

Instytut Informatyki
Wydział Elektrotechniki, Automatyki i Informatyki
Politechnika Opolska

1

PL-SQL

Opracowanie: dr inż. Ewelina Piotrowska
e.piotrowska@po.edu.pl
www.e.piotrowska.po.edu.pl

OPOLE 2019

Przedmioty:
Modelowanie baz danych

2

Podstawy

Wprowadzenie

3

- PL/SQL to proceduralne rozszerzenie SQL stosowane w szbd ORACLE
- PL/SQL jest językiem o strukturze blokowej.
- Za pomocą języka PL/SQL tworzy się:
 - anonimowe bloki programu
 - procedury i funkcje składowane
 - pakiety
 - wyzwalacze bazy danych (ang. triggers)
- Aplikacje korzystające z PL/SQL mogą wykonywać procedury i funkcje języka PL/SQL składowane w bazie danych oraz wysyłać własne programy do wykonania przez serwer.

Blok anonimowy

4

- Każdy blok anonimowy może składać się z trzech części: deklaracyjnej, wykonywalnej i obsługującej wyjątki
- Możliwe jest zagnieżdżanie bloków.
- PL/SQL umożliwia wykorzystanie
 - ▣ zmiennych i stałych
 - ▣ struktur kontroli, w tym instrukcji warunkowych, pętli, etykiet, instrukcji skoku, instrukcji wyboru warunkowego
 - ▣ kursorów
 - ▣ wyjątków i mechanizmu obsługi błędów.
- Jedynymi instrukcjami SQL, które mogą pojawić się w bloku PL/SQL są: SELECT, DELETE, INSERT, UPDATE, COMMIT, ROLLBACK, SAVEPOINT.
 - ▣ Ponadto można wykorzystywać wszystkie funkcje, operatory i pseudokolumny dostępne w SQL.

```
[DECLARE
... deklaracje ]
BEGIN
... rozkazy
[EXCEPTIONS
... wyjątki ]
END;
```

Zmienne

5

- Zmienne i stałe muszą być zadeklarowane w **sekcji deklaracyjnej** bloku.

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

- Przykład

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

Nadawanie wartości zmiennym

6

- Nadanie wartości poprzez przypisanie
 - UWAGA!!! Zmienna, która została zadeklarowana lecz nie została zainicjalizowana, posiada wartość NULL. Użycie takiej zmiennej może spowodować nieprawidłowe wyniki.

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

Nadawanie wartości zmiennym

7

- Nadanie wartości przez wczytanie danych z bazy danych do zmiennej

np.

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

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

Stałe

8

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

```
DECLARE
    godziny_pracy CONSTANT NUMBER := 42;
    pp CONSTANT VARCHAR2(30) := 'Politechnika Opolska';
```

Typy danych

9

- 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

%TYPE, %ROWTYPE

10

- Atrybuty %TYPE, %ROWTYPE umożliwiają określanie typów na podstawie wcześniej utworzonych zmiennych, kolumn tabeli lub wierszy tabeli.
 - Atrybut %TYPE zawiera typ innej zmiennej lub typ atrybutu w bazie danych.
 - 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
    v_nazwisko PRAC.NAZWISKO%TYPE;
    v_nazwa_zespolu ZESP.NAZWA%TYPE;
DECLARE
    v_pracownik PRAC%ROWTYPE;
    v_zespol ZESP%ROWTYPE;
```

Typ rekordowy

11

- Zmienną typu rekordowego można utworzyć również korzystając z zdefiniowanego wcześniej własnego typu rekordowego

```
DECLARE
    TYPE pracownik_rek IS RECORD (
        nazwisko prac.nazwisko%TYPE,
        zespól zesp.nazwa%TYPE,
        pensja NUMBER(9,2) NOT NULL := 900);

    nowy_prac pracownik_rek;
```

Deklaracja nowej zmiennej

Typ tablicowy

12

- Zmienne typu tablicowego mają strukturę dwukolumnowej matrycy.
 - ▣ Pierwsza kolumna jest wykorzystywana do indeksowania wierszy tablicy i musi być typu `binary_integer`.
 - ▣ Druga kolumna może być jednego z typów prostych np. `char`, `number`, `date`...

```
DECLARE
    TYPE nazwa_typu IS TABLE OF typ_danych
    [NOT NULL] INDEX BY BINARY_INTEGER;
```

```
DECLARE
    TYPE tab_liczbowa IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
    tablica tab_liczbowa;
    n binary_integer := 0;
    wartosc number(2);
```

zmienna typu tablicowego

Zmienna wykorzystana do adresowania komórek tablicy

wartość komórki tablicy

Podtypy

13

- Każdy typ danych definiuje zbiór poprawnych wartości i zbiór operatorów, które mogą być zastosowane 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 IS DATE NOT NULL;
    SUBTYPE Pieniadze IS NUMBER(9,2);
    ...
    v_moje_urodziny DataUr;
    v_moja_pensja Pieniadze;
```

- Konwersje typów mogą być realizowane niejawnie lub jawnie za pomocą funkcji konwertujących np. to_char() lub to_date().

INSTRUKCJE STERUJĄCE

14

- Instrukcja warunkowa IF-THEN-ELSE występuje w trzech wariantach:

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

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

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

Pętla LOOP

15

- 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

16

```
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

17

- Przed każdą iteracją sprawdzany jest warunek.
- Pętla jest wykonywana 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.
- UWAGA!!!
 - ▣ W sekwencji operacji musi znaleźć się polecenie, które zmieni warunek, w przeciwnym przypadku pętla jest nieskończona.

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

Przykład: Wyznaczanie NWD

18

```
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;
/
```

Kiedy używać ; i /

19

- I wanted to clarify some more use between the ; and the /
- 59 In SQLPLUS:
1. ; means "terminate the current statement, execute it and store it to the SQLPLUS buffer"
 2. <newline> after a D.M.L. (SELECT, UPDATE, INSERT,...) statement or some types of D.D.L (Creating Tables and Views) statements (that contain no ;), it means, store the statement to the buffer but do not run it.
 3. / after entering a statement into the buffer (with a blank <newline>) means "run the D.M.L. or D.D.L. or PL/SQL in the buffer."
 4. RUN or R is a sqlplus command to show/output the SQL in the buffer and run it. It will not terminate a SQL Statement.
 5. / during the entering of a D.M.L. or D.D.L. or PL/SQL means "terminate the current statement, execute it and store it to the SQLPLUS buffer"

NOTE: Because ; are used for PL/SQL to end a statement ; cannot be used by SQLPLUS to mean "terminate the current statement, execute it and store it to the SQLPLUS buffer" because we want the whole PL/SQL block to be completely in the buffer, then execute it. PL/SQL blocks must end with:

```
END;
/
```

[share](#) [improve this answer](#)
[edited Jan 15 '14 at 18:57](#)
[answered Jan 15 '14 at 18:16](#)

<https://stackoverflow.com/questions/1079949/when-do-i-need-to-use-a-semicolon-vs-a-slash-in-oracle-sql>

Pętla FOR

20

- 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 $-2^{31} \div 2^{31}$
- UWAGI
 - ▣ 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

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

Polecenia sterujące GOTO i NULL

21

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

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

22

