3.6. Podprogramy

Podprogramy (procedury i funkcje) dostępne od wersji Oracle 7 w przeciwieństwie do bloków anonimowych są blokami posiadającymi nazwę i mogą być składowane w bazie jako obiekty bazy danych. Do podprogramów składowanych w bazie danych można się odwoływać z innych bloków (podprogramów) a dla funkcji dodatkowo (od Oracle 7.3) z poziomu poleceń DML i polecenia SELECT. Funkcje składowane w bazie danych mogą więc rozszerzać możliwości SQL'a.

Blok procedury jest definiowany według składni:

PROCEDURE nazwa_procedury [({parametr [, ...]})]
[**AUTHID {DEFINER** | **CURRENT_USER**}]
{**IS** | **AS**}
  [-- *definicje obiektów PL/SQL dla procedury*]
BEGIN
  -- *część wykonywalna procedury*
[**EXCEPTION**
  -- *obsługa wyjątków*]
END [nazwa_procedury];

Blok funkcji jest definiowany według składni:

FUNCTION nazwa_funkcji [({parametr [, …]})] **RETURN** typ_zwr
[**AUTHID** {**DEFINER** | **CURRENT_USER**}]
{**IS** | **AS**}
  [-- *definicje obiektów PL/SQL dla funkcji*]
BEGIN
  -- *część wykonywalna funkcji z co najmniej jednym*
  -- *poleceniem:* RETURN wyrażenie;
[**EXCEPTION**
  -- *obsługa wyjątków*]
END [nazwa_funkcji];

Polecenie RETURN bez wyrażenia może wystąpić także w części wykonywalnej procedury. Zatrzymuje ono wtedy wykonanie procedury i przekazuje sterowanie do programu wywołującego.

Podprogram tak zdefiniowany może być podprogramem wewnętrznym (sekcja DECLARE) innego bloku (podprogramu) lub elementem pakietu (pakiet to biblioteka zgrupowanych pod jedną nazwą obiektów takich jak typy, procedury, funkcje, zmienne, stałe, kursory i wyjątki).

Klauzula AUTHID określa, z jakimi prawami będzie uruchamiany podprogram (domyślnymi definiującego czy wywołującego). Parametr w obu definicjach ma składnię:

nazwa_parametru [{**IN** | **OUT** [**NOCOPY**] | **IN OUT** [**NOCOPY**]}]
typ [{:=|**DEFAULT**} wartość_początkowa];

Parametry podprogramu, w przypadku ich wystąpienia, pełnią rolę parametrów formalnych. Typ parametru formalnego (definiowany także za pomocą atrybutu %TYPE) oraz typ zwracany w definicji funkcji oznacza dowolny typ predefiniowany lub definiowany (bez długości!). Parametry formalne mogą posiadać jeden z trzech trybów:

- IN - parametr tylko do odczytu, któremu w podprogramie nie można przypisać nowej wartości (jest to tryb domyślny),
- OUT - parametr tylko do zapisu, który w ramach podprogramu nabiera wartości (wartość z wywołania podprogramu jest ignorowana), a którego nie można odczytywać,
- IN OUT - parametr do odczytu i zapisu (cechy trybu IN i OUT).

Parametry w trybie IN przekazywane są przez odwołanie (wskaźnik) a w trybach OUT i IN OUT przez wartość. Modyfikator NOCOPY (od Oracle 8i) powoduje, że kompilator próbuje przekazać parametr przez odwołanie a nie przez wartość. Modyfikator ten jest ignorowany jeśli:

- parametr aktualny jest elementem tabeli indeksowej,
- typ parametru aktualnego posiada ograniczenie długości (nie dotyczy to jednak parametrów o typach znakowych), dokładności lub parametr posiada ograniczenie NOT NULL,
- parametry aktualny i formalny są rekordami zadeklarowanymi niejawnie jako zmienne sterujące pętli FOR lub są zadeklarowane za pomocą pseudoatrybutu %ROWTYPE, a ograniczenia odpowiadających sobie pól w rekordach różnią się od siebie,
- podczas przekazywania parametrów aktualnych wystąpi autokonwersja typów.

Modyfikator NOCOPY stosowany jest przede wszystkim w celu przyśpieszenia przekazywania dużych tabel. Dodatkowo umożliwia on zachowanie wartości parametrów wyznaczonych w podprogramie w przypadku wystąpienia błędu (standardowo w takiej sytuacji, dla przekazywania przez wartość, parametr aktualny zachowuje wartość sprzed wywołania).

Podprogram jest wywoływany według składni:

nazwa_podprogramu(argument [, ...])

Argumenty wywołania pełnią rolę parametrów aktualnych. Można je specyfikować w **notacji pozycyjnej** (odpowiedniość parametrów aktualnych i formalnych) lub w **notacji imiennej** (dowolna kolejność parametrów aktualnych). W notacji imiennej parametr jest specyfikowany według składni:

nazwa_parametru_formalnego=>wartość_aktualna

W przypadku braku wartości aktualnej parametru przekazywanego w trybie IN zostanie mu przypisana wartość domyślna.

Poniżej przedstawiono przykład nagłówka procedury oraz przykłady jej wywołania (poprawne i błędne).

```
PROCEDURE cos(zlecenie NUMBER, objetosc NUMBER:=450,
              dawca VARCHAR2:='Miodzio',
              pijca VARCHAR2:='Opoj');

cos;                                  -- źle
cos(221);                             -- dobrze
cos(221,'Baczek');                    -- źle
cos(221,pijca=>'Baczek');             -- dobrze
cos(zlecenie=>221,pijca=>'Baczek');   -- dobrze
```

Podprogram jest zapisywany jako obiekt bazy danych za pomocą polecenia składowej DDL SQL’a określonego składnią:

CREATE [**OR REPLACE**] definicja_bloku_podprogramu;

Uwaga!!! Oracle zapisuje w bazie danych także podprogram z błędami kompilacji. Na ekranie pojawia się wtedy odpowiednie ostrzeżenie. Polecenie SHOW ERRORS wyświetla listę błędów.

Podprogram składowany w bazie danych może być wywołany z wnętrza dowolnego bloku PL/SQL. Procedura składowana w bazie danych (poprzedzona nazwą użytkownika), w ramach SQL Developer'a, wywoływana jest pleceniem EXECUTE. W środowisku SQL*Plus procedura taka uruchamiana jest poleceniem EXEC[UTE] lub poleceniem CALL (od wersji Oracle 8.1). W obu środowiskach wykonawczych składowana w bazie danych funkcja może być wywołana także w ramach polecenia DML lub w ramach polecenia SELECT. Musi ona wtedy spełniać następujące warunki:

- polecenia w jej ciele nie mogą modyfikować zawartości bazy,
- parametry muszą być przekazywane w trybie IN,
- parametry i wyznaczana wartość muszą być typu prostego.

Ponadto polecenie DML nie może wykorzystywać funkcji operującej na relacji, której atrybuty to polecenie DML modyfikuje.

