3. Język PL/SQL

Język PL/SQL (Procedural Language /SQL) jest proceduralnym rozszerzeniem języka SQL zaproponowanym przez firmę Oracle umożliwiającym wykorzystanie rozkazów SQL w strukturze bloków stanowiących narzędzie programowania transakcji. W ramach programów tego języka, oprócz poleceń SQL, możliwe jest wykorzystanie występujących w językach proceduralnych konstrukcji takich jak:

- zmienne i typy danych (predefiniowane jak, i definiowane przez użytkownika),
- struktury sterowania takie jak np. instrukcje warunkowe i instrukcje pętli,
- procedury i funkcje,
- typy obiektowe i metody.

Można wymienić następujące istotne powody konieczności rozszerzenia możliwości języka SQL:

- manipulacja danymi realizowana przez SQL nie jest jedynym zadaniem obsługi bazy danych (dodatkowe zadania realizowane są na poziomie aplikacji),
- reguły biznesowe (także te dotyczące bezpieczeństwa danych) kontrolowane są na poziomie aplikacji (można je ominąć stosując inne narzędzie dostępu do bazy danych). Naraża to dane na celowe lub przypadkowe modyfikacje,
- każde zapytanie SQL wymaga osobnego połączenia z serwerem bazy danych co zwiększa obciążenie sieci (lepiej byłoby wysyłać wiele zapytań podczas jednego połączenia).

PL/SQL poprzez możliwość definicji podprogramów i pakietów składowanych w bazie danych umożliwia "przerzucenie" wielu zadań związanych z szeroko rozumianą obsługą bazy danych na serwer bazy danych sprowadzając aplikację klienta do typowego interfejsu ekranowego. Pozwala to na szybszy dostęp do danych zwiększając jednocześnie ich bezpieczeństwo. Wysyłanie wielu rozkazów SQL w ramach jednego bloku PL/SQL daje zmniejszenie obciążenia sieci i prowadzi do szybszego działania aplikacji.

3.1 Podstawowe informacje i pojęcia

Poniżej przedstawiono zestaw informacji i pojęć wprowadzających do programowania w języku PL/SQL.

3.1.1 Struktura blokowa programu PL/SQL

Program PL/SQL składa się z jednego lub większej liczby bloków. Bloki mogą być niezależne lub zagnieżdżone jeden w drugim. Wyróżnia się dwa rodzaje bloków: blok anonimowy i blok nazwany.

Blok anonimowy: blok PL/SQL bez nazwy, deklarowany w takim miejscu aplikacji, w którym będzie wykonany.

Blok anonimowy jest zazwyczaj przekazywany z programu działającego po stronie klienta w celu wywołania podprogramów zapisanych (składowanych) w bazie danych.

Struktura bloku anonimowego:

```
[DECLARE
   -- definicje i deklaracje obiektów PL/SQL dla bloku]
BEGIN
   --  zdania części wykonywalnej bloku
[EXCEPTION
   -- zdania obsługi wyjątków]
END;
/
-- linia pusta w przypadku uruchamiania ze skryptu w MS Windows
```

Definicje obiektów bloku, zdania części wykonywalnej (musi tu wystąpić przynajmniej jedno zdanie) i zdania obsługi wyjątków oddzielane są znakiem średnika. Blok kończy swoje działanie gdy wykonane zostaną wszystkie zdania części wykonywalnej lub wystąpi sytuacja wyjątkowa (błąd), obsłużona lub nie w części obsługi wyjątków. W ramach części wykonywalnej i części obsługi wyjątków mogą wystąpić bloki wewnętrzne. W części wykonywalnej bloki wewnętrzne stosuje się zwykle do obsługi wyjątków, które nie mają kończyć działania bloku zewnętrznego a w części obsługi wyjątków

do obsługi wyjątków od wyjątków. Każdy obiekt zdefiniowany w danym bloku jest dostępny tylko w tym bloku (a więc i w jego blokach wewnętrznych). W przypadku zdefiniowania obiektów o tych samych nazwach w blokach zewnętrznym i wewnętrznym obowiązuje zasada przykrywania nazw (na czas działania bloku wewnętrznego obiekt zdefiniowany w bloku zewnętrznym nie jest dostępny).

Blok nazwany: blok PL/SQL, któremu przypisano nazwę, do której można się odwołać.

Można wyróżnić trzy rodzaje bloków nazwanych:

- **blok oznaczony etykietą**: jest to blok anonimowy, któremu przydzielono etykietę będącą jego nazwą. Nazwa ta pozwala na odwoływanie się do zmiennych bloku z bloku wewnętrznego w przypadku tych samych nazw zmiennych w obu blokach,
- **blok podprogramu**: jest to procedura lub funkcja, do której można się jawnie odwoływać (po nazwie) z każdego rodzaju bloku. Podprogram można składować w bazie danych,
- **blok wyzwalacza**: jest blok składowany w bazie danych wykonywany niejawnie w przypadku zaistnienia określonego w definicji wyzwalacza zdarzenia.

Struktura bloku oznaczonego etykietą jest taka sama jak struktura bloku anonimowego. Struktura podprogramu i wyzwalacza różni się praktycznie od struktury bloku anonimowego tylko wystąpieniem nagłówka. Jako pierwsze więc zostaną omówione bloki anonimowe.

3.1.2. Wyświetlanie na ekranie komunikatów diagnostycznych

Do wyświetlania komunikatów diagnostycznych z poziomu bloku PL/SQL służą procedury z pakietu DBMS_OUTPUT o nazwach PUT_LINE, PUT, NEW_LINE. PUT_LINE umieszcza w buforze komunikat o maksymalnej długości 255 (ze znakiem przejścia do nowej linii), PUT komunikat bez przejścia do nowej linii - to przejście realizuje jawnie funkcja NEW_LINE. Aby komunikaty wprowadzane do bufora pojawiały się na ekranie, należy w środowisku SQL Developer wybrać w zakładce View element Dbms Output a w środowisku SQL*Plus należy wykonać polecenie:

SET SERVEROUTPUT {ON | OFF} [SIZE n]

ON w ramach polecenia powoduje włączenie wyświetlania komunikatów (domyślnie wyświetlanie jest wyłączone - OFF). Po SIZE określany jest w bajtach rozmiar bufora (maksymalnie 1000000 bajtów, domyślnie 2000 bajtów).

Procedury PUT_LINE i PUT wywoływane są zgodnie ze składnią:

PUT_LINE(komunikat [, VARCHAR2|NUMBER|DATE]);
PUT(komunikat [, VARCHAR2|NUMBER|DATE]);

Wywołanie procedur z pakietów poprzedza się nazwą pakietu np.:

```
DBMS_OUTPUT.PUT_LINE('Wiwat TYGRYS, Pan nad Pany!!!');
```

Funkcje pakietu DBMS_OUTPUT służą jedynie do testowania działania bloków PL/SQL. Rzeczywiste wyprowadzanie informacji na ekran odbywa się w ramach interfejsu ekranowego aplikacji bazodanowej.

***Zad**. Zdefiniować blok anonimowy wstawiający do relacji Kocury nowe krotki poprzez zdefiniowaną wcześniej perspektywę Banda4. W przypadku wprowadzenia błędnych danych (np. powtarzającego się imienia – na imię nałożony ma być unikalny indeks) na ekranie mają się pojawić odpowiednie komunikaty.*

```
SQL> CREATE UNIQUE INDEX unikalne_imie ON Kocury(imie);
Indeks został utworzony.
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2   INSERT INTO Banda4 VALUES ('&pseudo','&imie','&funkcja',
  3                              &przydzial_myszy,&nr_bandy);
  4   COMMIT;
  5  EXCEPTION
  6   WHEN DUP_VAL_ON_INDEX
  7   THEN DBMS_OUTPUT.PUT_LINE('Powtarzajace sie pseudo lub
  8                             imie!!! - BRAK WPISU!');
  9   WHEN OTHERS
 10   THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
 11  END;
 12  /
Proszę podać wartość dla pseudo: KOLES
Proszę podać wartość dla imie: MRUCZEK
```

```
Proszę podać wartość dla funkcja: KOT
stare   2:  INSERT INTO Banda4 VALUES
                                ('&pseudo','&imie','&funkcja',
nowe   2:  INSERT INTO Banda4 VALUES
                                ('KOLES','MRUCZEK','KOT',
Proszę podać wartość dla przydzial_myszy: 40
Proszę podać wartość dla nr_bandy: 4
stare   3:                     &przydzial_myszy,&nr_bandy);
nowe   3:                      40,4);
Powtarzajace sie pseudo lub imie!!! - BRAK WPISU!

Procedura PL/SQL została zakończona pomyślnie.

SQL>
```

Znak & przed zmienną oznacza, że jej wartość ma być wprowadzona z klawiatury. Polecenie WHEN … THEN obsługuje wyjątek. Jest on wymieniony z nazwy po WHEN a obsługiwany jest w sposób określony po THEN. DUP_VAL_ON_INDEX jest predefiniowanym wyjątkiem wskazującym na naruszenie unikalnego indeksu (dotyczy to także pseudonimu jako atrybutu kluczowego – na klucz nakładany jest automatycznie indeks a unikalność jest cechą klucza). OTHERS określa dowolny, inny od wcześniej wymienionych, wyjątek. SQLERRM jest funkcją wyświetlającą systemowy komunikat o błędzie (wyjątku).

