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**

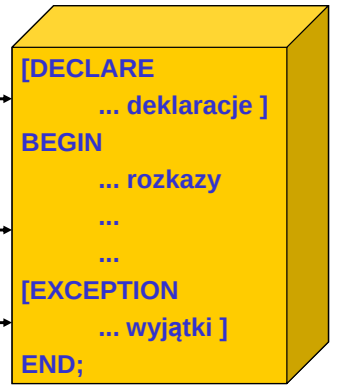
Przykładowy program w PL/SQL

```
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 liczbę 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**
- **Komentarze:**
  - **-- - jednoliniowy**
  - **/* … */ - wieloliniowy**



Zmienne proste

- **Zmienne proste (np. typu numerycznego, znakowego, daty)**

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

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

- **Zmienna zadeklarowana jako niepusta (NOT NULL) musi zostać zainicjalizowana**
- **Zmienna niezainicjalizowana posiada wartość pustą**

Atrybut %TYPE

- **Atrybut %TYPE pozwala zadeklarować zmienną w oparciu o typ innej zmiennej lub typ kolumny relacji w bazie danych.**

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


Zmienne rekordowe (1)

- **Rekord to zbiór powiązanych danych różnych typów, opisujących jedno i to samo pojęcie. Przed zadeklarowaniem zmiennej rekordowej trzeba zdefiniować typ rekordowy**
- **Deklaracja zmiennej w oparciu o zdefiniowany przez użytkownika typ rekordowy:**

```
DECLARE
  TYPE TPracownik IS RECORD (
        nazwisko VARCHAR2(50),
        pesel NUMBER(11),
        data_zatrudnienia DATE DEFAULT SYSDATE);
        r_pracownik TPracownik;
BEGIN
        r_pracownik.nazwisko := 'Kowalski';
        r_pracownik.pesel := 70120100000;
        ...
```


Zmienne rekordowe (2)

- **Deklaracja w oparciu o definicję relacji z bazy danych, kursora lub innej zmiennej rekordowej:**

```
DECLARE
        r_pracownik_1 PRACOWNICY%ROWTYPE;
        r_pracownik_2 r_pracownik_1%TYPE;
        r_zespol ZESPOLY%ROWTYPE;
```

Nadawanie wartości zmiennym (1)

- Nadanie wartości przez inicjalizację przy deklaracji

```
DECLARE
        v_licznik NUMBER := 10;
        v_nazwa VARCHAR2(30) DEFAULT 'Politechnika Poznańska';
```

- Nadanie wartości poprzez przypisanie w ciele programu

```
DECLARE
        v_flaga BOOLEAN; v_podatek NUMBER(10,2);
        v_zysk NUMBER(10,2);
BEGIN
        v_flaga := FALSE;
        v_podatek := v_suma * v_stawka_pod;
        v_zysk := f_oblicz_zysk('01-01-1999', v_today);
        v_identyfikator := seq_pracownicy.nextval; -- od Oracle11g
```


Nadawanie wartości zmiennym (2)

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

```
DECLARE
        v_id_zesp pracownicy.id_zesp%TYPE;
        v_etat pracownicy.etat%TYPE;
        r_zespol zespoly%ROWTYPE;
BEGIN
        SELECT id_zesp, etat INTO v_id_zesp, v_etat
        FROM pracownicy WHERE nazwisko = 'HAPKE';
        SELECT * INTO r_zespol FROM zespoly
        WHERE id_zesp = v_id_zesp;
        SELECT seq_zespoly.nextval INTO v_id_zesp FROM dual;
...
```

- Uwaga! Polecenie SELECT musi zwrócić dokładnie jeden rekord.

Nadawanie wartości zmiennym (3)

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

```
DECLARE
  v_id pracownicy.id_prac%TYPE;
  v_nowa_placa pracownicy.placa_pod%TYPE;
BEGIN
  INSERT INTO pracownicy (id_prac, nazwisko, etat, placa_pod)
  VALUES (prac_seq.NEXTVAL, 'NOWAK', 'ADIUNKT', 1000)
  RETURNING id_prac INTO v_id;
  UPDATE pracownicy
  SET placa_pod = 1.1 * placa_pod
  WHERE id_prac = v_id RETURNING placa_pod INTO v_nowa_placa;
...
```


Operacje na zmiennych rekordowych (1)

- Dopuszczalne operacje:
  - przypisanie zmiennej rekordowej wartości NULL,
  - przypisanie zmiennej rekordowej wartości innej zmiennej rekordowej (tylko gdy mają identyczną strukturę),
  - przypisanie polu zmiennej rekordowej wartości pola innej zmiennej rekordowej (o ile typy pól są zgodne lub można wykonać konwersję wartości),
  - porównywanie zmiennych rekordowych możliwe tylko na poziomie pól.

Operacje na zmiennych rekordowych (2)