```
DECLARE
    v_tek VARCHAR2(20);
BEGIN
    <<poczatek>>
    v_tek := 'JEDEN ';
    GOTO dwa;
    <<trzy>>
    v_tek := v_tek || 'TRZY ';
    GOTO drukuj;
    <<drukuj>>
    DBMS_OUTPUT.PUT_LINE(v_tek);
    GOTO koniec;
    <<dwa>>
    v_tek := v_tek || 'DWA ';
    GOTO trzy;
    <<koniec>>
    NULL;
END;
/
```

23

Kursory

Instrukcja SELECT w PL/SQL

24

- Instrukcja SELECT ma w PL/SQL swoją specjalną postać.
- Wynik zapytania nie jest wyświetlany na ekranie, tylko zostaje zapisany na zmiennych - zamieszczanych w klauzuli INTO - która jest wymaganą klauzulą instrukcji SELECT w PL/SQL.
- Instrukcja SELECT musi zwracać dokładnie jeden wiersz wyników.
- Składnia instrukcji SELECT w PL/SQL:

```
SELECT wyrażenie, wyrażenie,...  
INTO zmienna, zmienna,...  
FROM tabela, tabela,...  
[WHERE ...]  
[GROUP BY ...]  
[HAVING ...]  
[FOR UPDATE OF ...];
```

25

- Wartości, na których operujemy w bloku PL/SQL, pochodzą na ogół z bazy danych, gdzie został określony ich typ danych.
- W związku z tym, wygodnie jest w bloku PL/SQL określać typ danych przez odniesienie do kolumny w bazie danych jako "typ danych wymienionej kolumny"

```
nazwisko prac.nazwisko%TYPE;
```

```
SELECT p.nazwisko
INTO nazwisko
FROM prac p
WHERE p.ID_prac = 103;
```

Zmienne „wierszowe”

26

- W PL/SQL jest możliwość użycia zmiennych "wierszowych", na których można zapamiętać cały wiersz pochodzący z tabeli.
- Dostęp do poszczególnych pól wiersza odbywa się tak, jak w przypadku rekordów: przez kropkę i podanie nazwy kolumny.
- Zmiennej wierszowej nie można użyć bezpośrednio po słowie kluczowym VALUES w instrukcji INSERT INTO, to znaczy trzeba między nawiasami dokładnie wyspecyfikować wstawiane wartości jedna po drugiej.

```
DECLARE
  osoba prac%ROWTYPE; /* Typ wierszowy */
BEGIN

  SELECT * INTO osoba FROM prac p WHERE p.nazwisko = 'Tomacki';

  osoba.placa_pod := 1.1*osoba.placa_pod;

  INSERT INTO Dziennik
  VALUES (osoba.nazwisko, osoba.etat, osoba.placa_pod, SYSDATE);

END;
/
```

KURSORY

27

- Oracle tworzy robocze obszary nazywane "private SQL areas" lub "obszar kontekstu" dla wykonywania instrukcji SQL i przechowywania informacji o procesach.
- Konstrukcja PL/SQL nazywana "**kursorem**" pozwala utworzyć i wykorzystać prywatny obszar SQL.
- Rozróżnia się dwa rodzaje kursorów: jawne i niejawne.
 - PL/SQL **niejawnie** deklaruje kursor dla wszystkich instrukcji manipulowania danymi włączając zapytania zwracające tylko jeden wiersz.
 - Dla zapytań zwracających więcej niż jeden wiersz można **jawnie** zadeklarować kursor przetwarzający indywidualnie wiersze.

Kursor z parametrami

28

- Jawne kursory mogą zawierać parametry.

```
CURSOR nazwa [ (param1 typ1 [, param2 typ2, ...]) ]
[RETURN typ zwracany]
IS zapytanie_sql;
```

- Zakres parametrów kursora jest lokalny co oznacza, że można odwoływać się do parametrów tylko w sekcji deklaracji kursora.
- Wartości parametrów kursora są kojarzone w zapytaniu w momencie otwarcia kursora.

Jawne kursory

29

- Zbiór wierszy zwróconych przez wielowierszowe zapytanie nazywa się "aktywnym zbiorem".
 - Jego rozmiar odpowiada liczbie wierszy spełniających kryterium wyszukiwania.
- Jawny kursor **wskazuje bieżący wiersz** w aktywnym zbiorze - pozwala programowi przetwarzać pojedynczy wiersz.
- Kursor może być kontrolowany przez trzy rozkazy: **OPEN**, **FETCH**, i **CLOSE**.
 - Najpierw należy zainicjalizować kursor rozkazem OPEN, który identyfikuje aktywny zbiór.
 - Potem, można używając rozkazu FETCH pobierać kolejno wiersze ze zbioru.
 - Po przetworzeniu ostatniego wiersza należy zwolnić kursor poleceniem CLOSE.

Instrukcja OPEN

30

- Instrukcja OPEN wykonuje zapytanie skojarzone z jawnie zadeklarowanym kurem.
 - Otwarcie kursora wykonuje zapytanie i identyfikuje aktywny zbiór, który zawiera wiersze spełniające kryterium wyszukiwania.
 - Dla kursorów zadeklarowanych z użyciem klauzuli FOR UPDATE, instrukcja OPEN blokuje te wiersze.
- ```
OPEN c1;
```
- Przekazywanie parametrów:
  - Przykład:
 

```
CURSOR c1 (m_nazwa CHAR, m_numer NUMBER) IS SELECT ...
```
  - instrukcje otwierające ten kursor:
 

```
OPEN c1('ABCDE', 300);
OPEN c1(nazwisko, id_szefa);
OPEN c1('Kowalski', id_zesp);
```
  - Parametrom formalnym zadeklarowanym z wartościami domyślnymi nie muszą odpowiadać parametry aktualne.
  - Każdy parametr aktualny musi mieć typ danych kompatybilny z typem odpowiadającego parametru formalnego.

## Instrukcja FETCH

31

- Instrukcja FETCH pobiera kolejno wiersze z aktywnego zbioru.
- Za każdym razem wykonywania instrukcji FETCH kursor jest przesuwany do następnego wiersza aktywnego zbioru.
- Typowy sposób użycia instrukcji FETCH

```
OPEN c1;
LOOP
 FETCH c1 INTO my_record;
 EXIT WHEN c1%NOTFOUND;
 -- przetwarzanie danych
END LOOP;
```

## Przykład

32

```
DECLARE
...
cursor pracownik_kursor is
select nazwisko, placa_pod, id_zesp from prac;
...
osoba_nazwisko prac.nazwisko%TYPE;
osoba_placa prac.placa_pod%TYPE;
osoba_id_zesp prac.id_zesp%TYPE;
pracownik_dane pracownik_kursor%ROWTYPE;
...
BEGIN
...
fetch pracownik_kursor into osoba_nazwisko, osoba_placa,
osoba_id_zesp;
...
fetch pracownik_kursor into pracownik_dane;
...
END;
/
```



## Instrukcja CLOSE

33

- Instrukcja CLOSE zwalnia kursor a aktywny zbiór przestaje być zdefiniowany.