***Zad**. Zapisać w bazie danych funkcję określającą minimalny przydział myszy w określonej parametrem bandzie a następnie wykorzystać zdefiniowaną funkcję w zapytaniu SELECT oraz poleceniu DML.*

```
SQL> CREATE OR REPLACE FUNCTION min_przydzial(nrb NUMBER)
  2  RETURN NUMBER
  3  AS
  4   minp Kocury.przydzial_myszy%TYPE;
  5  BEGIN
  6   SELECT MIN(przydzial_myszy) INTO minp
  7   FROM Kocury WHERE nr_bandy=nrb;
  8   RETURN minp;
  9  EXCEPTION
 10   WHEN NO_DATA_FOUND THEN NULL;
 11   WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
 12  END;
 13  /

Funkcja została utworzona.
```

```
SQL> SELECT pseudo
  2  FROM Kocury WHERE przydzial_myszy=min_przydzial(2);

PSEUDO
--------------
LASKA

SQL> UPDATE Kocury
  2  SET przydzial_myszy=min_przydzial(3) WHERE nr_bandy=2;
ORA-04091: tabela Z.KOCURY ulega mutacji, wyzwalacz/funkcja
może tego nie widzieć
SET przydzial_myszy=min_przydzial(3) WHERE nr_bandy=2
                    *
BŁĄD w linii 2:
ORA-06503: PL/SQL: Powrót z funkcji bez wartości
ORA-06512: przy "Z.MIN_PRZYDZIAL", linia 12

SQL>
```

Zdefiniowana funkcja bez problemu została wywołana w ramach polecenia SELECT natomiast podczas jej wykorzystania w poleceniu UPDATE pojawił się błąd. Wynika on z wspomnianego zakazu wykorzystania w poleceniu DML funkcji, która operuje na relacji modyfikowanej – funkcja `min_przydzial` operuje na relacji `Kocury`, którą modyfikuje polecenie UPDATE.

Podprogram jest usuwany z bazy danych za pomocą polecenia DDL o składni:

DROP {**PROCEDURE** | **FUNCTION**} nazwa_podprogramu;

Listę wszystkich podprogramów użytkownika można znaleźć w perspektywie systemowej `USER_OBJECTS` a ich treść w perspektywie `USER_SOURCE`.

Podprogramy w PL/SQL można wywoływać rekurencyjnie.

Zad. *Wyznaczyć rekurencyjnie wszystkich szefów wybranego kota.*

```
SQL> CREATE OR REPLACE
  2  FUNCTION szefowie_rek(pseudo_kota Kocury.pseudo%TYPE)
  3  RETURN VARCHAR2
  4  AS
  5    ps_szefa Kocury.szef%TYPE; im_szefa Kocury.imie%TYPE;
  7  BEGIN
  8    SELECT K1.szef,K2.imie INTO ps_szefa,im_szefa
  9    FROM Kocury K1,Kocury K2
 10    WHERE K1.szef=k2.pseudo AND K1.pseudo=pseudo_kota;
 11    DBMS_OUTPUT.PUT('Pseudonim szefa: ');
 12    DBMS_OUTPUT.PUT(RPAD(ps_szefa,10));
 13    DBMS_OUTPUT.PUT_LINE(' Imie: '||RPAD(im_szefa,10));
 14    RETURN szefowie_rek(ps_szefa);
 15  END szefowie_rek;
 16  /

Funkcja została utworzona.

SQL> SET VERIFY OFF
SQL> DECLARE
  2    ps Kocury.pseudo%TYPE:='&1';
  3    ps_kota Kocury.pseudo%TYPE; im_kota Kocury.imie%TYPE;
  5  BEGIN
  6    SELECT imie INTO im_kota
  7    FROM Kocury
  8    WHERE pseudo=ps;
  9    DBMS_OUTPUT.PUT('Pseudonim kota:  ');
 10    DBMS_OUTPUT.PUT(RPAD(ps,10));
 11    DBMS_OUTPUT.PUT_LINE(' Imie: '||RPAD(im_kota,10));
 12    ps_kota:=szefowie_rek(ps);
 13  EXCEPTION
 14    WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('Koniec');
 15  END;
 16  /
Podaj wartość dla 1: ZERO
Pseudonim kota:  ZERO       Imie: LUCEK
Pseudonim szefa: KURKA      Imie: PUNIA
Pseudonim szefa: ZOMBI      Imie: KOREK
Pseudonim szefa: TYGRYS     Imie: MRUCZEK
Koniec

Procedura PL/SQL została zakończona pomyślnie.

SQL>SET VERIFY ON
SQL>
```

3.7. Pakiety

PL/SQL daje możliwość grupowania pewnych obiektów (typy, procedury, funkcje, zmienne, stałe, kursory i wyjątki) w biblioteki zwane pakietami. Pakiet jest kolejnym nazwanym blokiem PL/SQL zapisywanym jako obiekt bazy danych. W skład pakietu wchodzi:

- specyfikacja - zawiera ona deklaracje i niektóre definicje (stałe, typy, kursory) udostępnianych przez pakiet obiektów (jest to interfejs pakietu),
- ciało - zawiera ono definicje obiektów pakietu oraz deklaracje i definicje obiektów wewnętrznych ciała dostępnych tylko dla pakietu.

Specyfikacja pakietu definiowana jest według składni:

CREATE [**OR REPLACE**] **PACKAGE** nazwa_pakietu
[**AUTHID {DEFINER** | **CURRENT_USER**}]
{**IS** | **AS**}
/* *definicje stałych, typów, kursorów, deklaracje podprogramów (ich nagłówki), zmiennych, wyjątków dostępnych z zewnątrz pakietu* */
END [nazwa_pakietu];

Ciało pakietu definiowane je według składni:

CREATE [**OR REPLACE**] **PACKAGE BODY** nazwa_pakietu
{**IS** | **AS**}
/* *definicje i deklaracje obiektów lokalnych dla pakietu* */
/* *definicje podprogramów zadeklarowanych w specyfikacji pakietu* */
[**BEGIN**
/* *opcjonalny blok inicjujący zmienne pakietu w momencie pierwszego odwołania się do pakietu* */]
END [nazwa_pakietu];

Ciało pakietu jest elementem opcjonalnym (nie występuje jeśli specyfikacja nie zawiera deklaracji podprogramów). W przypadku gdy jakiś podprogram pakietu odwołuje się w ciele pakietu do innego podprogramu pakietu wtedy podprogram ten musi być zdefiniowany przed podprogramem, z którego następuje odwołanie. Jeśli z jakichś powodów jest to niemożliwe, należy zadeklarować wywoływany podprogram w postaci jego nagłówka umieszczonego przed definicją podprogramu wywołującego (wczesne deklarowanie). Odwołanie do obiektów pakietu następuje według składni:

nazwa_pakietu.nazwa_obiektu_pakietu

Podprogramy wewnątrz pakietu mogą być przeciążane tzn. może w nim istnieć więcej niż jedna procedura lub funkcja o tej samej nazwie lecz o różnych parametrach. Przeciążanie podprogramów podlega następującym ograniczeniom:

- nie można przeciążać dwóch podprogramów, jeśli ich parametry różnią się tylko nazwą lub trybem przekazywania,
- nie można przeciążać dwóch funkcji różniących się tylko typem zwracanej wartości,
- nie można przeciążać dwóch funkcji, dla których parametry posiadają typy z tej samej rodziny (np. CHAR i VARCHAR2).

W przypadku wykorzystywania przez polecenie SQL lub przez blok PL/SQL funkcji umieszczonych w pakiecie system, do wersji Oracle 8.1 włącznie, nie może sprawdzić warunku na ich użycie (tj. braku modyfikacji zawartości relacji przez polecenia funkcji - dostępna jest wtedy tylko specyfikacja pakietu). Od wersji Oracle 9i ograniczenie to dotyczy tylko funkcji pakietowych przeznaczonych do wywoływania z poziomu bloku. W takim przypadku należy w specyfikacji pakietu, po nagłówku funkcji, umieścić odpowiednią informację dotyczącą tzw. poziomu czystości funkcji. Wykorzystywana jest w tym celu dyrektywa kompilatora RESTRICT_REFERENCES. Jej składnia jest następująca:

PRAGMA RESTRICT_REFERENCES(nazwa_funkcji,
WNDS [, **WNPS**][, **RNDS**][, **RNPS**][,**TRUST**]);

Poniżej przedstawiono znaczenie parametrów w określających poziom czystości funkcji.