```
DECLARE
   r_pracownik_1 PRACOWNICY%ROWTYPE;
   r_pracownik_2 PRACOWNICY%ROWTYPE;
   r_zespol ZESPOLY%ROWTYPE;
   v_zmienna BOOLEAN;
BEGIN
   r_pracownik_1 := r_pracownik_2; -- OK.
   r_zespol := r_pracownik_2; -- Błąd
   r_zespol.id_zesp := r_pracownik_2.id_zesp;  -- OK.

   v_zmienna := r_pracownik_1 = r_pracownik_2; -- Błąd
   v_zmienna := r_pracownik_1.etat = r_pracownik_2.etat;  -- OK.

   v_zmienna := r_pracownik_1 = r_zespol; -- Błąd
   v_zmienna := r_pracownik_1.nazwisko = r_zespol.nazwa; -- OK.
END;
```


Użycie zmiennych rekordowych w DML

- **Rozszerzona składnia INSERT i UPDATE dla PL/SQL:**

```
INSERT INTO relacja [(lista kolumn)] VALUES zmienna_rekordowa;

UPDATE relacja SET ROW = zmienna rekordowa [WHERE warunek];
```

- **Przykład:**

```
DECLARE r_zespol zespoly%ROWTYPE;
BEGIN
        SELECT * INTO r_zespol FROM zespoly
        WHERE nazwa = 'ALGORYTMY';

        r_zespol.id_zesp := seq_zespoly.nextval;
        r_zespol.nazwa := r_zespol.nazwa || ' (NOWE)';

        INSERT INTO zespoly VALUES r_zespol;  -- Brak nawiasów!
END;
```


Stałe

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

```
DECLARE
        c_godziny_pracy CONSTANT NUMBER := 42;
        c_pp CONSTANT VARCHAR2(30) := 'Politechnika Poznańska';
        c_vat CONSTANT NUMBER(2,2) DEFAULT 0.23;
        c_pi CONSTANT NUMBER(3,2); -- BŁĄD, brak inicjalizacji
BEGIN
        c_vat := 0.08; -- BŁĄD!
        …
```


Typy danych

Typy liczbowe	Typy znakowe	Inne
BINARY_INTEGER	CHAR	ROWID
DEC	CHARACTER	UROWID
DECIMAL	LONG	
DOUBLE PRECISION	NCHAR	Typy LOB
FLOAT	NVARCHAR2	BFILE
INT	RAW	BLOB
INTEGER	STRING	CLOB
NATURAL	VARCHAR	NCLOB
NATURALN (not null)	VARCHAR2	
NUMBER		Typy wskaźnikowe
NUMERIC	Typ logiczny	REF CURSOR
PLS_INTEGER	BOOLEAN	REF object_type
POSITIVE		
POSITIVEN (not null)	Czas	
REAL	DATE	
SIGNTYPE	TIMESTAMP	
SMALLINT	INTERVAL	

Typ NUMBER

- Definicja identyczna jak w SQL
- Zaimplementowany w sposób niezależny od platformy
- Deklaracja typu zmiennoprzecinkowego: NUMBER
  - maks. 40 pozycji

```
v_liczba_zp NUMBER;
```

- Deklaracja typu stałoprzecinkowego: NUMBER(precyzja, skala):
  - precyzja: <1, 38> - całkowita liczba pozycji znaczących
  - skala: <-84, 127> - liczba pozycji na prawo (dodatnia) lub lewo (ujemna) od przecinka

