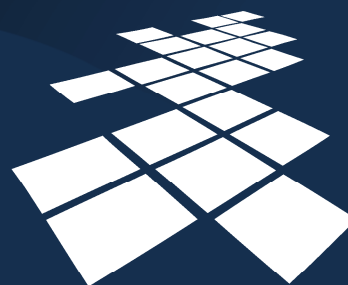


## Ćwiczenie 11 – PL/SQL

Wprowadzenie do języka  
PL/SQL



UCZELNIA  
ONLINE

Ćwiczenie 11 – PL/SQL

Celem ćwiczenia jest wprowadzeniem do programowania w języku PL/SQL. Język PL/SQL umożliwia tworzenie programów, przetwarzających dane w bazie danych.

Wymagania:

Umiejętność tworzenia zapytań w języku SQL, znajomość operacji z grup DML i DDL.



## Plan ćwiczenia

- Koncepcja języka PL/SQL.
- Struktura anonimowego bloku PL/SQL.
- Deklarowanie zmiennych i stałych.
- Przegląd podstawowych konstrukcji sterujących języka PL/SQL.
- Użycie w programie PL/SQL poleceń DML.

Ćwiczenie 11 – PL/SQL (2)

Zaprezentowana zostanie koncepcja języka, struktura bloku, sposób deklaracji i wykorzystania zmiennych oraz stałych, podstawowe struktury sterujące przebiegiem programu. Na końcu ćwiczenia wyjaśnimy, jak w programie PL/SQL użyć polecenia języka SQL z grupy DML.



## Konceptcja języka PL/SQL

- Rozwiązanie specyficzne – tylko w SZBD Oracle.
- Rozszerzenie języka SQL o elementy programowania proceduralnego i obiektowego.
- Język nastawiony na przetwarzanie danych.
- Umożliwia definiowanie:
  - anonimowych bloków programowych,
  - procedur i funkcji składanych w bazie danych,
  - pakietów (bibliotek) procedur i funkcji, składanych w bazie danych.
- W PL/SQL nie można umieszczać instrukcji DDL i DCL (sterowania sesją).

Ćwiczenie 11 – PL/SQL (3)

Język PL/SQL jest rozszerzeniem języka SQL o konstrukcje programowania proceduralnego i obiektowego. Język ten jest indywidualnym rozwiązaniem firmy Oracle, nie jest elementem standardu.

Język PL/SQL jest nastawiony na przetwarzanie danych, stąd posiada wiele konstrukcji, znacznie ułatwiających proces pobierania danych z bazy danych i ich przetwarzania wewnątrz programu. Język nie jest przeznaczony do interakcji z użytkownikiem, stąd brak w nim wielu mechanizmów, obecnych w innych językach programowania, a pozwalających np. na pobieranie informacji od użytkownika czy też wyrafinowane formatowanie wyników, wypisywanych przez program na ekranie.

W bieżącym ćwiczeniu będziemy zajmowali się jedynie anonimowymi blokami PL/SQL, które są wykonywane natychmiast po ich utworzeniu. Jednak PL/SQL pozwala również na definiowanie programów trwale zapisywanych w bazie danych, tzw. programów składanych, w postaci funkcji, procedur oraz bibliotek, nazywanych pakietami.

W programie PL/SQL można umieszczać polecenia DML, natomiast nie jest możliwe bezpośrednie wykonanie w programie poleceń z grupy DDL (poleceń tworzenia nowych obiektów, np. relacji) i DCL (poleceń sterowania przebiegiem sesji).



## Korzyści ze stosowania PL/SQL

- Większa łatwość wykonania niektórych zadań niż w SQL.
- Zwiększenie wydajności działania.
- Dostępność wielu mechanizmów, nieobecnych w SQL:
  - zmienne, stałe,
  - struktury sterujące,
  - obsługa błędów.
- Kod wykonywany na serwerze, zapewnia pełną przenaszalność pomiędzy platformami, na które oferowany jest SZBD Oracle.
- Możliwość wykorzystania predefiniowanych pakietów.

Ćwiczenie 11 – PL/SQL (4)

Zastosowanie języka PL/SQL pozwala w łatwy sposób rozwiązać problemy, których wykonanie z wykorzystaniem standardowych konstrukcji języka SQL byłoby bardzo trudne czy nawet niemożliwe. Co więcej, często zastosowanie PL/SQL może podnieść wydajność działania aplikacji w przypadku, gdy generuje ona wiele zapytań do bazy danych. W takiej sytuacji aplikacja przesyła do serwera bazy danych cały blok PL/SQL, zawierający wiele zapytań, zamiast przysyłać te zapytania osobno.

W PL/SQL mamy możliwość skorzystania z wielu mechanizmów, niedostępnych w SQL, np. zmiennych do przechowywania tymczasowych wyników pewnych operacji, stałych, różnorodnych struktur sterujących (np. pętli, warunków), procedur obsługi błędów, jakie mogą pojawić się przy dostępie do bazy danych czy podczas działania programu.

Program PL/SQL jest najczęściej wykonywany na serwerze, są jednak sytuacje, w których program jest wykonywany bezpośrednio w narzędziu. SZBD Oracle dostępny jest na wielu platformach sprzętowych i programów, PL/SQL jest w pełni przenaszalny pomiędzy wszystkimi platformami.

Kolejną zaletą języka PL/SQL jest możliwość wykorzystania bogatej biblioteki predefiniowanych programów, które Oracle dostarcza w postaci zbioru pakietów. Np. pakiet UTL\_FILE pozwala na wykonywanie wewnątrz programu PL/SQL operacji na plikach, pakiet DBMS\_SQL umożliwia dynamiczną konstrukcję poleceń SQL w programie, itd.



## Anonimowy blok PL/SQL

- Podstawowa jednostka programowa PL/SQL.

### postać podstawowa

```
BEGIN
    polecenia programu
END;
```

### pełna postać

```
DECLARE
    deklaracje
BEGIN
    polecenia programu
EXCEPTION
    obsługa błędów
END;
```

- Bloki mogą tworzyć strukturę zagnieżdżoną (blok jest elementem innego bloku).

