Rozdział 9
Język PL/SQL
Wprowadzenie

Koncepcja języka, zmienne i stałe, typy zmiennych, nadawanie wartości zmiennym, instrukcje warunkowe, pętle, sterowanie przebiegiem programu

Wprowadzenie do języka PL/SQL

Język PL/SQL to rozszerzenie SQL o elementy programowania proceduralnego i obiektowego. PL/SQL umożliwia wykorzystanie:

- **zmiennych i stałych**
- **struktur kontrolnych, w tym instrukcji warunkowych, pętli, etykiet i instrukcji skoku, instrukcji wyboru warunkowego**
- **kursorów**
- **wyjątków i mechanizmu obsługi błędów**

Za pomocą języka PL/SQL tworzy się

- **anonimowe bloki programu**
- **procedury i funkcje składowane**
- **pakiety**
- **wyzwalacze bazy danych**

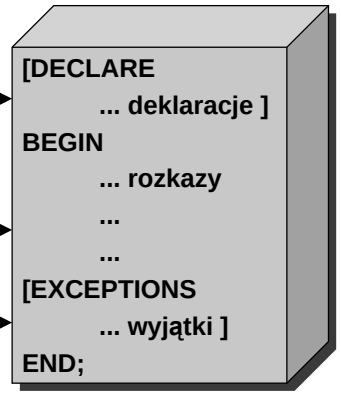
Przykładowy program w PL/SQL

```
DECLARE
   v_magazyn NUMBER(5);
BEGIN
   SELECT liczba_sztuk INTO v_magazyn FROM zapasy
   WHERE produkt = 'MLEKO UHT'
   FOR UPDATE OF liczba_sztuk; --odczytujemy liczbę w magazynie

   IF (v_magazyn > 0) THEN –-sprawdzamy ilość w magazynie
      UPDATE zapasy SET liczba_sztuk = liczba_sztuk - 1
      WHERE produkt = 'MLEKO UHT';
      INSERT INTO historia_zakupow
      VALUES ('Kupiono mleko UHT', SYSDATE);
   ELSE
      INSERT INTO historia_zakupow
      VALUES ('Brak mleka UHT w magazynie', SYSDATE);
   END IF;
   COMMIT;
END;
```


Struktura blokowa programu

- **Program składa się z jednostek zwanych blokami.**
- **Każdy blok odpowiada problemowi (podproblemowi)**
- **Bloki mogą być dowolnie zagnieżdżone**
- **Każdy blok składa się z trzech części:**
  - **deklaracji (o)**
  - **rozkazów (w)**
  - **obsługi błędów (o)**
- **Bloki mogą być zagnieżdżane w części rozkazów lub/i części obsługi błędów**



Zmienne

Zmienne proste (np. typu numerycznego, znakowego, daty)
Zmienne podtypu (np. zdefiniowanego przez użytkownika)
Zmienne złożone (np. rekordy, tablice, kolekcje, obiekty)

```
DECLARE nazwa_zmiennej typ(długość)
        [ DEFAULT wartość domyślna ]
        [ NOT NULL ];
```

```
DECLARE
        licznik NUMBER(4);
        znak CHAR(1) DEFAULT 'A';
        flaga BOOLEAN DEFAULT TRUE;
        data_pocz DATE DEFAULT SYSDATE NOT NULL;
```


Zmienne rekordowe

- Rekord to zbiór powiązanych danych różnych typów, opisujących jedno i to samo pojęcie. Przed zadeklarowaniem zmiennej rekordowej trzeba zdefiniować typ rekordowy

```
TYPE Pracownik IS RECORD (
nazwisko VARCHAR2(50),
pesel NUMBER(11),
data_zatrudnienia DATE DEFAULT SYSDATE );
...
```

```
...
        kowalski Pracownik;
BEGIN
        kowalski.nazwisko := 'Kowalski';
        kowalski.pesel := 70120100000;
```


Zmienne tablicowe

- Tablice indeksowane (index-by tables): nieograniczony rozmiar, automatycznie rozszerzane, nie mogą być typem atrybutu w bazie danych, indeks tablicy może być ujemny
- Tablice zagnieżdżone (nested tables): nieograniczony rozmiar, rozszerzane za pomocą procedury EXTEND, mogą być typem atrybutu w relacji, mogą być użyte w poleceniach DML, nie zainicjalizowana tablica jest pusta, przechowywane fizycznie w osobnej relacji (store table), indeks tablicy musi być dodatni
- Kolekcje o zmiennym rozmiarze (varrays): ograniczony i rozszerzalny rozmiar, przechowywane w zwartej postaci in-line, indeks kolekcji musi być dodatni

```
TYPE PracTab IS TABLE OF VARCHAR2(50)
        INDEX BY BINARY_INTEGER;
```

```
TYPE PracRecTab IS TABLE OF Pracownik;
```

```
TYPE SwietaVar IS VARRAY(50) OF DATE;
```