```
CLOSE c1;
```

- Zaraz po zamknięciu kursora można otworzyć go ponownie.
- Jakakolwiek operacja na zamkniętym kursorze zgłasza predefiniowany wyjątek INVALID\_CURSOR.

## Przykład

34

- Deklaracja kursora:

```
DECLARE
cursor osoba_kursor is
select nazwisko, nazwa
from prac, zesp
where prac.id_zesp = zesp.id_zesp and zesp.nazwa = `IT`;
```

- Otwarcie kursora:

```
open osoba_kursor;
```

- Pobranie krotki wyznaczonej przez kursor:

```
fetch ... into...
```

- Zamknięcie kursora:

```
close osoba_kursor;
```

35

```

DECLARE
 -- Deklarujemy kursor z dwoma parametrami
 CURSOR osoba (n prac.nazwisko%TYPE, e prac.etat%TYPE) IS
 SELECT nazwisko, etat FROM prac p
 WHERE UPPER(p.nazwisko) = UPPER(n) AND UPPER(p.etat) = UPPER(e)
 ORDER BY 1 DESC;
 zm_osoba osoba%ROWTYPE;
BEGIN
 -- Otwarcie kursora. Wielkość liter w łańcuchu bez znaczenia.
 OPEN osoba('DaBAcki','ADIUNKT');
 LOOP
 FETCH osoba INTO zm_osoba;
 EXIT WHEN osoba%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE(osoba%ROWCOUNT || ' ' ||
 zm_osoba.nazwisko || ' ' || zm_osoba.etat);
 END LOOP;
 CLOSE osoba;
END;
/

```

## Atrybuty kursora

36

- Każdy kursor jawnie zdefiniowany ma cztery atrybuty:
  - ▣ %NOTFOUND
  - ▣ %FOUND
  - ▣ %ROWCOUNT
  - ▣ %ISOPEN
- Po dołączeniu do nazwy kursora nazwy atrybutu można uzyskać informację o wykonaniu wielowierszowego zapytania.
- Niejawny kursor SQL ma te same atrybuty co jawny kursor.
  - ▣ Pozwalają one uzyskać informację o wykonaniu instrukcji INSERT, UPDATE, DELETE i SELECT INTO.
- Atrybutów cursorów można używać w instrukcjach proceduralnych ale nie w instrukcjach SQL.

## Niejawne kursory

37

- Oracle niejawnie otwiera kursor do przetwarzania każdej instrukcji SQL nie połączonej z jawnie zadeklarowanym kursorem.
- PL/SQL pozwala odwoływać się do ostatniego niejawnego kursora jak do kursora SQL.
  - Tym samym można używać atrybutów kursora w celu pozyskania informacji o ostatnio wykonanej instrukcji SQL.
- Przed otwarciem kursora SQL atrybuty kursora niejawnego są ustawiane na wartość NULL.

## Atrybut %NOTFOUND - Jawne kursory

38

- Przed pierwszym pobraniem z aktywnego zbioru %NOTFOUND ma wartość NULL.
- Jeżeli ostatnie pobranie wiersza nie powiodło się, to %NOTFOUND ma wartość TRUE.

```

LOOP
 FETCH c1 INTO m_nazwisko, m_id_zesp;
 EXIT WHEN c1%NOTFOUND;
 ...
END LOOP;
```

- Jeżeli kursor nie jest otwarty odwołanie do atrybutu %NOTFOUND zgłosi predefiniowany wyjątek INVALID\_CURSOR.

## Atrybut %NOTFOUND - Niejawne kursory

39

- %NOTFOUND ma wartość TRUE jeżeli INSERT, UPDATE, lub DELETE nie zadziała na żadnym wierszu lub SELECT INTO nie zwróci żadnego wiersza.
- Poza tymi przypadkami %NOTFOUND ma wartość FALSE.

```
UPDATE prac
SET placa_pod = placa_pod * 1.05
WHERE id_prac = m_numer;

IF SQL%NOTFOUND THEN -- aktualizacji nie powiodła się
 INSERT INTO temp VALUES (...);
END IF;
```

40

- Jeżeli SELECT INTO nie zwróci wiersza, to predefiniowany wyjątek NO\_DATA\_FOUND jest zgłaszany bez względu na to czy jest sprawdzany %NOTFOUND w następnej linii, czy nie.
- W takiej sytuacji %NOTFOUND jest użyteczny w programie obsługi wyjątków OTHERS. Zamiast kodować program obsługi NO\_DATA\_FOUND, można sprawdzić zgłoszenie wyjątku %NOTFOUND.

```
EXCEPTION
 WHEN OTHERS THEN
 IF SQL%NOTFOUND THEN -- sprawdzenie 'no data found'
 ...
 END IF;
 ...
END;
```

## Atrybut %FOUND - Jawne kursory

41

- %FOUND jest logicznym przeciwieństwem %NOTFOUND.
- Po otwarciu jawnego kursora a przed pierwszym pobraniem %FOUND ma wartość NULL.

```

LOOP
 FETCH c1 INTO m_nazwisko, m_id_zesp;
 IF c1%FOUND THEN -- pobranie udane
 INSERT INTO temp VALUES (...);
 ELSE -- pobranie nieudane
 EXIT;
 END IF;
 ...
END LOOP;

```

## Atrybut %FOUND - Niejawne kursory

42

- %FOUND jest logicznym przeciwieństwem %NOTFOUND.

```

DELETE FROM temp_prac
WHERE id_prac = m_id_prac;

IF SQL%FOUND THEN -- usunięcie udane
 INSERT INTO prac_kopia VALUES (m_numer, m_nazwisko, ...);
END IF;

```

## Atrybut %ROWCOUNT - Jawne kursory

43

- Otwarcie jawnego kursora zeruje atrybut %ROWCOUNT.
- Przed pierwszym pobraniem wiersza %ROWCOUNT zwraca zero.
- Następnie zwraca liczbę wierszy dotąd pobranych.
- Licznik jest inkrementowany jeżeli ostatnie pobranie wiersza zakończyło się pomyślnie.

```

LOOP
 FETCH c1 INTO m_nazwisko, m_zespol ;
 IF c1%ROWCOUNT > 10 THEN
 ...
 END IF;
 ...
END LOOP;

```

## Atrybut %ROWCOUNT - Niejawne kursory

44

- %ROWCOUNT zwraca liczbę wierszy na których zadziałała instrukcja INSERT, UPDATE, DELETE lub zwróconych przez instrukcję SELECT INTO.

```

DELETE FROM prac WHERE ...
IF SQL%ROWCOUNT > 10 THEN
 ...
END IF;

```

- Jeżeli SELECT INTO zwraca więcej niż jeden wiersz, to jest zgłaszany predefiniowany wyjątek TOO\_MANY\_ROWS i %ROWCOUNT jest ustawiany na 1, a nie na aktualną liczbę wierszy.

## Atrybut %ISOPEN -

45

### □ Jawne kursory

- %ISOPEN ma wartość TRUE jeżeli cursor jest otwarty, w przeciwnym razie ma wartość FALSE.