Parametr	Opis
WNDS	Funkcja nie modyfikuje zawartości relacji (za pomocą polecenia DML).
WNPS	Funkcja nie modyfikuje wartości zmiennych pakietowych (zmienne pakietowe nie są wykorzystywane przez operator przypisania ani przez polecenie FETCH).
RNDS	Funkcja nie odczytuje zawartości relacji (za pomocą polecenia SELECT) .
RNPS	Funkcja nie odczytuje wartości zmiennych pakietowych (zmienne pakietowe nie są wykorzystywane po prawej stronie operatora przypisania ani jako część wyrażenia SQL lub PL/SQL).
TRUST	Funkcja może wywoływać inne funkcje z nieokreślonym poziomem czystości.

Zad. *Utworzyć pakiet zawierający dwie funkcje, jedna wyznaczająca minimalny przydział myszy dla bandy określonej parametrem, druga wyznaczająca średni przydział myszy dla bandy określonej parametrem. Wykorzystać funkcje pakietu do znalezienia kotów, których przydział myszy jest większy od średniego przydziału w ich bandach wyświetlając dodatkowo różnicę między ich przydziałem myszy a minimalnym przydziałem w ich bandzie.*

```
SQL> CREATE OR REPLACE PACKAGE pakiet_funkcji AS
  2   FUNCTION min_przydzial(nrb NUMBER) RETURN NUMBER;
  3   FUNCTION sred_przydzial(nrb NUMBER) RETURN NUMBER;
  4  END pakiet_funkcji;
  5  /

Pakiet został utworzony.
```

```
SQL> CREATE OR REPLACE PACKAGE BODY pakiet_funkcji AS
  2   FUNCTION min_przydzial(nrb NUMBER) RETURN NUMBER
  3   IS
  4    mp Kocury.przydzial_myszy%TYPE;
  5   BEGIN
  6    SELECT MIN(NVL(przydzial_myszy,0)) INTO mp FROM Kocury
  7    WHERE nr_bandy=nrb;
  8    RETURN mp;
  9   END min_przydzial;
 10   FUNCTION sred_przydzial(nrb NUMBER) RETURN NUMBER
 11   IS
 12    sp NUMBER(10,3);
 13   BEGIN
 14    SELECT AVG(NVL(przydzial_myszy,0)) INTO sp FROM Kocury
 15    WHERE nr_bandy=nrb;
 16    RETURN sp;
 17   END sred_przydzial;
 18  END pakiet_funkcji;
 19  /

Ciało pakietu zostało utworzone.

SQL> SELECT imie "Imie",nr_bandy "Nr bandy",
  2         NVL(przydzial_myszy,0)-
  3         pakiet_funkcji.min_przydzial(nr_bandy)
  4        "Nadmiar"
  5  FROM Kocury
  6  WHERE przydzial_myszy>
  7                  pakiet_funkcji.sred_przydzial(nr_bandy)
  8  ORDER BY nr_bandy;

Imie              Nr bandy    Nadmiar
--------------- ---------- ----------
MRUCZEK                  1         81
BOLEK                    2         48
ZUZIA                    2         41
JACEK                    2         43
KOREK                    3         55
PUNIA                    3         41
MELA                     4         11
KSAWERY                  4         11
PUCEK                    4         25

9 wierszy zostało wybranych.

SQL>
```

Funkcje pakietu zostały zdefiniowane do ich wykorzystania z poziomu SQL’a więc nie było konieczne określenie ich poziomu czystości.

3.8. Wyzwalacze

Wyzwalacze są kolejnym nazwanym blokiem zapisywanym w bazie danych jako obiekt bazy danych. W przeciwieństwie do jawnie wykonywanych, poprzez wywołanie, podprogramów wyzwalacze są wykonywane niejawnie pod wpływem wystąpienia określonego zdarzenia wyzwalającego. Zdarzenie to może dotyczyć operacji DML na relacji (Oracle 7.0 i wyżej), operacji DML na perspektywie (Oracle 8.0 i wyżej), operacji DDL oraz zdarzeń bazy danych takich jak np. logowanie (Oracle 8.1 i wyżej). Wyzwalacz definiowany jest zgodnie ze składnią:

CREATE [**OR REPLACE**] **TRIGGER** nazwa_wyzwalacza
{**BEFORE** | **AFTER**} | **INSTEAD OF** zdarzenie_wyzwalające
ON {nazwa_relacji | **DATABASE**} | nazwa_perspektywy
[**REFERENCING** [**OLD AS** nazwa_OLD]
[**NEW AS** nazwa_NEW]
[**PARENT AS** nazwa_PARENT]]
[**FOR EACH ROW**]
[**FOLLOWS** nazwa_wyzwalacza]
[**WHEN** warunek_wyzwalacza]
{ blok PL/SQL | **CALL** procedura};

Rozmiar treści wyzwalacza nie może przekroczyć 32K. W przypadku większego wyzwalacza można zredukować jego objętość przenosząc część kodu w postaci podprogramów do pakietu. Wyzwalacze mogą posiadać te same nazwy co podprogramy czy relacje (posiadają inną przestrzeń nazw).

Poniżej Oracle 8i treścią wyzwalacza mógł być tylko blok PL/SQL. Od Oracle 8i jego treścią może być także procedura składowana w bazie danych (niekoniecznie napisana w PL/SQL!) wywoływana poleceniem CALL.

Wyzwalacze typu BEFORE aktywowane są przed wykonaniem zdarzenia wyzwalającego (operacje DML na relacji, operacje DDL oraz zdarzenia bazy danych) a typu AFTER po wykonaniu zdarzenia wyzwalającego. Wyzwalacze typu INSTEAD OF wykonywane są zamiast operacji DML (zdarzenie wyzwalające) na perspektywie.

Można wyróżnić następujące zdarzenia wyzwalające DML dla wyzwalaczy typu BEFORE | AFTER ... ON nazwa_relacji oraz dla wyzwalacza typu INSTEAD OF ... ON nazwa_perspektywy:

- INSERT - wstawianie nowego wiersza bezpośrednio w relacji lub pośrednio w relacji (relacjach) *via* perspektywa,
- UPDATE [OF lista_atrybutów] - poprawa wartości atrybutów bezpośrednio w relacji lub pośrednio w relacji (relacjach) *via* perspektywa (lista_atrybutów określa atrybuty, których poprawa ma uaktywniać wyzwalacz; lista ta jest niedostępna dla wyzwalacza INSTEAD OF),
- DELETE - usuwanie krotek bezpośrednio w relacji lub pośrednio w relacji (relacjach) *via* perspektywa.

Można wyróżnić następujące zdarzenia wyzwalające DDL dla wyzwalaczy typu BEFORE i AFTER ... ON DATABASE:

- CREATE - tworzenie nowego obiektu bazy danych,
- ALTER - zmiany w istniejącym obiekcie bazy danych,
- DROP - usuwanie obiektu bazy danych.

Zdarzenia wyzwalające bazy danych dla wyzwalaczy typu BEFORE | AFTER … ON DATABASE to:

- SERVERERROR - pojawienie się komunikatu o błędzie serwera (tylko typ AFTER),
- LOGON - zalogowanie się użytkownika (tylko typ AFTER),
- LOGOFF - wylogowanie się użytkownika (tylko typ BEFORE),
- STARTUP - otwarcie bazy danych (tylko typ AFTER),
- SHUTDOWN - zamknięcie bazy danych (tylko typ BEFORE).

Zdarzenia wyzwalające mogą być łączone operatorem logicznym OR.

Klauzula FOR EACH ROW dotyczy tylko wyzwalaczy aktywowanych przez polecenie DML i umieszczana jest wtedy, gdy wyzwalacz ma być uruchamiany dla każdego modyfikowanego wiersza (tzw. wyzwalacz wierszowy). W przeciwnym wypadku (brak klauzuli) wyzwalacz uruchamiany jest tylko raz, niezależnie od liczby wierszy modyfikowanych przez polecenie DML (tzw. wyzwalacz poleceniowy). Wyzwalacz INSTEAD OF jest zawsze wyzwalaczem wierszowym, stąd klauzula FOR EACH ROW tam nie występuje.

