Oracle9*i*. Programowanie w języku PL-SQL

Autor: Scott Urman
Tłumaczenie: Radosław Meryk
ISBN: 83-7361-065-0
Format: B5, stron: 536

Wykorzystanie wbudowanego w system Oracle języka PL/SQL w znaczący sposób zwiększa wydajność programisty systemów bazodanowych. PL/SQL łączy w sobie możliwości i elastyczność języka czwartej generacji (4GL) SQL z konstrukcjami proceduralnymi języka trzeciej generacji (3GL). Konstrukcje proceduralne są w pełni zintegrowane z Oracle SQL, co daje w rezultacie język strukturalny o ogromnym potencjale. Programy napisane w tym języku umożliwiają obsługę danych zarówno w samym systemie Oracle, jak i w zewnętrznych aplikacjach.

Książka „Oracle9i. Programowanie w języku PL/SQL" wyjaśnia główne właściwości języka oraz różnice w PL/SQL dla różnych wersji bazy danych. Dzięki niej nauczysz się projektować, testować i uruchamiać aplikacje PL/SQL działające w wielu środowiskach, jak również poznasz szczegóły zastosowania języków SQL i PL/SQL, obsługi błędów, zbioru podprogramów i pakietów, a także wiele zaawansowanych właściwości. Niniejsza pozycja umożliwia:

- Zapoznanie się z różnymi środowiskami programistycznymi języka PL/SQL, których kopie znajdują się na dołączonej płycie CD
- Poznanie szczegółów składni języka PL/SQL: zmienne, typy danych, wyrażenia, operatory oraz struktury sterujące
- Zapewnienie spójności danych dzięki instrukcjom sterowania transakcjami dostępnym w SQL-u
- Wykorzystanie kursorów, które pozwalają na tworzenie zapytań zwracających wiele wierszy oraz jawną kontrolę przetwarzania instrukcji SQL
- Tworzenie programów PL/SQL, które wykrywają i inteligentnie reagują na błędy fazy wykonania
- Wykorzystanie możliwości tworzenia kolekcji wielopoziomowych w systemie Oracle9i
- Tworzenie i korzystanie z procedur, funkcji i pakietów
- Tworzenie wyzwalaczy DML zastępujących i systemowych w celu wymuszania złożonych ograniczeń danych
- Korzystanie z zalet języka PL/SQL, takich jak: procedury zewnętrzne, wbudowany dynamiczny SQL, masowe powiązania oraz typy obiektowe

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 11.

Wyzwalacze

Wyzwalacze stanowią czwarty typ bloków nazwanych PL/SQL. Posiadają wiele właściwości charakterystycznych dla podprogramów (które omówiono w poprzednich dwóch podrozdziałach), ale także różnią się od nich w istotny sposób. W niniejszym rozdziale Czytelnik zapozna się ze sposobem tworzenia wyzwalaczy oraz z niektórymi możliwymi zastosowaniami tych obiektów.

Rodzaje wyzwalaczy

Wyzwalacze są podobne do procedur lub funkcji pod tym względem, że są nazwanymi blokami PL/SQL, w których występują sekcje deklaracji, wykonania i obsługi wyjątków. Podobnie jak pakiety, wyzwalacze muszą być składowane jako samodzielne obiekty w bazie danych i nie mogą występować lokalnie w bloku lub w pakiecie. Jak napisano w ostatnich dwóch rozdziałach, procedura jest wykonywana jawnie z innego bloku za pośrednictwem wywołania procedury, które daje możliwość przekazania argumentów. Wyzwalacz natomiast jest wykonywany niejawnie, jeśli wystąpi zdarzenie wyzwalające. Poza tym wyzwalacz nie akceptuje argumentów. Rozpoczęcie wykonywania wyzwalacza jest nazywane uruchamianiem wyzwalacza (*firing*). Zdarzeniem wyzwalającym może być operacja DML (instrukcje `INSERT`, `UPDATE` lub `DELETE`) wykonywana dla tabeli bazy danych lub odpowiedniego rodzaju perspektyw. W systemie Oracle8i rozszerzono możliwości uruchamiania wyzwalaczy o zdarzenia systemowe, takie jak: uruchomienie lub zamknięcie bazy danych, a także określone rodzaje operacji DDL. Zdarzenia wyzwalające zostaną szczegółowo omówione w dalszej części tego rozdziału.

Wyzwalacze można stosować do wielu celów, przykładowo:

- Utrzymywanie złożonych więzów integralności, niemożliwych do uzyskania przez więzy deklaracji uaktywniane podczas tworzenia tabeli.
- Kontrola danych w tabeli przez rejestrowanie dokonywanych zmian oraz autorów tych zmian.
- Automatyczne przekazywanie do innych programów informacji, że jest wymagane podjęcie określonych działań w razie dokonania zmian w tabeli.
- Publikowanie informacji na temat różnego rodzaju zdarzeń w środowisku publikowanie-subskrypcja.

Istnieją trzy podstawowe rodzaje wyzwalaczy: DML, zastępujące (*instead-of*) oraz systemowe. W następnych podrozdziałach będzie omówiony każdy z tych typów, szczegółowo w podrozdziale „Tworzenie wyzwalaczy".

W systemie Oracle8i oraz w wersjach wyższych istnieje możliwość pisania wyzwalaczy zarówno w języku PL/SQL, jak i innych. Wyzwalacze te można następnie wywołać jako procedury zewnętrzne. Więcej informacji na ten temat znajduje się w podrozdziale „Treść wyzwalaczy", a także w rozdziale 12.

Wyzwalacze DML

Wyzwalacze DML są uruchamiane przez instrukcje DML, a typ wyzwalacza DML jest określany przez typ instrukcji. Można je definiować dla operacji `INSERT`, `UPDATE` lub `DELETE`, uruchamiać przed operacją lub po niej, a także w związku z operacjami dotyczącymi wiersza lub instrukcji.