Parametr SERVEROUTPUT środowiska SQL*Plus pozostanie dalej, na potrzeby wykładu, ustawiony na wartości ON.

3.1.3 Środowisko dla PL/SQL

Bloki PL/SQL przetwarzane są przez maszynę PL/SQL rezydującą w SZBD lub w programie narzędziowym. Jeśli wywołanie bloku następuje z poziomu aplikacji stworzonej za pośrednictwem programu narzędziowego z zaimplementowaną maszyną PL/SQL, blok ten przetwarzany jest przez tą maszynę (wykonanie po stronie klienta), w przeciwnym wypadku (np. wywołanie z poziomu SQL*Plus) blok przetwarzany jest przez maszynę PL/SQL rezydującą w SZBD (wykonanie po stronie serwera). W obu tych przypadkach jednak maszyna PL/SQL wykonuje tylko rozkazy proceduralne a rozkazy SQL przesyła do wykonawcy rozkazów SQL w SZBD.

3.1.4 Identyfikatory i dostępne symbole (ograniczniki) w PL/SQL

Identyfikator w PL/SQL zaczyna się od litery, po której może nastąpić dowolna sekwencja znaków złożona z liter, cyfr, oraz znaków '$', '_', i '#'. Identyfikator zadeklarowany w apostrofach ("identyfikator") może zawierać dowolne znaki. Identyfikator może się składać maksymalnie z 30 znaków.

Dostępne w ramach Oracle symbole mogą się składać z jednego lub dwóch znaków.

Symbole składające się z jednego znaku:

+, -, *, / - operatory arytmetyczne	=, >, < - operatory relacji
(,), - ograniczniki wyrażenia	; - ogranicznik zdania
, - separator pozycji	. - ogranicznik składnika
@ - wskaźnik łącza bazy danych	' - ogranicznik ciągu znaków
" - ogranicznik cytowanego ciągu	: - wskaźnik zmiennej zewn.
% - wskaźnik atrybutu	

Symbole złożone z dwóch znaków:

** - operator potęgowania	:= - operator przypisania
<>, !=, ^= ,~= - różny	<= - mniejszy równy
>= - większy równy	=> - operator skojarzenia
.. - operator zakresu	|| - operator konkatenacji
<<, >> - ograniczniki etykiety	-- - komentarz (jedna linia)
/*, */ - początek i koniec komentarza	

3.1.5 Zmienne i stałe

W ramach bloków PL/SQL możliwa jest deklaracja następujących rodzajów zmiennych:

- skalarne: typy znane z dialektu SQL Oracle'a,
- złożone: typ rekordowy, typy tablicowe zwane kolekcjami (do Oracle 7 włącznie tablice indeksowe, od Oracle 8 dodatkowo tablice zagnieżdżone i tablice o zmiennym rozmiarze, od Oracle 9i kolekcje wielopoziomowe – kolekcje kolekcji),

- odwołania (wskaźniki, do Oracle 7 włącznie REF CURSOR, od Oracle 8 REF typ_objektu),
- LOB: BFILE, CLOB, NCLOB, BLOB (dostępne od Oracle 8 poprzez pakiet DBMS_LOB, obsługują duże obiekty binarne lub znakowe do 4 GB bez ograniczeń charakterystycznych dla LONG i LONG RAW, np. tylko jeden atrybut tego typu w relacji czy brak możliwości manipulacji na nich za pomocą wyzwalaczy).

Deklaracja zmiennej następuje zgodnie ze składnią:

nazwa_zmiennej typ [[**NOT NULL] {DEFAULT** | :=} wyrażenie];

Jeśli w deklaracji wystąpi NOT NULL, obowiązkowe jest określenie wartości domyślnej.

Bardzo często wykorzystywanym mechanizmem jest deklaracja zmiennej o typie zgodnym z typem atrybutu relacji. Odbywa się to zgodnie ze składnią:

nazwa_zmiennej nazwa_relacji.atrybut**%TYPE**;

Dla danego typu można zdefiniować jego podtyp zgodnie ze składnią:

SUBTYPE nowy_typ **IS** oryginalny_typ;

Oryginalny typ, do Oracle 8i, nie mógł posiadać ograniczenia długości. Od Oracle 8i zakaz ten jest zniesiony.

Stała w ramach bloku PL/SQL jest definiowana zgodnie ze składnią:

nazwa_stałej **CONSTANT** typ := wyrażenie;

Poniżej przedstawiono przykładowe deklaracje zmiennych i definicje stałych.

```
SUBTYPE  T IS VARCHAR2;
SUBTYPE  OPIS IS Wrogowie_kocurow.opis_incydentu%TYPE;
nazwisko T(25):='Kowalski';
imie     T(15):='Jan';
inicjal  T(4) :=
             SUBSTR(nazwisko,1,1)||'.'||SUBSTR(imie,1,1)||'.';
licznik  INTEGER NOT NULL:=0;
pesel    NUMBER(11);
dzieci   BOOLEAN:=FALSE;
koniec   BOOLEAN;
data     DATE;
dane     OPIS;
ps       Kocury.pseudo%TYPE;
pi       CONSTANT NUMBER(9,5):=3.14159;
```

3.1.6. Operacja przypisania

Operacja przypisania w ramach bloku PL/SQL wykonywana jest zgodnie ze składnią:

nazwa_zmiennej:=wyrażeniePL/SQL;

W ramach wyrażenia PL/SQL można stosować wszystkie znane z SQL'a funkcje (oprócz GRATEST, LEAST, DECODE i funkcji grupowych) oraz dodatkowo, jeśli wyrażenie wystąpi w ramach sekcji EXCEPTION bloku, funkcje SQLCODE (zwraca nr sytuacji wyjątkowej) i SQLERRM (zwraca komunikat o wyjątku łącznie z jego numerem).

Poniżej przedstawiono przykładowe operacje przypisania.

```
licznik:=licznik+1;
dane:=licznik||'. '||inicjal||' '||pesel;  -- autokonwersja
dzieci:=NOT dzieci;
koniec:=licznik=100;
```

Operatory w wyrażeniu PL/SQL wykonywane są w innej kolejności niż w wyrażeniu SQL. Poniżej przedstawiono kolejność wykonywania (priorytet) operatorów w PL/SQL.

1. **, NOT
2. +, - (jako znaki liczb)
3. *, /
4. +, - , ||
5. =, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN
6. AND
7. OR

Dla wyrażeń logicznych występujących w ramach poleceń PL/SQL, w których pojawi się wartość NULL, obowiązują te same zasady (logika) jak te obowiązujące dla wyrażeń logicznych w ramach klauzul SQL.

3.1.7. PL/SQL w SQL*Plus

Bloki w PL/SQL w ramach środowiska SQL*Plus mogą być przetwarzane na dwa sposoby:

a) poprzez wprowadzenie bloku do bufora i uruchomienie bufora - rozpoczęcie wprowadzania bloku do bufora to klauzula DECLARE lub BEGIN, koniec to znak kropki `'.'`, uruchomienie bloku z bufora to rozkaz RUN lub znak `'/'`
b) uruchomienie bloku ze skryptu poleceniem START z ewentualnym parametrami (zmienne zdefiniowane w skrypcie po znaku &).

3.1.8. Polecenia SQL w PL/SQL

W ramach bloku PL/SQL dozwolone jest stosowanie wprost poleceń DML, DCL i polecenia SELECT o zmodyfikowanej składni, zwracającego tylko jedną krotkę (składnia tego polecenia przedstawiona zostanie w dalszej części). Polecenia DDL można umieszczać w bloku tylko za pośrednictwem podprogramów pakietu DBMS_SQL realizującego tzw. **dynamiczny SQL** lub poprzez zastosowanie tzw. **wewnętrznego** (nazywanego też pierwotnym lub rodzimym) **dynamicznego SQL'a** (native dynamic SQL).

