Systemy Baz Danych

Wykład IX

PL / SQL język proceduralny serwera ORACLE – cz. 1

Materiał wykładu

Tematem kolejnych dwóch wykładów będzie Pl/SQL - język proceduralny serwera bazy danych ORACLE. Zostaną omówione podstawy składni tego języka, oraz. podstawowe struktury programistyczne. Ze zrozumiałych względów będziemy dokonywali porównań pomiędzy poznanym wcześniej T-SQL a Pl/SQL.

Pierwszy wykład zawiera omówienie podstawowej składni, deklaracji zmiennych, podstawowych konstrukcji programistycznych, jakimi są bloki anonimowe, instrukcje warunkowe i pętle, a także kursory.

Wykład jest przeznaczony dla studentów przedmiotu:

Systemy Baz Danych – studia inżynierskie na Wydziale Informatyki PJWSTK,

oraz jako materiał uzupełniający dla słuchaczy przedmiotu

Komunikacja z Bazami Danych – studia podyplomowe na Wydziale Informatyki PJWSTK

Język proceduralny

Z pojęciem języka proceduralnego zapoznaliśmy się w dwóch wykładach dotyczących języka T-SQL – języka SZBD MS SQL Server. Język PL/SQL pełni tą samą rolę w środowisku SZBD ORACLE, czyli jest proceduralnym rozszerzeniem języka SQL.

Składnia języka Pl/SQL w porównaniu z T-SQL jest znacznie bardziej uporządkowana i sformalizowana, przez co staje się trudniejsza w użyciu. Wyposażona jest też w nieco więcej narzędzi niż T-SQL. Jednak w swojej zasadniczej warstwie obie składnie umożliwiają realizację tych samych celów. W obu środowiskach mamy do dyspozycji zmienne, instrukcje sterujące, kursory, procedury, wyzwalacze, narzędzia obsługi błędów.

Język Pl/SQL jest też znacznie bogatszy niż T-SQL, a co za tym idzie, niestety, także bardziej skomplikowany. Z uwagi na to w trakcie obecnych wykładów zostaną zaprezentowane tylko jego podstawowe konstrukcje.

Programy narzędziowe ORACLE

Tak jak narzędziowym, deweloperskim klientem MS SQL Server jest Management Studio, dla ORACLE analogiczną rolę pełnią trzy aplikacje:

- SQL*Plus – najstarsza aplikacja uruchamiana z wiersza poleceń (Command Line),
- iSQLPlus – klient uruchamiany na przeglądarce, obecnie już nie wspierany przez ORACLE; do bazy w PJWSTK możemy się połączyć przez adres https://ora.pjwstk.edu.pl/isqlplus
- Sqldeveloper – klient stacjonarny napisany w środowisku Eclipse; aplikacja jest dostępna na stronach firmy ORACLE, nie wymaga instalowania.

Ponadto język ten może być użyty w prekompilatorach, procedurach, wyzwalaczach, aplikacjach na stronach WWW, w narzędziu ORACLE Developer 2000 (to ostatnia wersja) oraz w SQL Data Modeler który je zastąpił.

SQL*Plus

Najstarszym klientem bazy ORACLE jest prosty program SQL*Plus. Uruchamiany z linii poleceń, działa z każdą instalacją ORACLE na każdej platformie systemowej. Jest też domyślnie instalowany z każdą instalacją ORACLE. Umożliwia on nawiązanie połączenia z bazą danych ORACLE, wydawanie poleceń w językach SQL i Pl/SQL oraz wykonywanie skryptów z poleceniami w tych językach.

SQL*Plus jest interakcyjnym systemem umożliwiającym wprowadzanie i wykonywanie poleceń SQL, skryptów złożonych z poleceń języka SQL i Pl/SQL, uruchamianie bloków i procedur Pl/SQL. Operuje stosunkowo niewielką liczbą własnych instrukcji, umożliwia także operowanie zmiennymi dwóch rodzajów: wiązania i podstawienia, których można używać w instrukcjach SQL i Pl/SQL do wzajemnej komunikacji wartości.

Jakkolwiek niezbyt wygodny w użyciu i dość archaiczny z dzisiejszego punktu widzenia, z uwagi na prostotę, a przede wszystkim możliwość uruchomienia w każdym środowisku, nadal jest używany przez programistów i administratorów baz ORACLE.

Sqldeveloper

Aplikacją narzędziową o funkcjonalności zbliżonej do Management Studio jest „okienkowy” Sqldeveloper. Jest to bardzo mocne narzędzie, dające użytkownikowi wiele możliwości w operowaniu na bazie danych – począwszy od wykonywania zapytań SQL, po działania administratorskie.