Przykładowo, może zachodzić potrzeba śledzenia danych statystycznych, dotyczących różnych specjalności, włącznie z liczbą studentów zarejestrowanych i liczbą zaliczeń. Dane te mogą być przechowywane w tabeli `statystyka_specjalnosci`:

Dostępne na płycie CD jako część skryptu *tabele.sql*.

```
CREATE TABLE spec_stats (
  specjalnosc         VARCHAR2(30),
  ogolem_zaliczenia   NUMBER,
  ogolem_studenci     NUMBER);
```

W celu zapewnienia ciągłej aktualności danych z tabeli `spec_stats` można utworzyć wyzwalacz dla tabeli `studenci`, który będzie poprawiał tę pierwszą po każdej zmianie dokonanej w tabeli `studenci`. Zadanie to może wykonywać poniższy wyzwalacz `UaktualnijSpecStats`. Po każdej operacji DML wykonanej na tabeli `studenci` nastąpi uruchomienie wyzwalacza. Zawartość wyzwalacza wykonuje zapytanie na tabeli `studenci` i uaktualnia statystykę w `spec_stats`:

Dostępne na płycie CD w skrypcie *UaktualnijSpecStats.sql*.

```
CREATE OR REPLACE TRIGGER UaktualnijSpecStats
  /* Zapewnia ciągłą aktualność tabeli spec_stats,
     wprowadzając zmiany dokonywane w tabeli studenci. */
  AFTER INSERT OR DELETE OR UPDATE ON studenci
DECLARE
  CURSOR k_Statystyka IS
    SELECT specjalnosc, COUNT(*) ogolem_studenci,
           SUM(biezace_zaliczenia) ogolem_zaliczenia
      FROM studenci
      GROUP BY specjalnosc;
```

```
BEGIN
  /* Najpierw zostaną usunięte wiersze z tabeli spec_stats. Spowoduje to wyzerowanie
     statystyki i konieczność rozliczenia wszystkich studentów określonej
     specjalności */
  DELETE FROM spec_stats;
  /* Następnie będzie wykonana pętla dla każdej specjalności i wstawione odpowiednie
     wiersze do tabeli spec_stats  */
  FOR z_RekordStat in k_Statystyka LOOP
    INSERT INTO spec_stats(specjalnosc, ogolem_zaliczenia, ogolem_studenci)
      VALUES(z_RekordStat.specjalnosc, z_RekordStat.ogolem_zaliczenia,
             z_RekordStat.ogolem_studenci);
  END LOOP;
END UaktualnijSpecStats;
```

Wyzwalacz można uruchomić dla więcej niż jednego typu instrukcji wyzwalających. Na przykład, wyzwalacz `UaktualnijSpecStats` można uruchomić dla instrukcji `INSERT`, `UPDATE` oraz `DELETE`. Zdarzenie wyzwalające określa jedną operację DML lub więcej, które powinny spowodować uruchomienie wyzwalacza.

Wyzwalacze zastępujące

W systemie Oracle8 wprowadzono dodatkowy typ wyzwalacza. Wyzwalacze zastępujące (*Instead-of*) mogą być definiowane tylko dla perspektyw (relacyjnych lub obiektowych). Inaczej niż wyzwalacze DML, które uruchamiają się obok operacji DML, wyzwalacze zastępujące uruchamiają się zamiast wyzwalającej je instrukcji DML. Wyzwalacze zastępujące działają na poziomie wierszy. Dla przykładu proszę przeanalizować perspektywę `grupy_pokoje`:

Dostępne na płycie CD jako część skryptu *Zamiast.sql*.

```
CREATE OR REPLACE VIEW grupy_pokoje AS
  SELECT wydzial, kurs, budynek, pokoj_numer
  FROM pokoje, grupy
  WHERE pokoje.pokoj_id = grupy.pokoj_id;
```

Bezpośrednia operacja wstawienia wierszy do tej perspektywy jest nieprawidłowa, ponieważ jest to złączenie dwóch tabel, a do wykonania operacji `INSERT` potrzebna jest modyfikacja obu tabel. Taką sytuację pokazano w poniższej sesji programu SQL*Plus:

Dostępne na płycie CD jako część skryptu *Zamiast.sql*.

```
SQL>  INSERT INTO grupy_pokoje (wydzial, kurs, budynek, pokoj_numer)
  2      VALUES ('MUZ', 100, 'Budynek Muzyki', 200);
INSERT INTO grupy_pokoje (wydzial, kurs, budynek, pokoj_numer)
                          *
BŁĄD w linii 1:
ORA-01776: nie można modyfikować więcej niż jednej tabeli bazowej za pośrednictwem
perspektywy będącej złączeniem
```

Można jednak utworzyć wyzwalacz zastępujący, wykonujący poprawne operacje dla instrukcji INSERT, a mianowicie modyfikację obu tabel:

Dostępne na płycie CD jako część skryptu *Zamiast.sql*.

```
CREATE TRIGGER InsertGrupyPokoje
  INSTEAD OF INSERT ON grupy_pokoje
DECLARE
  z_pokojID pokoje.pokoj_id%TYPE;
BEGIN
  -- Najpierw określono identyfikator pokoju
  SELECT pokoj_id
    INTO z_pokojID
    FROM pokoje
    WHERE budynek = :new.budynek
    AND pokoj_numer = :new.pokoj_numer;

  -- Teraz uaktualnia się tabelę grupy
  UPDATE GRUPY
    SET pokoj_id = z_pokojID
    WHERE wydzial = :new.wydzial
    AND kurs = :new.kurs;
END GrupyPokojeInsert;
```

Dzięki wyzwalaczowi GrupyPokojeInsert wykonanie instrukcji INSERT kończy się powodzeniem.

