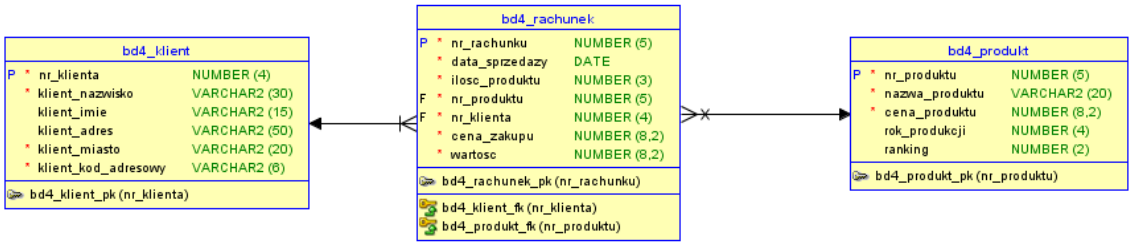
	Bazy Danych laboratorium	**Laboratorium BD11**

Projektowanie programowego generatora danych dla modelu transakcyjnego

Na potrzeby tego laboratorium zostanie wykorzystany model BD4_RACHUNEK przedstawiony poniżej:



Należy go zaimplementować w swoim schemacie bazodanowym przy pomocy skryptów zawartych w *Laboratorium BD11.zip* lub zrealizować to samodzielnie[1].

Celem tego laboratorium jest opracowanie oprogramowania, które będzie generowało transakcje w sposób losowy symulując realizacje procesu sprzedaży.

I. Generowanie liczb pseudolosowych o rozkładzie równomiernym

Podobnie do innych języków programowania, język PL/SQL posiada oprogramowanie umożliwiające generowanie liczb pseudolosowych o rozkładzie równomiernym w zadanym przedziale wartości.
Do tych celów został stworzony pakiet programowy *dbms_random* zawierający szereg funkcji służących generowaniu wartości numerycznych, dat i losowych ciągów znakowych.

W tym materiale omówione zostanie losowanie tylko wartości numerycznych.

W celu wylosowania liczby z przedziału [0..1] należy użyć funkcji:

select dbms_random.value from dual;

Przez wielokrotne wykonanie tego zapytania można otrzymać niepowtarzające się wartości liczbowe:

VALUE
0,067331443283191169941219250927532338 57
0,384775138028344609824492691174697117 53
0,096213584138320739502332666141318340 18
0,005602470989239461157409148790523419 13

....

z przedziału [0..1].

[1] Model ten różni się od opracowanego wcześniej modelu BD4. Chcąc z niego korzystać do budowy generatora danych należy poprzednią wersję usunąć skryptem *bd4_rachunek_drop*.

Chcąc zasymulować rzut kostką trzeba użyć innej formy tej funkcji:

```
select round ( dbms_random.value ( 1,6 ))  "rzut kostką"  from dual;
```

rzut kostką
4
2
3
5

....

W praktycznych zastosowaniach można spotkać taką konstrukcję:

```
select * from (
        select nr_produktu from  bd4_produkt
                order by dbms_random.value)
where rownum = 1
```

NR_PRODUKTU
10
1
8
5

....

, w której podzapytanie losowo porządkuje zbiór numerów produktów, a zapytanie główne wybiera pierwszy numer produktu.

II. Koncepcja budowy generatora danych modelu transakcyjnego

Przedstawiony na diagramie model rachunku jednopozycyjnego składa się z trzech tabel, z których dwie: BD4_KLIENT i BD4_PRODUKT mają charakter statyczny, to znaczy, że są one wypełnione danymi dotyczącymi klientów i produktów i zawartość ich nie musi się zmieniać.

Tabela BD4_RACHUNEK zawierająca transakcje kupna produktów przez klientów może być na początku pusta, a działanie generatora spowoduje, że programowo zostaną wypełnione danymi w dowolnej ilości.

Należy zatem jako czynność wstępną opracować skrypt zawierający zdania DDL tworzące wszystkie tabele modelu oraz inne niezbędne obiekty jak sekwencje i wyzwalacze bazodanowe. Model może być wzbogacony o inne tabele typu słownikowego, na przykład wymiarowanie klientów regionem zamieszkania, a produktów grupą asortymentową.

Drugim krokiem powinno być wprowadzenie do tabel BD4_KLIENT i BD4_PRODUKT (i ewentualnie słowników) danych demonstracyjnych w niedużych ilościach (do 20 wierszy w tabeli).

Zadaniem generatora danych będzie możliwość utworzenia pełnej transakcji zakupu towaru przez klienta i zapisanie tej transakcji w tabeli BD4_RACHUNEK.