Sqldeveloper implementuje część instrukcji pochodzących z SQL*Plus. Znaczna część instrukcji SQL*Plus stała się zbędna w środowisku „okienkowym” i nie jest w nim realizowana.

W dalszej części wykładu, jak też w trakcie ćwiczeń, Sqldeveloper będzie traktowany jako podstawowe środowisko pracy. W PJWSTK można go pobrać z zasobu \\pjwstk.edu.pl\apps\ folder Program Files\Sqldeveloper4.

Instrukcje SQL*Plus dostępne w Sqldeveloper

Istotne instrukcje SQL*Plus

- **SET AUTO[COMMIT] ON | OFF** – włącza/wyłącza automatyczne zatwierdzanie każdej poprawnie wykonanej instrukcji SQL,
- **SET SERVEROUTPUT ON | OFF** – włącza/wyłącza wyświetlanie wyników działania bloków i procedur,
- **SET ECHO** {**OFF|ON**} – wyłącza/włącza wypisanie instrukcji przed jej wykonaniem
- **SET FEEDBACK** – wyłącza/włącza komunikat o liczbie zwróconych rekordów
- **SET VERIFY OFF | ON** – wyłącza / włącza wyświetlanie instrukcji przed i po podstawieniu zmiennej z klawiatury
- **EXEC[UTE]** `nazwa_procedury (parametry…)` – uruchamia procedurę,
- **VARIABLE** `X typ_danych` – deklaracja zmiennej wiązania X,
- **EXECUTE** `:X := wyrażenie` – podstawienie wartości na zmienną wiązania
- **ACCEPT** `nazwa_zmiennej` **PROMPT** `'tekst_komunikatu'` – deklaracja i pobranie od użytkownika wartości zmiennej podstawienia

Blok anonimowy

Podstawowa konstrukcja programistyczną w języku Pl/SQL jest blok - struktura w języku ORACLE Pl/SQL. W przeciwieństwie do MS SQL Server posiada on swoją określoną strukturę. W Sqldeveloper, podobnie jak w Management Studio, poza strukturą bloku mogą być uruchamiane polecenia SQL. Jednak już wszelkie deklaracje zmiennych i uruchamianie instrukcji sterujących musi być wykonywane w obrębie bloku.

ORACLE stosuje się do zapisów standardu języka SQL, zgodnie z którym, wszystkie słowa kluczowe, nazwy obiektów i zmiennych są *Not Case Sensitive* – mogą być pisane z dowolnym użyciem małych i dużych liter. Dotyczy to zarówno składni SQL jak i Pl/SQL.

Natomiast Pl/SQL restrykcyjnie przestrzega konieczności zakończenia każdej instrukcji średnikiem, co w porównaniu z T-SQL wprowadza porządek w kodzie.

Bloki Pl/SQL mogą być zagnieżdżane, z czym wiąże się zasięg dostępności zmiennych deklarowanych w poszczególnych blokach. Blok zagnieżdżony jest traktowany przez blok zewnętrzny jako pojedyncza instrukcja.

Blok anonimowy

Składnia bloku anonimowego jest następująca:

DECLARE

Deklaracje obiektów Pl/SQL (zmienne, stałe, wyjątki, procedury, funkcje)

BEGIN

Ciąg instrukcji SQL i Pl/SQL

EXCEPTION

Obsługa wyjątków

END;

- Sekcje **DECLARE** i **EXCEPTION** są opcjonalne,
- bloki mogą być zagnieżdżane,
- w bloku mogą się pojawić instrukcje **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **COMMIT**, **ROLLBACK**.

Komentarze

Część kodu nie przeznaczona do wykonywania (wyłączona), nazywana jest komentarzem. W trakcie wykonywania programu będzie ona ignorowana przez interpreter. W obu środowiskach (ORACLE i MS SQL) komentarz oznaczany jest jednakowo.

Komentarz blokowy, dowolna liczba linii, pomiędzy nawiasami:

```
/* te wiersze są zakomentowane
i nie będą brane pod uwagę w trakcie
wykonywania programu*/
```

Komentarz jednoliniowy – do końca linii:

```
-- a to są zakomentowane dwa  pojedyncze wiersze,
-- one też zostaną pominięte
SELECT * FROM emp;  --polecenie odczytania danych
```

Skrótem klawiszowym komentowania / odkomentowania jest CTRl + „/”

Typy zmiennych

W Pl/SQL są dostępne typy danych z języka SQL a ponadto m.in.

- **`BOOLEAN`** – wartości logiczne
- **`BINARY_INTEGER`** typ liczb całkowitych – niezależny od podtypów typu **`NUMBER`** i przez to wymagający przy zapisie mniej miejsca w pamięci RAM

Zmienne

W języku Pl/SQL użycie każdej zmiennej musi zostać poprzedzone jej deklaracją.