W obecnej postaci wyzwalacz GrupyPokojeInsert nie sprawdza błędów. Niedogodność ta zostanie zlikwidowana w dalszej części tego rozdziału, w podrozdziale „Tworzenie wyzwalaczy zastępujących".

Wyzwalacze systemowe

W systemie Oracle8i oraz w systemach wyższych wprowadzono trzeci typ wyzwalaczy — wyzwalacze systemowe, które uruchamiają się w przypadku wystąpienia zdarzenia systemowego, takiego jak uruchomienie lub zatrzymanie bazy danych, nie zaś w przypadku wykonania instrukcji DML na tabeli. Wyzwalacz systemowy można także uruchomić dla operacji DDL, jak np. utworzenia tabeli. Przykładowo, chęć zarejestrowania utworzenia obiektu słownika danych można zrobić poprzez następującą tabelę:

Dostępne na płycie CD jako część skryptu *ZarejestrujOperTworzenia.sql*.

```
CREATE TABLE tworzenie_ddl(
   id_uzytkownika     VARCHAR2(30),
   typ_obiektu        VARCHAR2(20),
   nazwa_obiektu      VARCHAR2(30),
   wlasciciel_obiektu VARCHAR2(30),
   data_utworzenia    DATE);
```

Po sformułowaniu tej tabeli można utworzyć wyzwalacz systemowy, którego zadaniem będzie rejestrowanie interesujących informacji. Zadanie to wypełni wyzwalacz ZarejestrujOperTworzenia — po każdej operacji CREATE w bieżącym schemacie zarejestruje informacje na temat właśnie utworzonego obiektu w tabeli tworzenie_ddl:

Dostępne na płycie CD jako część skryptu *ZarejestrujOperTworzenia.sql*.

```
CREATE OR REPLACE TRIGGER ZarejestrujOperTworzenia
  AFTER CREATE ON DATABASE
BEGIN
  INSERT INTO tworzenie_ddl (id_uzytkownika, typ_obiektu, nazwa_obiektu,
                             wlasciciel_obiektu, data_utworzenia)
    VALUES (USER, SYS.DICTIONARY_OBJ_TYPE, SYS.DICTIONARY_OBJ_NAME,
            SYS.DICTIONARY_OBJ_OWNER, SYSDATE);
END ZarejestrujOperTworzenia;
```

Tworzenie wyzwalaczy

Niezależnie od typu wszystkie wyzwalacze tworzy się za pomocą tej samej składni, która — ogólnie — jest następująca:

```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza
 {BEFORE | AFTER | INSTEAD OF} zdarzenie_wyzwalajace
[klauzula_z_odwolaniem]
 [WHEN warunek_wyzwalacza]
 [FOR EACH ROW]
 tresc_wyzwalacza ;
```

W składni *nazwa_wyzwalacza* jest nazwą wyzwalacza, *zdarzenie_wyzwalajace* określa zdarzenie uruchamiające wyzwalacz (może również zawierać odwołanie do tabeli lub perspektywy), natomiast *tresc_wyzwalacza* jest głównym kodem wyzwalacza. Argument *klauzula_z_odwolaniem* jest wykorzystywany w celu odwołania do danych w modyfikowanym wierszu pod inną nazwą. Najpierw następuje ocena warunku *warunek_wyzwalacza* w klauzuli WHEN. Treść wyzwalacza jest wykonywana tylko wtedy, kiedy ten warunek przyjmie wartość TRUE. Więcej przykładów różnych rodzajów wyzwalaczy przeanalizowano w kolejnych podrozdziałach.

Rozmiar treści wyzwalacza nie może przekroczyć 32K. W przypadku większego wyzwalacza można zredukować jego objętość poprzez przeniesienie fragmentów kodu do oddzielnie skompilowanych pakietów lub procedur składowanych i wywołanie ich z treści wyzwalacza. Ze względu na dużą częstotliwość wykonywania dobrą praktyką jest utrzymywanie małego rozmiaru treści wyzwalaczy.

Tworzenie wyzwalaczy DML

Wyzwalacze DML są uruchamiane dla operacji INSERT, UPDATE lub DELETE dotyczących tabeli bazy danych. Następuje to przed wykonaniem określonej operacji lub po. Wyzwalacze mogą być wykonane raz w stosunku do wiersza, którego operacja dotyczy lub raz

w stosunku do określonej operacji. Połączenie tych czynników określa rodzaj wyzwalacza. W sumie istnieje 12 możliwych typów: 3 rodzaje instrukcji × 2 rodzaje czasów × 2 możliwe poziomy. Przykładowo, wszystkie podane niżej typy wyzwalacza są prawidłowe:

- BEFORE (przed) na poziomie instrukcji UPDATE.
- AFTER (po) instrukcji INSERT na poziomie wiersza.
- BEFORE (przed) instrukcją DELETE na poziomie wiersza.

W tabeli 11.1 znajduje się zestawienie tych możliwości. Wyzwalacz jest także uruchamiany dla więcej niż jednego typu instrukcji DML dla określonej tabeli, np. INSERT oraz UPDATE. Kod w wyzwalaczu jest wykonywany razem z samą instrukcją wyzwalającą, jako część tej samej transakcji.

Tabela 11.1. *Typy wyzwalaczy*

Kategoria	Wartości	Komentarz
Instrukcja	INSERT, DELETE, UPDATE	Definiuje rodzaj instrukcji DML powodującej uruchomienie wyzwalacza.
Czas	BEFORE (przed) lub AFTER (po)	Określa, czy wyzwalacz zostanie uruchomiony przed wykonaniem instrukcji czy też po jej wykonaniu.
Poziom	Wiersz lub instrukcja	Jeżeli dany wyzwalacz istnieje na poziomie wiersza, jest uruchamiany po jednym razie dla każdego wiersza, którego dotyczy instrukcja. Jeżeli dany wyzwalacz jest wyzwalaczem na poziomie instrukcji, to jest uruchamiany jeden raz — przed wykonaniem instrukcji albo po. Wyzwalacz na poziomie wiersza jest identyfikowany przez klauzulę FOR EACH ROW w definicji wyzwalacza.