3.1.9. Kursor

Każde polecenie SQL umieszczone w programie PL/SQL przetwarzane jest w obszarze pamięci zwanym obszarem roboczym lub obszarem kontekstowym. Serwer bazy danych wykorzystuje ten obszar do przechowywania danych wynikowych zapytania oraz do przechowywania dodatkowych informacji dotyczących stanu zapytania czyli tzw. atrybutów. Kursor jest identyfikatorem tego obszaru. Rozróżnia się dwa rodzaje kursorów:

- niejawny (implicit) stosowany automatycznie podczas wykonywania poleceń INSERT, UPDATE, DELETE i SELECT.
- jawny (explicit) stosowany do obsługi zapytań operujących na wielu krotkach (gdy te krotki wymagają "indywidualnego traktowania") i w tzw. pętlach z kursorem. Kursor taki jest jawnie definiowany i obsługiwany przez programistę.

Kursory jawne zostaną przedstawione w dalszej części wykładu.

Kursor niejawny posiada następujące atrybuty:

Atrybut	Opis
SQL%ROWCOUNT	Określa liczbę krotek przetworzonych przez polecenie SQL.
SQL%FOUND	Przyjmuje wartość TRUE jeśli polecenie przetworzyło przynajmniej jedną krotkę, FALSE w przeciwnym wypadku.
SQL%NOTFOUND	Przyjmuje wartość TRUE jeśli żadna krotka nie została przetworzona, FALSE w przeciwnym wypadku.
SQL%ISOPEN	Przyjmuje wartość TRUE jeśli kursor jest otwarty, FALSE w przeciwnym wypadku. Kursor niejawny jest automatycznie zamykany, stąd atrybut ten ma zawsze wartość FALSE.

Od wersji Oracle 8i istnieje, dla kursora niejawnego, dodatkowy atrybut SQL%BULK_ROWCOUNT a od Oracle 9i atrybut SQL %BULK_EXCEPTION, oba związane z tzw. pierwotnym wiązaniem masowym (inaczej masowym SQL - zagadnienie to zostanie przedstawione w dalszej części wykładu). Atrybuty kursorów mogą być wykorzystywane w poleceniach PL/SQL jednak nie w samych poleceniach SQL.

***Zad**. Zmodyfikować relację `Kocury` tak, aby każdy kot o przydziale myszy extra większym niż 20 uzyskał w ramach tego przydziału dodatkową jedną mysz. Wyświetlić liczbę zmodyfikowanych krotek.*

```
SQL> DECLARE
  2   liczba_pop NUMBER;
  3  BEGIN
  4   UPDATE Kocury SET myszy_extra=myszy_extra+1
  5   WHERE myszy_extra>20;
  6   liczba_pop:=SQL%ROWCOUNT;
  7   DBMS_OUTPUT.PUT_LINE('Poprawiono krotek: '||liczba_pop);
  8  END;
  9.
SQL> /
Poprawiono krotek: 6

Procedura PL/SQL została zakończona pomyślnie.
SQL> ROLLBACK;
```

Należy pamiętać że, **blok PL/SQL jest traktowany jako jednostka transakcji**, stąd nawet po ustawieniu parametru AUTOCOMMIT na ON rozkazy DML z bloku zostaną zatwierdzone dopiero po jego zakończeniu (chyba, że zatwierdzenia wystąpią w bloku). Podobnie w przypadku wystąpienia nie obsłużonego błędu blok kończy się niepowodzeniem a rozkazy DML z bloku są wycofane.

3.1.9. Polecenie SELECT

Polecenie SELECT umieszczone w bloku PL/SQL może zwrócić tylko jedną krotkę. Brak zwróconych krotek skutkuje wyjątkiem NO_DATA_FOUND a liczba zwróconych krotek większa niż 1 wyjątkiem TOO_MANY_ROWS. W poleceniu takim dopuszczalne są następujące klauzule:

SELECT
INTO
FROM
[**WHERE**]
[**GROUP BY**]
[**HAVING**]
[**ORDER BY**]
[**FOR UPDATE** [**OF** {atrybut [, ...]}]] [**NOWAIT** | **WAIT** n]

Po nieistniejącej w SQL klauzuli INTO występuje lista zmiennych, do których przypisywane są wartości wybrane w klauzuli SELECT. Gdy wystąpi klauzula FOR UPDATE (blokada krotek/atrybutów do poprawy) zabronione jest stosowania klauzul GROUP BY i HAVING oraz kwalifikatora DISTINCT. Klauzula WAIT określająca maksymalny czas oczekiwania polecenia na dostęp do krotki/atrybutów (n oznacza liczbę sekund) dostępna jest od Oracle 9i.

***Zad**. Obliczyć, jaki procent kotów otrzymuje dodatkowy przydział myszy.*

Zawartość bloku zapisanego w pliku `dodatki.sql`:

```
DECLARE
  l_kotow NUMBER;
  l_z_dodatkami NUMBER;
BEGIN
  SELECT COUNT(*) INTO l_kotow
  FROM Kocury;
  SELECT COUNT(myszy_extra) INTO l_z_dodatkami
  FROM Kocury;
```

```
DBMS_OUTPUT.PUT_LINE('*'||LPAD('*',54,'*')||'*');
  DBMS_OUTPUT.PUT_LINE('*'||LPAD(' ',54,' ')||'*');
  DBMS_OUTPUT.PUT_LINE('*  W stadku '||
                   ROUND(l_z_dodatkami/l_kotow*100,2)||
                   '% kotow ma dodatkowy przydzial myszy  *');
  DBMS_OUTPUT.PUT_LINE('*'||LPAD(' ',54,' ')||'*');
  DBMS_OUTPUT.PUT_LINE('*'||LPAD('*',54,'*')||'*');
EXCEPTION
  WHEN ZERO_DIVIDE
  THEN DBMS_OUTPUT.PUT_LINE('Brak kotow w stadzie!!!');
  WHEN OTHERS
  THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

Uruchomienie w środowisku SQL*Plus bloku zapisanego w pliku.

```
SQL> START C:\dodatki
******************************************************
*                                                    *
*  W stadku 38,89% kotow ma dodatkowy przydzial myszy  *
*                                                    *
******************************************************

Procedura PL/SQL została zakończona pomyślnie.

SQL>
```

3.2. Wyjątki

Wykonanie bloku kończone jest zawsze gdy wystąpi sytuacja wyjątkowa. Istnieją dwie klasy wyjątków:

- predefiniowane - zdefiniowane przez konstruktora systemu, posiadające swoje numery (kody),
- zdefiniowane przez użytkownika.

Poniżej przedstawiono wybrane wyjątki predefiniowane.

Nr	Nazwa	Opis
ORA-0001	DUP_VAL_ON_INDEX	Naruszenie ograniczenia unikalności.
ORA-0051	TIMEOUT_ON_RESOURCE	Upłynął czas na przydział zasobu.

ORA-0061	TRANSACTION_BACKED_OUT	Transakcja wycofana z powodu zakleszczenia.
ORA-1001	INVALID_CURSOR	Nieprawidłowa operacja na kursorze.
ORA-1012	NOT_LOGGED_ON	Brak połączenia z bazą danych.
ORA-1017	LOGIN_DENIED	Nieuprawniony użytkownik lub hasło.
ORA-1403	NO_DATA_FOUND	Nie odnaleziono danych.
ORA-1422	TOO_MANY_ROWS	SELECT INTO zwraca więcej niż jedną krotkę.
ORA-1476	ZERO_DIVIDE	Dzielenie przez zero.
ORA-1722	INVALID_NUMBER	Błąd SQL w konwersji do wartości liczbowej.
ORA-6500	STORAGE_ERROR	Błąd pamięci lub brak pamięci.
ORA-6501	PROGRAM_ERROR	Błędne działanie programu PL/SQL.
ORA-6502	VALUE_ERROR	Błąd PL/SQL obcięcia lub konwersji.
ORA-6511	CURSOR_ALREADY_OPEN	Próba otwarcia kursora już otwartego.
	OTHERS	Inny wyjątek - obsługiwany jako ostatni.

3.2.1. Obsługa wyjątków

Wyjątki predefiniowane i zdefiniowane przez użytkownika mogą być obsłużone (obsługa ta nie jest obowiązkowa) w sekcji EXCEPTION bloku za pomocą polecenia:

WHEN identyfikator_wyjątku **THEN** działanie;

Nazwy wyjątków po klauzuli WHEN można łączyć operatorami logicznymi. Aby obsłużyć w sekcji EXCEPTION wyjątek jawnie zdefiniowany przez użytkownika, należy go zadeklarować w sekcji DECLARE bloku zgodnie ze składnią:

identyfikator_wyjątku **EXCEPTION**;

Wyjątek jawnie definiowany wywoływany jest za pomocą polecenia:

RAISE identyfikator_wyjątku;

Brak obsługi w sekcji EXCEPTION tak wywołanego wyjątku powoduje pojawienie się błędu ORA-06510 (obsłużonego lub nie).

Zad. *Dla kotów pełniących podaną z klawiatury funkcję zmienić przydział myszy na wartość podaną z klawiatury. Obsłużyć wszystkie wyjątki.*

