Rozdział 11
Procedury i funkcje składowane

Zmienne podstawienia i zmienne wiązane, podprogramy, procedury składowane, typy argumentów, wywoływanie procedur, funkcje składowane, poziomy czystości funkcji, funkcje tablicowe

Zmienne podstawienia

Zmienne definiowane przez użytkownika mogą być użyte w miejsce nazw relacji, atrybutów lub jako wartości atrybutów. Zmienna podstawienia nie może być pierwszym słowem polecenia. Zmienne podstawienia są zawsze typu CHAR.

Polecenie DEFINE bez parametrów wyświetla listę wszystkich zmiennych podstawienia.

```
DEFINE myTable = 'PRACOWNICY'
DEFINE myValue = 'PROFESOR'

SELECT * FROM &myTable
WHERE etat = '&myValue';

UNDEFINE myTable
UNDEFINE myValue
```


Zmienne wiązane

Zmienne wiązane to zmienne deklarowane w SQL*Plus lub innym środowisku zewnętrznym, które są dostępne w programach PL/SQL. Mogą służyć do przekazywania wartości z PL/SQL do SQL i do optymalizacji zapytań. W PL/SQL zachowują się jak zwykłe zmienne.

```
VARIABLE x NUMBER

BEGIN
  SELECT COUNT(*) INTO :x FROM pracownicy;
END;

PRINT x
```


Zmienne wiązane kursorowe

Zmienne wiązane kursorowe pozwalają na odczytanie w SQL*Plus wyniku zapytania umieszczonego w bloku PL/SQL. Mogą być stosowane zarówno w anonimowych blokach PL/SQL, jak i jako parametr lub typ wynikowy procedury lub funkcji.

```
VARIABLE x REFCURSOR

BEGIN
  OPEN :x FOR SELECT * FROM pracownicy;
END;

PRINT x
```

Podprogramy

Procedury (wykonują określone akcje), funkcje (wykonują obliczenia i zwracają wartości) i pakiety (zbierają w całość logicznie powiązane procedury, funkcje, zmienne i kursory):

- **przechowywane w bazie danych w postaci skompilowanej i źródłowej (źródło dostępne poprzez USER_SOURCE)**
- **postać skompilowana zwiększenie szybkości działania**
- **współdzielone przez wielu użytkowników**

Zalety:

- **rozszerzalność**
- **modularność**
- **łatwość pielęgnowania kodu**
- **możliwość wielokrotnego użycia kodu**
- **ukrycie szczegółów implementacji**

Definiowanie procedury

```
CREATE [OR REPLACE] PROCEDURE
nazwa_procedury [ (argument, ...) ] IS      <- specyfikacja
      .....                                 <- deklaracje stałych, zmiennych, kursorów
BEGIN
      .....                                 <- ciało - polecenia PL/SQL i SQL
END [nazwa_procedury];
```

- **nazwa procedury musi być unikalna w ramach schematu (lub pakietu)**
- **między słowami kluczowymi IS i BEGIN umieszczamy deklaracje wszystkich zmiennych i kursorów lokalnych**
- **między słowami kluczowymi BEGIN i END umieszczamy kod PL/SQL, który wykonuje dana procedura**

Argumenty procedur

nazwa [IN | OUT | IN OUT] typ [DEFAULT wartość]

- **w liście argumentów nie podajemy rozmiaru argumentu (tylko typ)**
- **argument formalny: używany w deklaracji procedury i w części wykonywalnej PL/SQL**
- **argument aktualny: używany przy wywoływaniu procedury**

IN	OUT	IN OUT
Wartość przekazywana do programu przez referencję	Wartość zwracana do środowiska przez kopiowanie	Wartość przekazywana do programu i zwracana do środowiska przez kopiowanie
W programie zachowuje się jak stała	W programie zachowuje się jak nie zainicjalizowana zmienna	W programie zachowuje się jak zainicjalizowana zmienna
Musi być literałem, wyrażeniem, stałą lub zmienną	Musi być zmienną	Musi być zmienną

Przykład procedury

```
CREATE OR REPLACE PROCEDURE sprawdz_asystentow (p_id_zesp IN NUMBER,
p_ilu_asystentow OUT NUMBER) IS
  v_starszy_asystent pracownicy%ROWTYPE;
BEGIN
  SELECT COUNT(*) INTO p_ilu_asystentow
  FROM pracownicy WHERE id_zesp = p_id_zesp AND etat = 'ASYSTENT';
  FOR cur_rec IN
    (SELECT * FROM pracownicy WHERE id_zesp = p_id_zesp AND etat = 'ASYSTENT')
    LOOP
    IF (MONTHS_BETWEEN(SYSDATE, cur_rec.zatrudniony)/12) > 5 THEN
      DBMS_OUTPUT.PUT_LINE('Asystent ' || cur_rec.nazwisko || ' pracuje ponad 5 lat');
    END IF;
  END LOOP;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('W zespole ' || p_id_zesp || ' nie ma asystentow ');
END sprawdz_asystentow;
/
```