Dla tabeli można zdefiniować dowolną liczbę wyzwalaczy włącznie z możliwością zdefiniowania więcej niż jednego wyzwalacza dla określonego rodzaju instrukcji DML, np. można zdefiniować dwa wyzwalacze AFTER DELETE poziomu instrukcji. Wszystkie wyzwalacze tego samego typu będą uruchamiały się sekwencyjnie (więcej informacji na temat kolejności uruchamiania wyzwalaczy znajduje się w następnym podrozdziale).

Przed wydaniem języka PL/SQL w wersji 2.1 (system Oracle7 wydanie 7.1) dla tabeli można było zdefiniować tylko jeden wyzwalacz określonego typu — maksymalnie 12. A zatem, aby można było definiować podwójne wyzwalacze tego samego typu dla określonej tabeli, parametr inicjalizacji COMPATIBLE musi mieć wartość 7.1 lub wyższą (co jest niemal pewne).

Zdarzenie wyzwalające dla wyzwalacza DML określa nazwę tabeli (oraz kolumny), dla której nastąpi uruchomienie wyzwalacza. W systemie Oracle8i oraz w wersjach wyższych wyzwalacze można także uruchamiać dla kolumny tabeli zagnieżdżonej. Więcej informacji na temat tabel zagnieżdżonych znajduje się w rozdziale 8.

Kolejność uruchamiania wyzwalaczy

Uruchamianie wyzwalaczy następuje podczas wykonywania instrukcji DML. Poniżej podano algorytm ich wykonywania:

1. Wykonanie wyzwalacza BEFORE na poziomie instrukcji, o ile taki istnieje.
2. Dla każdego wiersza, na którym wykonywana jest instrukcja:
   - wykonanie wyzwalacza BEFORE na poziomie wiersza, o ile taki istnieje;
   - wykonanie samej instrukcji, o ile taka istnieje;
   - wykonanie wyzwalacza AFTER na poziomie wiersza, o ile taki istnieje.
3. Wykonanie wyzwalacza AFTER na poziomie instrukcji, o ile taki istnieje.

Aby to zilustrować, proszę sobie wyobrazić, że utworzono wszystkie cztery rodzaje wyzwalaczy UPDATE dla tabeli grupy — BEFORE oraz AFTER — na poziomie instrukcji oraz na poziomie wiersza. Zostaną utworzone trzy wyzwalacze BEFORE poziomu wiersza oraz dwa wyzwalacze AFTER poziomu instrukcji:

Dostępne na płycie CD w skrypcie *KolejnoscUruchamiania.sql*.

```
CREATE SEQUENCE kolejn_wyzwalaczy
  START WITH 1
  INCREMENT BY 1;

CREATE OR REPLACE PACKAGE PakietWyzwalaczy AS
  -- Globalny licznik do wykorzystania w wyzwalaczach
  z_Licznik NUMBER;
END PakietWyzwalaczy;


CREATE OR REPLACE TRIGGER GrupyBInstrukcja
  BEFORE UPDATE ON grupy
BEGIN
  -- Najpierw  licznik zostanie wyzerowany.
  PakietWyzwalaczy.z_Licznik := 0;

  INSERT INTO tabela_tymcz (kol_num, kol_znak)
    VALUES (kolejn_wyzwalaczy.NEXTVAL,
      'BEFORE, na poziomie instrukcji: licznik = ' || PakietWyzwalaczy.z_Licznik);

  -- A teraz zostanie zwiększona jego wartość o jeden dla kolejnego wyzwalacza.
  PakietWyzwalaczy.z_Licznik := PakietWyzwalaczy.z_Licznik + 1;
END GrupyBInstrukcja;


CREATE OR REPLACE TRIGGER GrupyAInstrukcja1
  AFTER UPDATE ON grupy
BEGIN
  INSERT INTO tabela_tymcz (kol_num, kol_znak)
    VALUES (kolejn_wyzwalaczy.NEXTVAL,
      'AFTER, na poziomie instrukcji 1.: licznik = ' || PakietWyzwalaczy.z_Licznik);

  -- Zwiększenie o jeden dla następnego wyzwalacza.
  PakietWyzwalaczy.z_Licznik := PakietWyzwalaczy.z_Licznik + 1;
END GrupyAInstrukcja1;
```

```
CREATE OR REPLACE TRIGGER GrupyAInstrukcja2
  AFTER UPDATE ON grupy
BEGIN
  INSERT INTO tabela_tymcz (kol_num, kol_znak)
    VALUES (kolejn_wyzwalaczy.NEXTVAL,
      'AFTER, na poziomie instrukcji 2.: licznik = ' || PakietWyzwalaczy.z_Licznik);

  -- Zwiększenie licznika dla następnego wyzwalacza.
  PakietWyzwalaczy.z_Licznik := PakietWyzwalaczy.z_Licznik + 1;
END GrupyAInstrukcja2;

CREATE OR REPLACE TRIGGER GrupyBWiersz1
  BEFORE UPDATE ON grupy
  FOR EACH ROW
BEGIN
  INSERT INTO tabela_tymcz (kol_num, kol_znak)
    VALUES (kolejn_wyzwalaczy.NEXTVAL,
      'BEFORE, na poziomie wiersza 1.: licznik = ' || PakietWyzwalaczy.z_Licznik);

  -- Zwiększenie licznika dla następnego wyzwalacza.
  PakietWyzwalaczy.z_Licznik := PakietWyzwalaczy.z_Licznik + 1;
END GrupyBWiersz1;


CREATE OR REPLACE TRIGGER GrupyBWiersz2
  BEFORE UPDATE ON grupy
  FOR EACH ROW
BEGIN
  INSERT INTO tabela_tymcz (kol_num, kol_znak)
    VALUES (kolejn_wyzwalaczy.NEXTVAL,
      'BEFORE, na poziomie wiersza 2.: licznik = ' || PakietWyzwalaczy.z_Licznik);

  -- Zwiększenie licznika dla następnego wyzwalacza.
  PakietWyzwalaczy.z_Licznik := PakietWyzwalaczy.z_Licznik + 1;
END GrupyBWiersz2;

CREATE OR REPLACE TRIGGER GrupyBWiersz3
  BEFORE UPDATE ON grupy
  FOR EACH ROW
BEGIN
  INSERT INTO tabela_tymcz (kol_num, kol_znak)
    VALUES (kolejn_wyzwalaczy.NEXTVAL,
      'BEFORE, na poziomie wiersza 3.: licznik = ' || PakietWyzwalaczy.z_Licznik);

  -- Zwiększenie licznika dla następnego wyzwalacza.
  PakietWyzwalaczy.z_Licznik := PakietWyzwalaczy.z_Licznik + 1;
END GrupyBWiersz3;

CREATE OR REPLACE TRIGGER GrupyAWiersz
  AFTER UPDATE ON grupy
  FOR EACH ROW
BEGIN
  INSERT INTO tabela_tymcz (kol_num, kol_znak)
    VALUES (kolejn_wyzwalaczy.NEXTVAL,
      'AFTER, na poziomie wiersza: licznik = ' || PakietWyzwalaczy.z_Licznik);

  -- Zwiększenie licznika dla następnego wyzwalacza.
     PakietWyzwalaczy.z_Licznik := PakietWyzwalaczy.z_Licznik + 1;
  END GrupyAwiersz;
```

