik powinien zawierać dwie kolumny *data* o typie *timestamp*, oraz *stan* o typie *numeric(11,2)*. W kolumnie *data* znajdują się momenty w czasie, gdy stan konta został zmieniony. W kolumnie *stan* znajdują się odpowiadające stany konta. Historia konta powinna być posortowana rosnąco po datach.

14. Napisz funkcję *moment_rozspojniajacy*, która zwróci pierwszy moment w czasie kiedy pewna transakcja rozspójniła dane zawarte w bazie danych doprowadzając do ujemnego stanu pewnego konta. Gdy nie ma takiego momentu, funkcja powinna zwrócić null. Możesz skorzystać z rozwiązania zadania 12.