Poniżej zostanie zaprezentowana metoda tworzenia takiego oprogramowania metodą top_down (od ogółu do szczegółu).

Poziom 0:

Utworzenie pojedynczej transakcji będzie realizowane przez procedurę o specyfikacji:

```
procedure pr_generuj_transakcje (v_data_sprzedazy date default sysdate);
```

Będzie ona odpowiedzialna za umieszczenie w tabeli BD4_RACHUNEK jednego wiersza stanowiącego kompletną transakcję.

Poziom 1:

Procedura pr_generuj_transakcje:

Powinna określić wszystkie niezbędne dane, aby możliwe było dokonanie wpisu do tabeli transakcyjnej.

Scenariusz może być następujący:

- NR_RACHUNKU - będzie wyznaczany sekwencją (na przykład *seq_rachunek*),
- DATA_SPRZEDAZY - parametr aktualny procedury,
- ILOSC_PRODUKTU – losowanie funkcją fn_daj_ilosc_produktu,
- NR_PRODUKTU – losowanie funkcją fn_daj_numer_produktu,
- NR_KLIENTA - losowanie funkcją fn_daj_numer_klienta,
- CENA_ZAKUPU – funkcja fn_daj_cene_produktu,

Uwagi:

1. Użycie *sysdate* jako daty transakcji jest prawidłowe, gdy generator będzie symulował działanie aplikacji czyli proces sprzedaży będzie rozłożony w czasie, na przykład przy wykorzystaniu automatów czasowych jakimi są *joby*. W przypadku użycia generatora do natychmiastowego wygenerowania określonej liczby transakcji należy odpowiednio ustawiać wartość parametru aktualnego v_data_sprzedazy. Zostanie to omówione w dalszej części materiału.

2. Losowanie ilości zakupionego towaru musi być ograniczone do pewnego zakresu dopuszczalnych wartości (przedziału liczbowego) w zależności od asortymentu produktów.

Ogólny schemat tej procedury będzie wyglądał następująco:

```
procedure pr_generuj_transakcje (v_data_sprzedazy date default sysdate);

begin
/*
wyznaczenie wszystkich niezbędnych wartości zgodnie z powyższym scenariuszem

        v_nr_rachunku     := seq_rachunek.nextval;
        v_ilosc_produktu  := fn_daj_ilosc_produktu (min_wartosc, max_wartosc);
        v_nr_produktu     := fn_daj_numer_produktu ();
        v_cena_produktu := fn_daj_cene_produktu (v_nr_produktu);
        v_wartosc         :-= v_cena_produktu * v_ilosc_produktu;
        v_nr_klienta       := fn_daj_numer_klienta ();

oraz wpisanie ich do tabeli BD4_RACHUNEK zdaniem:

        insert into bd4_rachunek values (v_nr_rachunku, v_data_sprzedazy
                                         v_ilosc_produktu, v_nr_produktu, v_nr_klienta,
                                         v_cena_produktu, v_wartosc);
        commit;

*/
null;

end;
```

Poziom 2:

Procedura *pr_generuj_transakcje* zawiera w swoim kodzie funkcje odpowiedzialne za określenie odpowiednich wartości niezbędnych do utworzenia transakcji. Wszystkie one opierać będą swoje działanie o losowanie wartości zgodnie z rozkładem równomiernym czyli wylosowanie każdego klienta z tabeli BD4_KLIENT jest równoprawdopodobne. Analogicznie jest z produktami oraz ich ilością.

Funkcja fn_daj_numer_klienta:

Specyfikacja tej funkcji nie zawiera żadnych parametrów formalnych i zwraca wartość numeryczną:

```
function fn_daj_numer_klienta return numer;
```

Taka specyfikacja funkcji ma tę zaletę, że nie ma znaczenia, czy zbiór numerów klientów jest zbiorem ciągłym czy nie. Przy zastosowaniu techniki zaprezentowanej wcześniej kod tej funkcji zawiera zdanie SQL:

```
        ……..
select * into v_nr_klienta from (
        select nr_klienta from  bd4_klient
        order by dbms_random.value)
where rownum = 1
        ……..
```

Funkcja fn_daj_numer_produktu:

Specyfikacja tej funkcji jest analogiczna do funkcji *fn_daj_numer_klienta*:

```
function fn_daj_numer_produktu return numer;
```

Dodatkową zaletą może być uniezależnienie losowanej wartości od typu kolumny *nr_produktu*.

Jeśli kody produktów byłyby typu *varchar2*, np. A101, A102, B35,…. to specyfikację tej funkcji należało by zmienić na:

```
function fn_daj_numer_produktu return varchar2;
```

, ale algorytm pozostał by ten sam.