```
IF c1%ISOPEN THEN -- czy cursor otwarty
 ...
ELSE -- cursor zamknięty
 OPEN c1;
END IF;
```

### □ Niejawne kursory

- Oracle automatycznie zamyka cursor SQL po wykonaniu instrukcji SQL. Wynikiem czego %ISOPEN zawsze ma wartość FALSE.

## Kursor pętli FOR

46

- Kursor FOR... LOOP niejawnie deklaruje indeks pętli jako %ROWTYPE rekord, otwiera kursor, kolejno pobiera wiersze aktywnego zbioru podstawiając do rekordu, i zamyka kursor po zakończeniu przetwarzania.

```
DECLARE
 zarobki REAL := 0.0;
 CURSOR c1 IS SELECT id_prac, placa_pod, zatrudnienie, id_zesp
 FROM prac;
 ...
BEGIN
 FOR prac_rec IN c1 LOOP
 ...
 zarobki := zarobki + prac_rec.placa_pod;
 END LOOP;
 ...
END;
```

- Dopuszcza się używanie aliasów dla komponentów instrukcji SELECT:

```
CURSOR c1 IS
 SELECT id_prac, placa_pod1+NVL(placa_dod,0) zarobki,
 etat FROM ...
```

## Przekazywanie parametrów

47

```
DECLARE
 CURSOR c1 (n_z NUMBER)
 IS SELECT placa_pod, placa_dod
 FROM prac WHERE id_zesp = n_z;
 ...
BEGIN
 FOR prac_rec IN c1(20) LOOP
 ...
 END LOOP;
 ...
END;
```

48

## Wyzwalacze



## Wyzwalacze BD (ang. triggers)

49

- Wyzwalacz bazy danych jest procedurą składowaną w bazie powiązaną z jedną konkretną tablicą.
- Z pojedynczą tablicą może być związane wiele wyzwalaczy, natomiast pojedynczy wyzwalacz może być związany tylko z jedną tablicą.
- Wyzwalacze w zależności od czasu, kiedy są uruchamiane dzielą się na następujące grupy:
  - ▣ before – uruchamiane przed poleceniem SQL,
  - ▣ after – uruchamiane po poleceniu SQL,
  - ▣ instead of (operujące na perspektywach)
- Mogą być wykonywane w wyniku poleceń: insert, update, delete.

## Definicja

50

```

CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]

ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]

WHEN (condition)

BEGIN
 --- sql statements

END;
```

## Przykład

51

```
create or replace trigger test
 after delete on prac
declare
/*deklaracje zmiennych stałych, kursorów, wyjątków lokalnych
wyzwalacza*/
begin
/*ciało wyzwalacza*/
dbms_output.put_line ('Wiersz opisujący pracownika
usuniętego');
end;
/
```

## Ograniczenia

52

- Ciało wyzwalacza może zawierać polecenia PL/SQL i SQL oraz wywołania podprogramów, z następującymi ograniczeniami:
  - nie wolno stosować poleceń DDL,
  - nie wolno stosować poleceń sterowania transakcjami SQL, takich jak **commit**, **rollback**, **savepoint** ani też wywoływać podprogramów zawierających te polecenia.
- Dla wyzwalaczy uruchamianych przez polecenia DML (**insert**, **update**, **delete**) można określić listę kolumn, których uaktualnienie uruchomi wyzwalacz.
- Przykład

```
create or replace trigger test
 after insert or delete or update of etat, placa_pod on prac
begin
...
end;
/
```

## for each row

53

- Można określić, czy wyzwalacz ma być uruchamiany dla każdego wiersza tabeli, czy też niezależnie od ilości wierszy – tylko jeden raz.
- Do tego celu służy klauzula **for each row**, która jeśli występuje w definicji wyzwalacza, to będzie on uruchamiany dla każdego wiersza spełniającego warunek polecenia.
- W definicji wyzwalacza **for each row** może być umieszczona opcjonalna klauzula **when**, określająca dodatkowe warunki jego uruchomienia.
- Wyrażenie wartościujące klauzuli musi być poleceniem SQL (bez podzapytań) i może ono zawierać dwa szczególne kwalifikatory (pseudorekordy) **new** i **old**.
  - ▣ Odwołania do tych kwalifikatorów tylko w klauzuli **when** nie poprzedza się dwukropkiem.

## Przykład

54

- Przy modyfikacji wierszy w tabeli prac, jeśli placa\_dod jest mniejsza od 100 pracownik dostaje podwyżkę.

```
create or replace trigger podwyzka
 before update on prac
for each row
when (old.placa_dod<100)
begin
...
end;
/
```

## Uwagi

55

- W klauzuli **when** można stosować klasyczne operatory matematyczne (**=, <=, >=, !=**), logiczne (**and, or, not**) oraz operatory SQL (**is null, like, between ... and, in**)
- np.: **when(old.placa\_dod between 100 and 300) ...**
- Dla wyzwalacza uruchamianego poleceniem **insert**, kwalifikator **old** przyjmuje wartość **null**, natomiast uruchamianego poleceniem **delete** wartość **null** przyjmuje kwalifikator **new**.
- Wyzwalacz uruchamiany w **update** ma dostęp do nowych i starych wartości niezależnie od sposobu uruchomienia.

## Przykład

56

- Wyzwalacz uruchamiany jest przed użyciem każdego wiersza z tabeli zesp.
  - ▣ Pozwoli on na modyfikację jedynie tych zespołów, w których nikt nie pracuje.

```
create or replace trigger czy_zesp_ma_prac
before update on zesp
for each row
declare
v_prac number(2) := 0;
begin
...
select count(*) into v_prac from prac where id_zesp=:old.id_zesp;

if v_prac>0 then
raise_application_error(-20000, 'W tym zespole są zatrudnieni pracownicy');
end if;
end;
/
```

### Uwaga!

Wszystkie odwołania do kwalifikatorów **new** i **old** oprócz klauzuli **when** muszą być poprzedzone dwukropkiem.

## 57 Predykaty warunkowe

- Ciało wyzwalacza, które jest uruchamiane przez więcej niż jedno polecenie DML może zawierać tzw. predykaty warunkowe **inserting**, **deleting** i **updating**, które umożliwiają wykonanie odpowiedniego fragmentu podprogramu, np. dla wyzwalacza zdefiniowanego poniżej:

```
create or replace trigger test
 before insert or update or delete on prac
begin
 if inserting then ...
end if;
 if updating then ...
end if;
 if deleting then ...
end if;
end;
```

- Przykład: polecenia predykatu **updating** zostaną wykonane tylko w przypadku uaktualnienia atrybutu `placa_pod`:

```
update prac
set placa_pod=placa_pod*1.2
where id_zesp=10;
```

## 58 Przykład 1

- Automatyczna zmiana daty zatrudnienia przy zmianie zespołu