Wprowadzona od wersji Oracle 11g klauzula FOLLOWS pozwala na wskazanie wyzwalacza DML tego samego typu dla danej tabeli, który ma być wykonywany przed definiowanym wyzwalaczem. W niższych wersjach Oracle kolejność ta była nieokreślona.

Klauzula WHEN umożliwia zdefiniowanie dodatkowego warunku (w nawiasach i bez podzapytań!) zawężającego liczbę wyzwoleń wyzwalacza. Dla wierszowych wyzwalaczy DML (tzn. posiadających klauzulę FOR EACH ROW lub wyzwalaczy INSTEAD OF) klauzula ta umożliwia definicję warunku selekcji wierszy, których modyfikacja będzie aktywowała wyzwalacz. Wyzwalacze takie (wierszowe, żadne inne!) mogą dodatkowo wykorzystywać w klauzurze WHEN oraz w swoim ciele dwa kwalifikatory: NEW i OLD (w ciele wyzwalacza poprzedzane znakiem dwukropka - :). Umożliwiają one dostęp do nowej (poprawionej) i starej wartości atrybutu modyfikowanego przez polecenie DML. Dostęp ten jest realizowany zgodnie ze składnią:

[**:**]**NEW**.nazwa_atrybutu [**:**]**OLD**.nazwa_atrybutu

Jeśli zachodzi potrzeba nadania innych nazw kwalifikatorom NEW i OLD, wykorzystuje się klauzulę REFERENCING. Od wersji Oracle 8i wprowadzono dodatkowy kwalifikator PARENT. Dotyczy on wyzwalaczy aktywowanych modyfikacją tzw. tabeli zagnieżdżonej (typ ten zostanie omówiony podczas przedstawiania rozszerzeń obiektowych Oracle’a).

Dla poleceń INSERT i DELETE odpowiednio, kwalifikatory OLD i NEW przyjmują wartość NULL.

W przypadku wyzwalaczy aktywowanych poleceniami DDL i zdarzeniami bazy danych (nazywanych razem systemowymi) istnieją ograniczenia dotyczące rodzajów warunków w klauzuli WHEN:

- dla wyzwalaczy STARTUP i SHUTDOWN nie można określić żadnych warunków,
- dla wyzwalaczy SERVERERROR można jedynie wykorzystać zmienną `ERRNO` określającą numer błędu serwera,
- dla wyzwalaczy LOGON i LOGOFF można jedynie sprawdzać hasło i nazwę użytkownika (`ORA_DES_ENCRYPTED_PASSWORD` i `ORA_LOGIN_USER`),
- dla wyzwalaczy DDL można jedynie sprawdzać typ i nazwę obiektu (`ORA_DICT_OBJ_TYPE` i `ORA_DICT_OBJ_NAME`) oraz hasło i nazwę użytkownika.

Tworzenie wyzwalaczy systemowych jest możliwe tylko z uprawnieniami ADMINISTER DATABASE TRIGGER.

Zad. *Tygrys postanowił zabezpieczyć się przed usunięciem go ze stada poprzez likwidację jego bandy (zawsze może być dodefiniowane ograniczenie ON DELETE CASCADE dla klucza obcego `nr_bandy` w relacji `Kocury`). Postanowił także dodatkowo zabezpieczyć swojego zausznika Łysego (banda nr 2). Zdefiniować wyzwalacz aktywowany przez usunięcie krotki w relacji `Bandy` realizujący te zabezpieczenia.*

```
SQL> CREATE OR REPLACE TRIGGER czy_usunac_bande
  2  BEFORE DELETE ON Bandy
  3  FOR EACH ROW WHEN (OLD.nr_bandy IN (1,2))
  5  DECLARE
  6   ile_czlonkow NUMBER(3):=0;
  7  BEGIN
  8   SELECT COUNT(*) INTO ile_czlonkow
  9   FROM Kocury WHERE nr_bandy=:OLD.nr_bandy;
 11   IF ile_czlonkow>0 THEN
 12      RAISE_APPLICATION_ERROR(-20105,
 13      'Banda '||:OLD.nazwa||' z obsada jest nieusuwalna!');
 14   END IF;
 15  END;
SQL> /

Wyzwalacz został utworzony.
SQL> DELETE FROM Bandy WHERE nr_bandy=1;
DELETE FROM Bandy WHERE nr_bandy=1
                            *
BŁĄD w linii 1:
```

```
ORA-20105: Banda SZEFOSTWO z obsada jest nieusuwalna!
ORA-06512: at line 8
ORA-04088: error during execution of trigger
           'Z.CZY_USUNAC_BANDE'
SQL> DELETE FROM Bandy WHERE nr_bandy=5;

1 wiersz został usunięty.

SQL> ROLLBACK;

Wycofanie zostało zakończone.

SQL>
```

Wyzwalacz zablokował usunięcie bandy nr 1. Bez problemu natomiast została usunięta banda nr 5. Zdefiniowany wyzwalacz ma znaczenie tylko wtedy, jeśli dla klucza obcego `nr_bandy` w relacji `Kocury` zdefiniowano ograniczenie ON DELETE CASCADE. Inaczej obsadzone bandy będą i tak chronione poprzez ograniczenie referencyjne - błąd `"ORA-02292: integrity constraint (Z.NR_BANDY_FK) violated - child record found"`.

Zad. *Wykorzystując perspektywę `Kotki` wybierającą atrybuty `pseudo`, `imie`, `w_stadku_od`, `nazwa` i `szef` z relacji `Kocury` i `Bandy` dopisać do relacji `Kocury` nowego kota.*

```
SQL> CREATE OR REPLACE VIEW Kotki AS
  2  SELECT pseudo,imie,w_stadku_od,nazwa,szef
  3  FROM Kocury K,Bandy B
  4 WHERE K.nr_bandy=B.nr_bandy;

Perspektywa została utworzona.

SQL> CREATE OR REPLACE TRIGGER dopisz_kota
  2  INSTEAD OF INSERT ON Kotki
  3  DECLARE
  4    nb NUMBER; l NUMBER;
  5  BEGIN
  6    SELECT COUNT(*) INTO l FROM Bandy
  7    WHERE nazwa=:NEW.nazwa;
  8    IF l=0
  9       THEN RAISE_APPLICATION_ERROR(-20001,'Zla nazwa bandy!');
 10    END IF;
 11    SELECT nr_bandy INTO nb FROM Bandy
 12    WHERE nazwa=:NEW.nazwa;
 13    IF :NEW.w_stadku_od>SYSDATE
 14       THEN RAISE_APPLICATION_ERROR(-20002,'Data powyzej biezacej!');
 15    END IF;
 16    SELECT COUNT(*) INTO l FROM Kocury
 17    WHERE pseudo=:NEW.pseudo;
 18    IF l=1
 19       THEN RAISE_APPLICATION_ERROR(-20003,'Istniejacy pseudonim!');
 20    END IF;
```

```
 21    SELECT COUNT(*) INTO l FROM Kocury
 22    WHERE szef=:NEW.szef;
 23    IF l=0
 24       THEN RAISE_APPLICATION_ERROR(-20004,'Nieistniejacy szef!');
 25    END IF;
 26    INSERT INTO Kocury (pseudo,imie,w_stadku_od,nr_bandy,szef)
 27           VALUES (:NEW.pseudo,:NEW.imie,:NEW.w_stadku_od,nb,:NEW.szef);
 28  END;
 29  /

Wyzwalacz został utworzony.

SQL> INSERT INTO Kotki VALUES ('GRUBY','RYCHO','2019-11-16',
  2                            'CZARNI RYCERZE','LYSY');

1 wiersz został utworzony.

SQL> ROLLBACK;

Wycofanie zostało zakończone

SQL> INSERT INTO Kotki VALUES ('TYGRYS','RYCHO','2019-11-16',
  2                            'CZARNI RYCERZE','LYSY');
                                          *
BŁĄD w linii 1:
ORA-20002: Istniejacy pseudonim!
ORA-06512: przy "Z.DOPISZ_KOTA", linia 17
ORA-04088:    błąd    w    trakcie    wykonywania    wyzwalacza
'Z.DOPISZ_KOTA'

SQL>
```

Dzięki wyzwalaczowi INSTEAD OF możliwa była modyfikacja relacji `Kocury` poprzez perspektywę `Kotki`, mimo że jest to perspektywa niemodyfikowalna. Wyzwalacz taki charakteryzuje się tym, że operacja powodująca jego „odpalenie” jest ignorowana i zastępowana przez operację zdefiniowaną w jego ciele.