Nazwy zmiennych „zwyczajne”:

- Muszą zaczynać się od litery,
- mogą zawierać litery, cyfry oraz znaki $, #, _
- nie mogą być słowami kluczowymi SQL ani Pl/SQL,
- nie powinny pokrywać się z nazwami kolumn i tabel,
- maksymalna liczba znaków w nazwie zmiennej nie może przekraczać 30 bajtów,
- nie można łączyć deklaracji kilku zmiennych jednego typu, a każda deklaracja musi być zakończona średnikiem,
- zmiennej w trakcie deklaracji można nadać wartość początkową pod postacią stałej, wyrażenia lub nazwy funkcji.

Nazwami zmiennych mogą też być ciągi dowolnych znaków ujęte w podwójne cudzysłowy, np. **” x+y”**, **”last minute”**, **”on/off switch”**, ale nie polecam!

Podstawienie wartości na zmienną

Aby na zmienną można był podstawić wartość, musi ona zostać wcześniej zadeklarowana. Wartość początkowa może zostać nadana zmiennej łącznie z jej deklaracją.

Operatorem podstawienia w Pl/SQL jest **:=**

```
nazwa_zmiennej := wyrażenie;
```

Drugim sposobem podstawienia jest skorzystanie z instrukcji **SELECT**

```
SELECT wyrażenie, wyrażenie (,…)
INTO   zmienna, zmienna (,…)
FROM   źródła_danych….)
[WHERE …] [GROUP BY…] [HAVING…] (…);
```

Powyższa instrukcja musi zwracać dokładnie jeden rekord, w przeciwnym wypadku wystąpi błąd. Wyrażenia mogą zawierać nazwy kolumn.

ORACLE implementuje dwie pomocnicze tabele, zwracające pojedyncze wartości, które mogą być wykorzystywane w instrukcjach powyższego typu:

DUMMY zwraca 0, **DUAL** zwraca 'X'

Deklaracja zmiennych

Przykład

```
Set Serveroutput on;
DECLARE
     v_nazwisko Varchar2(30); v_placa Int := 1200;
     v_date DATE; v_info VARCHAR2(100);
BEGIN
     v_nazwisko := 'Kowalski';
     v_info := 'Pracownik ' || ' zarabia ' || v_placa
     || od dnia ' || v_date ;
     dbms_output.put_line(v_info);
END;
```

Instrukcja SQL*Plus **`Set Serveroutput on`** to dyrektywa wypisania wyniku działania bloku na konsolę (domyślnie jest zapisywany tylko w logu).

Instrukcja Pl/SQL `dbms_output.put_line(v_info);` to polecenie wypisania z wnętrza bloku wyrażenia tekstowego lub wartości zmiennej podanej jako parametr.

Deklaracja zmiennych i stałych

W Pl/SQL wszystkie niezainicjalizowane zmienne mają wartość NULL.

Przykład :

```
DECLARE
  v_ile INTEGER;    --zmienna nie została zainicjowana
  v_max CONSTANT INTEGER := 10; --stała musi zostać
BEGIN                                --zainicjowana
  v_ile := v_ile + 1;       --zmienna nadal ma wartość NULL
  dbms_output.put_line(v_ile); --czyli nic nie zostanie
END;                                  --wypisane!
```

Stałe podlegają tym samym regułom nazewnictwa i użycia co zmienne, ale musi im zostać nadana wartość podczas deklaracji.

Stałe i zmienne mogą być używane wewnątrz poleceń SQL.

Zmienna może być zadeklarowana jako NOT NULL, ale wówczas trzeba w trakcie deklaracji nadać jej wartość początkową.

Zmienne %TYPE

Deklaracja typu zmiennej może się odwoływać do typu elementu, który został zdefiniowany wcześniej, ale nie znamy jego typu(!). Może to być inna zmienna lub kolumna tabeli. Jeżeli definicja typu „wzorca” ulegnie zmianie, również ulegnie zmianie typ zmiennej lub stałej odwołującej się do tego „wzorca”.

Przykład :

```
DECLARE
   v_ile INTEGER;
   v_max v_ile%type :=10;
   v_sal CONSTANT emp.sal%type := 1100;
BEGIN
 v_ile := v_max + 1;
 dbms_output.put_line(v_ile);
END;
```

Ten sposób deklarowania typów zmiennych jest szczególnie wygodny w odniesieniu do zmiennych przechowujących wartości odczytywane z bazy danych.

