Rozdział 10
Kursory i wyjątki

Kursory i praca z kursorami, kursory jawne i niejawne, otwieranie kursora, pobieranie z kursora, zamykanie kursora, wyjątki systemowe i użytkownika, zgłaszanie i obsługa wyjątków

Kursor

Każde zapytanie SQL umieszczone w programie PL/SQL może zwrócić zero, jedną bądź wiele krotek. Aby efektywnie przetworzyć krotki zwrócone przez zapytanie korzystamy z kursorów. Kursor jest obiektem związanym z zapytaniem. Programista może:

- Otworzyć kursor (zidentyfikować zbiór wynikowy)
- Pobrać daną do kursora (odczytać kolejną krotkę z wyniku zapytania i wpisać ją do kursora)
- Zamknąć kursor (zwolnić obszar pamięci przydzielony kursorowi)

Kursor to nazwa obszaru roboczego, w którym mieści się wynik zapytania (*result set*). Wewnątrz kursora wyróżniamy bieżący wiersz (*current row*). Kursor może być jawny (*explicit*) lub niejawny (*implicit*).

Deklarowanie kursora

```
DECLARE CURSOR nazwa [ ( lista parametrów) ]
        [ RETURN typ zwracany ] IS zapytanie SQL;
```

```
parametr [ IN ] typ [ { := | DEFAULT } wartość ]
```

- Nazwa kursora nie jest zmienną, lecz identyfikatorem. Do kursora nie można przypisać wartości
- Parametry są widoczne tylko wewnątrz kursora, nie można związać z nimi żadnych ograniczeń

```
DECLARE CURSOR c_pracownicy (zespol NUMBER DEFAULT 10)
IS
        SELECT nazwisko, etat, zatrudniony, placa_pod
        FROM pracownicy
        WHERE id_zesp = zespol;
```


Otwieranie kursora

Otwarcie kursora powoduje wykonanie związanego z nim zapytania i zidentyfikowanie zbioru wynikowego, zawierającego krotki spełniające kryteria wyszukiwania.

```
OPEN nazwa_kursora [ (lista parametrów aktualnych) ]
```

```
DECLARE
  CURSOR c_prac (zespol NUMBER, posada VARCHAR2)
  IS SELECT * FROM pracownicy
     WHERE id_zesp = zespol AND etat = posada;
BEGIN
        ...
        v_etat := 'PROFESOR';
        OPEN c_prac(10, 'ASYSTENT');
        OPEN c_prac(50, v_etat);
```

Pobieranie z kursora

```
FETCH nazwa_kursora INTO lista zmiennych | rekord;
```

Każde wykonanie polecenia FETCH powoduje odczytanie bieżącego wiersza kursora i przesunięcie znacznika kursora na kolejny wiersz. Na liście zmiennych musi się znajdować taka sama liczba zmiennych jak liczba atrybutów w kursorze. Odpowiednie zmienne i atrybuty muszą się zgadzać co do typu.

```
DECLARE
        CURSOR c_prac IS SELECT nazwisko, etat FROM pracownicy;
        v_nazwisko PRACOWNICY.NAZWISKO%TYPE;
        v_etat PRACOWNICY.ETAT%TYPE;
BEGIN
        OPEN c_prac;
        FETCH c_prac INTO v_nazwisko, v_etat;
```


Zamykanie kursora

```
CLOSE nazwa_kursora;
```

Zamknięcie kursora powoduje, że kursor staje się nieaktywny a zbiór wynikowy związany z kursorem staje się niezdefiniowany. Zamknięty kursor można powtórnie otworzyć, np. z innymi parametrami. Każde odwołanie się do zamkniętego (lub jeszcze nie otwartego) kursora powoduje błąd INVALID_CURSOR.

Atrybuty kursora

- %FOUND – wartością atrybutu jest TRUE jeśli ostatnia operacja FETCH odczytała krotkę z kursora. W przeciwnym wypadku (tzn. kiedy odczyt się nie udał) atrybut przyjmuje wartość FALSE. Przed pierwszym odczytem atrybut ma wartość NULL
- %NOTFOUND – wartością atrybutu jest FALSE jeśli ostatnia operacja FETCH odczytała krotkę z kursora. W przeciwnym wypadku (tzn. kiedy odczyt się nie udał) atrybut przyjmuje wartość TRUE. Przed pierwszym odczytem atrybut ma wartość NULL
- %ROWCOUNT – wartością atrybutu jest liczba odczytanych z kursora krotek. Przed pierwszym odczytem atrybut ma wartość 0
- %ISOPEN – wartością atrybutu jest TRUE jeśli kursor jest otwarty i FLASE jeśli kursor jest zamknięty.