Systemowe zdarzenia wyzwalające, czyli zdarzenia DDL i zdarzenia bazy danych, posiadają atrybuty, które można odczytać w ramach ciała wyzwalacza. Atrybutami tymi są:

- DICTIONARY_OBJ_NAME - nazwa obiektu bazy danych użyta w poleceniu DDL,
- DICTIONARY_OBJ_TYPE - rodzaj obiektu w poleceniu DDL,
- DICTIONARY_OBJ_OWNER - właściciel obiektu, którego nazwa została użyta w poleceniu DDL,
- IS_ALTER_COLUMN(atrybut IN VARCHAR2) - TRUE, jeśli definicja atrybutu została zmieniona,

- IS_DROP_COLUMN(atrybut IN VARCHAR2) - TRUE, jeśli atrybut został usunięty,
- IS_SERVERERROR(numer_błędu IN NUMBER) - TRUE, jeśli wystąpił błąd o podanym numerze,
- LOGIN_USER - nazwa użytkownika aktywującego wyzwalacz,
- SYSEVENT - nazwa zdarzenia, którego wystąpienie spowodowało uruchomienie wyzwalacza,
- CLIENT_IP_ADRES - adres IP komputera klienta,

Zad. *Zdefiniować wyzwalacz monitorujący w relacji Zdarzenia wykonywanie poleceń DDL.*

```
SQL> CREATE TABLE Zdarzenia
  2  (polecenie VARCHAR2(10), uzytkownik VARCHAR2(15),
  3   data DATE, obiekt VARCHAR2(10), nazwa VARCHAR2(14));
Tabela została utworzona.

SQL> CREATE OR REPLACE TRIGGER opis_zdarzenia
  1  BEFORE CREATE OR ALTER OR DROP ON DATABASE
  2  DECLARE
  3    pol Zdarzenia.polecenie%TYPE;
  4    uzy Zdarzenia.uzytkownik%TYPE; dat Zdarzenia.data%TYPE;
  5    obi Zdarzenia.obiekt%TYPE; naz Zdarzenia.nazwa%TYPE;
  6  BEGIN
  7    pol:=SYSEVENT;
  8    uzy:=LOGIN_USER; dat:=SYSDATE;
  9    obi:=DICTIONARY_OBJ_TYPE; naz:=DICTIONARY_OBJ_NAME;
  10   INSERT INTO Zdarzenia VALUES (pol,uzy,dat,obi,naz);
  11 END;
SQL>/

Wyzwalacz został utworzony

SQL> CREATE TABLE Nowa(kolumna NUMBER);

Tabela została utworzona.

SQL> SELECT * FROM Zdarzenia;

POLECENIE  UZYTKOWNIK     DATA       OBIEKT     NAZWA
---------- -------------- ---------- ---------- --------------
CREATE     Z              2019-11-16  TABLE      NOWA

SQL>
```

W przypadku, gdy wyzwalacz jest definiowany dla więcej niż jednego polecenia DML, ciało wyzwalacza można podzielić na sekcje uruchamiane przy wykonywaniu konkretnego polecenia. Służą do tego predykaty INSERTING, UPDATING, DELETING umieszczone w instrukcji warunkowej IF.

```
CREATE OR REPLACE cos
BEFORE INSERT OR UPDATE OR DELETE ON Kocury
BEGIN
 ...
 IF INSERTING THEN ...
    ...
 END IF;
 ...
 IF UPDATING THEN ...
    ...
 END IF;
 -- IF UPDATING('przydzial_myszy') OR
 --    UPDATING('myszy_extra') THEN ...
 --    ...
 -- END IF;
 ...
 IF DELETING THEN ...
    ...
 END IF
 ...
END;
```

3.8.1. Blokowanie i usuwanie wyzwalaczy

Po zdefiniowaniu wyzwalacz jest standardowo gotowy do działania (odblokowany). Blokowanie lub odblokowywanie konkretnego wyzwalacza realizowane jest zgodnie ze składnią:

ALTER TRIGGER nazwa_wyzwalacza {**DISABLE** | **ENABLE**};

Wszystkie wyzwalacze związane z konkretną relacją mogą być blokowane lub odblokowywane zgodnie ze składnią:

ALTER TABLE nazwa_relacji {**DISABLE ALL TRIGGERS** | **ENABLE ALL TRIGGERS**};

Wyzwalacz jest usuwany poleceniem:

DROP TRIGGER nazwa_wyzwalacza;

3.8.2. Ograniczenia wyzwalaczy DML

W wersji Oracle 7.0 z daną relacją może być związany tylko jeden wyzwalacz poleceniowy i tylko jeden wyzwalacz wierszowy każdego z typów: BEFORE INSERT, BEFORE UPDATE, BEFORE DELETE, AFTER INSERT, AFTER UPDATE, AFTER DELETE.

Od wersji Oracle 7.3 z relacją może być związanych wiele wyzwalaczy tego samego typu. Dodatkowo dla wyzwalaczy obowiązują następujące ograniczenia:

- wyzwalacze wierszowe i wszystkie wyzwalacze uruchamiane pośrednio w wyniku działania ograniczenia ON DELETE CASCADE lub ograniczenia ON DELETE SET NULL nie mogą odczytywać ani zmieniać zawartości relacji modyfikowanej (mutating table). Przez relację modyfikowaną rozumiana jest relacja wyzwalacza lub relacja odwołująca się do relacji wyzwalacza poprzez wyżej wymienione ograniczenia. Nie dotyczy to wyzwalaczy INSTEAD OF,
- wyzwalacze nie mogą wykonywać poleceń DDL ani poleceń DCL. Wyjątkiem są tu dostępne od Oracle 8i wyzwalacze z tzw. transakcją autonomiczną,
- w treści wyzwalacza nie można deklarować żadnych zmiennych typu LONG lub LONG RAW. Także kwalifikatory OLD i NEW nie mogą się odwoływać do atrybutów tych typów określonych w relacji, dla której zdefiniowano wyzwalacz,
- poniżej wersji Oracle 8 w wyzwalaczach nie można się odwoływać do atrybutów typu LOB (Large Objects). Od wersji Oracle 8 można to robić, ale nie można modyfikować ich wartości.

Jedynymi przypadkami wyzwalaczy wierszowych, które mogą odczytywać lub modyfikować relację wyzwalającą są wyzwalacze BEFORE i AFTER dla instrukcji INSERT dotyczącej tylko jednego wiersza (np. nie dla instrukcji INSERT ALL lub dla instrukcji INSERT INTO relacja SELECT... nawet jeśli działa ona tylko na jednym wierszu).