Wywołanie procedur i funkcji (PL/SQL)

```
VARIABLE nr_zespolu NUMBER
VARIABLE liczba_asystentow NUMBER
BEGIN
  :nr_zespolu := 20;
  sprawdz_asystentow(:nr_zespolu,:liczba_asystentow);
END;
/
PRINT liczba_asystentow
```

Wywołanie funkcji (SQL)

```
SELECT moja_funkcja(placa_pod) FROM pracownicy;
```

UWAGA: wołanie funkcji z poziomu SQL jest możliwe tylko wtedy gdy funkcja posiada odpowiedni poziom „czystości”

Czystość funkcji

Aby funkcja mogła być wywoływana z poziomu SQL, musi ona posiadać odpowiedni poziom czystości

- **funkcja wywoływana z instrukcji SELECT nie może modyfikować żadnych wartości w bazie danych**
- **funkcja wywoływana z instrukcji INSERT, UPDATE, DELETE nie może odczytywać i modyfikować żadnej tabeli, której dotyczy instrukcja**
- **funkcja wywoływana z instrukcji SELECT, INSERT, UPDATE, DELETE nie może zawierać instrukcji sterujących sesją i transakcjami oraz instrukcji DDL**

Poziom czystości deklaruje się za pomocą dyrektywy RESTRICT_REFERENCES:

- **RNDS, RNPS – Reads No Database/Package State**
- **WNDS, WNPS – Writes No Database/Package State**
- **TRUST – brak kontroli czystości funkcji**

Definiowanie funkcji

```
CREATE [OR REPLACE] FUNCTION
nazwa_funkcji [ (argument, ...) ]
RETURN typ IS
        .....
BEGIN
        .....
END [nazwa_funkcji];
```

specyfikacja

typ zwracany

deklaracje stałych, zmiennych, kursorów

ciało - polecenia PL/SQL i SQL

- **nazwa funkcji musi być unikalna w ramach schematu (lub pakietu)**
- **po słowie kluczowym RETURN umieszczamy typ zwracany przez funkcję**
- **między słowami kluczowymi IS i BEGIN umieszczamy deklaracje wszystkich zmiennych i kursorów lokalnych**
- **między słowami kluczowymi BEGIN i END umieszczamy kod PL/SQL, który wykonuje dana funkcja**
- **w kodzie PL/SQL musi się znaleźć instrukcja RETURN**

Przykład funkcji

```
CREATE OR REPLACE FUNCTION podatek (p_id_prac IN NUMBER) RETURN
NUMBER IS
  CURSOR c_pracownik IS  SELECT * FROM pracownicy  WHERE id_prac = p_id_prac;
  v_pracownik pracownicy%ROWTYPE;
  v_roczne_zarobki NUMBER;
  v_podatek NUMBER;
BEGIN
  OPEN c_pracownik;
  FETCH c_pracownik INTO v_pracownik;
  CLOSE c_pracownik;
  v_roczne_zarobki := 12 * v_pracownik.placa_pod + NVL(v_pracownik.placa_dod, 0);
  IF (v_roczne_zarobki > 5000) THEN v_podatek := 0.40 * v_roczne_zarobki;
  ELSIF (v_roczne_zarobki > 3000) THEN v_podatek := 0.30 * v_roczne_zarobki;
  ELSE v_podatek := 0.19 * v_roczne_zarobki;
  END IF;
  RETURN v_podatek;
END podatek;
```

Funkcje tablicowe

Funkcje tablicowe dają w wyniku kolekcję krotek. Nazwy tych funkcji mogą być wykorzystywane zamiast nazw tabel. Funkcje tablicowe mogą również przyjmować kolekcję krotek jako parametr. Funkcje mogą być zrównoleglone oraz potokowane, co zwiększa efektywność przetwarzania poprzez:

- **wielowątkowe wykonanie funkcji**
- **eliminację przechowywania wyników pośrednich**
- **zmniejszenie czasu odpowiedzi na pierwsze wyniki**
- **iteracyjne dostarczanie kolejnych krotek wyniku**

W przypadku funkcji potokowanych do dostarczenia krotki do wyniku służy komenda PIPE ROW (...)

Przykład funkcji tablicowej

```
CREATE TYPE CharTyp AS TABLE OF VARCHAR2(20);

CREATE OR REPLACE FUNCTION FPracownicy(p_etat CHAR)
RETURN CharTyp PIPELINED AS
BEGIN
  FOR x IN ( SELECT * FROM pracownicy WHERE etat=p_etat) LOOP
    PIPE ROW (x.nazwisko);
  END LOOP;
  RETURN;
END FPracownicy;

SELECT * FROM TABLE( FPracownicy('ASYSTENT') );
SELECT * FROM TABLE( FPracownicy('PROFESOR') );
```