Teraz można wydać następującą instrukcję UPDATE:

Dostępne na płycie CD jako część skryptu *KolejnoscUruchamiania.sql*.

```
UPDATE grupy
  SET liczba_zaliczen = 4
  WHERE wydzial IN ('HIS', 'INF');
```

Powyższa instrukcja dotyczy czterech wierszy. Każdy z wyzwalaczy BEFORE oraz AFTER poziomu instrukcji jest wykonywany raz oraz wyzwalacze BEFORE i AFTER poziomu wiersza są wykonywane po cztery razy. Jeżeli następnie będzie pobrany wiersze z tabeli tabela_tymcz, to uzyska się następujący wynik:

Dostępne na płycie CD w skrypcie *KolejnoscUruchamiania.sql*.

```
SQL> SELECT *  FROM tabela_tymcz
  2  ORDER BY kol_num;
  KOL_NUM   KOL_ZNAK
---------   ----------------------------------------------
        1   BEFORE, na poziomie instrukcji: licznik = 0
        2   BEFORE, na poziomie wiersza 3.: licznik = 1
        3   BEFORE, na poziomie wiersza 2.: licznik = 2
        4   BEFORE, na poziomie wiersza 1.: licznik = 3
        5   AFTER, na poziomie wiersza: licznik = 4
        6   BEFORE, na poziomie wiersza 3.: licznik = 5
        7   BEFORE, na poziomie wiersza 2.: licznik = 6
        8   BEFORE, na poziomie wiersza 1.: licznik = 7
        9   AFTER, na poziomie wiersza: licznik = 8
       10   BEFORE, na poziomie wiersza 3.: licznik = 9
       11   BEFORE, na poziomie wiersza 2.: licznik = 10
       12   BEFORE, na poziomie wiersza 1.: licznik = 11
       13   AFTER, na poziomie wiersza: licznik = 12
       14   BEFORE, na poziomie wiersza 3.: licznik = 13
       15   BEFORE, na poziomie wiersza 2.: licznik = 14
       16   BEFORE, na poziomie wiersza 1.: licznik = 15
       17   AFTER, na poziomie wiersza: licznik = 16
       18   AFTER, na poziomie instrukcji 2.: licznik = 17
       19   AFTER, na poziomie instrukcji 1.: licznik = 18
```

Dla każdego uruchamianego wyzwalacza są widoczne zmiany dokonane przez wcześniejsze wyzwalacze, jak również inne zmiany w bazie danych dokonane za pomocą instrukcji. Można to zauważyć, obserwując wartość licznika wyświetlaną przez każdy z wyzwalaczy (więcej informacji na temat zastosowania zmiennych pakietowych znajduje się w rozdziale 10.).

Kolejność uruchamiania wyzwalaczy tego samego typu nie jest określona. Jak można zauważyć w powyższym przykładzie, dla każdego z wyzwalaczy są widoczne zmiany wykonane przez wyzwalacz uruchomiony wcześniej. Jeżeli kolejność wykonywanych operacji jest istotna, można je wszystkie połączyć w jednym wyzwalaczu.

Kiedy tworzy się migawkę rejestrującą dla tabeli, wówczas system Oracle automatycznie utworzy wyzwalacz AFTER na poziomie wiersza, który będzie uaktualniał dziennik po wykonaniu każdej instrukcji DML. Należy o tym pamiętać, chcąc utworzyć dla tej tabeli dodatkowy wyzwalacz AFTER na poziomie wiersza. Są jeszcze inne ograniczenia dotyczące wyzwalaczy i migawek (w systemie Oracle9i znanych pod nazwą materializowanych perspektyw). Więcej informacji na ten temat znajduje się w dokumentacji *Oracle Server Replication*.