3.8.3. Kolejność wykonywania wyzwalaczy DML

W przypadku, gdy z relacją związane są co najmniej dwa wyzwalacze aktywowane zdarzeniem DML, istotna jest kolejność ich wykonywania. Jest ona następująca:

1. Wykonywany jest wyzwalacz poleceniowy BEFORE (jeśli istnieje),
2. Dla każdego wiersza wykonywany jest wyzwalacz wierszowy BEFORE (jeśli istnieje),
3. Wykonywana jest sama instrukcja,
5. Dla każdego wiersza wykonywany jest wyzwalacz wierszowy AFTER (jeśli istnieje),
6. Wykonywany jest wyzwalacz poleceniowy AFTER (jeśli istnieje).

Do wersji Oracle 11g kolejność wykonywania wyzwalaczy DML tego samego typu związanych z jedną relacją była nieokreślona. Od wersji Oracle 11g decyduje o tym przedstawiona wcześniej opcjonalna klauzula nagłówka wyzwalacza FOLLOWS.

3.8.4. COMPOUND TRIGGER

Od wersji Oracle 11g akcje, dotyczące jednej relacji, realizowane dotychczas przez wiele osobnych wyzwalaczy DML (poleceniowych, wierszowych, w trybie BEFORE i AFTER) mogą być zdefiniowane w ramach jednego wyzwalacza określanego w jego definicji jako COMPUND TRIGGER (wyzwalacz złożony). Duża zaletą takiego rozwiązania jest możliwość dostępu, przez każdą z wspomnianych akcji, do wspólnych danych pamiętanych w zmiennych lokalnych wyzwalacza. We wcześniejszych wersjach Oracle kolejne wyzwalacze mogły dzielić dane pamiętane w zmiennych, zdefiniowanej do tego celu, specyfikacji pakietu.

Istnieją dwie podstawowe sytuacje, w których COMPOUND TRIGGER może być wykorzystany:

1. Przygotowanie danych do przetwarzania masowego (wiązanie masowe – zagadnienie to zostanie przedstawione w dalszej części wykładu).
2. W celu uniknięcia błędu *ORA-04091: mutating-table error*.

Poniżej przedstawiono składnię wyzwalacza COMPOUND.

CREATE [**OR REPLACE TRIGGER**] nazwa_wyzwalacza
FOR zdarzenie_wyzwalające_DML
ON nazwa_relacji | nazwa_perspektywy
[**FOLLOWS** nazwa_wyzwalacza]
[**WHEN** warunek_wyzwalacza]
COMPOUND TRIGGER
[-- *definicje i deklaracje obiektów PL/SQL dla wyzwalacza*]

BEFORE STATEMENT IS
[-- *definicje i deklaracje obiektów PL/SQL dla sekcji*]
BEGIN
-- *zdania sekcji realizującej część poleceniową BEFORE*
[**EXCEPTION**
-- *zdania obsługi wyjątków sekcji*]
END BEFORE STATEMENT;

BEFORE EACH ROW IS
[-- *definicje i deklaracje obiektów PL/SQL dla sekcji*]
BEGIN
-- *zdania sekcji realizującej część wierszową BEFORE*
[**EXCEPTION**
-- *zdania obsługi wyjątków sekcji*]
END BEFORE EACH ROW;

```
 AFTER EACH ROW IS
    [-- definicje i deklaracje obiektów PL/SQL dla sekcji]
    BEGIN
      -- zdania sekcji realizującej część wierszową AFTER
    [EXCEPTION
      -- zdania obsługi wyjątków sekcji]
    END AFTER EACH ROW;

    AFTER STATEMENT IS
    [-- definicje i deklaracje obiektów PL/SQL dla sekcji]
    BEGIN
      -- zdania sekcji realizującej część poleceniową AFTER
    [EXCEPTION
      -- zdania obsługi wyjątków sekcji]
    END AFTER STATEMENT;

END nazwa_wyzwalacza;
```

Wyzwalacz jest aktywowany wskazanym rodzajem zdarzenia DML (INSERT, DELETE lub UPDATE [OF lista_atrybutów]; zdarzenia mogą być powiązane operatorem logicznym OR) na wskazanej relacji lub perspektywie. Zawiera on opcjonalną część deklaratywną, w której pamiętane mogą być, między innymi, dzielone przez wszystkie sekcje wyzwalacza dane. Sekcje wyzwalacza realizują kolejno akcje: części poleceniowej BEFORE, części wierszowej BEFORE, części wierszowej AFTER i części poleceniowej AFTER. W ramach sekcji wierszowych można wykorzystywać kwalifikatory :OLD, :NEW i :PARENT. Sekcje mogą wystąpić tylko w kolejności zgodnej z powyższą składnią. W ramach wyzwalacza musi wystąpić przynajmniej jedna z tych sekcji.

Wykorzystanie wyzwalacza COMPOUND związane jest z pewnymi ograniczeniami.

1. Wyzwalacz mogą aktywować jedynie polecenia DML na relacji lub perspektywie.
2. Wyzwalacz nie może zawierać transakcji autonomicznej (dyrektywa PRAGMA AUTONOMOUS_TRANSACTION w części deklaratywnej – patrz następny podrozdział).
3. Wyjątki muszą być obsłużone w ramach sekcji, w której wystąpią.
4. Skoki (polecenie GOTO) mogą odbywać się tylko w ramach konkretnej sekcji.
5. Wartość :NEW może być modyfikowana jedynie w sekcji BEFORE EACH ROW.
6. Po wystąpieniu wyjątku DML, w ramach którejś z sekcji, wartości zmiennych lokalnych sekcji są re-inicjalizowane (tracone są ich wartości), jednak dokonane wcześniej modyfikacje nie są wycofywane.

Przykład wyzwalacza COMPOUND realizującego przygotowanie danych do przetwarzania masowego zostanie przedstawiony w części wykładu dotyczącej wiązania masowego a przykład ilustrujący sposób uniknięcia błędu *ORA-04091: mutating-table error* będzie tematem jednego z zadań na listach projektowych.

3.8.4. Wyzwalacze z transakcją autonomiczną

Od wersji Oracle 8.1 możliwe jest wykorzystywanie tzw. transakcji autonomicznej. Transakcja taka jest niezależną transakcją osadzoną w transakcji głównej wykonywaną w trakcie zawieszonej transakcji głównej. Po jej zakończeniu transakcja główna jest kontynuowana. Transakcja autonomiczna musi być zawsze zakończona (zatwierdzona lub wycofana) inaczej wystąpi wyjątek "`ORA-06519: active autonomous transaction detected and rolled back`". Wycofanie transakcji głównej nie ma wpływu na transakcję autonomiczną. Transakcja autonomiczna definiowana jest za pomocą dyrektywy kompilatora:

PRAGMA AUTONOMOUS_TRANSACTION;