```
SQL> DECLARE
  2    maxm Funkcje.max_myszy%TYPE;
  3    minm Funkcje.min_myszy%TYPE;
  4    p1   Funkcje.funkcja%TYPE:='&funkcja';
  5    p2   Kocury.przydzial_myszy%TYPE:=&nowy_przydzial;
  6    za_malo_lub_za_duzo EXCEPTION;
  7  BEGIN
  8    SELECT max_myszy,min_myszy INTO maxm,minm FROM Funkcje
  9    WHERE funkcja=p1;
 10    IF p2 BETWEEN minm AND maxm
 11       THEN UPDATE Kocury SET przydzial_myszy=p2
 12            WHERE funkcja=p1;
 13       ELSE RAISE za_malo_lub_za_duzo;
 14    END IF;
 15  EXCEPTION
 16    WHEN NO_DATA_FOUND
 17         THEN DBMS_OUTPUT.PUT_LINE('Bledna funkcja!!!');
 18    WHEN za_malo_lub_za_duzo
 19         THEN DBMS_OUTPUT.PUT_LINE('Poza widelkami!!!');
 20    WHEN OTHERS
 21         THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
 22  END;
 23  .
SQL> /
Podaj wartość dla funkcja: LAZIK
stare   4:   p1   Funkcje.funkcja%TYPE:='&funkcja';
nowe   4:   p1   Funkcje.funkcja%TYPE:='LAZIK';
Podaj wartość dla nowy_przydzial: 50
stare   5:   p2  Kocury.przydzial_myszy%TYPE:=&nowy_przydzial;
nowe   5:   p2   Kocury.przydzial_myszy%TYPE:=50;
Bledna funkcja!!!

Procedura PL/SQL została zakończona pomyślnie.

SQL>
```

Do obsługi wyjątków można wykorzystać komunikaty systemowe o błędach zwracane przez funkcję SQLERRM. Jeśli naruszone zostanie jakieś ograniczenie zdefiniowane w ramach polecenia CREATE TABLE, nazwa ograniczenia będzie wchodziła w skład komunikatu systemowego zwracanego przez funkcję SQLERRM. Przykładowo, przy założeniu, że podczas tworzenia relacji `Funkcje` ograniczenie `CHECK(max_myszy<200)` nazwane zostało `ile_moglby_zjesc`, w przypadku realizacji operacji DML ustawiającej wartość `max_myszy` powyżej 199 wyjątek OTHERS mógłby być obsłużony w następujący sposób:

```
...
EXCEPTION
 WHEN OTHERS
      THEN IF UPPER(SQLERRM) LIKE '%ILE_MOGLBY_ZJESC%'
              THEN DBMS_OUTPUT.PUT_LINE('Przydzial>=200!!! ');
              ELSE DBMS_OUTPUT.PUT_LINE('Blad: '||SQLERRM);
           END IF;
END;
```

Innym sposobem obsługi błędów jest zastosowanie wprowadzonej od Oracle 7 funkcji RAISE_APPLICATION_ERROR, umożliwiającej tworzenie własnych komunikatów o błędzie. Składnia wywołania tej funkcji jest następująca:

RAISE_APPLICATION_ERROR(nr_wyjątku, komunikat, [dołącz]);

Funkcja, w przeciwieństwie do wyjątku obsługiwanego w sekcji EXCEPTION, zatrzymuje działanie bloku, wycofuje wszystkie zmiany oraz wyświetla określony przez programistę nr wyjątku i komunikat. Nr wyjątku jest parametrem o wartości od -20000 do -20999. Opcjonalny logiczny parametr dołącz (domyślnie FALSE) decyduje o tym, czy błąd zostanie dołączony do listy wcześniejszych błędów, czy też zastąpi tą listę. Blok z poprzedniego zadania, przy zastosowaniu funkcji RAISE_APPLICATION_ERROR, ma następujący kształt:

```
SQL> SET VERIFY OFF
SQL> DECLARE
  2    maxm Funkcje.max_myszy%TYPE;
  3    minm Funkcje.min_myszy%TYPE;
  4     p1   Funkcje.funkcja%TYPE:='&funkcja';
  5     p2   Kocury.przydzial_myszy%TYPE:=&nowy_przydzial;
  6  BEGIN
  7    SELECT max_myszy,min_myszy INTO maxm,minm FROM Funkcje
  8    WHERE funkcja=p1;
  9    IF p2 BETWEEN minm AND maxm
 10       THEN UPDATE Kocury SET przydzial_myszy=p2
 11            WHERE funkcja=p1;
 12       ELSE
 13       RAISE_APPLICATION_ERROR(-20001,'Poza widelkami!!!');
 14    END IF;
 15  EXCEPTION
 16    WHEN NO_DATA_FOUND THEN
 17      RAISE_APPLICATION_ERROR(-20002,'Bledna funkcja!!!');
 18    WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
 19  END;
 20  .
SQL> /
Proszę podać wartość dla funkcja: SZEFUNIO
Proszę podać wartość dla nowy_przydzial: 250
DECLARE
*
BŁĄD w linii 1:
ORA-20001: Poza widelkami!!!
ORA-06512: przy linia 13
SQL> SET VERIFY ON
SQL>
```

Zmienna środowiskowa VERIFY SQL*Plus'a ustawiona na OFF powoduje brak wyświetlenia komunikatu potwierdzającego wartości zmiennych wprowadzanych z klawiatury.

3.2.2. Dyrektywy PRAGMA

PL/SQL posiada pewną liczbę dyrektyw pełniących funkcję instrukcji dla kompilatora (koncepcja wspólna z ADĄ). Wykorzystanie tych dyrektyw anonsowane jest w sekcji deklaracji bloku według składni:

PRAGMA nazwa_dyrektywy;

Kolejne dyrektywy kompilatora będą omawiane wraz z ich pojawianiem się w trakcie wykładu.

3.2,3. Dyrektywa EXCEPTION_INIT

W Oracle istnieje możliwość nadawania własnych nazw predefiniowanym wyjątkom. Służy to tego dyrektywa kompilatora EXCEPTION_INIT. Składnia tej dyrektywy jest następująca:

PRAGMA EXCEPTION_INIT(nazwa_wyjątku, numer_wyjątku);

gdzie nazwa_wyjątku oznacza własną nazwę dla predefiniowanego wyjątku Oracle a numer_wyjątku kod wyjątku zwracany przez funkcję SQLCODE. W poniższym przykładzie zmieniona zostanie nazwa wyjątku z predefiniowanej DUP_VAL_ON_INDEX (standardowo opisywanego jako: *ORA-00001: naruszono więzy unikatowe*) na własną `wyj`.

```
SQL> DECLARE
  2    wyj EXCEPTION;
  3    PRAGMA EXCEPTION_INIT(wyj,-1);
  4  BEGIN
  5    INSERT INTO Bandy
  6    VALUES (5,'NOWORYSZE','LASEK');
  7  EXCEPTION
  8    WHEN wyj THEN
  9      DBMS_OUTPUT.PUT_LINE
 10         ('Nr bandy wpisany przez Ciebie juz istnieje!!!');
 11      DBMS_OUTPUT.PUT_LINE
 12         ('Zastosuj sekwencję Numery_band!!!');
 13    WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
 14  END;
 15  /
Nr bandy wpisany przez Ciebie juz istnieje!!!
Zastosuj sekwencję Numery_band!!!

Procedura PL/SQL została zakończona pomyślnie.

SQL>
```

3.2.3. Rozprzestrzenianie się wyjątków

Jeśli w ramach bloku wewnętrznego pojawił się wyjątek, którego ten blok nie obsługuje, następuje rozprzestrzenienie wyjątku. Oznacza to, że wyjątek jest przekazywany przez kolejne bloki nadrzędne aż do napotkania obsługi. Jeśli taka obsługa nie istnieje w żadnym z bloków, wówczas wyjątek przechodzi do środowiska zewnętrznego.