Identyfikatory korelacji w wyzwalaczach na poziomie wiersza

Wyzwalacze na poziomie wiersza uruchamiają się raz dla wiersza przetwarzanego przez instrukcję wyzwalającą. Wewnątrz wyzwalacza można uzyskać dostęp do danych w obecnie przetwarzanym wierszu. Można to zrobić za pomocą dwóch identyfikatorów korelacji — :old oraz :new. *Identyfikator korelacji* jest specjalnym rodzajem zmiennej dowiązanej PL/SQL. Dwukropek na początku każdego z nich wskazuje, że są to zmienne dowiązane w znaczeniu zmiennych hosta wykorzystywanych we wbudowanym PL/SQL i znak ten określa, że nie są to zwykłe zmienne PL/SQL. Kompilator PL/SQL będzie traktował je jako rekordy typu:

```
tabela_wyzwalajaca%ROWTYPE;
```

Tu *tabela_wyzwalajaca* jest tabelą, dla której zdefiniowano wyzwalacz. Zatem odwołanie postaci:

```
:new.pole
```

będzie prawidłowe tylko wtedy, gdy *pole* jest polem w tabeli wyzwalającej. Znaczenie wartości :old oraz :new opisano w tabeli 11.2. Chociaż syntaktycznie identyfikatory korelacji są traktowane jako rekordy, w rzeczywistości nie są nimi (zagadnienie to omówiono w podrozdziale „Pseudorekordy"). Z tego powodu :old i :new nazywane są także *pseudorekordami*.

Tabela 11.2. *Identyfikatory korelacji :old oraz :new*

Instrukcja wyzwalająca	Wartości :old	Wartości :new
INSERT	Nieokreślone — wszystkie pola mają wartość NULL.	Wartości, które będą wstawione po wykonaniu instrukcji.
UPDATE	Oryginalne wartości wiersza przed uaktualnieniem.	Nowe wartości, które będą uaktualnione po wykonaniu instrukcji.
DELETE	Oryginalne wartości przed usunięciem wiersza.	Nieokreślone — wszystkie pola mają wartość NULL.

Wartość :old jest nieokreślona dla instrukcji INSERT, a wartość :new jest nieokreślona dla instrukcji DELETE. Kompilator PL/SQL nie wygeneruje błędu w razie zastosowania :old dla instrukcji INSERT lub :new dla instrukcji DELETE, ale wartości obydwóch pól wyniosą NULL.

W systemie Oracle8i zdefiniowano dodatkowy identyfikator korelacji — :parent. Jeżeli wyzwalacz zdefiniowano dla tabeli zagnieżdżonej, to wartości :old oraz :new odnoszą się do wierszy w tabeli zagnieżdżonej, podczas gdy wartość :parent odnosi się do bieżącego wiersza w tabeli nadrzędnej. Więcej informacji na temat zastosowania identyfikatora :parent znajduje się w dokumentacji systemu Oracle.

Zastosowanie identyfikatorów :old oraz :new

W wyzwalaczu WygenerujIDStudenta, którego kod przedstawiono w poniższym przykładzie, wykorzystano wartość :new. Jest to wyzwalacz typu BEFORE na poziomie instrukcji INSERT i ma służyć do wypełniania pola ID tabeli studenci wartością generowaną z sekwencji studenci_sekwencja:

Dostępne na płycie CD jako część skryptu *WygenerujIDstudenta.sql*.

```
CREATE OR REPLACE TRIGGER WygenerujIDStudenta
  BEFORE INSERT OR UPDATE ON studenci
  FOR EACH ROW
BEGIN
  /* Wypełnienie pola ID tabeli studenci następną wartością z sekwencji
     studenci_sekwencja. ID jest kolumną tabeli studenci, zatem
     :new.ID jest prawidłowym odwołaniem. */
  SELECT studenci_sekwencja.nextval
    INTO :new.ID
    FROM dual;
END WygenerujIDStudenta;
```

Wyzwalacz WygenerujIDStudenta faktycznie modyfikuje wartość :new.ID. Jest to jedna z użytecznych cech wartości :new — w czasie wykonywania instrukcji zostaną zastosowane bieżące wartości :new. Dzięki wyzwalaczowi WygenerujIDStudenta można wydać następującą instrukcję INSERT:

Dostępne na płycie CD jako część skryptu *WygenerujIDstudenta.sql*.

```
INSERT INTO studenci (imie, nazwisko)
  VALUES ('Lolita', 'Lazarus');
```

Instrukcja ta jest wykonywana bez wygenerowania błędu. Nawet w przypadku nieokreślenia wartości klucza głównego kolumny ID (co jest wymagane) wyzwalacz uzupełni brakującą wartość. W rzeczywistości w razie zdefiniowania wartości dla kolumny ID wartość ta zostanie zignorowana, ponieważ wyzwalacz ją zmieni. Podobnie będzie po wydaniu poniższej instrukcji:

Dostępne na płycie CD jako część skryptu *WygenerujIDstudenta.sql*.

```
INSERT INTO studenci (ID, imie, nazwisko)
  VALUES (-7, 'Zelda', 'Zoom');
```

Kolumna ID będzie wypełniona wartością studenci_sekwencja.nextval, nie zaś wartością -7.

Z powyższego opisu wynika, że nie można zmienić wartości :new w wyzwalaczu AFTER na poziomie wiersza, ponieważ odpowiednia instrukcja została już wykonana. Uogólniając, wartość :new jest modyfikowana tylko w wyzwalaczu BEFORE na poziomie wiersza, a wartość :old nigdy nie jest modyfikowana, tylko odczytywana.

Rekordy :new oraz :old są prawidłowe tylko wewnątrz wyzwalaczy na poziomie wiersza. Przy próbie odwołania się do tych wartości wewnątrz wyzwalacza na poziomie instrukcji wystąpi błąd kompilacji. Wyzwalacz na poziomie instrukcji jest wykonywany tylko raz, nawet jeżeli dana instrukcja przetwarza wiele wierszy, a zatem wartości :new oraz :old nie mają żadnego znaczenia. W takiej sytuacji nie istnieje bowiem żaden wiersz, do którego wartości te mogłyby się odwołać.