Podstawową jednostką logiczną w programie PL/SQL jest anonimowy blok. Jak sama nazwa wskazuje, blok anonimowy nie posiada nazwy i jest wykonywany natychmiast po utworzeniu. W najprostszej postaci blok składa się z dwóch słów kluczowych: słowa BEGIN, które rozpoczyna blok i słowa END, które blok kończy. Pomiędzy BEGIN i END znajdują się polecenia programu, jest to tzw. sekcja wykonywalna bloku. Zauważmy, że polecenie END kończymy średnikiem, w przeciwieństwie do polecenia BEGIN.

Pełna postać anonimowego bloku PL/SQL składa się z dwóch dodatkowych elementów. Pierwszy z nich, tzw. sekcja deklaracji, służy do zadeklarowania elementów (zmiennych, stałych, itd.), które zostaną następnie użyte w programie. Sekcja deklaracji rozpoczyna się słowem kluczowym DECLARE, jej koniec wyznacza słowo BEGIN. Druga, opcjonalna sekcja bloku PL/SQL to sekcja obsługi błędów (nazywana także sekcją obsługi wyjątków). Sekcja ta znajduje się na końcu bloku, za wszystkimi poleceniami programu. Rozpoczyna się od słowa kluczowego EXCEPTION, kończy się słowem END, kończącym również cały blok PL/SQL.

Bloki PL/SQL mogą tworzyć strukturę zagnieżdżoną, czyli w sekcjach: wykonywalnej lub obsługi błędów może znajdować się nowy blok. Nie można umieścić bloku w sekcji deklaracyjnej.



## Typy danych PL/SQL

<u>Typy liczbowe</u> BINARY_INTEGER DEC DECIMAL DOUBLE PRECISION FLOAT INT INTEGER NATURAL NATURALN (not null) NUMBER NUMERIC PLS_INTEGER POSITIVE POSITIVEN (not null)	<u>Typy liczbowe cd</u> REAL SIGNTYPE SMALLINT <u>Typy znakowe</u> CHAR CHARACTER LONG NCHAR NVARCHAR2 RAW STRING VARCHAR VARCHAR2	<u>Typ logiczny</u> BOOLEAN, literały: TRUE (prawda), FALSE (fałsz) <u>Typy czasowe</u> DATE TIMESTAMP INTERVAL <u>Typy złożone</u> RECORD TABLE VARRAY <u>Typy wskaźnikowe</u> REF CURSOR REF object_type
---	---	--

Powyższy slajd przedstawia zbiór typów danych, które mogą być używane w bloku PL/SQL. Nie wnikając w szczegóły, w PL/SQL możemy użyć wszystkich tych typów danych, które są obecne w SQL, dodatkowo PL/SQL posiada swoje własne typy danych. Najważniejsze z nich to: typ logiczny BOOLEAN, posiadający dwa zdefiniowane literały: TRUE (prawda) i FALSE (fałsz) oraz typ RECORD, umożliwiający zdefiniowanie zmiennej rekordowej. Pozostałe wymienione tutaj specyficzne typy PL/SQL nie będą nam potrzebne do dalszych rozważań, zainteresowanych rozszerzonymi informacjami o typach odsyłamy do dokumentacji SZBD Oracle.



## Zmienna

- Deklarowana w sekcji **DECLARE**:
- Rodzaje zmiennych:
  - prosta – liczba, ciąg znaków, data, wartość logiczna,
  - złożona – rekord, tablica, obiekt.
- Widoczna w bloku deklaracji i blokach zagnieżdżonych.
- Przykład – zmienne proste:

```
DECLARE  
nazwa_zmiennej typ(długość);
```

```
DECLARE  
v_i NUMBER(6);  
nazwa VARCHAR2(100);  
data_sprzedazy DATE;  
czy_w_magazynie BOOLEAN;
```

Ćwiczenie 11 – PL/SQL (7)

Zmienna jest elementem, służącym do krótkotrwałego przechowywania danych wewnątrz bloku PL/SQL. Zmienna przed użyciem musi zostać zadeklarowana w sekcji deklaracji bloku. Deklaracja zmiennej wymaga podania nazwy zmiennej i określenia jej typu (również długości jeśli typ tego wymaga). Dobrym nawykiem jest tworzenie nazw zmiennych przez użycie przedrostka „v\_” w nazwie. Taka konwencja nazewnicza znacznie poprawia czytelność programu.

W PL/SQL wyróżniamy dwa rodzaje zmiennych: zmienne proste i złożone. Zmienna prosta służy do przechowywania wartości podstawowych typów danych: liczb, ciągów znaków, dat i wartości logicznych. W przykładzie zaprezentowanym na bieżącym slajdzie zadeklarowano cztery zmienne proste: v\_i typu numerycznego, nazwa będąca ciągiem znaków, data\_sprzedazy, która jest datą i czy\_w\_magazynie, będącą zmienną logiczną.

Z kolei zmienna złożona posiada wewnętrzną strukturę. Przykładami zmiennych złożonych są tablice, rekordy i obiekty.



## Zainicjalizowanie zmiennej

- Zmienna niezainicjalizowana posiada wartość pustą.
- Sposoby inicjalizowania zmiennej:
  1. przez przypisanie wartości,
  2. przez określenie wartości domyślnej (słowo DEFAULT).
- Dla zmiennej zainicjalizowanej można wymusić obowiązkowość wartości (słowo NOT NULL).