```
CREATE TRIGGER zmiana_zespolu
 BEFORE UPDATE OF id_zesp ON prac
 FOR EACH ROW
BEGIN
 :NEW.zatrudniony := SYSDATE;
END;
```

## Uwagi

59

- Do zablokowania (odblokowania) wyzwalaczy służy polecenie

```
alter trigger nazwa_wyzwalacza disable [enable];
```

- Wyzwalacze związane z daną tabelą można zablokować (odblokować) poleceniem

```
alter table nazwa_tabeli disable [enable] all triggers;
```

- Opis wyzwalaczy zdefiniowanych przez użytkownika można odczytać przy pomocy zapytania do perspektywy systemowej *USER\_TRIGGERS*.
- Wyzwalacze usuwamy poleceniem:

```
drop trigger nazwa_wyzwalacza;
```

- Uwaga! W zależności od wersji serwera bazy danych Oracle z daną tabelą może być związana różna ilość wyzwalaczy tego samego typu.
  - Istnieje pewne ograniczenie stosowania wyzwalaczy zdefiniowanych z wykorzystaniem klauzuli **for each row**.
  - W takim wyzwalaczu nie można odwoływać się do tej samej tabeli, w której wyzwalacz został zdefiniowany.

## Przykład 2.

60

- Wyzwalacz nadający nowemu pracownikowi kolejny numer, który jest pobierany z licznika *seq\_prac*.

```
create or replace trigger generuj_numer
before insert on prac
for each row
begin
 if :new.numer is null
 then select seq_prac.nextval into :new.numer from dual;
 end if;
end;
/
```

### Przykład 3

61

- Tabela *historia\_pracownik*, przechowuje dane historyczne dotyczące pracowników, której schemat stanowi rozszerzenie schematu tabeli *prac* o trzy następujące atrybuty:„
  - ▣ *uzytkownik* - przechowuje nazwę użytkownika, który dokonał zmian w zawartości tabeli *prac*,
  - ▣ *data\_operacji* - datę dokonania tych zmian
  - ▣ *rodzaj\_operacji* - tj. 'UPDATE' lub 'DELETE'.
- Przedstawiony dalej wyzwalacz, na skutek uaktualnienia wartości kolumn *etat*, *szef*, *placa\_pod*, *placa\_dod*, *id\_zesp* lub usunięcia wierszy z tabeli *prac* wpisze dane z przed modyfikacji lub usunięcia do tabeli *historia\_pracownik*.
  - ▣ Wyzwalacz powinien również wpisać do tej tabeli nazwę użytkownika, wykonującego operację, datę wykonania i rodzaj operacji.

### Tworzenie tabeli

62

```
create table historia_pracownik (
 numer number(4),
 nazwisko varchar2(12),
 etat varchar2(10),
 szef number(4),
 zatrudniony date,
 placa_pod number(6,2),
 placa_dod number(6,2),
 id_zesp number(2),
 uzytkownik varchar2(12),
 data_operacji date,
 rodzaj_operacji varchar2(6));
```

## Tworzenie wyzwalacza

63

```

create or replace trigger historia
 after update of etat, szef, placa_pod, placa_dod, id_zesp
 or delete on prac
for each row
begin
 if updating then
 insert into historia_pracownik values(
 :old.numer, :old.nazwisko, :old.etat, :old.szef,
 :old.zatrudniony, :old.placa_pod, :old.placa_dod,
 :old.id_zesp, user, sysdate, 'UPDATE');
 elsif deleting then
 insert into historia_pracownik values(
 :old.numer, :old.nazwisko, :old.etat, :old.szef,
 :old.zatrudniony, :old.placa_pod, :old.placa_dod,
 :old.id_zesp, user, sysdate, 'DELETE');
 end if;
end;
/

```

## Przykład 4

64

- Wyzwalacz definiujący ograniczenie integralnościowe zapewniające, że numer zespołu dla wprowadzanego lub uaktualnianego wiersza dotyczącego pracownika jest numerem jednego z już istniejących zespołów.

```

create or replace trigger czy_zesp_istnieje
before insert or update of id_zesp on prac
for each row
declare
 jest:=0;
begin
 select count(*) into jest from zesp
 where id_zesp=new.id_zesp;

 if jest=0 then
 raise_application_error(-20000,'Zespól o podanym numerze nie istnieje');
 end if;
end;
/

```



## Wyzwalacze **instead of**

66

- Wyzwalacze **instead of** pozwalają użytkownikowi na modyfikowanie perspektyw, których zawartości nie można zmieniać używając poleceń DML (np. w przypadku perspektyw korzystających ze złączeń).
- Cechy odróżniające wyzwalacze **instead of** od wyzwalaczy **before** i **after** to:
  - ▣ przy uruchamianiu wyzwalacza **instead of** nie jest wykonywana instrukcja aktywująca wyzwalacz,
  - ▣ w przypadku wyzwalacza **instead of** ograniczenia integralnościowe typu **check** nie są sprawdzane,
  - ▣ nie można definiować wyzwalaczy **instead of** na tabelach,
  - ▣ wszystkie wyzwalacze **instead of** są typu wierszowego.

## Wyzwalacze systemowe

67

- Wyzwalacze systemowe mogą być uruchamiane jako reakcja na:
  - ▣ zdarzenie wywołane stanem systemu (uruchomienie lub zamknięcie instancji, błąd serwera);
  - ▣ zdarzenie wywołane przez użytkownika (zalogowanie lub wylogowanie się, wydanie polecenia DDL, albo DML).
- Wyzwalacze wywoływane zdarzeniami systemowymi mogą być definiowane na poziomie bazy danych (dla wszystkich użytkowników) lub na poziomie schematu (dla jednego użytkownika).

## Przykłady zdarzeń systemowych

68

Poniżej przedstawiono listę niektórych zdarzeń systemowych i opis sytuacji, w jakich aktywowany jest wyzwalacz reagujący na określone zdarzenie:

|               |                                            |
|---------------|--------------------------------------------|
| STARTUP       | po otwarciu bazy danych                    |
| SHUTDOWN      | tuż przed zatrzymaniem instancji           |
| SERVERERROR   | po wystąpieniu błędu                       |
| AFTER LOGON   | po zalogowaniu się użytkownika             |
| BEFORE LOGOFF | tuż przed wylogowaniem użytkownika         |
| BEFORE CREATE | przed utworzeniem obiektu bazy danych      |
| AFTER CREATE  | po utworzeniu obiektu bazy danych          |
| BEFORE ALTER  | przed zmianą definicji obiektu bazy danych |
| AFTER ALTER   | po zmianie definicji obiektu bazy danych   |

## Przykłady zdarzeń cd.

69

|                |                                            |
|----------------|--------------------------------------------|
| BEFORE DROP    | przed usunięciem obiektu bazy danych       |
| AFTER DROP     | po usunięciu obiektu bazy danych           |
| BEFORE AUDIT   | przed wykonaniem polecenia AUDIT           |
| AFTER AUDIT    | po wykonaniu polecenia AUDIT               |
| BEFORE NOAUDIT | przed wykonaniem polecenia NOAUDIT         |
| AFTER NOAUDIT  | po wykonaniu polecenia NOAUDIT             |
| BEFORE DDL     | przed wykonaniem większości instrukcji DDL |
| AFTER DDL      | po wykonaniu większości instrukcji DDL     |
| BEFORE GRANT   | przed wykonaniem polecenia GRANT           |
| AFTER GRANT    | po wykonaniu polecenia GRANT               |
| BEFORE RENAME  | przed wykonaniem polecenia RENAME          |
| AFTER RENAME   | po wykonaniu polecenia RENAME              |
| BEFORE REVOKE  | przed wykonaniem polecenia REVOKE          |
| AFTER REVOKE   | po wykonaniu polecenia REVOKE              |

## Przykład

70