Zmienne wierszowe %ROWTYPE

Wzorcem dla deklarowanej zmiennej może być cały wiersz, albo instrukcja **SELECT** definiująca kursor. Na zmienną podstawiana jest wówczas cała zawartość rekordu, a odczyt wartości z poszczególnych pól odbywa się poprzez odwołanie:

nazwa_zmiennej.nazwa_kolumny

Zmiennej wierszowej nie można użyć bezpośrednio po słowie kluczowym **VALUES** w instrukcji **INSERT INTO**.

Przykład :

```
DECLARE
   v_test emp%rowtype; v_ename emp.ename%type;
   v_sal emp.sal%type;
BEGIN
   SELECT * INTO v_test FROM emp WHERE empno = 7788;
   v_ename := v_test.ename;
   v_sal := v_test.sal;
   dbms_output.put_line(v_ename || ' zarabia ' || v_sal);
END;
```

Zmienne rekordowe

W Pl/SQL można zdefiniować własny typ rekordu, definiując jego wzorzec:

```
TYPE Typ_rekordowy IS RECORD (
nazwa_zmiennej Typ_danych,(….));
```

Przykład:

```
DECLARE
   TYPE typ_dane IS RECORD (
   naz Varchar2(20),
   plac Number);
   dane typ_dane;
   v_ename Varchar2(20);
   v_sal Number;
BEGIN
   SELECT ename, sal INTO dane FROM emp WHERE empno = 7788;
   v_ename := dane.naz;
   v_sal := dane.plac;
   dbms_output.put_line(v_ename || ' zarabia ' || v_sal);
END;
```

Zasięg działania zmiennych

Zasięg zmiennej (lub stałej) w Pl/SQL to obszar bloków, z których można się do tej zmiennej odwołać. Ponieważ bloki mogą być zagnieżdżane, zatem zasięg zmiennych jest zależny od tego, w jakim bloku została zadeklarowana. Zmienna zadeklarowana w bloku jest dostępna w blokach w nim zagnieżdżonych (**wewnętrznych** względem bloku deklaracji). Zmienna zadeklarowana w bloku nie jest dostępna w bloku **zewnętrznym** w stosunku do bloku jej deklaracji, czyli bloku, w którym blok jej deklaracji jest zagnieżdżony.

Zmienne zadeklarowane w bloku nie są dostępne w blokach „sąsiednich", utworzonych na tym samym poziomie.

Odwołanie do zmiennej znajdującej się poza zasięgiem wywołuje błąd.

Zmienne w blokach zagnieżdżonych mogą mieć te same nazwy, ale są traktowane jako różne zmienne!

Zasięg działania zmiennych

Przykład :

```
DECLARE
  v_tytul Varchar2(20);
BEGIN
  v_tytul := 'ORACLE jest latwy';
  DECLARE
    v_tytul2 Varchar2(20);
    v_tytul Varchar2(20);
  BEGIN
    v_tytul2 := 'Pl/SQL jest latwy';
    v_tytul := 'Pl/SQL jest TRUDNY';
  END;
dbms_output.put_line(v_tytul);
--dbms_output.put_line(v_tytul2); zmienna poza zasięgiem
END;                                    --wygeneruje błąd
```

ORACLE jest latwy

Zasięg działania zmiennych

Przykład :

Zmienne o tej samej nazwie, zadeklarowane w zagnieżdżonych blokach, to dwie różne zmienne;

```
DECLARE
  v_tytul Varchar2(20);
BEGIN
  v_tytul := 'ORACLE jest latwy';
  DECLARE
    v_tytul Varchar2(20);
  BEGIN
    v_tytul := 'Pl/SQL jest TRUDNY';
    dbms_output.put_line(v_tytul);
  END;
dbms_output.put_line(v_tytul);
END;
```

```
Pl/SQL jest TRUDNY
ORACLE jest latwy
```

Zasięg działania zmiennych

Przykład :

Do zmiennej zadeklarowanej w bloku zewnętrznym odwołujemy się z bloku zewnętrznego i wewnętrznego:

```
DECLARE
     v_tytul Varchar2(20);
BEGIN
     v_tytul := 'ORACLE jest latwy';
     BEGIN
     v_tytul := 'Pl/SQL jest TRUDNY';
     dbms_output.put_line(v_tytul);
   END;
dbms_output.put_line(v_tytul);
END;
```

```
Pl/SQL jest TRUDNY
Pl/SQL jest TRUDNY
```

Zmienne podstawienia i zmienne wiązania

Oprócz zmiennych deklarowanych w bloku Pl/SQL mogą jeszcze występować zmienne z aplikacji korzystających z bloku Pl/SQL. Nazywane są **zmiennymi wiązania**, a w odwołaniach ich nazwa poprzedzana jest dwukropkiem. Deklarowane są poleceniem SQL*Plus **VARIABLE**, wypisywane poleceniem **PRINT**.