Pseudorekordy

Chociaż wartości :new oraz :old są syntaktycznie traktowane jako rekordy typu tabela_wyzwalajaca%ROWTYPE, to jednak w rzeczywistości nimi nie są. W efekcie operacje prawidłowe dla rekordów nie są poprawne dla wartości :new oraz :old, np. nie można przypisywać ich wartości jako całych rekordów, jedynie uzyskać dostęp do indywidualnych pól. Właściwość tę zilustrowano w poniższym przykładzie:

Dostępne na płycie CD jako skrypt *Pseudorekordy.sql*.

```
CREATE OR REPLACE TRIGGER UsunTymcz
  BEFORE DELETE ON tabela_tymcz
  FOR EACH ROW
DECLARE
  z_RekTymcz tabela_tymcz%ROWTYPE;
BEGIN
  /* To nie jest poprawne przypisanie ponieważ :old nie jest właściwe
     rekordem. */
  z_RekTymcz := :old;

  /* Można jednak uzyskać ten sam efekt, przypisując wartości
    poszczególnych pól pojedynczo. */
  z_RekTymcz.kol_znak := :old.kol_znak;
  z_RekTymcz.kol_num := :old.kol_num;
END UsunTymcz;
```

Ponadto nie jest możliwe przekazanie wartości :old oraz :new do procedur lub funkcji, które przyjmują argumenty typu tabela_wyzwalajaca%ROWTYPE.

Klauzula REFERENCING

Jeżeli jest taka potrzeba, to wykorzystuje się klauzulę REFERENCING w celu nadania innych nazw wartościom :new oraz :old. Klauzula ta występuje po zdarzeniu wyzwalającym, a przed klauzulą WHEN i posiada następującą składnię:

```
REFERENCING [OLD AS nazwa_old] [NEW AS nazwa_new]
```

Wewnątrz treści wyzwalacza można wykorzystywać nazwy *:nazwa_old* oraz *:nazwa_new* zamiast :old i :new. Proszę zauważyć, że w nazwach identyfikatorów korelacji w klauzuli REFERENCING nie używa się dwukropków. Poniżej znajduje się alternatywna wersja wyzwalacza WygenerujIDStudenta, gdzie wykorzystano klauzulę REFERENCING w celu umożliwienia odwoływania się do identyfikatora :new jako :nowy_student:

Dostępne na płycie CD jako część skryptu *WygenerujIDstudenta.sql*.

```
CREATE OR REPLACE TRIGGER WygenerujIDStudenta
  BEFORE INSERT OR UPDATE ON studenci
  REFERENCING new AS nowy_student
  FOR EACH ROW
BEGIN
  /* Wypełnienie pola ID tabeli studenci następną wartością z sekwencji
     studenci_sekwencja. ID jest kolumną tabeli studenci, zatem
     :new.ID jest prawidłowym odwołaniem. */
  SELECT studenci_sekwencja.nextval
    INTO :nowy_student.ID
    FROM dual;
END WygenerujIDStudenta;
```

Klauzula WHEN

Klauzula WHEN jest prawidłowa tylko dla wyzwalaczy na poziomie wiersza. Treść wyzwalacza, o ile istnieje, może być wykonywana tylko dla tych wierszy, które spełniają warunek określony przez tę klauzulę. Poniżej przedstawiono składnię klauzuli WHEN:

```
WHEN warunek
```

W niej *warunek* jest wyrażeniem logicznym (boolowskim), sprawdzanym dla każdego wiersza. Istnieje możliwość odwoływania się do pseudorekordów :new i :old również wewnątrz wyrażenia stanowiącego warunek, ale wtedy nie stosuje się znaku dwukropka. Uwzględnienie znaku dwukropka jest prawidłowe tylko w treści wyzwalacza. Przykładowo, treść wyzwalacza SprawdzZaliczenia jest wykonywana tylko wówczas, gdy liczba bieżących zaliczeń studenta jest większa niż 20:

```
CREATE OR REPLACE TRIGGER SprawdzZaliczenia
  BEFORE INSERT OR UPDATE OF biezace_zaliczenia ON studenci
  FOR EACH ROW
  WHEN (new.biezace_zaliczenia > 20)
BEGIN
  /* Tutaj powinna się znaleźć treść wyzwalacza */
END;
```

Wyzwalacz SprawdzZaliczenia można także zapisać w następujący sposób:

```
CREATE OR REPLACE TRIGGER SprawdzZaliczenia
  BEFORE INSERT OR UPDATE OF biezace_zaliczenia ON studenci
  FOR EACH ROW
BEGIN
  IF :new.biezace_zaliczenia > 20 THEN
  /* Tutaj powinna się znaleźć treść wyzwalacza */
  END IF;
END;
```

Korzystanie z predykatów wyzwalaczy INSERTING, UPDATING oraz DELETING

Omawiany już w niniejszym rozdziale wyzwalacz UaktualnijStatSpec jest wyzwalaczem na poziomie instrukcji INSERT, UPDATE oraz DELETE. Wewnątrz wyzwalacza tego typu (który jest uruchamiany dla różnych instrukcji DML) występują trzy funkcje logiczne (boolowskie), które mogą służyć do określenia rodzaju operacji. Tymi predykatami są INSERTING, UPDATING oraz DELETING. Sposób ich działania przedstawiono w poniższej tabeli.

Predykat	Działanie
INSERTING	TRUE, jeżeli instrukcją wyzwalającą jest INSERT, w przeciwnym razie FALSE.
UPDATING	TRUE, jeżeli instrukcją wyzwalającą jest UPDATE, w przeciwnym razie FALSE.
DELETING	TRUE, jeżeli instrukcją wyzwalającą jest DELETE, w przeciwnym razie FALSE.

W systemie Oracle8i zdefiniowano dodatkowe funkcje, podobne do predykatów wyzwalacza, które można wywołać z treści wyzwalacza. Więcej informacji na ten temat znajduje się w podrozdziale „Funkcje — atrybuty zdarzeń", w dalszej części niniejszego rozdziału.