Funkcja fn_daj_ilosc_produktu:

Ta funkcja musi mieć zdefiniowane parametry formalne, którymi jest wartość minimalna i maksymalna ilości produktu:

```
function fn_daj_ilosc_produktu ( min_ilosc number default 1,
                                 max_ilosc number default 10 ) return number;
```

W tej specyfikacji standardowo ustawiona jest minimalna i maksymalna wartość, co oznacza, że wywołanie jej może odbywać się na kilka sposobów:

```
v_ilosc_produktu  := fn_daj_ilosc_produktu ();
v_ilosc_produktu  := fn_daj_ilosc_produktu (5, 30);
v_ilosc_produktu  := fn_daj_ilosc_produktu (max_ilosc => 15);
```

Funkcja fn_daj_cene_produktu:

Ta funkcja ma pobierać cenę wylosowanego wcześniej produktu z tabeli BD4_PRODUKT z ewentualną możliwością symulacji negocjowania rabatu:

```
function fn_daj_cene_produktu ( v_numer_produktu number ) return number;
```

Podsumowanie:

W wyniku zastosowania takiej metody dekompozycji oprogramowania można wyspecyfikować wszystkie obiekty programowe niezbędne do zbudowania generatora danych.
Są to:

```
procedure pr_generuj_transakcje (v_data_sprzedazy date default sysdate);

function fn_daj_numer_klienta return numer;

function fn_daj_numer_produktu return numer;

function fn_daj_ilosc_produktu ( min_ilosc number default 1,
                                 max_ilosc number default 10 ) return number;

function fn_daj_cene_produktu ( v_numer_produktu number ) return number;
```

III. Zastosowanie pakietów PL/SQL do budowy generatora danych

Informacje wstępne:

Pakiety PL/SQL grupują logicznie powiązane składniki, takie jak, zmienne, struktury danych, wyjątki oraz procedury i funkcje. Składają się z dwóch części: specyfikacji i implementacji.

Umożliwiają serwerowi Oracle jednoczesne wczytywanie wielu obiektów do pamięci.
Pakiet nie może być wywoływany, zagnieżdżany ani nie może mieć parametrów. Po napisaniu i skompilowaniu pakietu jego zawartość może być współużytkowana przez wiele aplikacji czy sesji.

W momencie pierwszego odwołania się do pakietu jest on w całości ładowany do pamięci i następne próby używania konstrukcji zawartych w pakiecie nie wymagają już operacji dyskowych wejścia-wyjścia.

Budowa pakietu:

Specyfikacja pakietu definiowana jest jak poniżej:

```
create or replace package pkg_generator_danych is

procedure pr_generuj_transakcje (v_data_sprzedazy date default sysdate);

function fn_daj_numer_klienta return number;

function fn_daj_numer_produktu return number;

function fn_daj_ilosc_produktu ( min_ilosc number default 1,
                                 max_ilosc number default 10 ) return number;

function fn_daj_cene_produktu ( v_numer_produktu number ) return number;

end pkg_generator_danych;
```

Zgłaszane są specyfikacje funkcji i procedur. Dodatkowo można deklarować zmienne, własne typy danych, na przykład rekordowe lub tablicowe.

Tylko obiekty zdefiniowane w specyfikacji pakietu będą widoczne przez inne aplikacje czy też sesje.

Sposób odwoływania się do obiektu pakietowego wygląda tak:

```
-- funkcja pakietowa
v_nr_klienta := pkg_generator_danych.fn_daj_numer_klienta;
```

lub

```
-- procedura pakietowa
begin
   pkg_generator_danych.pr_generuj_transakcje ( '2020/09/18' );
end;
```

Implementacja pakietu definiowana jest jak poniżej:

```
create or replace package body pkg_generator_danych is

procedure pr_generuj_transakcje (v_data_sprzedazy default sysdate)
  begin
        null;
        /*  implementacja procedury */
 end pr_generuj_transakcje;

function fn_daj_numer_klienta  return number;
  begin
   return null;
        /* implementacja funkcji */
  end fn_daj_numer_klienta ;
                .....     -- kod wszystkich procedur i funkcji zdefiniowanych w specyfikacji
                          -- pakietu
end pkg_generator_danych;
```

Usuwanie pakietu:

Aby usunąć specyfikację i implementację pakietu należy użyć zdania:

```
drop package pkg_generator_danych;
```

Aby usunąć tylko implementację pakietu należy użyć zdania:

```
drop package body pkg_generator_danych;
```

Uwagi końcowe:

1. Nie można usunąć specyfikacji pakietu zostawiając jego implementację.

2. Procedury i funkcje w pakietach mogą być przeciążane, co oznacza, że można projektować obiekty o tych samych nazwach, lecz o argumentach formalnych różniących się istotnie między sobą.