```
SQL>create or replace trigger rejestr_log_wyz
after logon
on schema
begin
insert into rejestr_log
values('zalogowano sie' ||sysdate);
end;
/
```

*Trigger created.*

```
SQL> connect s12354/s12345
```

```
Connected
```

```
SQL> select * from rejestr_log;
```

## Atrybuty

71

- Każde zdarzenie systemowe posiada atrybuty, które można odczytać po uruchomieniu wyzwalacza. Poniżej przedstawiono listę niektórych atrybutów.

|                                           |                                                                                     |
|-------------------------------------------|-------------------------------------------------------------------------------------|
| ora_client_ip_address                     | adres IP komputera klienta (jeśli jest używany protokół TCP/IP)                     |
| ora_database_name                         | nazwa bazy danych                                                                   |
| ora_dict_obj_name                         | nazwa obiektu bazy danych, użyta w instrukcji DLL                                   |
| ora_dict_obj_owner                        | nazwa właściciela obiektu bazy danych, którego nazwa została użyta w instrukcji DLL |
| ora_dict_obj_type                         | rodzaj obiektu użytego w instrukcji DLL                                             |
| ora_is_alter_column (kolumna IN VARCHAR2) | TRUE, jeśli definicja podanej kolumny została zmieniona                             |
| ora_is_drop_column (kolumna IN VARCHAR2)  | TRUE, jeśli podana kolumna została usunięta                                         |
| ora_is_servererror (numer_błędu NUMBER)   | TRUE, jeśli wystąpił błąd o podanym numerze                                         |
| ora_login_user                            | nazwa użytkownika                                                                   |
| ora_server_error(n NUMBER)                | numer błędu n-ty na stosie błędów                                                   |
| ora_sysevent                              | nazwa zdarzenia systemowego, którego wystąpienie spowodowało odpalenie wyzwalacza   |

## Przykład

72

```
SQL>CREATE OR REPLACE TRIGGER obj_wyz
BEFORE CREATE ON DATABASE
DECLARE
 uzytk rejestr_log.uzytkownik%TYPE;
 data_utw rejestr_log.data_utw%TYPE;
 obiekt rejestr_log.obiekt%TYPE;
 nazwa rejestr_log.nazwa%TYPE;
BEGIN
 uzytk:= dictionary_obj_owner;
 data_utw:= SYSDATE;
 obiekt:= dictionary_obj_type;
 nazwa:= dictionary_obj_name;
INSERT INTO rejestr_log
VALUES (uzytk, data_utw, obiekt, nazwa);
END;
/

SQL> CREATE TABLE tabela (kolumna NUMBER);

SQL> SELECT * FROM rejestr_log;
```

74

## Podprogramy

## Podprogramy

75

- Podprogramy to:
  - ▣ procedury (wykonują określone akcje)
  - ▣ funkcje (wykonują obliczenia i zwracają wartości)
  - ▣ pakiety (zbierają w całość logicznie powiązane procedury, funkcje, zmienne i kursory)
- Są przechowywane w bazie danych w postaci skompilowanej i źródłowej (źródło dostępne poprzez USER\_SOURCE).
  - ▣ Mogą być współdzielone przez wielu użytkowników.
- Każdy podprogram składa się ze specyfikacji zawierającej słowo **PROCEDURE** lub **FUNCTION**, nazwy i listy parametrów w nawiasach oraz ciała, które jest normalnym blokiem PL/SQL z wyjątkiem części deklaracyjnej, która nie zawiera słowa DECLARE.

## Komunikacja

76

- W środowisku SQL\*Plus do współdziałania z PL/SQL używa się następujących komend:
  - ▣ ACCEPT - czyta wejście od użytkownika i zapamiętuje je na zmiennych
  - ▣ VARIABLE - deklaruje zmienną związaną lub zmienną operacyjną
  - ▣ PRINT - wyświetla obecną wartość zmiennych związanych
  - ▣ EXECUTE - wykonuje pojedynczą instrukcję PL/SQL

## Definiowanie procedury

77

- Nazwa procedury musi być unikalna w ramach schematu (lub pakietu).
- Między słowami kluczowymi IS i BEGIN umieszcza się deklaracje wszystkich zmiennych i kursorów lokalnych.
- Między słowami kluczowymi BEGIN i END umieszcza się kod PL/SQL, który wykonuje dana procedura
- Składnia procedury:

```
CREATE [OR REPLACE] PROCEDURE nazwa_procedury [(arg1[,
arg2, ...)]] IS
[deklaracje]
BEGIN
 instrukcje
[EXCEPTION
 obsługa wyjątków]
END [nazwa_procedury];
```

## Składnia funkcji

78

- Składnia funkcji jest podobna do procedury , ale jest rozszerzona o dodatkowe elementy:
  - Musi zawierać klauzulę RETURN, która określa typ danych zwracany przez funkcję.
  - W części wykonawczej, każda ścieżka musi prowadzić do wyrażenia RETURN

```
FUNCTION nazwa_funkcji [(arg1 [arg2, ...)]]
RETURN nazwa_typu IS
[deklaracje]
BEGIN
 instrukcje
[EXCEPTION
 obsługa wyjątków]
END [nazwa_funkcji];

parametr ::= nazwa [IN | OUT | IN OUT] typ [{:= | DEFAULT}
wyrażenie]
```

## Parametry podprogramów

79

- Przy definiowaniu podprogramów każdy zadeklarowany parametr musi być przekazany w jednym z następujących trybów:
  - ▣ IN (domyślnie) – przekazuje wartość do podprogramu;
    - parametr formalny - zachowuje się jak stała, nie można go zmieniać;
    - parametr aktualny (wywołania) - może być wyrażeniem, stałą literałem lub zmienną zainicjalizowaną
  - ▣ OUT – zwraca wartość do wywołującej jednostki programu.
    - Wewnątrz programu parametr zachowuje się jak zmienna, która nie została zainicjalizowana
  - ▣ IN OUT - kombinacja powyższych trybów umożliwiającą przekazywanie wartości do podprogramu oraz zwracanie wartości na zewnątrz

## Tworzenie podprogramów

80

- Procedury i funkcje tworzy się w sekcji deklaracyjnej dowolnego bloku PL/SQL, po wszystkich deklaracjach.
  - ▣ Możliwe jest użycie deklaracji zapowiadających funkcje i procedury zdefiniowane w dalszej części programu.