```
v_liczba_sp1 NUMBER(6,2);
v_liczba_sp2 NUMBER(4);
v_liczba_sp3 NUMBER(8,10);
v_liczba_sp4 NUMBER(1,-3);
```


Dodatkowe typy numeryczne w PL/SQL (1)

- Typy całkowitoliczbowe:
  - PLS_INTEGER – zakres: <−2 147 483 648, 2 147 483,647>, wydajny (wykorzystuje arytmetykę sprzętową), podtypy:
    - NATURAL – tylko liczby nieujemne,
    - NATURALN – tylko liczby nieujemne bez NULL,
    - POSITIVE – tylko liczby dodatnie,
    - POSITIVEN – tylko liczby dodatnie bez NULL,
    - SIGNTYPE – wartości: -1, 0, 1,
    - BINARY_INTEGER – od wersji Oracle10g identyczny z PLS_INTEGER, we wcześniejszych wersjach implementowany niezależnie od sprzętu,
  - SIMPLE_INTEGER – zakres identyczny z PLS_INTEGER, bez NULL, brak kontroli przepełnienia, dostępny od Oracle11g, znaczny wzrost wydajności w stosunku do PLS_INTEGER przy kompilacji natywnej.

Dodatkowe typy numeryczne w PL/SQL (2)

- Typy zmiennoprzecinkowe z reprezentacją binarną wg standardu IEEE-754:
  - SIMPLE_FLOAT, SIMPLE_DOUBLE – podtypy powyższych, dostępne od Oracle11g, brak NULL i kontroli przepełnienia, wyższa wydajność przy kompilacji natywnej.

Ciągi znaków zmiennej długości w PL/SQL (1)

- VARCHAR2 – ciąg znaków kodowany zestawem znaków domyślnym dla bazy, maks. długość 32676B,
  - długość podajemy w bajtach lub znakach:

```
v_S1 VARCHAR2(1000 BYTE);   v_S2 VARCHAR2(1000 CHAR);
```

  - pominięcie BYTE i CHAR – długość wyrażona w jednostkach określonych przez parametr sesji NLS_LENGTH_SEMANTICS

```
SELECT value FROM nls_session_parameters
WHERE parameter = 'NLS_LENGTH_SEMANTICS';
```

  - Uwaga! VARCHAR2 w SQL przechowuje maks. 4000B
- NVARCHAR2 – ciąg znaków kodowany w wielobajtowym Unicode, może wykorzystywać inny zestaw znaków niż domyślny dla bazy

Ciągi znaków zmiennej długości w PL/SQL (2)

- Synonimy:
  - VARCHAR2: CHAR VARYING, CHARACTER VARYING, STRING, VARCHAR
  - NVARCHAR2: NATIONAL CHAR VARYING, NCHAR VARYING, NATIONAL CHARACTER VARYING
- Uwaga!

  W Oracle pusty ciąg znaków jest równy NULL. Jest to sprzeczne ze standardem języka SQL.

```
DECLARE v_ciag VARCHAR2(10); v_zmienna BOOLEAN;
BEGIN
  v_ciag := '';
  v_zmienna := v_ciag IS NULL; -- v_zmienna = true
  v_ciag := NULL;
  v_zmienna := v_ciag IS NULL; -- v_zmienna = true  …
```


Ciągi znaków stałej długości w PL/SQL (1)

- CHAR – ciąg znaków kodowany zestawem znaków domyślnym dla bazy, maks długość. 32676B,
  - długość podajemy w bajtach lub znakach:

```
v_S1 CHAR(100 BYTE); v_S2 CHAR(100 CHAR); v_S3 CHAR;
```

  - pominięcie BYTE i CHAR – długość wyrażona w jednostkach określonych przez parametr sesji NLS_LENGTH_SEMANTICS,
  - pominięcie długości – długość ciągu = 1,
  - niewykorzystane pozycje dopełniane spacjami z prawej strony,
  - Uwaga! CHAR w SQL przechowuje maks. 2000B (255B w wersjach przed Oracle8i),
- NCHAR – ciąg znaków kodowany w wielobajtowym Unicode, może korzystać z innego zestawu znaków niż domyślny dla bazy

Ciągi znaków stałej długości w PL/SQL (2)

- Synonimy:
  - CHAR: CHARACTER
  - NCHAR: NATIONAL CHARACTER, NATIONAL CHAR

ROWID i UROWID (1)

- Typy dla zmiennych przechowujących adresy rekordów relacji.
- ROWID – typ dla adresu rekordu "zwykłych" relacji Oracle (implementowanych za pomocą sterty).
- UROWID – ang. *Universal ROWID* – typ dla adresów rekordów wszystkich rodzajów relacji, zarówno zwykłych jak i zewnętrznych, zorganizowanych jak indeks, obcych (np. z IBM DB2 dostępnych przez bramę).
- Użycie adresu rekordu może znacznie przyspieszyć operacje w programie PL/SQL.
- Uwaga! Adres rekordu może ulec zmianie!

ROWID i UROWID (2)