   Na przykład:

```
-- procedura generująca jedną transakcję poprzez generator
procedure pr_generuj_transakcje (v_data_sprzedazy date default sysdate);

-- procedura generująca wiele transakcji z jedną datą poprzez generator
procedure pr_generuj_transakcje (v_data_sprzedazy date default sysdate,
                                 v_ile number);

-- procedura generująca wiele transakcji w zadanym przedziale czasu poprzez generator
procedure pr_generuj_transakcje (pocz_data date, konc_data date);
```

Przykłady zastosowania i sposoby wywoływania tych procedur:

- wygenerowanie jednej transakcji z określoną datą:

```
begin
  pkg_generator_danych.pr_generuj_transakcje ('2020/09/18');
end;
```

- wygenerowanie kilku transakcji z tą samą datą:

```
begin
  pkg_generator_danych.pr_generuj_transakcje
                        (v_data_sprzedazy => '2020/09/18',
                        v_ile =>  10);
end;
```

- wygenerowanie wielu transakcji w określonym przedziale czasowym:

```
begin
  pkg_generator_danych . pr_generuj_transakcje
                        (pocz_data => '2020/01/01',
                         konc_data  =>  sysdate);
end;
```

W tym przypadku podstawą algorytmu generowania powinno być symulowanie upływu czasu czyli wyznaczanie w pierwszej kolejności daty i dla niej generowanie określonej liczby transakcji:

```
cur_date := pocz_data;
while cur_date <= konc_data
 loop
   pkg_generator_danych.pr_generuj_transakcje
                              (v_data_sprzedazy => cur_date,
                               v_ile =>  10);
  cur_date := cur_date + 1;
end loop;
```

Warto zwrócić uwagę, że w takim przypadku będziemy mieć do czynienia z wywoływaniem w procedurze *pr_generuj_transakcje* przeciążonej procedury.

Dodatkowo można utworzyć pakietową funkcję *fn_daj_liczbe_transakcji*, która będzie losowo określała liczbę transakcji dla każdego dnia i wtedy w powyższym algorytmie drugi argument procedury może wyglądać tak:

```
.......

v_ile =>  fn_daj_liczbe_transakcji (min_liczba => 1,
                                    max_liczba = > 10) ;

......
```

3. Można opracować automat czasowy (*job*), który z określonym interwałem czasowym (na przykład co jedną godzinę) będzie uruchamiał procedurę *pr_generuj_transakcje* z bieżącą datą. Do tego celu służy pakiet programowy *dbms_scheduler*, przy pomocy którego można definiować różne warianty realizacji zadań.

Zadania do samodzielnego wykonania:

1. Opracować i przetestować kompletny pakiet generatora danych omówiony w tym materiale.

   Scenariusz postępowania:
   - usunięcie z tabeli BD4_RACHUNEK wszystkich wierszy,
   - skasowanie sekwencji seq_rachunek,
   - założenie sekwencji seq_rachunek,
   - uruchomienie generatora:

     ```
     begin
       pkg_generator_danych.pr_generuj_transakcje
                                 (pocz_data => '2020/01/01', konc_data  => sysdate);
     end;
     ```

   - (opcjonalnie) opracowanie automatu czasowego (*job*), który z określonym interwałem będzie uruchamiał procedurę:

     ```
     begin
       pkg_generator_danych.pr_generuj_transakcje (sysdate);
     end;
     ```

2. Poddać analizie prawidłowość losowania liczb pseudolosowych przy pomocy pakietu d*bms_random*. W tym celu należy założyć tabelę testową, w której sumowane będą wylosowane wartości.

   Scenariusz tego eksperymentu w oparciu o rzut kostką:
   - założenie tabeli zawierającej trzy kolumny: wylosowana wartość, ilość wystąpień, procentowy rozkład,
   - wpisanie do niej sześciu wierszy (x, 0, 0), gdzie x - wartość [1..6],
   - napisanie bloku PL/SQL lub procedury, wykonującej określoną liczbę losowań (na przykład 6000) i aktualizującą po każdym losowaniu kolumnę ilość wystąpień dla wylosowanej wartości,
   - zaktualizowanie kolumny procentowy rozkład.

3. Na podstawie zasad budowy zadań zawartych w *Laboratorium BD9* oraz skryptu *job_generowanie_transakcji.sql* (*Laboratorium BD11.zip*) opracować zadanie (*job*) generujące pojedynczą transakcję z określonym interwałem czasowym zawartym w przedziale [3 ..6] godzin.