Atrybuty %TYPE, %ROWTYPE

- Atrybut %TYPE zawiera typ innej zmiennej lub typ atrybutu w bazie danych.

```
DECLARE
        v_nazwisko PRACOWNICY.NAZWISKO%TYPE;
        v_nazwa_zespolu ZESPOLY.NAZWA%TYPE;
```

- Atrybut %ROWTYPE zawiera typ rekordowy reprezentujący strukturę pojedynczej krotki z danej relacji. Atrybuty w krotce i odpowiadające im pola w rekordzie mają te same nazwy i typy.

```
DECLARE
        r_pracownik PRACOWNICY%ROWTYPE;
        r_zespol ZESPOLY%ROWTYPE;
```

Nadawanie wartości zmiennym

- Nadanie wartości poprzez przypisanie

```
DECLARE
        v_licznik NUMBER := 10;
        v_nazwa VARCHAR2(30) := 'Politechnika Poznańska';
        v_flaga BOOLEAN := FALSE;
        ...
        v_podatek NUMBER(10,2) := v_suma * v_stawka_pod;
        v_zysk NUMBER(10,2) := f_oblicz_zysk('01-01-1999', v_today);
```

UWAGA!!!

Zmienna, która została zadeklarowana lecz nie została zainicjalizowana, posiada wartość NULL. Użycie takiej zmiennej może spowodować nieprawidłowe wyniki.

Nadanie wartości zmiennym (c.d.)

- Nadanie wartości przez wczytanie danych z bazy danych do zmiennej poleceniem SELECT ... INTO ...

```
SELECT nazwisko, etat INTO v_nazwisko, v_etat
FROM pracownicy WHERE placa_pod = (
        SELECT MAX(placa_pod) FROM pracownicy );
```

- Nadanie wartości przez przekazanie zmiennej jako parametru typu IN OUT lub OUT do procedury lub funkcji

```
DECLARE
        v_pensja NUMBER(7,2);
        PROCEDURE zarobki(id_prac INT IN, placa REAL OUT) IS ...
BEGIN
        SELECT AVG(placa_pod) INTO v_pensja FROM pracownicy;
        zarobki(100, v_pensja);
```


Nadanie wartości zmiennym (c.d.)

- Nadanie wartości przez wczytanie danych z bazy danych do zmiennej za pomocą klauzuli RETURNING poleceń INSERT/UPDATE/DELETE
- Typowe zastosowanie RETURNING: odczyt wartości ustawionych na poziomie bazy danych (np. wartości klucza głównego)

```
DECLARE
  id NUMBER;
  nowa_placa NUMBER;
BEGIN
  INSERT INTO pracownicy (id_prac, nazwisko, etat, placa_pod, id_zesp)
  VALUES (prac_seq.NEXTVAL, 'NOWAK', 'ADIUNKT', 1000, 20)
  RETURNING id_prac INTO id;
  UPDATE pracownicy
  SET placa_pod = 1.1 * placa_pod
  WHERE id_prac = id
  RETURNING placa_pod INTO nowa_placa;
...
```

Stałe

- Stałe deklarujemy z użyciem słowa kluczowego CONSTANT. Stała musi zostać zainicjalizowana podczas deklaracji. Po utworzeniu stałej jakiekolwiek modyfikacje jej wartości są niedozwolone.

```
DECLARE
        v_godziny_pracy CONSTANT NUMBER := 42;
        v_pp CONSTANT VARCHAR2(30) := 'Politechnika Poznańska';
        v_vat CONSTANT NUMBER(2,2) := 0.22;
```

Typy danych

Numerical Types	Character Types	Composite Types
BINARY_INTEGER	CHAR	RECORD
DEC	CHARACTER	TABLE
DECIMAL	LONG	VARRAY
DOUBLE PRECISION	NCHAR	
FLOAT	NVARCHAR2	**LOB Types**
INT	RAW	BFILE
INTEGER	STRING	BLOB
NATURAL	VARCHAR	CLOB
NATURALN (not null)	VARCHAR2	NCLOB
NUMBER		
NUMERIC	**Boolean Type**	**Reference Types**
PLS_INTEGER	BOOLEAN	REF CURSOR
POSITIVE		REF object_type
POSITIVEN (not null)	**Date Type**	
REAL	DATE	
SIGNTYPE	TIMESTAMP	
SMALLINT	INTERVAL	