### DECLARE

```
v_i NUMBER(6) NOT NULL := 10;  
nazwa VARCHAR2(100) := 'ALGORYTMY';  
data_sprzedazy DATE DEFAULT DATE '2006-04-01';  
czy_w_magazynie BOOLEAN NOT NULL DEFAULT TRUE;
```

Ćwiczenie 11 – PL/SQL (8)

Wartość zmiennej, która została zadeklarowana, jednak nie przeprowadzono jej zainicjalizowania, jest pusta (zmienna posiada wartość NULL). Zainicjalizowanie zmiennej to nadanie jej początkowej wartości bezpośrednio przy deklaracji. Inicjalizować zmienną można przez wykonanie operacji przypisania wartości do zmiennej przy wykorzystaniu operatora przypisania (:=) lub przez użycie słowa DEFAULT. W przykładzie zmienna v\_i została zainicjalizowana wartością 10, zmienna nazwa ciągiem znaków „ALGORYTMY”, data\_sprzedazy bieżącą datą systemową, a czy\_w\_magazynie wartością TRUE. Dla zainicjalizowanej zmiennej przy jej deklaracji można dodać słowo NOT NULL jeśli zależy nam na tym, aby zmienna nigdy nie miała wartości pustej (w przypadku, gdy nastąpi przypisanie wartości pustej do takiej zmiennej, program zostanie przerwany z komunikatem o błędzie). W przykładzie jako niepuste zadeklarowano zmienne v\_i i czy\_w\_magazynie.





## Zmienna rekordowa

- Rekord jest grupą powiązanych danych, składanych w polach, z których każde ma własną nazwę i typ.
- Kroki procesu deklarowania zmiennej rekordowej:
  1. zdefiniowanie typu rekordowego (TYPE nazwa IS RECORD),
  2. zadeklarowanie zmiennej typu zdefiniowanego w kroku 1.
- Użycie w programie – dostęp kropkowy

### DECLARE

```
TYPE DanePracownika IS RECORD (
    nazwisko VARCHAR2(100),
    imię VARCHAR2(100));
v_pracownik DanePracownika;
```

### BEGIN

```
v_pracownik.nazwisko := 'Kowalski';
v_pracownik.imię := 'Jan';
```

Ćwiczenie 11 – PL/SQL (9)

Omówimy teraz zmienną rekordową jako przykład zmiennej złożonej.

Rekord jest strukturą, umożliwiającą przechowywanie powiązanych logicznie danych. Dane rekordu składane są w polach, z których każde ma swoją własną nazwę i typ danych.

Aby w języku PL/SQL zadeklarować zmienną rekordową, należy najpierw zdefiniować tzw. typ rekordowy. Definicję typu przeprowadzamy w sekcji deklaracji bloku. Definicja typu rozpoczyna się od słowa kluczowego TYPE, po którym następuje nazwa definiowanego typu, następnie słowa kluczowe IS RECORD, po których w nawiasach umieszcza się oddzielone przecinkami definicje pól rekordu w postaci par nazwa\_pola typ\_pola. W zaprezentowanym przykładzie zdefiniowano typ o nazwie DanePracownika, którego struktura składa się z dwóch pól typu varchar2(100) o nazwach nazwisko i imię.

Po zdefiniowaniu typu rekordowego można już zadeklarować zmienną rekordową w standardowy sposób (nazwa\_zmiennej typ\_zmiennej). W przykładzie na slajdzie zadeklarowano zmienną v\_pracownik typu DanePracownika. Zmienna ta ma takie same pola, jak jej typ.

Dostęp do zmiennej rekordowej w bloku PL/SQL wykonuje się korzystając z tzw. notacji kropkowej. Odwołując się do pola zmiennej należy podać nazwę zmiennej, a następnie po kropce nazwę pola. W przykładzie zaprezentowano przypisanie wartości do obu pól zmiennej v\_pracownik.



## Atrybuty %TYPE i %ROWTYPE

- Atrybut %TYPE – do deklarowania zmiennej prostej na podstawie typu atrybutu relacji bazy danych lub typu innej zmiennej.
- Atrybut %ROWTYPE – do deklarowania zmiennej rekordowej w oparciu o schemat relacji, kursora lub typ innej zmiennej rekordowej.

### DECLARE

```
v_nazwisko pracownicy.nazwisko%TYPE;  
v_nazwisko_szefa v_nazwisko%TYPE := 'Kowalski';  
v_dane_pracownika pracownicy%ROWTYPE;
```

Mechanizmem bardzo ułatwiającym deklarowanie zmiennych są atrybuty %TYPE i %ROWTYPE. Atrybut %TYPE umożliwia zadeklarowanie zmiennej na podstawie definicji istniejącej zmiennej lub na podstawie definicji wskazanego atrybutu relacji z bazy danych. To drugie rozwiązanie stosuje się wszędzie tam, gdzie istnieje konieczność zadeklarowania zmiennej, która ma przechowywać dane pobrane z bazy danych. Z kolei atrybut %ROWTYPE pozwala zadeklarować zmienną rekordową na podstawie definicji innej zmiennej rekordowej lub kursora (kursorami zajmiemy się w następnym ćwiczeniu). Jednak najczęściej atrybutu %ROWTYPE używa się, gdy potrzebujemy zmiennej rekordowej, która ma przechować cały rekord ze wskazanej relacji.

W zaprezentowanym na slajdzie przykładzie zadeklarowano zmienną v\_nazwisko, której typ będzie taki sam, jak typ atrybutu NAZWISKO relacji PRACOWNICY. Następnie zadeklarowano kolejną zmienną, v\_nazwisko\_szefa, której typ będzie identyczny z typem zmiennej v\_nazwisko. Ostatnia zmienna o nazwie v\_dane\_pracownika jest zmienną rekordową, a jej struktura będzie identyczna ze strukturą rekordu relacji PRACOWNICY.



- Deklarowana w sekcji **DECLARE**:

**DECLARE**

```
nazwa_stalej CONSTANT typ(długość) := wartość;
```

- Musi zostać zainicjalizowana przy deklaracji, nigdy nie zmienia przypisanej wartości.

**DECLARE**

```
c_pi CONSTANT NUMBER(5,4) := 3.1415;  
c_fałsz CONSTANT BOOLEAN := FALSE;
```

Przejdziemy teraz do omówienia stałych. Stałą można rozumieć jako zmienną, która nie zmienia przypisanej jej podczas deklaracji wartości przez cały czas trwania programu. Stała może stać tylko po prawej stronie operacji przypisania – jakkolwiek próba zmiany wartości stałej powoduje przerwanie działania programu i wygenerowanie komunikatu o błędzie.