```
BEGIN
...
  BEGIN
  ...
    -- sytuacja wyjątkowa A
   ...
  EXCEPTION
    -- brak obsługi wyjątku A
  END;
...
EXCEPTION
  -- tutaj obsługa wyjątku A
END;
```

3.2.4. Uwagi dotyczące wyjątków

W obsłudze wyjątków przydatne mogą być przedstawione poniżej zasady.

1. Wyjątek zdefiniowany przez użytkownika, podobnie jak zmienna, posiada swój zasięg (blok, w którym został zdefiniowany). Jeśli wyjątek rozprzestrzenił się poza obszar swojego zasięgu, nie można odwoływać się do niego przez nazwę. Rozwiązaniem tego problemu jest zdefiniowanie wyjątku w pakiecie (pakiety zostaną omówione w najbliższej przyszłości). Wtedy jego nazwa będzie zawsze dostępna (pakiet jest obiektem składowanym w bazie danych).
2. Dla wszystkich poleceń bloku sprawdzana jest jedna sekcja obsługi wyjątków. Powoduje to, w przypadku użycia kilku poleceń SQL tego samego typu (np. kilku poleceń SELECT...INTO..), trudności w ustaleniu instrukcji powodującej wystąpienie błędu. Rozwiązaniem problemu jest umieszczenie każdego takiego polecenia w bloku zagnieżdżonym (blok taki posiada swoją sekcję obsługi wyjątków) lub oznaczenie każdego takiego polecenia unikalną wartością znacznika, który może być wykorzystany przy obsłudze związanego z poleceniem wyjątku.
3. Dobrym zwyczajem jest unikanie nieobsługiwanych wyjątków. Osiąga się to zwykle poprzez uwzględnienie klauzuli OTHERS w sekcji obsługi wyjątków.

3.3. Instrukcje

W bloku (programie) PL/SQL oprócz poleceń SQL mogą wystąpić struktury sterowania takie jak instrukcje warunkowe i instrukcje pętli. Poniżej przedstawiono ich składnię.

3.3.1. Instrukcja warunkowa IF

Składnia instrukcja warunkowej IF ma kształt:

IF warunek **THEN** {polecenie; [...]}
[**ELSIF** warunek **THEN** {polecenie; [...]}] |
[**ELSE** {polecenie; [...]}
END IF;

Wartość TRUE warunku powoduje wykonanie poleceń po THEN, wartość FALSE lub NULL pominięcie poleceń. Polecenie oznacza instrukcję PL/SQL lub SQL. Jeśli w klauzuli ELSE ma wystąpić jedynie instrukcja IF, wygodniej jest użyć klauzuli ELSIF (brak jest wtedy END IF na końcu każdej zagnieżdżonej instrukcji IF).

3.3.2. Instrukcja warunkowa CASE (od Oracle 9i)

Istnieją dwie formy instrukcji CASE: prosta (simple) i dokładna (searched).

Instrukcja prosta:

[<<etykieta>>] **CASE** selektor
{**WHEN** wyrażenie **THEN** {polecenie; [...]} [...]}
[**ELSE** {polecenie; […]}]
END CASE [etykieta];

Polecenie oznacza instrukcję PL/SQL lub SQL. Selektor jest wyrażeniem dowolnego typu, którego wartość porównywana jest z wartościami wyrażeń po klauzurze WHEN (ich typ musi być zgodny z typem selektora). Wykonywane są polecenia po pierwszej klauzurze WHEN, dla której wartość wyrażenia jest równa wartości selektora.

Jeśli brak jest wyrażenia o wartości równej wartości selektora, wykonywane są polecenia w klauzurze ELSE (jeśli pominięto klauzulę ELSE zgłaszany jest błąd `ORA-6592 CASE_NOT_FOUND`).

Instrukcja dokładna:

CASE
{**WHEN** warunek **THEN** {polecenie; [...]} [...]}
[**ELSE** {polecenie; […]}]
END CASE;

W instrukcji dokładnej wykonywane są polecenia z pierwszej klauzuli WHEN, dla której warunek ma wartość TRUE. Dla tej formy instrukcji warunkowej CASE wymagane jest, w przypadku braku klauzuli ELSE, użycie przynajmniej dwóch klauzul WHEN.

Podobnie jak dla instrukcji prostej, brak możliwości wykonania jakiegokolwiek polecenia (np. przy pominiętym ELSE) powoduje powstanie wyjątku.

3.3.3 Wyrażenie CASE

W PL/SQL słowo kluczowe CASE oprócz roli instrukcji może pełnić także rolę funkcji. Poniżej pokazano ilustrujący to fragment kodu.

```
...
ps:='&pseudonim';
SELECT plec INTO pl
FROM Kocury
WHERE pseudo=ps;
sex:=CASE pl
       WHEN 'M' THEN 'Kocur'
       WHEN 'D' THEN 'Kotka'
       ELSE 'Plec nieznana'
     END;   -- uwaga!!! END zamiast END CASE
...
```

Zmienna `sex` przyjmuje tu wartość `'Kocor'`, `'Kotka'` lub `'Plec nieznana'`.

3.3.4. Instrukcje pętli

Najprostszym typem pętli jest tzw. pętla podstawowa (prosta) o składni:

[<<etykieta>>] **LOOP**
{polecenie; [...]}
END LOOP [etykieta];

Polecenie oznacza tutaj instrukcję PL/SQL lub SQL. Wyjście z pętli następuje poprzez wykonanie jednego z poleceń: EXIT, GOTO lub znanego już polecenia RAISE.

EXIT [etykieta_pętli] [**WHEN** warunek];

Jeśli EXIT występuje jako samodzielna instrukcja, wtedy konieczna jest klauzula WHEN z warunkiem wyjścia. Klauzula ta nie jest konieczna gdy EXIT jest jednym z poleceń instrukcji warunkowej IF. W przypadku pętli zagnieżdżonych pętlę, z której następuje wyjście określa jej etykieta.

GOTO etykieta;

Instrukcja powyższa definiuje skok bezwarunkowy do etykiety <<etykieta>> w aktualnym bloku lub bloku zewnętrznym (ale nie do wnętrza innej instrukcji sterującej).

```
...
LOOP
  licznik:=licznik+1;
  ...
  IF licznik=10 THEN EXIT;
  END IF;
  ...
END LOOP;
...
```

```
...
LOOP
  ...
  EXIT WHEN przydzial>maxm;
  ...
END LOOP;
...

...
<<zewn>>LOOP
          ...
          LOOP
            licznik:=licznik+1;
            ...
            EXIT zewn WHEN przydzial>maxm;
            EXIT WHEN licznik=10;
            ...
          END LOOP;
         END LOOP zewn;
...
```

Pierwszy warunek w powyższym przykładzie powoduje wyjście z obu pętli, drugi tylko z pętli wewnętrznej.

Drugim rodzajem pętli w PL/SQL jest pętla FOR. Składnia pętli FOR jest następująca:

FOR zmienna_sterująca **IN** [**REVERSE**] wartość_pocz .. wartość_kon pętla_podstawowa;

Zmienna sterująca typu BINARY_INTEGER lub, od Oracle 9i, PLS_INTEGER jest deklarowana niejawnie i zmienia się co 1 od wartości początkowej do wartości końcowej (co -1 od wartości końcowej do początkowej w przypadku wykorzystania REVERSE). Wartości początkowa i końcowa mogą być wyrażeniami dowolnego typu konwertowalnymi do wartości liczbowej.

Trzecim rodzajem pętli w PL/SQL jest pętla WHILE. Składnia pętli WHILE jest następująca:

WHILE warunek
pętla_podstawowa;

Warunek sprawdzany jest na początku każdej iteracji. Zakończenie pętli następuje gdy warunek przyjmie wartość FALSE lub NULL.

Pętle FOR i WHILE można także zakończyć poprzez wykorzystanie poleceń EXIT, GOTO lub RAISE. Obie te pętle mogą być także opatrzone etykietami (np. w przypadku zagnieżdżenia pętli różnego rodzaju).

3.3.5. Instrukcja NULL

Jeśli kodzie PL/SQL występuje potrzeba wskazania, że nie będzie wykonywana żadna operacja, wtedy należy wykorzystać instrukcję NULL. Instrukcja ta służy tylko jako "wypełniacz". Przykładowo jeśli należy obsłużyć wyjątek bez żadnej akcji, wtedy kod po klauzuli EXCEPTION może mieć postać:

```
...
EXCEPTION
  ...
  WHEN OTHERS THEN NULL;
END;
...
```

3.3.6. Bloki z etykietami

Bloki mogą być oznaczone etykietami (bez bloku najbardziej zewnętrznego - można to jednak ominąć dodając sztuczne zewnętrzne BEGIN i END).