***Zad**. Tygrys postanowił rejestrować historię zmian przydziału myszy (także zmiany nie zatwierdzone) w relacji `Historia_zmian`. Zdefiniować wyzwalacz, który monitoruje każdą taką zmianę.*

```
SQL> CREATE TABLE Historia_zmian(
  2         nr_zmiany NUMBER(5),komu VARCHAR2(15),data DATE,
  3         przydzial NUMBER(5),extra NUMBER);

Tabela została utworzona.

SQL> CREATE SEQUENCE nr_w_historii;

Sekwencja została utworzona.
SQL> CREATE OR REPLACE TRIGGER co_z_myszkami
  2  BEFORE INSERT OR UPDATE OF przydzial_myszy,myszy_extra
  3  ON Kocury FOR EACH ROW
  4  DECLARE
  5    ps Kocury.pseudo%TYPE;
  6    pm Kocury.przydzial_myszy%TYPE;
  7    me Kocury.myszy_extra%TYPE;
  8    PRAGMA AUTONOMOUS_TRANSACTION;
  9  BEGIN
  10   IF INSERTING
  11      THEN ps:=:NEW.pseudo; pm:=:NEW.przydzial_myszy;
  12           me:=:NEW.myszy_extra;
  13      ELSE ps:=:OLD.pseudo;
  14   END IF;
  15   IF UPDATING('przydzial_myszy')
  16      THEN pm:=:NEW.przydzial_myszy;
  17      ELSIF NOT INSERTING THEN pm:=:OLD.przydzial_myszy;
  18   END IF;
  19   IF UPDATING('myszy_extra')
  20      THEN me:=:NEW.myszy_extra;
  21      ELSIF NOT INSERTING THEN me:=:OLD.myszy_extra;
  22   END IF;
  23   INSERT INTO Historia_zmian
  24          VALUES (nr_w_historii.NEXTVAL,ps,SYSDATE,pm,me);
  25   COMMIT;
  26 END;
  27 /

Wyzwalacz został utworzony

SQL> UPDATE Kocury  SET myszy_extra=50 WHERE pseudo='LOLA';

1 wiersz został zmodyfikowany.

SQL> ROLLBACK;

Wycofywanie zostało zakończone.

SQL> SELECT * FROM Historia_zmian;
```

```
 NR_ZMIANY KOMU            DATA       PRZYDZIAL      EXTRA
---------- --------------- ---------- ---------- ----------
         1 LOLA            2019-11-16         25         50

SQL>
```

Dzięki zastosowaniu transakcji autonomicznej możliwe było wycofanie modyfikacji krotki w relacji `Kocury` bez jednoczesnego wycofania zmian w relacji `Historia_wydan` (zabroniona dla innych wyzwalaczy a tu wymagana operacja DCL - COMMIT).

Wyzwalacze, w których zdefiniowana jest transakcja autonomiczna mogą oprócz operacji DCL wykonywać także zabronione dla innych wyzwalaczy operacje DDL.

Zad. *Zdefiniować wyzwalacz tworzący nowego użytkownika bazy danych (bez uprawnień) w postaci nowego członka kociego stada.*

```
SQL> CREATE OR REPLACE TRIGGER nowy_uzytkownik
  1  BEFORE INSERT ON Kocury FOR EACH ROW
  2  DECLARE
  3    PRAGMA AUTONOMOUS_TRANSACTION
  4  BEGIN
  5    EXECUTE IMMEDIATE 'CREATE USER '||:NEW.pseudo||
  6                      ' IDENTIFIED BY '||:NEW.pseudo;
  7  END;
  8 /
Wyzwalacz został utworzony

SQL> INSERT INTO Kocury (pseudo,imie,w_stadku_od,szef,
  2                      przydzial_myszy,myszy_extra)
  3  VALUES ('GRUBY','RYCHO','2014-12-09','LYSY',50,10);

1 wiersz został utworzony.

SQL> ROLLBACK;

Wycofanie zostało zakończone

SQL> CONNECT GRUBY/GRUBY;

ERROR:
ORA-01045: user GRUBY lacks CREATE SESSION privilege; logon
denied
Ostrzeżenie: Nie ma już połączenia z ORACLE.

SQL>
```

Ostatni błąd jest związany z brakiem uprawnień do otwarcia sesji przez użytkownika `'GRUBY'`.

Wykorzystane w wyzwalaczu polecenie EXECUTE IMMEDIATE jest elementem tzw. wewnętrznego (rodzimego) dynamicznego SQL. Powoduje ono w tym przypadku stworzenie nowego użytkownika o danych określonych przez wyrażenie łańcuchowe (wykonywane jest polecenie DDL normalnie zabronione dla bloku!).

3.9. Wewnętrzny dynamiczny SQL

SQL jest nazywany dynamicznym, jeśli pełna instrukcja nie jest zdefiniowana aż do momentu wykonania programu. Dopiero wtedy jest ona tworzona w formie wyrażenia łańcuchowego. Wyrażenie takie może zawierać dowolne instrukcje SQL, także te niedostępne w blokach PL/SQL. W Oracle istnieją dwie realizacje dynamicznego SQL'a: niewygodny w użyciu choć mający spore możliwości pakiet DBMS_SQL oraz wprowadzony od wersji Oracle 8.1 tzw. wewnętrzny dynamiczny SQL (native dynamic SQL). Podstawowym poleceniem wewnętrznego dynamicznego SQL'a jest EXECUTE IMMEDIATE powodujące wykonanie dowolnego polecenia SQL (także bloku PL/SQL) zapisanego w postaci łańcucha znaków. Polecenie to posiada dwie postaci:

EXECUTE IMMEDIATE wyrażenie_łańcuchowe_polecenie_SQL
[[**INTO** {zmienna [, ...]}][**USING** {argument_dowiązany [, ..]}]];

EXECUTE IMMEDIATE wyrażenie_łańcuchowe_blok_PL/SQL
[**USING** {argument_dowiązany [, ..]}];

Pierwsze polecenie dotyczy dynamicznego SQL'a, drugie dynamicznego PL/SQL'a. Łańcuch znaków definiujący dynamiczny blok PL/SQL musi się kończyć znakiem średnika (;). Jeśli średnikiem zakończony zostanie łańcuch dynamicznego SQL'a, będzie on traktowany jak blok PL/SQL. Wyrażenie łańcuchowe (w szczególnym przypadku zmienna lub stała łańcuchowa) definiuje dowolne zapytanie SQL. Klauzula INTO dotyczy dynamicznej odmiany polecenia SELECT w bloku PL/SQL (klauzula ta nie może wchodzić w skład polecenia SELECT zdefiniowanego w postaci łańcucha). Klauzula USING definiuje tzw. argumenty dowiązane. Odpowiadające im zmienne, pełniące de facto rolę parametrów formalnych dynamicznego zapytania, wchodzą w skład wyrażenia

łańcuchowego, przy czym dla identyfikacji poprzedzone są znakiem dwukropka (:) a wiązanie odbywa się zgodnie z kolejnością ich wystąpienia w łańcuchu. Wiąże się je w znanych z podprogramów trybach IN, OUT, IN OUT (domyślne IN). Argumenty takie muszą posiadać typy dozwolone w SQL, a nie w PL/SQL i muszą mieć nazwy różne od nazw obiektów bazy danych.