Dobrym nawykiem jest tworzenie nazw stałych przez użycie przedrostka „c\_” w nazwie. Taka konwencja nazewnicza znacznie poprawia czytelność programu.

W przykładzie zadeklarowano dwie stałe: liczbową `c_pi` o wartości 3.1415 i logiczną `c_fałsz` o wartości `FALSE`.

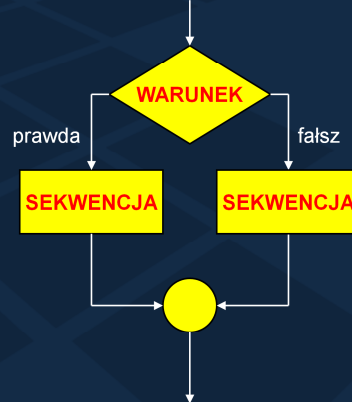


## Rodzaje struktur sterujących

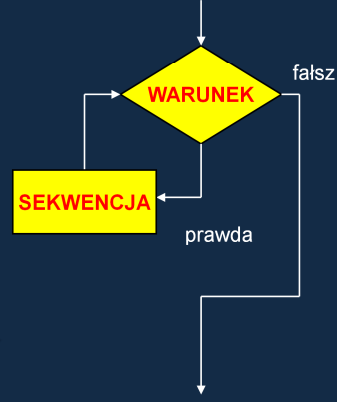
### SEKWENCJA



### SELEKCJA



### ITERACJA



Ćwiczenie 11 – PL/SQL (12)

Przejdziemy teraz do omówienia podstawowych struktur sterujących, wykorzystywanych w programach PL/SQL. Rozpoczniemy od sekwencji operacji.



## Sekwencja

- Sekwencja poleceń, wykonywanych w określonym porządku.
- Każde polecenie kończy się średnikiem.
- Przykład:

```
DECLARE
  v_i NUMBER(3) := 0;
  v_nazwa VARCHAR2(10);
BEGIN
  v_i := v_i + 1;
  v_nazwa := 'ABC';
  v_nazwa := v_nazwa || 'DEF';
END;
```

Ćwiczenie 11 – PL/SQL (13)

Sekwencja jest ciągiem poleceń, wykonywanych w określonym porządku. Sekwencja jest umieszczana w sekcjach: wykonywalnej lub obsługi błędów bloku PL/SQL, elementem sekwencji może być również blok zagnieżdżony. Każde polecenie w sekwencji kończy się średnikiem. W zaprezentowanym na slajdzie przykładzie w sekcji wykonywalnej bloku mamy sekwencję trzech poleceń przypisujących wartości do zadeklarowanych wcześniej zmiennych.



## 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);
```

- ustaw zmienną SERVEROUTPUT narzędzia SQL\*Plus na wartość ON przed wykonaniem programu.

```
SQL> SET SERVEROUTPUT ON
```

Ćwiczenie 11 – PL/SQL (14)

Jak już wcześniej wspomniano, język PL/SQL jest nastawiony na przetwarzanie danych i jego możliwości w zakresie interakcji z użytkownikiem są nader skromne.

Jeśli zachodzi konieczność wczytania wartości do programu, można użyć tzw. zmiennych podstawienia. Zmienna podstawienia to dowolny literał, rozpoczynający się od znaku „&”. Jeśli program został zapisany w narzędziu SQL\*Plus, przed wykonaniem programu narzędzie przegląda go w poszukiwaniu zmiennych podstawienia, jeśli je znajdzie, pyta użytkownika o wartości dla tych zmiennych. Podane przez użytkownika wartości zostają wstawione w miejsca zmiennych podstawienia i program zostaje wykonany. Podkreślmy – zmienne podstawienia są zamieniane na wartości przed wykonaniem programu, a nie w trakcie. Stąd nie można ich użyć np. do pytania użytkownika co do przebiegu programu (np. do jego rozgałęzienia).

Jeśli istnieje konieczność wypisania komunikatu z programu PL/SQL, można do tego celu użyć procedury PUT\_LINE z pakietu DBMS\_OUTPUT. Parametrem procedury jest ciąg znaków, który ma zostać wyświetlony na konsoli. Aby komunikaty pojawiały się na konsoli, konieczne jest ustawienie w narzędziu SQL\*Plus wartości zmiennej SETSERVEROUTPUT na ON. Należy jednak pamiętać, że komunikaty z programu, generowane przez wykonanie wspomnianej procedury, pojawią się na konsoli dopiero po wykonaniu całego programu, a nie w momencie wykonania linii zawierającej procedurę PUT\_LINE.



## Interakcja z użytkownikiem (2)

- Przykład:

```
SQL> 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;
```

Ćwiczenie 11 – PL/SQL (15)

Powyższy slajd przedstawia przykład programu, pobierającego od użytkownika dwie wartości przy wykorzystaniu zmiennych podstawienia &liczba i &tekst.

Wartości te zostają użyte do zainicjalizowania zmiennych v\_i i v\_nazwa.

Zwróćmy uwagę, że zmienna podstawienia &tekst została ujęta w apostrofy.

W sekcji wykonywalnej bloku następuje wypisanie na konsoli zdania „Zmienna v\_i: <wartość>”, gdzie pod <wartość> zostaje wstawiona wartość zmiennej v\_i, skonwertowana do ciągu znaków. Kolejna operacja dokleja do ciągu znaków, podanego przez użytkownika ciąg „ABC”. Ostatnia operacja wypisuje zawartość zmiennej v\_nazwa na konsoli.



## Selekcja – instrukcja IF ... THEN (1)

### postać podstawowa

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

### postać rozszerzona

```
IF warunek THEN
    sekwencja poleceń_1
ELSE
    sekwencja poleceń_2
END IF;
```

### pełna postać