```
BEGIN   -- sztuczne
  <<zewn>>DECLARE
            x NUMBER;
            ...
          BEGIN
            ...
            <<wewn>>DECLARE
                      x NUMBER:=10;
                      ...
                    BEGIN
                      zewn.x:=zewn.x+1;
                      /* użycie etykiety zewn określa
                         odwołanie do zmiennej x z bloku
                         zewnętrznego */
                      ...
                    END wewn;
...
          END zewn;
END;    -- sztuczne
```

Etykieta bloku umieszczona przed zmienną wskazuje, że zmienna pochodzi z bloku oznaczonego tą etykietą. Pozwala to na identyfikację zmiennych o tych samych nazwach, pochodzących z rożnych bloków a dostępnych w ramach jednego bloku.

3.4. Złożone typy danych

W blokach PL/SQL można korzystać z dwóch rodzajów złożonych typów danych. Pierwszym z nich są **rekordy**, drugim **kolekcje**. Wśród kolekcji można wyróżnić trzy ich rodzaje: **tabele indeksowe, tabele zagnieżdżone** (od Oracle 8) oraz **tablice o zmiennym rozmiarze** (od Oracle 8). Od wersji Oracle 9i możliwa jest także budowa kolekcji wielopoziomowych (kolekcji, których elementami są kolekcje). Tabele zagnieżdżone oraz tablice o zmiennym rozmiarze są elementami obiektowego rozszerzenia bazy danych Oracle, stąd zostaną przedstawione w części wykładu dotyczącej tego tematu.

3.4.1. Rekordy

Zmienna rekordowa może być deklarowana na dwa sposoby:
- poprzez użycie pseudoatrybutu %ROWTYPE,
- poprzez wykorzystanie jawnie zdefiniowanego typu rekordowego (TYPE ... IS RECORD ...).

Składnia deklaracji zmiennej rekordowej z użyciem pseudoatrybutu %ROWTYPE jest następująca:

nazwa_zmiennej_rekordowej nazwa_objektu**%ROWTYPE**;

Struktura tak zdefiniowanego rekordu jest zgodna ze strukturą wiersza obiektu (relacja, perspektywa (widok), kursor jawny). Nazwy pól w rekordzie są takie same jak nazwy atrybutów obiektu. Rekord taki może być wypełniony w klauzuli INTO polecenia SELECT lub poprzez przypisanie wartości poszczególnym polom. Dostęp do pola rekordu odbywa się według składni:

nazwa_zmiennej_rekordowej.nazwa_pola

W poniższym przykładzie polecenie SELECT wypełnia zmienną rekordową o strukturze krotki relacji `Kocury` danymi Tygrysa.

```
DECLARE
  kocury_r Kocury%ROWTYPE;
BEGIN
  SELECT * INTO kocury_r
  FROM Kocury
  WHERE pseudo='TYGRYS';
...
END;
```

Jedyną dozwoloną operacją na zmiennej rekordowej jest przypisanie jej wartości innej zmiennej rekordowej tego samego typu.

Typ rekordowy definiowany jest jawnie zgodnie ze składnią:

TYPE nazwa_typu
IS RECORD ({nazwa_pola typ [**NOT NULL**][:=wyrażenie] [, ...]});

Poniżej przedstawiono przykład definicji typu rekordowego i deklaracji zmiennej tego typu.

```
DECLARE
  TYPE o_kocurach IS RECORD(pseudonim VARCHAR2(15),
                            sex VARCHAR2(1) NOT NULL:='M',
                            poluje_od DATE);
  o_kocurach_r o_kocurach;
  ...
END;
```

3.4.2. Tabele indeksowe

Tabele indeksowe są pierwszym rodzajem kolekcji omawianym w ramach niniejszego wykładu. Pozostałe dwa rodzaje (tabele zagnieżdżone i tablice o zmiennym rozmiarze), ze względu na wykorzystanie elementów obiektowych, zostaną przedstawione po omówieniu rozszerzeń obiektowych systemu Oracle.

Typ tabeli indeksowej definiowany jest według składni:

TYPE nazwa_typu **IS TABLE OF** typ_elementu
INDEX BY typ_indeksu;

gdzie typ_indeksu jest typem całkowitym (BINARY_INTEGER lub, od Oracle 9i, PLS_INTEGER) lub typem łańcuchowym (VARCHAR2 z ograniczeniem długości lub LONG) a typ_elementu jest dowolnym typem skalarnym, typem rekordowym (od Oracle 7.3), typem obiektowym (od Oracle 8) lub dowolną kolekcją (od Oracle 9i).

Dostęp do pola tabeli indeksowej uzyskiwany jest według składni:

nazwa_zmiennej(nr_elementu)

Do definiowania typu tabeli indeksowej rekordów można wykorzystać pseudoatrybut %ROWTYPE. Poniżej przedstawiono przykład takiej definicji.

```
TYPE tabela_kotow IS TABLE OF Kocury%ROWTYPE
INDEX BY BINARY_INTEGER;
Koty tabela_kotow;
```

Pod względem składniowym tabela indeksowa traktowana jest jak tablica, jednak faktycznie jest ona podobna do relacji bazy danych (składa się z dwóch kolumn: KEY typu typ_indeksu i VALUE typu elementu tabeli). Tabela ta posiada nieograniczony rozmiar (jedyne ograniczenie to rozmiar typu indeksu) a jej elementy nie muszą być indeksowane sekwencyjnie (indeksem może być dowolne wyrażenie typu typ_indeksu).

Zad. *Zapamiętać w tabelach indeksowych dane kotów o najdłuższym stażu w każdej bandzie (po jednym reprezentancie).*

```
SQL> DECLARE
  2   TYPE rec_da IS RECORD (ps Kocury.pseudo%TYPE,
  3                          da DATE);
  4   TYPE tab_da IS TABLE OF rec_da INDEX BY BINARY_INTEGER;
  5   tab_re tab_da;
  6   i BINARY_INTEGER; lb NUMBER; l NUMBER;
  7  BEGIN
  8   SELECT MIN(nr_bandy),MAX(nr_bandy) INTO l,lb
  9   FROM Kocury;
 10   FOR i IN l..lb
 11   LOOP
 12    BEGIN
 13     SELECT pseudo,w_stadku_od INTO tab_re(i) FROM Kocury
 14     WHERE nr_bandy=i
 15           AND
 16           w_stadku_od=(SELECT MIN(w_stadku_od)
 17                        FROM Kocury
 18                        WHERE nr_bandy=i)
 19           AND ROWNUM=1;
 20     DBMS_OUTPUT.PUT('Banda '||i||' - najdluzej ');
 21     DBMS_OUTPUT.PUT(tab_re(i).ps||' od ');
 22  DBMS_OUTPUT.PUT_LINE(TO_CHAR(tab_re(i).da,'YYYY-MM-DD'));
 23    EXCEPTION
 24     WHEN NO_DATA_FOUND THEN NULL;
 25    END;
 26   END LOOP;
 27   -- dalszy ciąg programu wykorzystujący dane z tabel
 28  EXCEPTION
 29   WHEN NO_DATA_FOUND
 30        THEN DBMS_OUTPUT.PUT_LINE('Brak kotow');
 31   WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
 32  END;
 33  /
```

```
Banda 1 - najdluzej TYGRYS od 2002-01-01
Banda 2 - najdluzej SZYBKA od 2006-07-21
Banda 3 - najdluzej ZOMBI od 2004-03-16
Banda 4 - najdluzej RAFA od 2006-10-15

Procedura PL/SQL została zakończona pomyślnie.