Drugim rodzajem zmiennych są **zmienne podstawienia** SQL*Plus, które mogą występować wyłącznie w wyrażeniach po prawej stronie instrukcji przypisania. W odwołaniach ich nazwa poprzedzana jest znakiem &.

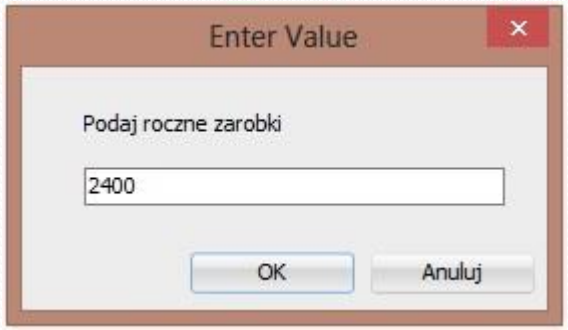
Oba te rodzaje zmiennych mogą służyć do wprowadzania wartości z klawiatury i przekazania ich do wykonania w bloku Pl/SQL.

Dalsze dwa przykłady pokazują użycie zmiennych wiązania i zmiennych podstawienia do wykonania operacji w bloku Pl/SQL.

Zmienne podstawienia i zmienne wiązania

Przykład użycia zmiennej podstawienia.

```
ACCEPT year_sal PROMPT 'Podaj roczne zarobki'
DECLARE
   sal NUMBER(8,2) := &year_sal;
BEGIN
   sal := sal/12;
   dbms_output.put_line(sal);
END;
```



W wyniku uruchomienia skryptu wyświetlane jest okno dialogowe pozwalające wprowadzić wartość zmiennej podstawienia, a po jej wprowadzeniu zwracany jest wynik.

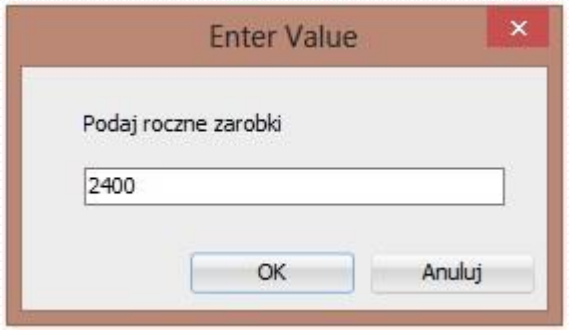
```
200
```

Zmienne podstawienia i zmienne wiązania

Przykład użycia zmiennej wiązania.

```
ACCEPT year_sal PROMPT 'Podaj roczne zarobki'
VARIABLE b_sal NUMBER;
BEGIN
  :b_sal := &year_sal /12;
END;
/
PRINT b_sal
```

Wynik jak poprzednio, ale zmienne deklarowane i odczytywane poza blokiem Pl/SQL.



200

Zmienne systemowe

- **SQL%Rowcount** – zwraca liczbę wierszy przetworzonych przez ostatnią instrukcję SQL,
- **SQL%Found** = TRUE jeśli został znaleziony rekord,
- **SQL%Notfound** = TRUE jeśli nie został znaleziony żaden rekord
- **SQLErrm** – tekst komunikatu o błędzie
- **SQLCode** – numer błędu

Dwie ostatnie zmienne mogą być użyte wyłącznie w sekcji **EXCEPTION**.

Przykład:

```
DECLARE ile number :=0;
BEGIN
   UPDATE emp SET sal = sal*1.1 WHERE deptno = 30;
   ile := SQL%Rowcount;
   dbms_output.put_line(ile);
END;
```

Instrukcje warunkowe

```
IF warunek THEN
   ciąg_instrukcji
END IF;

IF warunek THEN
   ciąg_instrukcji
ELSE
   ciąg_instrukcji
END IF;

IF warunek THEN
   ciąg_instrukcji
ELSIF warunek THEN
   ciąg_instrukcji
END IF;
```

- Instrukcje po **THEN** są wykonywane wtedy, gdy wartością `warunku` jest **TRUE**.
- Instrukcje po **ELSE** są wykonywane wtedy, gdy wartością `warunku` jest **FALSE** lub **NULL**.

Instrukcja iteracji (pętli) LOOP

```
LOOP ciąg_instrukcji END LOOP;
```

Instrukcje wewnątrz pętli wykonywane są w każdej iteracji. Wyjście z pętli następuje po wywołaniu instrukcji **EXIT** albo **EXIT WHEN** albo wywołaniu błędu.