Atrybuty kursora niejawnego

- Każde polecenie DML (INSERT, UPDATE, DELETE, SELECT FOR UPDATE) powoduje utworzenie kursora niejawnego (ang. *implicit cursor*). Dla takiego kursora można sprawdzać wartości następujących atrybutów:
  - SQL%ROWCOUNT: liczba wierszy zmodyfikowanych przez polecenie
  - SQL%FOUND: TRUE jeśli ostatnie polecenie zmodyfikowało jakiekolwiek wiersze
  - SQL%NOTFOUND: TRUE jeśli ostatnie polecenie nie zmodyfikowało żadnych wierszy
  - SQL%ISOPEN: zawsze FALSE (kursor niejawny jest zamykany natychmiast po zakończeniu polecenia)

Użycie kursora

```
DECLARE
        CURSOR c_prac (zespol NUMBER DEFAULT 10) IS
        SELECT nazwisko, etat FROM pracownicy
        WHERE id_zesp = zespol ORDER BY placa_pod DESC;
        v_nazwisko PRACOWNICY.NAZWISKO%TYPE;
        v_etat PRACOWNICY.ETAT%TYPE;
BEGIN
        OPEN c_prac(30);
        LOOP
                FETCH c_prac INTO v_nazwisko, v_etat;
                EXIT WHEN c_prac%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE(v_nazwisko || ' , ' || v_etat);
        END LOOP;
        CLOSE c_prac;
END;
```


Pętla FOR z kursorem

```
DECLARE
        CURSOR c (minplaca NUMBER) IS
        SELECT * FROM pracownicy WHERE placa_pod > minplaca;
BEGIN
        FOR c_rec IN c(800) LOOP
                DBMS_OUTPUT.PUT_LINE(c_rec.nazwisko ||
                ' zarabia ' || c_rec.placa_pod || ' i pracuje jako ' || c_rec.etat);
        END LOOP;
END;
```

- Zmienna sterująca pętlą jest deklarowana automatycznie jako zmienna typu *kursor*%ROWTYPE
- Kursor jest otwierany automatycznie
- W każdym przebiegu pętli jedna krotka jest pobierana z kursora i umieszczana w zmiennej sterującej pętlą
- Po pobraniu ostatniej krotki kursor jest automatycznie zamykany

Pętla FOR z podzapytaniem

```
BEGIN
        FOR c_rec IN ( SELECT * FROM pracownicy ) LOOP
                DBMS_OUTPUT.PUT_LINE(c_rec.nazwisko ||
                ' zarabia ' || c_rec.placa_pod || ' i pracuje jako ' || c_rec.etat);
        END LOOP;
END;
```

- Zmienna sterująca pętlą jest deklarowana automatycznie jako zmienna typu *podzapytanie*%ROWTYPE
- Kursor jest otwierany automatycznie
- W każdym przebiegu pętli jedna krotka jest pobierana z kursora i umieszczana w zmiennej sterującej pętlą
- Po pobraniu ostatniej krotki kursor jest automatycznie zamykany
- Kursor oparty na podzapytaniu nie może być parametryzowany ani wykorzystywany wielokrotnie

Klauzula WHERE CURRENT OF

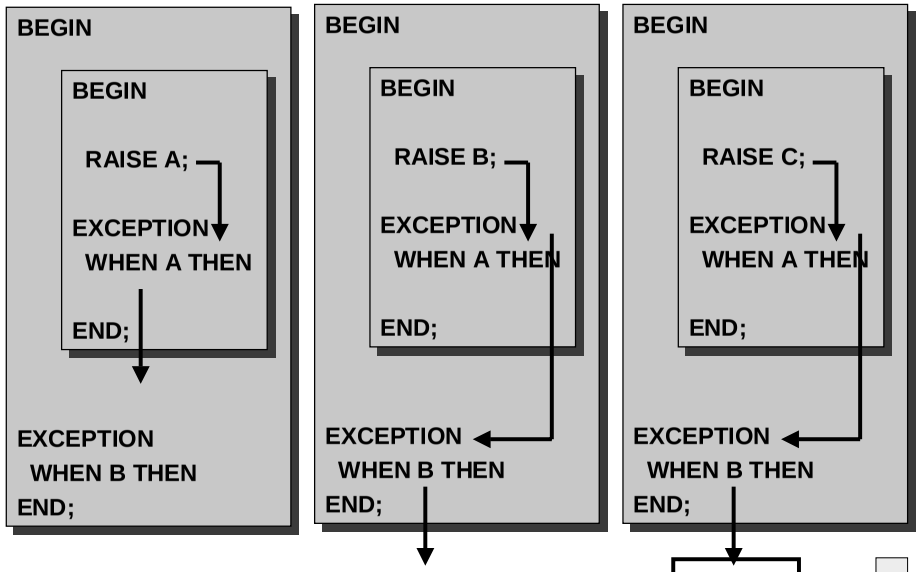
Klauzula WHERE CURRENT OF ma zastosowanie do poleceń UPDATE i DELETE umieszczonych wewnątrz kursora. Warunek jest spełniony tylko i wyłącznie dla bieżącej krotki w kursorze. Aby można było skorzystać z tej klauzuli, zapytanie definiujące kursor musi zawierać klauzulę FOR UPDATE OF (założenie blokady na odczytywanych krotkach)

```
DECLARE
 CURSOR c IS SELECT * FROM pracownicy JOIN zespoly USING (id_zesp)
  FOR UPDATE OF placa_pod;
BEGIN
 FOR c_rec IN c LOOP
  IF (c_rec.adres = 'PIOTROWO 3A') THEN
   UPDATE pracownicy SET placa_pod = 1.1 * placa_pod
   WHERE CURRENT OF c;
  END IF;
 END LOOP;
END;
```