SQL>
```

Od wersji Oracle 7.3 w obsłudze tabeli indeksowej można wykorzystywać dodatkowo jej atrybuty. Są to:

- EXISTS(n) - zwraca TRUE lub FALSE w zależności od tego, czy element o indeksie n istnieje lub nie (unika się błędu `ORA-1403`),
- COUNT - zwraca liczbę wierszy w tabeli,
- FIRST, LAST - zwracają odpowiednio wartość indeksu dla pierwszego i ostatniego wiersza tabeli,
- PRIOR(n), NEXT(n) - dla wiersza tabeli o indeksie n zwraca odpowiednio wartość indeksu poprzedniego i następnego wiersza,
- DELETE, DELETE(n), DELETE(m,n) - odpowiednio usuwa wszystkie wiersze tabeli, usuwa wiersz o indeksie n, usuwa wiersze o indeksach z przedziału od m do n.

Atrybuty wykorzystuje się zgodnie ze składnią:

nazwa_tabeli.atrybut

3.5. Kursor jawny

Kursor jawny umożliwia formułowanie zapytań w PL/SQL skierowanych do wielu wierszy oraz ich ewentualną obsługę. Jest on przetwarzany w następujących kolejnych krokach:

1. Definicja kursora.
2. Otwarcie kursora.
3. Pobranie wartości z bieżącego wiersza (wierszy) kursora.
4. Zamkniecie kursora.

Kursor jawny definiowany jest w sekcji DECLARE według składni:

CURSOR nazwa_kursora [({parametr typ [, ...]})] **IS**
polecenie_SELECT;

Polecenie SELECT nie zawiera klauzuli INTO. Parametry kursora pełnią rolę parametrów formalnych i są (jeśli wystąpią) wykorzystywane w poleceniu SELECT.

Kursor jawny otwierany jest poleceniem OPEN według składni:

OPEN nazwa_kursora [({argument [, ...]})];

Argumenty otwarcia kursora (jeśli wystąpią) pełnią rolę parametrów aktualnych. Odpowiadają one, kolejno, parametrom formalnym w definicji kursora. Parametrem kursora nie może być nazwa relacji ani perspektywy (widoku). Po otwarciu kursor wskazuje pierwszy wiersz relacji zwracanej przez polecenie SELECT kursora.

Wartość bieżącego wiersza relacji zwracanej przez polecenie SELECT kursora (bezpośrednio po otwarciu jest to pierwszy wiersz) jest pobierana przez polecenie FETCH. Polecenie to posiada składnię:

FETCH nazwa_kursora
INTO {zmienna [, ...]} | zmienna_rekordowa;

Wartości atrybutów pobranego wiersza wstawiane są do listy zmiennych (o typach zgodnym z typami kolejnych wyrażeń klauzuli SELECT kursora) lub do zmiennej rekordowej (o typie nazwa_kursora%ROWTYPE). Pobranie takie powoduje przejście do następnego wiersza kursora. Pierwsze polecenie FETCH bez wybranego wiersza (wszystkie już wybrano) nie spowoduje błędu a jedynie docelowe zmienne lub pola rekordu uzyskają wartość NULL. Od Oracle 8i istnieje możliwość jednorazowego pobrania zawartości całego kursora do kolekcji za pomocą polecenia BULK COLLECT. Jest to element tzw. pierwotnego wiązania masowego – temat ten zostanie omówiony w dalszej części wykładu.

Kursor zamykany jest poleceniem CLOSE według składni:

CLOSE nazwa_kursora;

Podobnie jak kursory niejawne, kursory jawne posiadają atrybuty %FOUND, %NOTFOUND, %ROWCOUNT, %ISOPEN, w tym przypadku poprzedzone przez nazwę kursora. Oprócz zwrócenia standardowych wartości TRUE, FALSE i NULL pobranie atrybutu może spowodować powstanie wyjątku `ORA-1001 INVALID_CURSOR`.

Atrybut	Opis
%ISOPEN	Zwraca TRUE jeśli kursor jest otwarty, FALSE w przeciwnym wypadku.
%FOUND	Zwraca TRUE jeśli wiersz został pomyślnie pobrany, FALSE gdy żaden wiersz nie został zwrócony. Przed pierwszym pobraniem wiersza zwracana jest wartość NULL. W przypadku gdy kursor nie jest jeszcze otwarty lub został już zamknięty, zwracany jest wyjątek INVALID_CURSOR.
%NOTFOUND	Zwraca TRUE jeśli żaden wiersz nie został zwrócony, FALSE gdy wiersz został pomyślnie pobrany. Przed pierwszym pobraniem wiersza zwracana jest wartość NULL. W przypadku gdy kursor nie jest jeszcze otwarty lub został już zamknięty, zwracany jest wyjątek INVALID_CURSOR.
%ROWCOUNT	Zwraca liczbę do tej pory pobranych wierszy kursora. W przypadku gdy kursor nie jest jeszcze otwarty lub został już zamknięty, zwracany jest wyjątek INVALID_CURSOR.

Pierwszy brak zwracanego wiersza spowoduje przyjęcie przez atrybut %NOTFOUND wartości TRUE a kolejne niepomyślne wywołanie błąd ORA-1002.

Zad. *Znaleźć sumaryczną liczbę myszy spożywanych miesięczne przez koty o indywidualnych przydziałach większych od średniej w stadzie.*

```
SQL> DECLARE
  2   CURSOR ponadsr IS
  3   SELECT NVL(przydzial_myszy,0) pm,
  4          NVL(myszy_extra,0) me
  5   FROM Kocury
  6   WHERE NVL(przydzial_myszy,0)+NVL(myszy_extra,0)>=
  7         (SELECT AVG(NVL(przydzial_myszy,0)+
  8                     NVL(myszy_extra,0))
  9          FROM Kocury);
 10   sp NUMBER(4):=0; se NUMBER(4):=0; pr ponadsr%ROWTYPE;
 11   sa_wiersze BOOLEAN:=FALSE;
 12  BEGIN
 13   OPEN ponadsr;
 14   LOOP
 15    FETCH ponadsr INTO pr; EXIT WHEN ponadsr%NOTFOUND;
 16    IF NOT sa_wiersze THEN sa_wiersze:=TRUE; END IF;
 17    sp:=sp+pr.pm; se:=se+pr.me;
 18   END LOOP; CLOSE ponadsr;
 19   IF sa_wiersze
 20   THEN
 21    DBMS_OUTPUT.PUT('Miesiecznie spozycie: ');
 22    DBMS_OUTPUT.PUT(TO_CHAR(sp+se,999));
 23    DBMS_OUTPUT.PUT(' (w tym dodatki: ');
 24    DBMS_OUTPUT.PUT_LINE(TO_CHAR(se,999)||')');
 25   ELSE
 26    DBMS_OUTPUT.PUT_LINE('Brak kotow!!!');
 27   END IF;
 28  END;
 29  /
Miesiecznie spozycie:  650 (w tym dodatki:  156)

Procedura PL/SQL została zakończona pomyślnie.

SQL>
```

Wykorzystanie w poleceniu SELECT kursora klauzuli FOR UPDATE spowoduje blokadę na wyłączność (blokadę znosi zamknięcie kursora - nie działa do tego momentu polecenie COMMIT) krotek do poprawy (polecenie UPDATE) lub do usunięcia (polecenie DELETE) w relacji modyfikowanej, wskazywanych przez kursor. Jeśli po FOR UPDATE wystąpi słowo kluczowe OF z listą atrybutów, spowoduje to zawężenie blokady tylko do tych atrybutów. Do krotek modyfikowanej relacji, wskazywanych aktualnie przez kursor, można się odnieść poprzez, umieszczony po klauzurze WHERE polecenia UPDATE lub DELETE, dostępny tylko w PL/SQL, warunek:

CURRENT OF nazwa_kursora

Zad. *Zapewnić obsadę bandy nr 5 ('ROCKERSI') przydzielając do niej koty z najmniejszym przydziałem myszy w swojej aktualnej bandzie. Na szefa bandy oddelegować kotkę o pseudonimie 'LOLA' oraz zrealizować jej postulat, aby w nowej bandzie nie było kotów pełniących funkcję 'MILUSIA'.*

```
SQL> DECLARE
  2    CURSOR do_zm IS
  3    SELECT pseudo FROM Kocury
  4    WHERE (((przydzial_myszy,nr_bandy) IN
  5           (SELECT MIN(NVL(przydzial_myszy,0)),nr_bandy
  6            FROM Kocury
  7            GROUP BY nr_bandy)) AND funkcja<>'MILUSIA')
  8          OR
  9          pseudo='LOLA'
 10    FOR UPDATE OF nr_bandy;
 11    re do_zm%ROWTYPE; sa_wiersze BOOLEAN:=FALSE;
 12    brak_kotow EXCEPTION;
 13  BEGIN
 14    OPEN do_zm;
 15    LOOP
 16      FETCH do_zm INTO re;
 17      EXIT WHEN do_zm%NOTFOUND;
 18      IF NOT sa_wiersze THEN sa_wiersze:=TRUE;
 19      END IF;
 20      UPDATE Kocury
 21      SET nr_bandy=5
 22      WHERE CURRENT OF do_zm;
 23    END LOOP;
 24    CLOSE do_zm;
```

```
 25    IF NOT sa_wiersze
 26       THEN RAISE brak_kota;
 27    END IF;
 28    UPDATE Bandy
 29    SET nazwa='LOLERSI',
 30        szef_bandy='LOLA'
 31    WHERE nr_bandy=5;
 32    -- COMMIT;
 33  EXCEPTION
 34    WHEN brak_kota THEN DBMS_OUTPUT.PUT_LINE('Brak kota');
 35    WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
 36  END;
 37  /

Procedura PL/SQL została zakończona pomyślnie.