Typy danych - uwagi

- **Typy danych dostępne w PL/SQL nie odpowiadają dokładnie analogicznym typom dostępnym w SQL.**
- **Typy BINARY_INTEGER i PLS_INTEGER: $-2^{31} \div 2^{31}$. Ich podtypami są typy: POSITIVE, POSITIVEN, NATURAL, NATURALN i SIGNTYPE (-1, 0, 1).**
- **Typ NUMBER: $10^{-130} \div 10^{125}$. Jego podtypami są typy: DECIMAL, INTEGER, FLOAT, REAL, NUMERIC.**
- **Typy CHAR, VARCHAR2, RAW, LONG: 32767 bajtów**
- **Typ BOOLEAN: TRUE, FALSE, NULL**

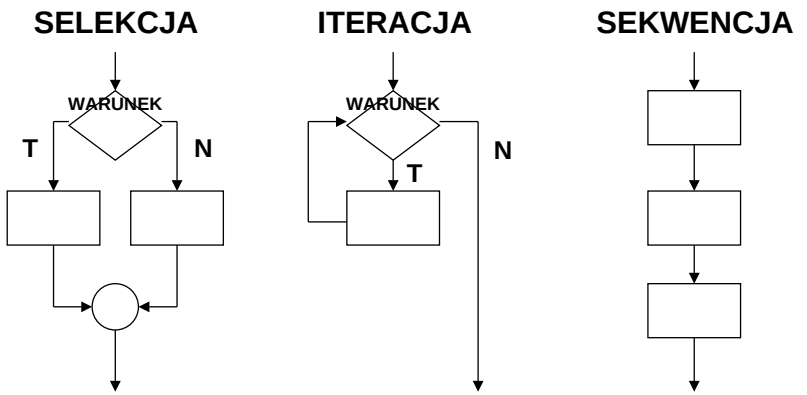
Podtypy

Każdy typ danych definiuje zbiór poprawnych wartości i zbiór operatorów, które mogą być zastosowany do zmiennej danego typu. Podtyp definiuje ten sam zbiór operatorów co jego typ nadrzędny, lecz zawęża zbiór poprawnych wartości.

```
SUBTYPE nazwa IS typ bazowy [ (ograniczenie) ] [ NOT NULL ];
```

```
DECLARE
        SUBTYPE DataUr DATE NOT NULL;
        SUBTYPE Pieniadze NUMBER(9,2);
        ...
        v_moje_urodziny DataUr;
        v_moja_pensja Pieniadze;
```


Struktury kontrolne



Instrukcja warunkowa IF-THEN-ELSE

```
IF warunek THEN
        sekwencja poleceń;
END IF;
```

Warunek musi zwracać wartość logiczną. Sekwencja poleceń jest wykonywana, gdy wartością warunku jest TRUE. Jeśli wartością warunku jest FALSE lub UNKNOWN to sekwencja nie jest wykonywana.

```
IF warunek THEN
        sekwencja poleceń;
ELSE
        sekwencja poleceń;
END IF;
```

```
IF warunek_1 THEN
        sekwencja poleceń;
ELSIF warunek_2 THEN
        sekwencja poleceń;
ELSE
        sekwencja poleceń;
END IF;
```


Przykład instrukcji warunkowej

Przed wykonaniem ćwiczenia ustaw zmienną środowiska SQL*Plus:

```
SQL> SET SERVEROUTPUT ON SIZE 1000000
```

```
DECLARE
        v_hello VARCHAR2(20) := 'Hello, ';
        v_kogo_witamy NUMBER(1) := 0;
BEGIN
        IF (v_kogo_witamy = 0) THEN
                v_hello := v_hello || 'world!';
        ELSE
                v_hello := v_hello || 'universe!';
        END IF;
        DBMS_OUTPUT.PUT_LINE(v_hello);
END;
```


Instrukcja wyboru wielokrotnego CASE

Instrukcja CASE może występować z selektorem (selektorem może być dowolnie złożone wyrażenie, ale najczęściej jest to jedna zmienna) lub z listą wyrażeń (*searched CASE*)

```
CASE selektor
        WHEN wartość_1 THEN polecenie_1;
        WHEN wartość_2 THEN polecenie_2;
        WHEN ...
        ELSE polecenie_n;
END CASE;
```

```
CASE
        WHEN wyrażenie_1 THEN polecenie_1;
        WHEN wyrażenie_2 THEN polecenie_2;
        WHEN ...
        ELSE polecenie_n;
END CASE;
```


Przykład instrukcji CASE

```
DECLARE
        v_ocena NUMBER(2,1) := 2.0;
        v_slownie CHAR(16);
BEGIN
        CASE v_ocena
                WHEN 2.0 THEN v_slownie := 'niedostateczny';
                WHEN 3.0 THEN v_slownie := 'dostateczny';
                WHEN 3.5 THEN v_slownie := 'dostateczny plus';
                WHEN 4.0 THEN v_slownie := 'dobry';
                WHEN 4.5 THEN v_slownie := 'dobry plus';
                WHEN 5.0 THEN v_slownie := 'bardzo dobry';
        END CASE;
        DBMS_OUTPUT.PUT_LINE(v_slownie);
END;
```