Zad. *Myśli Tygrysa opanowała spiskowa wizja świata. Spisek miał się zawiązać wśród kotów będących starymi obywatelami Unii Europejskiej, a miał polegać na przymusowej wymianie (pod przykrywką wymiany handlowej) zdrowych polskich myszy na sztucznie „pędzone" europejskie. Aby zaradzić zagrożeniu Tygrys postanowił założyć członkom swojego stada tajne „myszowe" konta, na których przechowywana będzie część upolowanych myszy. Napisać blok realizujący to zadanie. Przekazać na początek każdemu kotu po liczbie myszy proporcjonalnej do jego pozycji w stadzie.*

```
SQL> DECLARE
  2   CURSOR kotki IS SELECT level,pseudo FROM Kocury
  3   START WITH szef IS NULL CONNECT BY PRIOR pseudo=szef;
  4   dyn_lanc VARCHAR2(1000); maxl NUMBER(2):=0;
  5   ile NUMBER(4);
  6  BEGIN
  7   FOR ko IN kotki
  8   LOOP
  9    IF ko.level>maxl THEN maxl:=ko.level; END IF;
 10    SELECT COUNT(*) INTO ile
 11    FROM USER_TABLES WHERE table_name=ko.pseudo;
 12    IF ile=1 THEN
 13       EXECUTE IMMEDIATE 'DROP TABLE '||ko.pseudo;
 14    END IF;
 15    dyn_lanc:='CREATE TABLE '||ko.pseudo||'
 16               (data_wpisu DATE,data_wypisu DATE)';
 17    EXECUTE IMMEDIATE dyn_lanc;
 18   END LOOP;
 19   FOR ko IN kotki
 20   LOOP
 21    dyn_lanc:='INSERT INTO '||ko.pseudo||
 22                ' (data_wpisu) VALUES (:da_wp)';
 23    FOR i IN 1..maxl-ko.level+1
 24    LOOP
 25     EXECUTE IMMEDIATE dyn_lanc USING SYSDATE;
 26    END LOOP;
 27   END LOOP;
 28   FOR ko IN kotki
```

```
 29   LOOP
 30    dyn_lanc:='SELECT COUNT(*)-COUNT(data_wypisu) FROM '
 31                ||ko.pseudo;
 32    EXECUTE IMMEDIATE dyn_lanc INTO ile;
 33    DBMS_OUTPUT.PUT_LINE(RPAD(ko.pseudo,10)||
 34                      ' - Liczba myszy na stanie: '||ile);
 35   END LOOP;
 36  END;
 37  /

TYGRYS     - Liczba myszy na stanie: 4
BOLEK      - Liczba myszy na stanie: 3
LOLA       - Liczba myszy na stanie: 3
LYSY       - Liczba myszy na stanie: 3
LASKA      - Liczba myszy na stanie: 2
PLACEK     - Liczba myszy na stanie: 2
RURA       - Liczba myszy na stanie: 2
SZYBKA     - Liczba myszy na stanie: 2
MALA       - Liczba myszy na stanie: 3
RAFA       - Liczba myszy na stanie: 3
DAMA       - Liczba myszy na stanie: 2
MALY       - Liczba myszy na stanie: 2
MAN        - Liczba myszy na stanie: 2
UCHO       - Liczba myszy na stanie: 2
ZOMBI      - Liczba myszy na stanie: 3
KURKA      - Liczba myszy na stanie: 2
ZERO       - Liczba myszy na stanie: 1
PUSZYSTA   - Liczba myszy na stanie: 2

Procedura PL/SQL została zakończona pomyślnie.

SQL>
```

Zad. *Strach przed spiskiem sprawił, że przywódca stada utajnił swój pseudonim. Zrobił to tak skutecznie, że w końcu sam go zapomniał. Napisać blok uruchamiający dynamiczny blok PL/SQL znajdujący na podstawie imienia pseudonim dowolnego kota.*

```
SQL> DECLARE
  2    im Kocury.imie%TYPE:='&1';li NUMBER(2);
  3  BEGIN
  4    SELECT COUNT(*) INTO li FROM Kocury WHERE imie=im;
  5    IF li=0 THEN
  6       RAISE_APPLICATION_ERROR(-20105,'Bledne imie!');
  7    END IF;
```

```
  8    EXECUTE IMMEDIATE
  9      'DECLARE
 10         CURSOR imiennicy IS
 11         SELECT pseudo FROM Kocury WHERE imie=:im;
 12       BEGIN
 13         FOR i IN imiennicy
 14         LOOP
 15           DBMS_OUTPUT.PUT_LINE
 16                       ('||'''Pseudonim - '''||'||i.pseudo);
 17         END LOOP;
 18       END;'
 19    USING im;
 20  END;
SQL> /

Proszę podać wartość dla 1: MRUCZEK
stare   2:   im Kocury.imie%TYPE:='&1';li NUMBER(2);
nowe   2:   im Kocury.imie%TYPE:='MRUCZEK';li NUMBER(2);
Pseudonim – TYGRYS

SQL>
```

W statycznym SQL wielowierszowe polecenia SELECT były obsługiwane albo przez zadeklarowanie jawnego kursora, albo przez zmienne kursora. W wewnętrznym dynamicznym SQL wykorzystywana jest zmienna kursora, której wartość (polecenie SELECT) definiowana jest w postaci łańcucha znaków. Nowym elementem składni jest tu klauzula USING w poleceniu OPEN z listą argumentów dowiązanych. Składnia tego polecenia jest następująca:

OPEN zmienna_kursora **FOR** wyrażenie_łańcuchowe
[**USING** {argument_dowiązany [, …]}];

Wyrażenie łańcuchowe definiuje zapytanie SELECT kursora.

Zad. *Tygrys doszedł do wniosku, że warto by (w ramach ochrony przed spiskiem) ukryć część przydziałów myszy poprzez zafałszowanie (zmniejszenie) liczby myszy dodatkowych uwzględnianych w funkcji wyświetlającej statystykę miesięcznego spożycia. Liczba myszy dodatkowych spożywanych przez każdego kota uwzględniana w oficjalnym zestawieniu miała by być równa połowie średniej wartości dodatkowego spożycia. Napisać blok wyświetlający dla wybranych kotów zmodyfikowany przydział myszy.*

```
SQL> CREATE OR REPLACE PACKAGE kursor AS
  2    TYPE k IS REF CURSOR;
  3  END kursor;
  4  /

Pakiet został utworzony.

SQL> CREATE OR REPLACE
  2   FUNCTION cos_o_kotach(dodatek NUMBER,
  3                         warunekWHERE VARCHAR2)
  4   RETURN kursor.k AS
  5    kur kursor.k;
  6    zapytanie VARCHAR2(1000);
  7  BEGIN
  8   zapytanie:=
  9           'SELECT pseudo,NVL(przydzial_myszy,0)+NVL(:do,0)
 10            FROM Kocury WHERE '||warunekWHERE;
 11   OPEN kur FOR zapytanie
 12   USING dodatek;
 13   RETURN kur;
 14  END cos_o_kotach;
 15  /

Funkcja została utworzona.

SQL> DECLARE
  2   warunek VARCHAR2(500):='&warunek'; wynkur kursor.k;
  3   do NUMBER(3);ps VARCHAR(15);sp NUMBER(3);
  4  BEGIN
  5   SELECT ROUND(AVG(NVL(myszy_extra,0))/2,0)
  6          INTO do FROM Kocury;
  7   wynkur:=cos_o_kotach(do,warunek);
  8   DBMS_OUTPUT.PUT_LINE
  9                   ('Spozycie myszy przez wybrane koty');
 10   LOOP
 11    FETCH wynkur INTO ps,sp;
 12    EXIT WHEN wynkur%NOTFOUND;
 13    DBMS_OUTPUT.PUT_LINE
 14                   ('  '||' Kot '||ps||' zjada '||sp);
 15   END LOOP;
 16   CLOSE wynkur;
 17  EXCEPTION
 18   WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
 19  END;
 20  /
```

```
Proszę podać wartość dla warunek: funkcja=''KOT''
stare   2:  warunek VARCHAR2(500):='&warunek';
                                                wynkur kursor.k;
nowe   2:  warunek VARCHAR2(500):='funkcja=''KOT''';
                                                wynkur kursor.k;
Spozycie myszy przez wybrane koty
Kot ZERO zjada 49
Kot UCHO zjada 46
Kot MALY zjada 46

Procedura PL/SQL została zakończona pomyślnie.

SQL> /
Proszę podać wartość dla warunek: przydzial_myszy>40
stare   2:  warunek VARCHAR2(500):='&warunek';
                                                wynkur kursor.k;
nowe   2:  warunek VARCHAR2(500):='przydzial_myszy>40';
                                                wynkur kursor.k;
Spozycie myszy przez wybrane koty
Kot TYGRYS zjada 109
Kot BOLEK zjada 56
Kot ZOMBI zjada 81
Kot LYSY zjada 78
Kot SZYBKA zjada 71
Kot RAFA zjada 71
Kot KURKA zjada 67
Kot MAN zjada 57
Kot DAMA zjada 57
Kot PLACEK zjada 73
Kot RURA zjada 62
Kot ZERO zjada 49

Procedura PL/SQL została zakończona pomyślnie.

SQL>
```