```
CREATE OR REPLACE PROCEDURE sprawdz_asystentow
 (p_id_zesp IN NUMBER, p_ilu_asystentow OUT NUMBER) IS
 CURSOR c_asystenci IS
 SELECT * FROM prac WHERE id_zesp=p_id_zesp AND etat ='ASYSTENT';
 v_starszy_asystent prac%ROWTYPE;
BEGIN
 SELECT COUNT(*) INTO p_ilu_asystentow
 FROM prac WHERE id_zesp=p_id_zesp AND etat ='ASYSTENT';
 FOR cur_rec IN c_asystenci LOOP
 IF (ROUND (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 procedury

81

```

DECLARE
 p_in NUMBER :=&numer_zespolu;
 p_out NUMBER :=0;
BEGIN
 sprawdz_asystentow(p_in, p_out);
 DBMS_OUTPUT.PUT_LINE('Liczba asystentow:' || p_out);
END;
/

```

## Wywołanie procedur i funkcji (PL/SQL)

83

- Wszędzie, gdzie można umieścić funkcję wbudowaną SQL, można umieścić funkcję PL/SQL zdefiniowaną przez użytkownika.
- 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 niezależnej

84

```
CREATE or REPLACE FUNCTION polowa(v_ile IN NUMBER)
RETURN NUMBER IS
BEGIN
 RETURN (v_ile * 0.5);
END polowa;
/
```

### □ Wywołanie funkcji z SQL\*Plus:

```
SQL> VARIABLE x NUMBER;
SQL> EXECUTE :x := polowa(100);
SQL> PRINT x;
```

### □ lub

```
SQL> SELECT nazwisko, placa_pod, polowa(placa_pod)
 FROM prac_kopia
 WHERE id_prac >150;
```

## Przykład:

85

### □ Utworzenie funkcji **podatek**:

```
CREATE OR REPLACE FUNCTION podatek (kwota_opod IN NUMBER)
RETURN NUMBER
IS
BEGIN
 RETURN (kwota_opod * 0.2);
END podatek;
/
```

### □ Użycie funkcji **podatek** w instrukcji SQL:

```
SQL> SELECT placa_pod, podatek(placa_pod)
 FROM prac
 WHERE id_prac >150;
```

## Podprogramy składowane w bazie danych

86

- Procedury i funkcje przestają istnieć po zakończeniu macierzystego programu.
- Aby umożliwić innym aplikacjom korzystanie z nich należy umieścić je w bazie danych.
  - Umożliwia to instrukcja **CREATE [ OR REPLACE]**, która tworzy funkcję lub procedurę składowaną.
  - Uwaga: Każdy podprogram pisany interaktywnie można umieścić w pliku tekstowym i zapamiętać w bazie danych poleceniem **save prog.sql**, gdzie prog jest nazwą pliku z tekstem podprogramu.
    - Z poziomu SQL\*Plus uruchamia się go poleceniem **start prog** lub **@prog**,
- Wywołanie z poziomu SQL\*Plus podprogramów składowanych w bazie danych umożliwia polecenie **EXECUTE**, a uzyskiwanie informacji o ich parametrach polecenie **DESCRIBE**.
- Do wykrywania błędów pomocne jest polecenie **SHOW ERRORS**, a do oglądania ich treści - perspektywa słownika danych o nazwie **USER\_SOURCE**.

## Przykład

87

```
SQL> CREATE OR REPLACE
PROCEDURE Podnies_place(suma NUMBER)
IS
BEGIN
 UDATE prac SET placa_pod = placa_pod + suma;
END;
/
SQL>SHOW ERRORS
...
SQL> LIST 3
SQL> CHANGE /UBDATE/UPDATE
SQL> /
.....
SQL> DESCRIBE podnies_place;
SQL> EXECUTE podnies_place(50);
...
SQL> SELECT line, text FROM user_source
WHERE name = 'PODNIES_PLACE';

SQL> DROP PROCEDURE podnies_place;
```

## Pakiet

Przykłady: <http://psoug.org/reference/packages.html>

88

- Pakiet – zbiór logicznie powiązanych zmiennych, stałych, kursorów, wyjątków, procedur, funkcji itp., tworzących jeden nazwany, przechowywany trwale w bazie danych

- Składowe pakietu:

- specyfikacja ( interfejs ) - dostępna dla aplikacji

```
CREATE [OR REPLACE] PACKAGE <Nazwa>
{ IS | AS }
< publiczne-deklaracje >;
< specyfikacja_podprogramów>;
END [<Nazwa>];
```

- ciało ( implementacja ) - ukryte, opcjonalne

```
CREATE [OR REPLACE] PACKAGE BODY <Nazwa>
{ IS | AS }
<prywatne_deklaracje>;
< definicje_podprogramów >;
[BEGIN
 < instrukcje_inicjalizujące>;]
END [< Nazwa >];
```

## Przykład

89

- Deklaracja wszystkich zmiennych w specyfikacji

```
create or replace package pakiet
is
 zmienna number;
 function funkcja return number;
 function funkcja_druga(imie varchar2) return number;
 procedure procedura;
end;
```

- Implementacja pakietu

```
create or replace package body pakiet
is
 function funkcja
 return number
 is
 liczba number;
 begin
 liczba:=55;
 return liczba;
 end;
```

## Implementacja cd.

90

```
function funkcja_druga
(imie varchar2)
return number
is
liczba number;
begin
liczba:=100;
return liczba;
end;

procedure procedura
is
begin
dbms_output.put_line('jestem w procedurze procedura');
end;

begin
select funkcja_druga('Andrzej') into zmienna from dual;
end;
```

## Przykładowe wywołanie

91

```
select pakiet.funkcja() from dual;
select pakiet.funkcja_druga('Adam') from dual;
execute pakiet.procedura;

begin
DBMS_OUTPUT.put_line('pakiet.zmienna: '||pakiet.zmienna);
end;
```

## Pakiety wbudowane

92

- DBMS\_DATA\_MINIG – umożliwia eksplorację danych
- DBMS\_DATAPUMP – ułatwia przenoszenie danych ( metadanych) między bazami danych
- DBMS\_DIMISION – używany głównie do obsługi hurtowni danych
- DBMS\_JAVA – umożliwia dostęp do mechanizmów języka Java w języku PL/SQL
- DBMS\_OUTPUT – narzędzie do diagnozowania
- DBMS\_RANDOM – zawiera generator liczb losowych bazujący na języku C
- DBMS\_XMLGEN – umożliwia transformację wyniku SELECT na format XML

94

## Wyjątki

## Obsługa wyjątków w PL/SQL

95

- Błąd lub ostrzeżenie nazywamy w PL/SQL wyjątkiem (ang. *exception*).
- Wyjątki mogą być systemowe, predefiniowane - (dzielenie przez zero, brak wolnej pamięci, brak praw do obiektu) lub definiowane przez użytkownika (za niski budżet, za wysoka płaca, zbyt mała ilość towaru w magazynie).
- Wystąpienie błędu jest sygnalizowane przez wywołanie wyjątku.
  - Błędy systemowe sygnalizowane są automatycznie, błędy definiowane przez użytkownika są wywoływane ręcznie za pomocą polecenia RAISE.
- Każdy wyjątek predefiniowany ma przypisany:
  - typ błędu: **PLS** - błąd kompilacji lub **ORA** – błąd wykonania,
  - kod błędu: liczba ujemna wskazująca numer,
  - tekst błędu: opis złożony z maks. 512 bajtów.
- Wyjątków predefiniowanych nie trzeba ani deklarować ani zgłaszać
  - można jednak je zgłaszać poprzez RAISE.