Przykład:

```
DECLARE
  x NUMBER := 0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Wewnątrz:  x = ' || TO_CHAR(x));
    x := x + 1;
    IF x > 3 THEN
      EXIT;
    END IF;
  END LOOP;
DBMS_OUTPUT.PUT_LINE('Po wyjściu:  x = ' || TO_CHAR(x));
END;
```

```
Wewnątrz:  x = 0
Wewnątrz:  x = 1
Wewnątrz:  x = 2
Wewnątrz:  x = 3
Po wyjściu:  x = 4
```

Instrukcja iteracji (pętli) FOR

```
FOR i IN [REVERSE] wartość1..wartość2
LOOP ciąg_instrukcji END LOOP;
```

Przykład 1:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE ('w_dolna < w_górna');
  FOR k IN 1..3 LOOP DBMS_OUTPUT.PUT_LINE (k);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('w_dolna = w_górna');
  FOR i IN 2..2 LOOP DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;
END;
```

w_dolna < w_górna
1
2
3
w_dolna = w_górna
2

Instrukcj iteracji (pętli)

```
FOR i IN [REVERSE] wartość1..wartość2
LOOP ciąg_instrukcji END LOOP;
```

Przykład 2:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE ('w_dolna < w_górna');
  FOR k IN 3..1 LOOP DBMS_OUTPUT.PUT_LINE (k);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('w_dolna < w_górna');
  FOR i IN REVERSE 1..2 LOOP DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;
END;
```

w_dolna < w_górna
w_górna < dolna
2
1

Instrukcje iteracji (pętli)

```
WHILE warunek LOOP ciąg_instrukcji END LOOP;
```

Iteracja wykonywana jest tak długo, jak długo **warunek** ma wartość TRUE.

Przykład:

```
DECLARE
  done  BOOLEAN := FALSE;
BEGIN
  WHILE done LOOP
    DBMS_OUTPUT.PUT_LINE ('Ta linia nie zostanie wypisana.');
    done := TRUE;  -- To podstawienie nie zostanie wykonane.
  END LOOP;

  WHILE NOT done LOOP
    DBMS_OUTPUT.PUT_LINE ('Hello, world!');
    done := TRUE;
  END LOOP;
END;
```

Hello, world!

Kursory

Zasada działania i użycia kursorów Pl/SQl jest taka sama jak T-SQl. Różnice są wyłącznie składniowe. Ponieważ Pl/SQL nie wyprowadza danych w postaci Result Set, zatem znacznie częściej zachodzi konieczność użycia kursora.

Zdefiniowanie kursora:

```
CURSOR nazwa_kursora IS instrukcja_SELECT;
```

Otwarcie kursora:

```
OPEN nazwa_kursora ;
```

Pobieranie kolejnych wierszy:

```
FETCH nazwa_kursora INTO zmienna (,…);
```

Wyjście z pętli po odczytaniu wszystkich wierszy:

```
EXIT WHEN nazwa_kursora%NOTFOUND;
```

Zamknięcie kursora:

```
CLOSE nazwa_kursora;
```

Kursory – zmienne stanu kursora

- **nazwa_kursora%FOUND** = TRUE jeśli z bazy sprowadzono kolejny rekord,
- **nazwa_kursora%NOTFOUND** = TRUE jeśli kolejny rekord nie został znaleziony – koniec sprowadzania rekordów,
- **nazwa_kursora%ROWCOUNT** liczba sprowadzonych dotąd rekordów,
- **nazwa_kursora%ISOPEN** = TRUE jeżeli kursor jest otwarty

```
IF NOT nazwa_kursora%ISOPEN THEN
  OPEN nazwa_kursora;
END IF;
LOOP
  FETCH nazwa_kursora INTO …
```

Kursory

Przykład:

```
SET SERVEROUTPUT ON
DECLARE
  v_dname Varchar2(10); v_deptno INT; v_dsal Number(8,2) := 0;
  CURSOR Cur_Dept IS SELECT deptno, dname FROM dept;
BEGIN
  OPEN Cur_Dept;
  LOOP
    FETCH Cur_Dept INTO v_deptno, v_dname;
    EXIT WHEN Cur_Dept%NOTFOUND;
    SELECT NVL(SUM(sal), 0) INTO v_dsal
    FROM emp WHERE deptno = v_deptno;
    dbms_output.put_line('Budżet dzialu ' || v_dname
                         || ' wynosi ' || v_dsal);
    v_dsal := 0;
  END LOOP;
  CLOSE Cur_Dept;
END;
```

Budżet dzialu ACCOUNTING wynosi 8750
Budżet dzialu RESEARCH wynosi 10075
Budżet dzialu SALES wynosi 9000
Budżet dzialu OPERATIONS wynosi 0

Kursory – aliasy kolumn

W deklaracji kursora, na liście **SELECT** mogą znajdować się dowolne wyrażenie, przy czym jeśli nie są to „czyste” nazwy kolumn, muszą być stosowane aliasy.