```
IF warunek_1 THEN
    sekwencja poleceń_1
ELSIF warunek_2 THEN
    sekwencja poleceń_2
ELSIF warunek_3 THEN
    sekwencja poleceń_3
...
ELSE
    sekwencja poleceń_n
END IF;
```

Ćwiczenie 11 – PL/SQL (16)

Przejdziemy teraz do omawiania kolejnej konstrukcji sterującej języka PL/SQL, tzw. selekcji. Zajmiemy się najpierw instrukcją IF ... THEN. W postaci podstawowej składa się ona ze słowa kluczowego IF, po którym następuje warunek logiczny. Jeśli warunek jest spełniony, wówczas następuje wykonanie sekwencji poleceń podanych po słowie THEN. Całą konstrukcję kończy słowo kluczowe END IF. W postaci rozszerzonej instrukcja uzupełniona jest przez słowo kluczowe ELSE, po którym następuje sekwencja poleceń, jakie mają być wykonane w przypadku, gdy warunek logiczny po słowie IF jest fałszywy.

Instrukcja w pełnej postaci pozwala na testowanie prawdziwości wielu warunków. Jeśli warunek\_1 po słowie IF jest prawdziwy, wówczas wykonywana jest sekwencja poleceń\_1. W przeciwnym wypadku sprawdzany jest warunek\_2 po słowie ELSIF. Jeśli jest on prawdziwy, wykonywana jest sekwencja poleceń\_2, jeśli fałszywy, następuje sprawdzenie warunku\_3, itd. Gdy żaden z warunków nie jest prawdziwy, wykonywana jest sekwencja poleceń\_n, umieszczona po słowie kluczowym ELSE.





## Selekcja – instrukcja IF ... THEN (2)

- przykład:

```
DECLARE
  v_prawda BOOLEAN := true;
BEGIN
  IF v_prawda THEN
    dbms_output.put_line('prawda');
  ELSE
    dbms_output.put_line('fałsz');
  END IF;
END;
```

Przykład na powyższym slajdzie pokazuje użycie selekcji w postaci rozszerzonej.



## Selekcja – instrukcja CASE (1)

### postać prosta

```

CASE wyrażenie
  WHEN wartość_1 THEN
    sekwencja poleceń_1
  WHEN wartość_2 THEN
    sekwencja poleceń_2
  ...
[ELSE
  sekwencja poleceń_n]
END [CASE];

```

### postać z listą wyrażeń

```

CASE
  WHEN warunek_1 THEN
    sekwencja poleceń_1
  WHEN warunek_2 THEN
    sekwencja poleceń_2
  ...
[ELSE
  sekwencja poleceń_n]
END [CASE];

```

Ćwiczenie 11 – PL/SQL (18)

Drugim rodzajem operacji selekcji jest instrukcja CASE. Pozwala ona na testowanie wielu warunków, jej zapis jest bardziej zwięzły niż zapis instrukcji IF ... THEN.

Instrukcja CASE ma dwie postaci. W pierwszej po słowie CASE umieszczamy wyrażenie. Wartość wyrażenia jest dopasowywana do jednej z wartości, umieszczonych po słowach WHEN. Jeśli dopasowanie zakończy się powodzeniem dla wartość\_i, wykonywana jest sekwencja poleceń\_i. Opcjonalna sekcja, rozpoczynająca się od słowa ELSE, umożliwia zdefiniowanie sekwencji poleceń, które zostaną wykonane w przypadku, gdy nie zajdzie żadne dopasowanie. Istotnym ograniczeniem tej postaci instrukcji CASE jest możliwość testowania tylko równości wyrażenia z dostarczonymi wartościami

Druga postać instrukcji CASE nie posiada powyższego ograniczenia. W każdej z klauzul WHEN znajduje się warunek logiczny, jeśli jest on prawdziwy, wykonana zostaje sekwencja poleceń umieszczona po słowie THEN danej klauzuli WHEN. Podobnie jak w pierwszej postaci można dodać sekcję ELSE, której sekwencja poleceń zostanie wykonana w przypadku, gdy żaden z warunków nie będzie prawdziwy.

W obu postaciach instrukcji CASE po znalezieniu pierwszego dopasowania lub pierwszego prawdziwego warunku dalsze poszukiwanie nie jest kontynuowane.



## Selekcja – instrukcja CASE (2)

- Przykłady:

```
DECLARE
v_vat number(2,2) := 0.22;
v_proc varchar2(20);
BEGIN
v_proc := CASE v_vat
  WHEN 0 THEN '0%'
  WHEN 0.7 THEN '7%'
  WHEN 0.22 THEN '22%'
END;
dbms_output.put_line(v_proc);
END;
```

```
DECLARE
v_vat number(2,2) := 0.22;
v_proc varchar2(20);
BEGIN
  CASE
    WHEN v_vat = 0 THEN
      v_proc := '0%';
    WHEN v_vat = 0.7 THEN
      v_proc := '7%';
    WHEN v_vat = 0.22 THEN
      v_proc := '22%';
  END CASE;
dbms_output.put_line(v_proc);
END;
```

Powyższy slajd przedstawia dwie wersje tego samego programu, zrealizowane przy zastosowaniu obu postaci instrukcji CASE. Program umieszcza w zmiennej v\_proc ciąg znaków, określający procent podatku VAT w zależności od wartości zmiennej v\_vat.



## Iteracja – instrukcja LOOP (1)

### pętla bezwarunkowa