Pętla LOOP

Prosta pętla wykonuje się w nieskończoność. Wyjście z pętli jest możliwe tylko jako efekt wykonania polecenia EXIT lub EXIT WHEN. W każdym przebiegu pętli wykonuje się sekwencja poleceń. Po ich wykonaniu kontrola powraca do początku pętli.

```
LOOP
        sekwencja poleceń;
        IF warunek THEN
                EXIT;
        END IF;
END LOOP;
```

```
LOOP
        sekwencja poleceń;
        EXIT WHEN warunek;
END LOOP;
```


Przykład prostej pętli

```
DECLARE
        v_suma NUMBER := 0;
        v_i INTEGER := 0;
        v_koniec CONSTANT INTEGER := &koniec;
BEGIN
        LOOP
                v_suma := v_suma + v_i;
                EXIT WHEN (v_i = v_koniec);
                v_i := v_i + 1;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('Suma liczb od 0 do ' || v_koniec);
        DBMS_OUTPUT.PUT_LINE(v_suma);
END;
```


Pętla WHILE

Przed każdą iteracją sprawdzany jest warunek. Pętla jest wykonywano tak długo, jak długo warunek ma wartość TRUE. Jeżeli wartość warunku wynosi FALSE lub UNKNOWN to kontrola przechodzi do pierwszego polecenia po pętli. Jeżeli warunek na samym początku nie był spełniony, to pętla nie wykona się ani razu.

```
WHILE warunek LOOP
        sekwencja poleceń;
END LOOP;
```

UWAGA!!!

Pamiętaj, aby w sekwencji operacji znalazło się polecenie, które zmieni warunek, w przeciwnym przypadku grozi pętla nieskończona.

Przykład pętli WHILE

```
DECLARE
        a NUMBER := &pierwsza_liczba;
        b NUMBER := &druga_liczba;
BEGIN
        WHILE (a != b) LOOP
                IF (a > b) THEN
                        a := a - b;
                ELSE
                        b := b - a;
                END IF;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(' NWD = ' || a);
END;
```


Pętla FOR

Pętla FOR wykonuje się określoną liczbę razy. Liczba iteracji jest określona przez zakres podany między słowami kluczowymi FOR i LOOP. Zakres musi być typu numerycznego, w przedziale $-2^{31} \div 2^{31}$

```
FOR licznik IN  [ REVERSE ] dolna_gr .. górna_gr LOOP
        sekwencja poleceń;
END LOOP;
```

- Słowo kluczowe REVERSE odwraca kierunek iteracji
- Wewnątrz pętli nie wolno nadawać wartości zmiennej iterującej
- Jeśli dolna granica jest wyższa niż górna granica to pętla nie wykona się ani razu
- Obie granice zakresu iteracji nie muszą być statyczne
- Zmienna iterująca nie musi być wcześniej deklarowana ani inicjalizowana
- Do wcześniejszego wyjścia z pętli można użyć polecenia EXIT

Przykład pętli FOR

```
DECLARE
        a NUMBER := &koniec;
        is_prime BOOLEAN;
BEGIN
        FOR i IN 1 .. a LOOP
                is_prime := TRUE;
                FOR j IN 2 .. i/2 LOOP
                        IF (MOD(i,j) = 0) THEN
                                is_prime := FALSE;
                        END IF;
                END LOOP;
                IF (is_prime) THEN
                        DBMS_OUTPUT.PUT_LINE(i || ' jest liczba pierwsza');
                END IF;
        END LOOP;
END;
```


Polecenia sterujące GOTO i NULL

Polecenie GOTO bezwarunkowo przekazuje kontrolę wykonywania programu do miejsca wskazywanego przez etykietę związaną z poleceniem. Polecenie NULL nie wykonuje żadnej akcji.

```
GOTO etykieta;
...
<<etykieta>>
```

```
NULL;
```

- Etykieta musi poprzedzać polecenie wykonywalne
- GOTO nie może przeskakiwać do warunkowych części poleceń IF-THEN-ELSE, CASE, do polecenia LOOP i do bloku podrzędnego
- GOTO nie może wyskakiwać z podprogramu oraz procedury obsługi błędu

Przykład polecenia GOTO

```
DECLARE
        v_tekst VARCHAR2(20);
BEGIN
<<początek>>
        v_tekst := 'Ala '; GOTO ma;
<<asa>>
        v_tekst := v_tekst || 'asa '; GOTO drukuj;
<<drukuj>>
        DBMS_OUTPUT.PUT_LINE(v_tekst); GOTO koniec;
<<ma>>
        v_tekst := v_tekst || 'ma '; GOTO asa;
<<koniec>>
        NULL;
END;
```