Przykład:

```
CURSOR      Budzety_dzialow IS
SELECT      Deptno, Dname, Sum(sal) AS Budzet_dzialu
FROM        emp
INNER JOIN  Dept
ON          emp.deptno = dept.deptno;
```

Kursory – iteracja z kursorem

Odwołać się do kursora można także w instrukcji iteracji. W tym przypadku operacje **OPEN**, **FETCH** i **CLOSE** wykonywane są niejawnie, a kursor zakończy się i zostanie zamknięty po podstawieniu wszystkich rekordów.

Przykład:

```
SET Serveroutput ON
DECLARE
CURSOR kursor IS SELECT dname, loc
                 FROM dept;
BEGIN
   FOR kursor_rek IN kursor
   LOOP
     dbms_output.put_line(kursor_rek.dname || ' ' ||
                          kursor_rek.dname);
   END LOOP;
END;
```

ACCOUNTING
RESEARCH
SALES
OPERATIONS

UWAGA: w powyższym przykładzie `kursor_rek` to dowolna nazwa.

Kursory – kursor z parametrami

W instrukcji **SELECT** w kursorze mogą występować parametry. Ten sam kursor może być otwarty wielokrotnie – z różnymi wartościami parametrów.

```
CURSOR nazwa_kursora(parametr typ_danych, ....)
IS     instrukcja_SELECT;
```

Przykład pokazano na następnym slajdzie.

Kursory – kursor z parametrami

Przykład:

```
SET serveroutput on
DECLARE
   CURSOR Dept_job (v_deptno Int, v_job Varchar2)
   IS  SELECT Empno, Ename, sal
   FROM emp WHERE deptno = v_deptno AND job = v_job;
   v_ees Dept_job%rowtype;
BEGIN
   OPEN Dept_job(10, 'CLERK');
   LOOP
     FETCH Dept_job INTO v_ees;
     EXIT WHEN Dept_job%NOTFOUND;
     dbms_output.put_line(v_ees.empno || ' ' || v_ees.ename
                          || ' ' || v_ees.sal);
   END LOOP;
   CLOSE Dept_job;
END;
```

7934 MILLER 1300

Aktualizacja rekordów za pomocą kursora

Przy wykonywaniu instrukcji SELECT definiującej kursor mogą być zakładane blokady na wiersze, w celu ich modyfikacji.

Dyrektywa

```
…FOR UPDATE [OF kolumna, kolumna…]
```

określa tabele, lub kolumny tabel, w których rekordy mają zostać zablokowane w celu umożliwienia modyfikacji.

Stowarzyszona z nią w instrukcji **UPDATE** lub **DELETE** klauzula

```
…WHERE CURRENT OF nazwa_kursora
```

umożliwia modyfikację lub usunięcie wiersza aktualnie sprowadzonego przez kursor z danej tabeli.

Przykład został pokazany na kolejnym slajdzie.

Aktualizacja rekordów za pomocą kursora

```
DECLARE
CURSOR ename_sal IS SELECT ename, sal FROM emp FOR UPDATE OF sal;
v_rekosoba ename_sal%ROWTYPE; v_newsal NUMBER(8,2);
BEGIN
   OPEN ename_sal;
   LOOP
     FETCH ename_sal INTO v_rekosoba;
     EXIT WHEN ename_sal%NOTFOUND;
     IF v_rekosoba.sal < 1000 THEN
       v_newsal := v_rekosoba.sal*1.1;
       UPDATE emp SET sal = v_rekosoba.sal * 1.1
       WHERE CURRENT OF ename_sal;
       dbms_output.put_line('Podniesiono place ' || ' ' ||
                  v_rekosoba.ename || ' do ' || v_newsal);
     END IF;
   END LOOP;
CLOSE ename_sal;
COMMIT;
END;
```

Podniesiono place SMITH do 968

Obsługa błędów – niektóre nazwane wyjątki

- **dup_val_on_index** – powtarzająca się wartość w indeksie jednoznacznym (UNIQUE),
- **no_data_found** – instrukcja **SELECT** nie zwróciła wartości przeznaczonych do podstawienia na zmienne w klauzuli **SELECT** … **INTO**…
- **too_many_rows** – instrukcja SELECT zwróciła więcej niż jeden wiersz zawierający wartości przeznaczone do podstawienia na zmienne w klauzuli **SELECT** … **INTO**…
- **zero_divide** – błąd dzielenia przez zero,
- **timeout_on_resource** – przekroczony limit czasu oczekiwania na zwrot żądanych zasobów,
- **invalid_cursor** – próba wykonania niepopranej operacji na kursorze,
- **invalid_number** – błąd (próby) konwersji na liczbę,
- **cursor_already _open** – próba otwarcia kursora, który jest już otwarty