Obsługa wyjątków w PL/SQL

- **Błąd lub ostrzeżenie nazywamy w PL/SQL wyjątkiem (ang. *exception*). Wyjątki mogą być systemowe (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.**
- **Po wystąpieniu wyjątku kontrola przechodzi do procedury obsługi wyjątku (ang. *exception handler*). Po jej wykonaniu kontrola przechodzi do kolejnego bloku nadrzędnego. Jeśli procedura obsługi danego błędu nie zostanie znaleziona, to wykonywanie programu zostanie przerwane.**

Kontrola obsługi wyjątku



Predefiniowane wyjątki systemowe

Exception	Oracle Error	SQLCODE Value
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
STORAGE_ERROR	ORA-06500	-6500
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Definiowanie własnych wyjątków

```
DECLARE
        v_liczba NUMBER := 0;
        ex_moj_wyjatek EXCEPTION;
...
```

Wyjątki mogą być deklarowane w blokach deklaracji dowolnych 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 jakiejkolwiek operacji arytmetycznej.

Wywoływanie własnych wyjątków

Użytkownik może wywoływać ręcznie zarówno błędy systemowe, jak i zdefiniowane przez siebie. Każdy wywołany błąd powinien zostać obsłużony przez odpowiednią procedurę obsługi wyjątku.

```
DECLARE
        v_liczba NUMBER := 0;
        v_zespol NUMBER := &zespol;
        ex_za_malo_pracownikow EXCEPTION;
BEGIN
        SELECT COUNT(*) INTO v_liczba
        FROM pracownicy WHERE id_zesp = v_zespol;
        IF (v_liczba < 3) THEN RAISE ex_za_malo_pracownikow; END IF;
EXCEPTION
        WHEN ex_za_malo_pracownikow THEN
        DBMS_OUTPUT.PUT_LINE('W zespole ' || v_zespol || ' brakuje ludzi!');
END;
```


Funkcje SQLCODE i SQLERRM

- **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`**”**

```
DECLARE
        err_num NUMBER; err_msg VARCHAR2(100);
BEGIN
...
EXCEPTION
        ...
        WHEN OTHERS THEN
                err_num := SQLCODE;
                err_msg := SUBSTR(SQLERRM, 1, 200);
                INSERT INTO errors VALUES (err_num, err_msg);
END;
```


Procedura RAISE_APPLICATION_ERROR

- **Procedura pozwala na przesyłanie komunikatów o błędach zdefiniowanych przez użytkownika do programu nadrzędnego.**

```
RAISE_APPLICATION_ERROR(error_number, message, [ , TRUE | FALSE ] )
```

- ***error_numer*: liczba ujemna z przedziału –20000 ÷ -20999**
- ***message*: łańcuch znaków o rozmiarze do 2048 bajtów**
- **TRUE, FALSE: czy błąd ma być umieszczony na szczycie stosu błędów, czy też ma je zastąpić**

```
BEGIN
 FOR prac_record IN ( SELECT * FROM PRACOWNICY ) LOOP
  IF ( prac_record.placa_pod < 300 ) THEN
   RAISE_APPLICATION_ERROR(-20010, 'Uwaga, za niskie pensje pracowników');
  ELSE
   DBMS_OUTPUT.PUT_LINE('Płaca pracownika: ' || prac_record.placa_pod);
  END IF;
 END LOOP;
END;
```