```
LOOP
  sekwencja poleceń
END LOOP;
```

### pętla z EXIT WHEN

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

### pętla z EXIT

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

Przejdziemy teraz do omawiania kolejnego rodzaju instrukcji sterujących – pętli. Na początek zajmiemy się pętlą LOOP.

Pętla LOOP występuje w trzech postaciach. Pierwsza postać, określana jako pętla bezwarunkowa, rozpoczyna się od słowa kluczowego LOOP, po którym następuje sekwencja poleceń, które mają być iterowane. Konstrukcja kończy się słowem kluczowym END LOOP. Omawiana postać nie posiada żadnego warunku wyjścia z pętli – oznacza to, że iteracja będzie realizowana w nieskończoność.

Druga postać pętli LOOP posiada wewnątrz pętli warunek wyjścia z pętli. Jest to konstrukcja EXIT WHEN <warunek>. Iteracje pętli realizowane są tak długo, aż wartość wyrażenia nie będzie prawdą. W takim przypadku pętla jest przerywana i sterowanie przechodzi do pierwszej instrukcji po słowie END LOOP.

Odmianą pętli LOOP z konstrukcją EXIT WHEN jest pętla LOOP z EXIT. Umieszczone wewnątrz pętli polecenie EXIT powoduje przerwanie pętli i przejście do pierwszej instrukcji po słowie END LOOP. Polecenie EXIT najczęściej umieszcza się w instrukcji warunkowej, jednak nic nie stoi na przeszkodzie, aby umieścić je w innej instrukcji.



## Iteracja – instrukcja LOOP (2)

- Przykład:

```
DECLARE
  v_licznik number(2) := 0;
  c_liczba_iteracji CONSTANT number(2) := 5;
BEGIN
  LOOP
    v_licznik := v_licznik + 1;
    dbms_output.put_line('Iteracja nr ' || to_char(v_licznik));
    EXIT WHEN v_licznik = c_liczba_iteracji;
  END LOOP;
END;
```

Przykład, zaprezentowany na powyższym slajdzie, pokazuje zastosowanie pętli LOOP z EXIT WHEN. Pętla będzie wykonywana tak długo, dopóki wartość zmiennej v\_licznik nie będzie równa stałej c\_liczba\_iteracji. W każdej iteracji pętli wartość zmiennej v\_licznik jest zwiększana o 1 i wypisywana na konsoli.



## Iteracja – instrukcja WHILE

- Postać: **WHILE** warunek **LOOP**  
sekwencja poleceń  
**END LOOP;**

- Przykład: **DECLARE**  
v\_licznik NUMBER(2) := 0;  
c\_liczba\_iteracji CONSTANT NUMBER(2) := 5;  
**BEGIN**  
  **WHILE** v\_licznik < c\_liczba\_iteracji **LOOP**  
    v\_licznik := v\_licznik + 1;  
    dbms\_output.put\_line('Iteracja nr ' || to\_char(v\_licznik));  
  **END LOOP;**  
**END;**

Drugim rodzajem iteracji jest pętla WHILE. Po słowie kluczowym WHILE umieszcza się warunek logiczny, iteracje pętli realizowane są wtedy, gdy warunek logiczny jest prawdziwy. Pomiedzy słowami kluczowymi LOOP i END LOOP znajduje się iterowana sekwencja poleceń. Pętla WHILE może zostać również przerwana wykonaniem polecenia EXIT lub EXIT WHEN.

Przykład zaprezentowany na slajdzie pokazuje realizację tego samego zadania co przykład objaśniający zastosowanie pętli LOOP.



## Iteracja – instrukcja FOR

- Postać:

```
FOR licznik IN [REVERSE] dolna_granica .. górna_granica LOOP
    sekwencja poleceń
END LOOP;
```

- Przykład:

```
DECLARE
    c_liczba_iteracji CONSTANT NUMBER(2) := 5;
BEGIN
    FOR v_licznik IN 1..c_liczba_iteracji LOOP
        dbms_output.put_line('Iteracja nr ' || to_char(v_licznik));
    END LOOP;
END;
```

Ostatnim rodzajem iteracji jest pętla FOR. Różni się ona od poprzednich rodzajów pętli tym, że pozwala ona dokładnie określić liczbę iteracji. Po słowie FOR umieszcza się zmienną licznikową (nie należy jej deklarować!), po słowie IN podaje się najpierw dolną, a po dwóch kropkach górną granicę przedziału, w ramach którego ma zmieniać się wartość zmiennej licznikowej. Domyślnie wartości zmiennej licznikowej zmieniają się od dolnej do górnej granicy, jeśli interesuje nas porządek odwrotny, po słowie IN należy dodać słowo kluczowe REVERSE. Iterowaną sekwencję poleceń umieszczamy pomiędzy słowami LOOP i END LOOP.

Wewnątrz pętli można odczytywać wartość zmiennej licznikowej, natomiast jej zmienianie jest zabronione. Pętla FOR może zostać przerwana wykonaniem polecenia EXIT lub EXIT WHEN.

Zaprezentowany na powyższym slajdzie przykład realizuje to samo zadanie co przykład objaśniający zastosowanie pętli LOOP.



## Instrukcja NULL

- Nie wykonuje żadnej akcji.
- Używana na etapie projektowania programu do testowania struktur sterujących.
- Przykład:

```
DECLARE
    v_czy_zapłacona BOOLEAN := true;
BEGIN
    IF NOT v_czy_zapłacona THEN
        NULL;
    ELSE
        dbms_output.put_line('Faktura opłacona!');
    END IF;
END;
```

Ćwiczenie 11 – PL/SQL (24)

Uzupełnieniem zbioru instrukcji PL/SQL, jaki poznajemy w ramach niniejszego ćwiczenia, jest instrukcja NULL. Nie realizuje ona żadnej akcji i jest właściwie swego rodzaju wypełniaczem, wykorzystywanym podczas testowania programów, których nie wszystkie elementy zostały jeszcze zdefiniowane.

Zanalizujmy zaprezentowany na slajdzie przykład. Programista zdefiniował już akcję, jaka ma być zrealizowana w przypadku, gdy wartość zmiennej `v_czy_zapłacona` jest prawdziwa. Brak jednak jeszcze sekwencji poleceń w sytuacji, gdy zmienna `v_czy_zapłacona` jest fałszywa. Aby móc wykonać program celem przetestowania działania już istniejącej części, programista po słowie `THEN` wstawił instrukcję `NULL`, przez co uniknął wygenerowania błędu niepoprawnej składni instrukcji `IF-THEN`. Oczywiście instrukcja `NULL` w gotowym programie zostanie usunięta.





## DML w PL/SQL (1)

- Zapytanie musi zwrócić dokładnie jeden rekord.
- W zapytaniu dodatkowa klauzula **INTO**, w niej:
  - lista zmiennych prostych, liczba zmiennych musi odpowiadać liczbie wyrażeń w klauzuli **SELECT** zapytania, typy muszą być zgodne, lub
  - zmienna rekordowa o strukturze zgodnej ze strukturą rekordu, odczytywanego przez zapytanie.
- Średnik umieszczamy na końcu całego polecenia.

Ćwiczenie 11 – PL/SQL (25)

Zajmiemy się teraz umieszczaniem w bloku PL/SQL zapytań do bazy danych. Zapytanie nie różni się prawie od zwykłego zapytania SQL z dwoma wyjątkami. Po pierwsze, programista musi zagwarantować, że zapytanie odczyta dokładnie jeden rekord z bazy danych. Gdy zapytanie nie odczyta żadnego rekordu lub odczyta więcej niż jeden rekord, program zostanie przerwany z komunikatem o błędzie. Po drugie, po klauzuli **SELECT** zapytania umieszcza się dodatkową klauzulę **INTO** z listą zmiennych, do których trafią wartości odczytane przez zapytanie. W klauzuli **INTO** można podać listę zmiennych prostych, ich liczba i typy powinny odpowiadać liczbie i typom wartości wyrażeń, odczytywanych przez zapytanie z bazy danych. Innym rozwiązaniem jest umieszczenie w klauzuli **INTO** pojedynczej zmiennej rekordowej o strukturze odpowiadającej strukturze rekordu, odczytywanego przez zapytanie.

Średnik umieszczamy po całym zapytaniu (nie po końcu linii w przypadku zapytania zajmującego w programie wiele linii).



## DML w PL/SQL (2)

### DECLARE

```
v_suma_plac NUMBER(6,2);  
v_ilu_pracownikow NUMBER(5);  
v_zespol zespol%ROWTYPE;
```

### BEGIN

```
SELECT * INTO v_zespol FROM zespol  
WHERE nazwa = 'ADMINISTRACJA';  
SELECT sum(placa_pod), count(*)  
INTO v_suma_plac, v_ilu_pracownikow  
FROM pracownicy WHERE id_zesp = v_zespol.id_zesp;  
dbms_output.put_line('Suma płac: ' || to_char(v_suma_plac));  
dbms_output.put_line('Pracowników: ' || to_char(v_ilu_pracownikow));
```

### END;

Ćwiczenie 11 – PL/SQL (26)

W zaprezentowanym na bieżącym slajdzie przykładzie pierwsze zapytanie odczytuje z relacji ZESPOLY rekord, opisujący zespół o nazwie „ADMINISTRACJA”. Wartość rekordu zostaje zachowana w zmiennej rekordowej v\_zespol. Następnie na podstawie wartości pola id\_zesp zmiennej v\_rekord drugie zapytanie odczytuje sumę płac i liczbę pracowników w zespole ADMINISTRACJA. Wyliczone wartości trafiają do zmiennych prostych v\_suma\_plac i v\_ilu\_pracownikow, a następnie zostają wypisane na konsoli.



## DML w PL/SQL (3)

- INSERT, UPDATE, DELETE – postać poleceń identyczna jak w SQL.
- Opcjonalna klauzula **RETURNING ... INTO**.
- Przykład:

### DECLARE

```
v_id_prac pracownicy.id_prac%TYPE;
```

### BEGIN

```
INSERT INTO pracownicy (id_prac, imie, nazwisko)
```

```
VALUES (400, 'Jacek', 'Kowalski')
```

```
RETURNING id_prac INTO v_id_prac;
```

```
dbms_output.put_line('Id nowego pracownika: ' || to_char(v_id_prac));
```

```
END;
```

Ćwiczenie 11 – PL/SQL (27)

Zastosowanie poleceń INSERT, UPDATE i DELETE w programie PL/SQL w podstawowej postaci nie różni się niczym od postaci stosowanej w SQL. Opcjonalnie do poleceń można dodać klauzulę RETURNING INTO, która pozwala na zapisanie we wskazanej zmiennej:

- wartości atrybutów rekordu, wstawionego przez zlecenie INSERT,
- wartości atrybutów rekordu, zmodyfikowanego przez zlecenie UPDATE,
- wartości atrybutów rekordu, usuniętego przez zlecenie DELETE.

Przykład zaprezentowany na powyższym slajdzie pokazuje polecenie INSERT, wstawiające do relacji PRACOWNICY rekord opisujący pracownika Jacka Kowalskiego z identyfikatorem 400. Dzięki klauzuli RETURNING INTO wartość, jaka w nowym rekordzie została zapisana w atrybucie ID\_PRAC, zostaje zapisana w zmiennej v\_id\_prac.



## DML w PL/SQL (4)

- Wykorzystanie zmiennych rekordowych:
  - INSERT
  - UPDATE – pseudokolumna **ROW**