Obsługa błędów

Ogólna składnia obsługi wyjątków wygląda następująco:

```
DECLARE
   Deklaracje zmiennych i kursorów
BEGIN
   Ciąg instrukcji Pl/SQL i SQL
EXCEPTION
WHEN nazwa_wyjątku_1 THEN
   Ciąg instrukcji
WHEN nazwa_wyjątku_2 THEN
   Ciąg instrukcji
   (…)
WHEN OTHERS THEN
   Ciąg instrukcji
END;
```

Obsługa błędów

Przykład:

```
SET Serveroutput ON
DECLARE
   v_ename emp.ename%TYPE;
   v_info Varchar2(100);
BEGIN
   SELECT ename INTO v_ename FROM emp WHERE sal = 1100;
   EXCEPTION
     WHEN no_data_found THEN
       dbms_output.put_line('Brak pracowników o pensji 1100');
     WHEN too_many_rows THEN
        dbms_output.put_line('Jest kilku pracowników o pensji
                              1100');
     WHEN OTHERS THEN
        V_info := 'Bląd nr ' || SQLCODE || ', komunikat: ' ||
                  Substr(SQLERRM, 1, 50);
     dbms_output.put_line(v_info);
END;
```

Obsługa błędów

Jeżeli blok, w którym wystąpił błąd, zawiera obsługę tego błędu, to po dokonaniu obsługi, sterowanie jest w zwykły sposób przekazywane do bloku go zawierającego (nadrzędnego).

Jeśli nie zawiera obsługi błędów, następuje przekazanie błędu do bloku w którym dany blok jest zagnieżdżony (czyli do bloku nadrzędnego) i albo tam nastąpi jego obsługa, albo błąd przechodzi do środowiska zewnętrznego.

Błąd, który wystąpił w sekcji wykonawczej bloku (między **BEGIN** i **END**) jest obsługiwany w sekcji **EXCEPTION** tego samego bloku. Błędy, które wystąpią w sekcji deklaracji lub w sekcji wyjątków są przekazywane do bloku nadrzędnego.

Dobra praktyka programistyczna wymaga, aby każdy błąd został obsłużony – w ostateczności w klauzuli **WHEN OTHERS THEN** najbardziej zewnętrznego bloku.

Aby móc stwierdzić, która instrukcja SQL spowodowała błąd:

- można używać podbloków z własną obsługą błędów, albo
- można używać licznika, zwiększającego się o jeden po wykonaniu każdej instrukcji SQL.

Obsługa błędów

Wyjątki mogą być deklarowane przez programistę. W tym celu w sekcji **DECLARE** wprowadzana zostaje nazwa wyjątku ze słowem kluczowym **EXCEPTION**.

```
DECLARE
   Nazwa_wyjątku EXCEPTION;
   (…)
```

W sekcji instrukcji tak zadeklarowane wyjątki mogą być podnoszone instrukcją

```
RAISE nazwa_wyjątku;
```

a następnie są obsługiwane w sekcji **EXCEPTION**

```
WHEN nazwa_wyjątku THEN …
```

Przykład znajduje się na kolejnym slajdzie.

Obsługa błędów

Przykład:

```
DECLARE
   v_budzet emp.sal%TYPE; v_info Varchar2(100);
   v_notenough EXCEPTION;
BEGIN
   SELECT Sum(sal+100) INTO v_budzet FROM emp WHERE deptno = 30;
   IF v_budzet <= 11000 THEN
      UPDATE emp SET sal = sal + 100 WHERE deptno = 30;
      V_info := 'Dokonano podwyzki w dziale 30';
      dbms_output.put_line(v_info);
   ELSE
      RAISE v_notenough;
   END IF;
   EXCEPTION
      WHEN v_notenugh THEN
         V_info := 'Zbyt maly budzet na podwyżki w dziale 30';
         dbms_output.put_line(v_info);
         RAISE wniosek;    --błąd obsłużony w bloku nadrzędnym
END;
```

Obsługa błędów

Istnieje także możliwość podniesienia wyjątku za pomocą procedury

Raise_application_error(numer_błędu, komunikat)

z przypisaniem mu numeru oraz treści komunikatu. Wyjątek taki może zostać obsłużony w tym samym bloku, albo w aplikacji zewnętrznej, w której to wywołanie zostało wykonane.

Dostępny jest zakres numeracji błędów od -20000 do -209999, jako numery błędów definiowane przez programistę.

```
BEGIN      .....
   Raise_Application_Error(-20100,'Błąd');
EXCEPTION
WHEN OTHERS THEN
   numer:=SQLCODE;
   IF numer = -20100 THEN
      Dbms_output.Put_line('Błąd przechwycony!');
   END IF;
END;
```