## Funkcje SQLCODE i SQLERRM

96

- Funkcja **SQLCODE** zwraca numer błędu, który wystąpił.
  - Numer jest zawsze ujemny, za wyjątkiem błędu NO\_DATA\_FOUND (+100) i błędów definiowanych przez użytkownika (+1)
- Funkcja **SQLERRM** zwraca treść błędu, który wystąpił.
- Jeśli nie wystąpił żaden błąd to SQLCODE zwraca 0, a SQLERRM zwraca : „ORA-0000: normal, successful completion”
- Po wystąpieniu wyjątku sterowanie przechodzi do procedury obsługi wyjątku (ang. *exception handler*).
  - Po jej wykonaniu sterowanie przechodzi do kolejnego bloku nadrzędnego.
  - Jeśli procedura obsługi danego błędu nie zostanie znaleziona, to wykonywanie programu zostanie przerwane.

## Kontrola obsługi wyjątku (z zapisem do tablicy *errors*):

97

```

DECLARE
v_a NUMBER := 0; v_b NUMBER := 0; v_c NUMBER;
BEGIN
 v_a := 10;
 BEGIN
 v_c := v_a / v_b;
 EXCEPTION
 WHEN NO_DATA_FOUND THEN
 INSERT INTO errors VALUES ('No data found', SYSDATE);
 END;
 SELECT COUNT(*) INTO v_a FROM pracownicy;
 ...
EXCEPTION
 WHEN ZERO_DIVIDE THEN
 INSERT INTO errors VALUES ('Division by 0', SYSDATE);
 WHEN OTHERS THEN
 INSERT INTO errors VALUES ('Other error', SYSDATE);
END;

```

## Definiowanie własnych wyjątków

98

- Zmienne typu **EXCEPTION** zadeklarowane w sekcji **DECLARE** bloków **PL/SQL**.
- Przed użyciem wyjątku musi on być zadeklarowany.
- Wyjątek jest widoczny w danym bloku i wszystkich jego blokach podrzędnych.
- Wyjątek nie jest daną, do wyjątku nie można przypisać żadnej wartości ani użyć wyjątku w jakiegokolwiek operacji arytmetycznej.
- **a) związane tylko z daną aplikacją PL/SQL i zgłaszane jawnie przy użyciu instrukcji RAISE i nazwy wyjątku.**
  - Funkcja **SQLCODE** zwraca dla nich zawsze wartość 1.

```

DECLARE
 v_liczba NUMBER := 0;
 moj_wyjatek EXCEPTION;
BEGIN
 ...
 RAISE moj_wyjatek;
 ...
END;

```

## Własne wyjątki cd.

99

- b) Wyjątki powiązane z określonym kodem błędu Oracle poprzez PRAGMA EXCEPTION\_INIT - zgłaszane niejawnie, obsługane poprzez mechanizm obsługi błędów użytkownika

```

DECLARE
 nazwa_w EXCEPTION;
 PRAGMA EXCEPTION_INIT(nazwa_w, kod_błędu);
BEGIN
 ...
 EXCEPTION
 WHEN nazwa_w THEN instrukcje1
 [WHEN OTHERS instrukcje2]
END;
```

## Uwagi

100

- Skojarzony wyjątek użytkownika zgłaszany jest automatycznie chociaż można go zgłosić ręcznie.
- PRAGMA musi być zadeklarowana w bloku deklaracji wyjątku użytkownika (najlepiej zadeklarować jedno pod drugim).
  - ▣ Można skojarzyć więcej niż jeden wyjątek użytkownika z tym samym numerem błędu
- Klauzula WHEN OTHERS przechwytuje wszystkie możliwe wyjątki pojawiające się w programie PL/SQL.
  - ▣ Powinna być umieszczana **po wszystkich innych** klauzulach obsługi wyjątków - **może wystąpić tylko jeden raz**.
- Kolejność obsługi wyjątku:
  - ▣ W momencie zaistnienia wyjątku wykonanie aktualnego bloku kończy się.
    - Wywołuje się funkcję obsługi tego wyjątku.
  - ▣ W przypadku zgłoszenia wyjątku, szukana jest klauzula w najbliższym bloku, potem w bloku wyżej, aż do klauzuli w bloku zewnętrznym (obejmującym).
  - ▣ Sterowanie jest zwracane do następnej instrukcji w bloku nadrzędnym (zawierającym blok, w którym wystąpił wyjątek).
    - Jeśli taki blok nie istnieje to sterowanie jest zwracane do systemu.



## Wyjątki dynamiczne w sekcji wykonawczej

101

- Konieczne zastosowanie funkcji

```
RAISE_APPLICATION_ERROR(nr_błędu, komunikat [, zapisywanie])
```

- Można użyć liczby z przedziału od -20999 do -20000.
- Przykład. Fragment procedury dodającej nowego pracownika:
 

```
-- sprawdzenie poprawności daty zatrudnienia przy dodawaniu
nowego pracownika
IF trunc(prac.zatrudniony) > trunc(sysdate) THEN
 RAISE_APPLICATION_ERROR (-20000,'data zatrudnienia nie może
 być w przyszłości');
END IF;
```
- Użytkownik może wywoływać ręcznie zarówno błędy systemowe, jak i zdefiniowane przez siebie.
- Każdy wywołany błąd może zostać obsłużony przez odpowiednią procedurę obsługi wyjątku.

## Literatura dodatkowa

103

- <https://edux.pjwstk.edu.pl/mat/118/lec/w1.html>
- <https://docs.oracle.com/database/121/LNPLS/toc.htm>
- <http://plsql-tutorial.com/>
- <http://kszpyt.blogspot.com/>
- <http://psoug.org/reference/library.html>
- <https://kursplsql.wordpress.com/>
- [http://www.dbf.pl/faq/oracle\\_plsql\\_podstawy.html](http://www.dbf.pl/faq/oracle_plsql_podstawy.html)

## Oracle SQL Data Modeler

104

Uwaga! Prezentacja przydatna przy wyjaśnianiu pojęć:

[http://www.cs.put.poznan.pl/bbebel/pi/03\\_PI\\_modelowanie\\_ER.pdf](http://www.cs.put.poznan.pl/bbebel/pi/03_PI_modelowanie_ER.pdf)