SQL> SELECT pseudo,nazwa
  2  FROM Kocury K,Bandy B
  3  WHERE K.nr_bandy=B.nr_bandy AND K.nr_bandy=5;

PSEUDO          NAZWA
--------------- --------------------
LOLA            LOLERSI
UCHO            LOLERSI
MALY            LOLERSI

SQL>ROLLBACK;

Wycofywanie zostało zakończone.

SQL>
```

Jak wspomniano wcześniej, kursor może posiadać parametry. Poniżej przedstawiono fragment kodu z przykładem takiego kursora.

```
...
CURSOR wybor (par1 NUMBER,par2 VARCHAR2) IS
SELECT imie,przydzial_myszy,w_stadku_od
FROM Kocury
WHERE nr_bandy=par1 AND funkcja=par2;
...
nrb:=1;fu:='MILUSIA';
...
OPEN wybor(nrb,fu);
...
```

Powyżej zdefiniowany i otwarty kursor zwraca dane kotów należących do bandy nr 1 pełniących funkcję `'MILUSIA'`.

3.5.1. Pętla FOR z kursorem

Przetwarzanie kursora można uprościć poprzez zastosowanie tzw. pętli FOR z kursorem. Pętla taka posiada następującą składnię:

FOR zmienna_rekordowa **IN** nazwa_kursora[({parametr typ [, ...]})]
pętla_podstawowa;

Liczba obiegów w pętli jest równa liczbie wierszy zwracanych przez kursor. Zmienna rekordowa jest deklarowana niejawnie jako nazwa_kursora%ROWTYPE. Kursor otwierany jest niejawnie w momencie zainicjowania pętli. Niejawne jest także pobieranie wierszy do zmiennej rekordowej i zamknięcie kursora (także jeśli wyjście nastąpi poprzez polecanie EXIT, GOTO lub RAISE). Poniżej przedstawiono fragment kodu z przykładem takiej pętli.

```
...
FOR nr IN wybor(nrb,fu)
LOOP
  s:=s+nr.przydzial_myszy   -- operacja na polu kursora!
END LOOP;
...
```

Powyższa pętla obsługuje kursor `wybor`. Wyznaczana jest w niej suma przydziałów myszy kotów należących do bandy o numerze `nrb` i pełniących funkcję `fu`. Należy tu zwrócić uwagę na fakt, że w przypadku takiej obsługi kursora pola zmiennej rekordowej (w powyższym przykładzie jest to zmienna `nr`) posiadają nazwy zgodne z nazwami pól kursora określonych w ramach jego definicji (klauzula SELECT polecenia SELECT kursora).

Zamiast w sekcji DECLARE bloku, kursor może być także definiowany bezpośrednio w pętli FOR z kursorem. Obsługa takiego kursora odbywa się według następującej składni:

FOR zmienna_rekordowa **IN** (polecenie_SELECT)
pętla_podstawowa;

Ciało (treść) kursora definiowane jest wtedy przez polecenie SELECT po słowie kluczowym IN. Jest to także kursor jawny, nie posiada on jednak nazwy (nie został zdefiniowany w sekcji DECLARE) więc brak jest dostępu do jego atrybutów a także nie może być on parametryzowany.

Zad. *Zastosować pętlę FOR z kursorem do znalezienia kotów o najdłuższym stażu w swojej bandzie.*

```
SQL> DECLARE
  2   sa_wiersze BOOLEAN:=FALSE;
  3   brak_kotow EXCEPTION;
  4  BEGIN
  5   FOR re IN (SELECT pseudo,w_stadku_od,nr_bandy
  6              FROM Kocury
  7              WHERE (w_stadku_od,nr_bandy) IN
  8                          (SELECT MIN(w_stadku_od),nr_bandy
  9                           FROM Kocury
 10                           GROUP BY nr_bandy))
 11   LOOP
 12    sa_wiersze:=TRUE;
 13    DBMS_OUTPUT.PUT
 14             ('Banda '||re.nr_bandy||' - najdluzej ');
 15    DBMS_OUTPUT.PUT(re.pseudo||' od ');
 16    DBMS_OUTPUT.PUT_LINE
 17               (TO_CHAR(re.w_stadku_od,'YYYY-MM-DD'));
 18   END LOOP;
 19   IF NOT sa_wiersze
 20      THEN RAISE brak_kotow;
 21   END IF;
 22  EXCEPTION
 23    WHEN brak_kotow THEN DBMS_OUTPUT.PUT_LINE
 24                                        ('Brak kotow');
 25    WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
 26  END;
 27  /
Banda 1 - najdluzej TYGRYS od 2002-01-01
Banda 2 - najdluzej SZYBKA od 2006-07-21
Banda 4 - najdluzej RAFA od 2006-10-15
Banda 3 - najdluzej ZOMBI od 2004-03-16

Procedura PL/SQL została zakończona pomyślnie.

SQL>
```

Dzięki zastosowaniu kursora jawnego w ramach bloku PL/SQL możliwa była obsługa zapytania SELECT zwracającego więcej niż jedną krotkę.

3.5.2. Zmienne kursora

Zmienna kursora jest typem odwołania czyli wskaźnikiem do obszaru pamięci, w którym przechowywany jest wynik zapytania oraz jego atrybuty. Kursor wykorzystywany do tej pory był odpowiednikiem stałej PL/SQL (kursor statyczny). Składnia służąca do definicji typu zmiennej odwołania jest następująca:

TYPE nazwa_typu_kursora
IS REF CURSOR [**RETURN** typ_rekordu];

Jeśli w ramach powyższej składni występuje klauzula RETURN, definiowany jest typ dla ograniczonej zmiennej kursora (wiersze kursora muszą mieć typ zgodny z rekordem - od Oracle 7.2) w przeciwnym wypadku dla zmiennej nieograniczonej (dostępnej dla każdego kursora - od Oracle 7.3).

Po definicji typu zmiennej kursora należy już tylko taką zmienną zadeklarować.

Podobnie jak kursor statyczny, zmienną kursora należy otworzyć (*de facto* nadać jej wartość). Odbywa się to według składni:

OPEN zmienna_kursora **FOR** polecenie_SELECT;

Polecenie SELECT definiuje kursor wskazywany przez zmienną kursora. Pobieranie wierszy ze zmiennej kursora i zamykanie zmiennej kursora odbywa się w sposób analogiczny do tych operacji dla kursora statycznego.

Poniżej przedstawiono przykład wykorzystania jednej zmiennej kursora dla różnych jej wartości.

```
DECLARE
  TYPE typ_kursora IS REF CURSOR;
  kursor typ_kursora;
  ps Kocury.pseudo%TYPE;
  pm Kocury.przydzial_myszy%TYPE;
  me Kocury.myszy_extra%TYPE;
  iw Wrogowie_kocurow.imie_wroga%TYPE;
  sw Wrogowie.stopien_wrogosci%TYPE;
  kod_relacji VARCHAR2(2):='&1';
BEGIN
  IF kod_relacji='KO'
     THEN OPEN kursor FOR
          SELECT pseudo,przydzial_myszy,myszy_extra
          FROM Kocury
          WHERE przydzial_myszy>50;
  ELSIF kod_relacji='WR'
        THEN OPEN kursor FOR
             SELECT imie_wroga,stopien_wrogosci
             FROM Wrogowie
             WHERE stopien_wrogosci>5;
  ELSE
   RAISE_APPLICATION_ERROR(-20103,'Brak obslugi tej relacji');
  END IF;
  LOOP
    IF kod_relacji='KO' THEN
       FETCH kursor INTO ps,pm,me;
       EXIT WHEN kursor%NOTFOUND;
       ...
    ELSE
       FETCH kursor INTO iw,sw;
       EXIT WHEN kursor%NOTFOUND;
       ...
    END IF;
  END LOOP;
  CLOSE kursor;
END;
```

Istnieją pewne ograniczenia dotyczące wykorzystania zmiennej kursora (dla Oracle 7.3 i wyżej). Przedstawiono je poniżej.

- nie można w ramach zmiennej kursora stosować atrybutu %ROWTYPE,
- kolekcje PL/SQL nie mogą przechowywać zmiennych kursora,
- dopiero od wersji Oracle 8i polecenie SELECT definiujące kursor może zawierać klauzulę FOR UPDATE,
- zmienne kursora nie mogą być deklarowane w pakiecie (można w nim jedynie definiować typ kursorowy).