```
DECLARE
  v_adres UROWID;
  v_placa pracownicy.placa_pod%TYPE;
BEGIN
  SELECT ROWID, placa_pod INTO v_adres, v_placa
  FROM pracownicy WHERE nazwisko = 'HAPKE';

  ... -- obliczenia nowej płacy, wynik w v_placa

  UPDATE pracownicy
  SET placa_pod = v_placa
  WHERE ROWID = v_adres;
END;
```


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 TDataUr IS DATE NOT NULL;
        SUBTYPE TPieniadze IS NUMBER(9,2);
        ...
        v_moje_urodziny TDataUr;
        v_moja_pensja TPieniadze;
```


Interakcja z użytkownikiem (1)

- **Pobieranie informacji od użytkownika – zmienne podstawienia.**

```
v_zmienna := &zmienna_podstawienia;
```

- **Wypisywanie informacji na konsoli – procedura PUT_LINE z pakietu DBMS_OUTPUT.**

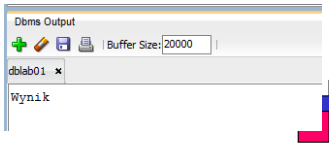
```
DBMS_OUTPUT.PUT_LINE(ciąg_tekstowy);
```

  - ***SQL*Plus* lub *iSQL*Plus*: ustaw zmienną SERVEROUTPUT na wartość ON przed wykonaniem programu.**

```
SET SERVEROUTPUT ON [SIZE rozmiar_bufora]
```

    - **domyślny rozmiar bufora: 2 000, dopuszczalne wartości: <2 000, 1 000 000>**

- ***Oracle SQL Developer*: włącz konsolę**



Interakcja z użytkownikiem (2)

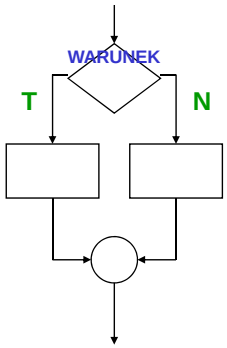
Przykład:

```
SET SERVEROUTPUT ON
```

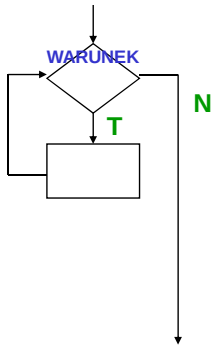
```
DECLARE
      v_i NUMBER(3) := &liczba;
      v_nazwa VARCHAR2(50) := '&tekst';
BEGIN
      dbms_output.put_line('Zmienna v_i: ' || to_char(v_i));
      v_nazwa := v_nazwa || ' ABC';
      dbms_output.put_line(v_nazwa);
END;
```


Struktury kontrolne

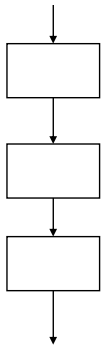
SELEKCJA



ITERACJA



SEKWENCJA



Instrukcja warunkowa IF-THEN-ELSE

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

```
IF warunek THEN
        sekwencja poleceń;
ELSE
        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 warunek1 THEN
        sekwencja poleceń;
ELSIF warunek2 THEN
        sekwencja poleceń;
ELSE
        sekwencja poleceń;
END IF;
```


Przykład instrukcji warunkowej

Przed wykonaniem ćwiczenia ustaw zmienną środowiska SQL*Plus:

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 wartość1 THEN polecenie1;
        WHEN wartość2 THEN polecenie2;
        WHEN ...
        ELSE polecenien;
END CASE;
```

```
CASE
        WHEN wyrażenie1 THEN polecenie1;
        WHEN wyrażenie2 THEN polecenie2;
        WHEN ...
        ELSE polecenien;
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;
        c_koniec CONSTANT INTEGER := &koniec;
BEGIN
        LOOP
                v_suma := v_suma + v_i;
                EXIT WHEN (v_i = c_koniec);
                v_i := v_i + 1;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('Suma liczb od 0 do ' || c_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;
```


Polecenie CONTINUE

- **Powoduje pominięcie bieżącej iteracji i przejście do następnej iteracji.**
- **Wersje:**
  - **bezwarunkowa – CONTINUE ,**
  - **warunkowa – CONTINUE WHEN <warunek logiczny>.**
- **Dostępne w Oracle od wersji 11g.**

```
BEGIN
  FOR i IN 1..100 LOOP
    CONTINUE WHEN mod(i, 2) = 1;
    DBMS_OUTPUT.PUT_LINE('Numer iteracji: ' || to_char(i));
  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;
```