ZAJĘCIA VI
BLOKI ANONIMOWE, KURSORY, INSTRUKCJE STERUJĄCE

Stworzyć tabele: Pracownik1 (kopia tabeli Pracownik) oraz Wiad składającą się z dwóch kolumn (jednej numerycznej, drugiej tekstowej). Tabela Pracownik1 powinna zawierać wszystkie wiersze tabeli Pracownik.

Zad. 1
Stworzyć blok PL/SQL, w którym wybrane by zostały tytuły dwóch najdroższych projektów. Przykład:
Przykład:

nazwa	*budzet*
"Projekt 1"	100
"Projekt 2"	90
"Projekt 3"	100
"Projekt 4"	50

Wynikiem są tytuły: "Projekt 1", "Projekt 3", "Projekt 2", ponieważ 100 i 90 są dwoma najwyższymi budżetami. Nazwa projektu wraz z budżetem powinny się znaleźć w tabeli Wiad. Powinny pojawić się także komunikaty: „Projekt o nazwie ... ma budzet ... (1 bądź 2) w kolejnosci".

Zad. 2 (NA KOPII TABELI Pracownik)
Stworzyć blok PL/SQL zwiększający pensję o 5% zaczynając od pracowników najmniej zarabiających. Jeśli suma pensji wszystkich pracowników (po podwyżce i przed podwyżką) przekroczy 35000, to pensja żadnego innego pracownika nie powinna ulec zmianie. Nazwisko pracownika, którego pensja została zmieniona jako ostatnia oraz liczba wskazująca na to, która to była zmiana, powinny znaleźć się w tabeli Wiad.

Zad. 3
Dodać do tabeli Wiad kolumnę tekstową. Stworzyć blok PL/SQL, który sprawdzi nazwy projektów realizowanych przez najmniejszą i największą liczbę pracowników. Informacje o nazwach projektów, liczbach pracowników oraz komentarze: „projekt z najwieksza/najmniejsza liczba pracownikow" powinny znaleźć się w tabeli Wiad. (Zadanie powinno zostać rozwiązane przy użyciu kursorów jawnych).

Zad. 4
Stworzyć tabelę Wiad2 z czteroma kolumnami numerycznymi, przy czym pierwsza kolumna stanowi klucz główny. Wstawić do niej wiersze postaci:

nr_przedziału_zarobkow *zarobki_od* *zarobki_do* *liczba_pracownikow*

Użytkownik podaje na ile przedziałów zarobków mamy rozbić. Należy wyznaczyć: dla atrybutu zarobki równe przedziały i liczbę pracowników z takim przedziałem - wstawić wyznaczone wartości do tabeli Wiad2.

POMOC

1. Tworzenie tabel

```
CREATE TABLE N_TABELI AS WYR_SELECT;
```

2. Wstawianie wierszy

```
INSERT INTO N_TABELI (LISTA KOLUMN) (WYR_SELECT);
```

3. Bloki PL/SQL anonimowe

```
[DECLARE
n_zmiennej typ_danych;.......]

BEGIN
instrukcje;
[EXCEPTION
obsluga wyjatkow]
END;
```

Aby wykonać instrukcje zawarte w bloku należy wpisać znak '/' i ENTER

```
n_zmiennej n_tabeli.n_kolumny%type;
n_zmiennej n_tabeli%rowtype;

n_zmiennej:=wartosc;
```

4. Instrukcja **SELECT INTO**

```
SELECT n_kolumn
INTO n_zmiennych
FROM n_tabeli..........
```

Wyrażenie SELECT INTO zwraca jeden wiersz!

- zmienne po przecinkach,
- SELECT INTO pobiera wartości i wstawia pod zmienne.

5. Kursor

Skupia wiele wierszy wybranych na podstawie jakiegoś WYR_SELECT, różny od polecenia SELECT INTO ze względu na ilość zwracanych wierszy (może być większa od 1).

a) jawny

Deklaracja (sekcja deklaracyjna bloku)

```
CURSOR n_kursora (parametry) IS WYR_SELECT;
```

Otwieranie kursora

```
OPEN n_kursora(parametry);
```

Pobieranie danych z kursora

```
FETCH n_kursora INTO zmienna (zmienne);
```

Zamykanie kursora

```
CLOSE n_kursora;
```

Pobieranie wyników zapytania

```
FOR i IN n_kursora LOOP
.....
END LOOP;
```

→ w tym przypadku nie trzeba poleceń `OPEN`, `FETCH`, `CLOSE`; `FOR` otwiera i później zamyka kursor; `FETCH` niepotrzebne, bo pod zmienna `i` automatycznie podstawiane są wyniki zapytania.

b) niejawny

```
FOR i IN (WYR_SELECT) LOOP
     instrukcje;
END LOOP;
```

`i.(nazwa kolumny wybranej w WYR_SELECT)` - pobranie wartości
`i` nie trzeba deklarować

6. Wypisywanie komunikatów

```
dbms_output.put_line ('komunikat');
np.: dbms_output.put_line (i.tytul||' kosztuje '||i.cena);
```

Aby było to możliwe:

```
set serveroutput on;
```

7. IF

```
IF warunek THEN
instr
ELSIF warunek THEN
     instr
....
ELSE
instr
END IF;
```

8. CASE

Bardziej wydajna "opcja" `ELSIF`. Jeśli zależy nam na wydajności należy używać `CASE`.

```
CASE coś
     WHEN war_1 THEN instr_1;
     WHEN war_2 THEN instr_2;
     …
     ELSE  instr_n;
END CASE;
```

W przypadku ominięcia `ELSE` jeśli zdarzy się, że nie ma wśród `war_1,...,war_k` odpowiedniej wartości, to podnoszony jest wyjątek `CASE_NOT_FOUND`.

9. Pętle

```
LOOP                    WHILE war LOOP
     instr                   instr
END LOOP;               END LOOP;

FOR i IN [REVERSE] min..max LOOP
     instr
END LOOP;
```

EXIT- bezwarunkowe wyjście z pętli
EXIT WHEN- warunkowe

10. Polecenia GOTO oraz NULL

Polecenie GOTO

<<etykieta>>

- musi być unikalna w pewnym zakresie widzenia,
- musi poprzedzać jakieś wyrażenia do wykonania.

Przykład:

```
BEGIN
     ...
     GOTO insert_row;
     ...
     <<insert_row>>
     INSERT INTO tabela VALUES ...
END;
```

```
DECLARE
   x NUMBER := 0;
BEGIN
   <<increment_x>>
   BEGIN
      x := x + 1;
   END;
   IF x < 10 THEN
      GOTO increment_x;
   END IF;
END;
```

```
DECLARE
   done BOOLEAN;
BEGIN
   FOR i IN 1..50 LOOP
      IF done THEN
         GOTO end_loop;
      END IF;
   <<end_loop>> -- BŁĄD
   END LOOP; -- etykieta nie poprzedza
-- polecenia
END;
```

```
FOR i IN 1..50 LOOP
   IF done THEN
      GOTO end_loop;
   END IF;
   ...
   <<end_loop>>
   NULL; -- polecenie, czyli OK
END LOOP;
```

Ograniczenia GOTO, nie można:

- przekierować do wnętrza IF, CASE, LOOP, podbloku,
- wyjść z podprogramu.

Polecenie NULL

Puste polecenie: nic się nie dzieje, kontrola przekazywana jest do następnego polecenia.

Parametr

&zmienna (w bloku PL/SQL) - podczas wykonywania użytkownik zostanie poproszony o podanie wartości dla parametru zmienna