```
DECLARE
  v_zespol zespoly%ROWTYPE;
BEGIN
  v_zespol.id_zesp := 70;
  v_zespol.nazwa := 'SIECI';
  v_zespol.adres := 'PIOTROWO 3A';
  INSERT INTO zespoly VALUES v_zespol;
  v_zespol.nazwa := 'SIECI KOMPUTEROWE';
  v_zespol.adres := 'WIENIAWSKIEGO';
  UPDATE zespoly SET row = v_zespol
  WHERE id_zesp = 70;
END;
```

Ćwiczenie 11 – PL/SQL (28)

Komentarza wymaga jeszcze użycie zmiennych rekordowych w poleceniach DML. W przypadku polecenia INSERT zmienną rekordową można podać bezpośrednio w klauzuli VALUES polecenia. W zaprezentowanym na slajdzie przykładzie do relacji ZESPOLY zostaje wstawiony rekord, którego wartości dla poszczególnych atrybutów przechowuje zmienna rekordowa v\_zespol.

Dla polecenia UPDATE istnieje możliwość modyfikacji wszystkich atrybutów rekordu relacji do wartości, jakie zawiera zmienna rekordowa. Do tego celu należy użyć konstrukcji SET ROW = v\_zmienna\_rekordowa. W przykładzie wartości atrybutów rekordu, opisującego zespół o identyfikatorze 70, zostają zmienione do wartości, przechowywanych w zmiennej v\_zespol.



## Zadania

1. Zadeklaruj zmienne `v_tekst` i `v_liczba` o wartościach odpowiednio „Witaj, świecie!” i 1000.456. Wyświetl wartości tych zmiennych.
2. Napisz program dodający do siebie dwie liczby całkowite. Liczby powinny być podawane przez użytkownika z konsoli.
3. Napisz program, który oblicza pole powierzchni i obwód koła o podanym promieniu. W programie posłuż się zdefiniowaną przez siebie stałą `c_pi = 3.14`.

Bieżący slajd rozpoczyna zbiór zadań, których celem jest utrwalenie wiadomości o podstawowych konstrukcjach języka PL/SQL.



## Zadania

4. Napisz program, który wyświetli informacje o najlepiej zarabiającym pracowniku w formacie „Najlepiej zarabia <imię> <nazwisko>. Pracuje on jako <etat>.”.
5. Napisz program, który dla podanego przez użytkownika n obliczy wartość wyrażenia  $n! = 1 * 2 * 3 * \dots * n$  (silnię).



```
1 DECLARE
    v_tekst VARCHAR2(100) := 'Witaj świecie!';
    v_liczba NUMBER(7,3) := 1000.456;
BEGIN
    dbms_output.put_line('Zmienna v_tekst: ' || v_tekst);
    dbms_output.put_line('Zmienna v_liczba: ' || to_char(v_liczba));
END;
```

```
2 DECLARE
    v_liczba_1 NUMBER(5) := &pierwsza_liczba;
    v_liczba_2 NUMBER(5) := &druga_liczba;
    v_wynik NUMBER(6);
BEGIN
    v_wynik := v_liczba_1 + v_liczba_2;
    dbms_output.put_line('Wynik dodawania: ' || v_wynik);
END;
```

Ćwiczenie 11 – PL/SQL (31)

Bieżący slajd przedstawia rozwiązania zadań (1) i (2), których treść zacytowano poniżej.

- (1) Zadeklaruj zmienne `v_tekst` i `v_liczba` o wartościach odpowiednio „Witaj, świecie!” i 1000.456. Wyświetl wartości tych zmiennych.
- (2) Napisz program dodający do siebie dwie liczby całkowite. Liczby powinny być podawane przez użytkownika z konsoli.



```
3 DECLARE
    v_r NUMBER(8,3) := &promien;
    c_pi CONSTANT NUMBER(3,2) := 3.14;
    v_pole NUMBER(10,3);
    v_obwod NUMBER(10,3);
BEGIN
    v_pole := c_pi * v_r * v_r;
    v_obwod := 2 * c_pi * v_r;
    dbms_output.put_line('Pole koła: ' || to_char(v_pole));
    dbms_output.put_line('Obwód koła: ' || to_char(v_obwod));
END;
```

Bieżący slajd przedstawia rozwiązanie zadania (3), którego treść zacytowano poniżej.

- (3) Napisz program, który oblicza pole powierzchni i obwód koła o podanym promieniu. W programie posłuż się zdefiniowaną przez siebie stałą  $c\_pi = 3.14$ .





```
DECLARE
    v_zdanie_1 VARCHAR2(100);
    v_zdanie_2 VARCHAR2(100);
BEGIN
    SELECT 'Najlepiej zarabia ' || imie || ' ' || nazwisko || ',
           'Pracuje on jako ' || etat || '.'
    INTO v_zdanie_1, v_zdanie_2
    FROM pracownicy WHERE placa_pod = (
        SELECT max(placa_pod) FROM pracownicy);
    dbms_output.put_line(v_zdanie_1 || v_zdanie_2);
END;
```

Bieżący slajd przedstawia rozwiązanie zadania (4), którego treść zacytowano poniżej.

- (4) Napisz program, który wyświetli informacje o najlepiej zarabiającym pracowniku w formacie „Najlepiej zarabia <imię> <nazwisko>. Pracuje on jako <etat>.”.



```
5 DECLARE
    n NUMBER(5) = &liczba;
    v_wynik NUMBER(10) := 1;
BEGIN
    FOR v_i IN 1.. n LOOP
        v_wynik := v_wynik * v_i;
    END LOOP;
    dbms_output.put_line(to_char(n) || '! = ' || to_char(v_wynik));
END;
```

Bieżący slajd przedstawia rozwiązanie zadania (5) którego treść zacytowano poniżej.

(5) Napisz program, który dla podanego przez użytkownika n obliczy wartość wyrażenia  $n! = 1 * 2 * 3 * \dots * n$  (silnie).



## Podsumowanie

- Język PL/SQL umożliwia konstrukcję programów, wykonujących określone operacje w bazie danych.
- Anonimowy blok jest podstawową konstrukcją języka PL/SQL.
- Blok może zawierać instrukcje sekwencji, selekcji oraz iteracji.
- W bloku PL/SQL można umieszczać operacje DML, umieszczanie operacji DDL i DCL jest zabronione.

Ćwiczenie 11 – PL/SQL (35)

W zakończonym ćwiczeniu została zaprezentowana koncepcja języka PL/SQL, umożliwiającego tworzenie programów, wykonujących określone operacje w bazie danych. Przedstawiono podstawową konstrukcję języka, anonimowy blok PL/SQL, oraz pozostałe konstrukcje: sekwencję, selekcję i iterację. Dalej omówiono zasady stosowania operacji DML w anonimowym bloku PL/SQL. Każde z omówionych zagadnień zostało utrwalone przez serię zadań.