W wyzwalaczu RejestrujZmianyZS wykorzystano powyższe predykaty w celu zapisu wszystkich zmian dokonywanych w tabeli zarejestrowani_studenci. Oprócz tego wyzwalacz zapisuje identyfikator użytkownika wprowadzającego zmiany. Poszczególne rekordy są przechowywane w tabeli ZS_Audyt. Poniżej przedstawiono kod służący do jej utworzenia:

Dostępne na płycie CD jako część skryptu *tabele.sql*.

```
CREATE TABLE ZS_audyt (
  zmiana_typ       CHAR(1)     NOT NULL,
  zmieniono_przez  VARCHAR2(8) NOT NULL,
  data_czas        DATE        NOT NULL,
  stary_student_id NUMBER(5),
  stary_wydzial    CHAR(3),
  stary_kurs       NUMBER(3),
  stara_ocena      CHAR(1),
  nowy_student_id  NUMBER(5),
  nowy_wydzial     CHAR(3),
  nowy_kurs        NUMBER(3),
  nowa_ocena       CHAR(1)
  );
```

Wyzwalacz RejestrujZmianyZS jest tworzony za pomocą poniższego kodu:

Dostępne na płycie CD w skrypcie *RejestrujZmianyZS.sql*.

```
CREATE OR REPLACE TRIGGER RejestrujZmianyZS
  BEFORE INSERT OR DELETE OR UPDATE ON zarejestrowani_studenci
  FOR EACH ROW
```

```
DECLARE
  z_ZmianaTyp CHAR(1);
BEGIN
  /* Zastosowano 'I' dla instrukcji INSERT, 'D' dla instrukcji DELETE oraz
     'U' dla instrukcji UPDATE. */
  IF INSERTING THEN
    z_ZmianaTyp := 'I';
  ELSIF UPDATING THEN
    z_ZmianaTyp := 'U';
  ELSE
    z_ZmianaTyp := 'D';
  END IF;

  /* Zapisano w tabeli ZS_Audyt wszystkie zmiany dokonane w tabeli
     zarejestrowani_studenci. Zastosowano funkcję SYSDATE do wygenerowania datownika
     oraz USER w celu uzyskania identyfikatora bieżącego użytkownika. */
  INSERT INTO ZS_Audyt
    (zmiana_typ, zmieniono_przez, data_czas,
     stary_student_id, stary_wydzial, stary_kurs, stara_ocena,
     nowy_student_id, nowy_wydzial, nowy_kurs, nowa_ocena)
  VALUES
    (z_ZmianaTyp, USER, SYSDATE,
     :old.student_id, :old.wydzial, :old.kurs, :old.ocena,
     :new.student_id, :new.wydzial, :new.kurs, :new.ocena);
END RejestrujZmianyZS;
```

Wyzwalacze zwykle są używane do prowadzenia kontroli zmian dokonywanych w danych, jak następowało w przypadku powyższego wyzwalacza `RejestrujZmianyZS`. Chociaż takie działanie jest dostępne jako jedna z właściwości bazy danych, to jednak zastosowanie wyzwalaczy pozwala na bardziej elastyczną kontrolę. Wyzwalacz `RejestrujZmianyZS` można zmodyfikować i prowadzić, przykładowo, zapis zmian wprowadzanych tylko przez niektórych użytkowników. Mógłby również sprawdzać, czy użytkownicy są uprawnieni do dokonywania zmian, a gdyby tak nie było, wywołać błąd (za pomocą procedury `RAISE_APPLICATION_ERROR`).

Tworzenie wyzwalaczy zastępujących

Inaczej niż wyzwalacze DML, które uruchamiają się dodatkowo, oprócz instrukcji `INSERT`, `UPDATE` lub `DELETE` (przed tymi operacjami lub po nich), wyzwalacze zastępujące uruchamiają się zamiast operacji DML. Ponadto zastępujące można definiować wyłącznie dla perspektyw, podczas gdy wyzwalacze DML są definiowane dla tabel. Wyzwalacze zastępujące wykorzystuje się w dwóch przypadkach:

- w celu umożliwienia modyfikowania perspektyw (bez wyzwalaczy zastępujących byłoby to niemożliwe),
- w celu modyfikowania kolumn tabeli zagnieżdżonej będącej kolumną w perspektywie.

Pierwszy z tych przypadków zostanie omówiony w tym podrozdziale. Więcej informacji na temat tabel zagnieżdżonych znajduje się w rozdziale 8.

Perspektywy modyfikowalne a perspektywy niemodyfikowalne

Perspektywa modyfikowalna to taka, dla której można wydać instrukcję DML. Ogólnie — perspektywa jest modyfikowalna, jeżeli nie zawiera żadnego z poniższych elementów:

- operatorów zbioru (`UNION`, `UNION ALL`, `MINUS`),
- funkcji agregacji (`SUM`, `AVG` itp.),
- klauzul `GROUP BY`, `CONNECT BY` lub `START WITH`,
- operatora `DISTINCT`,
- złączeń.

Istnieją jednak perspektywy zawierające złączenia, które są modyfikowalne. Następuje to wówczas, gdy operacja DML dla tej perspektywy modyfikuje tylko jedną tabelę bazową w danym czasie oraz jeżeli instrukcja DML spełnia warunki wymienione w tabeli 11.3 (więcej informacji na temat modyfikowalnych i niemodyfikowalnych perspektyw ze złączeniami znajduje się w *Oracle Concepts*). Jeżeli perspektywa jest niemodyfikowalna, można zapisać dla niej wyzwalacz zastępujący, który wykonuje pożądane działania, tzn. umożliwia jej modyfikację. Wyzwalacz zastępujący można także zapisać dla perspektywy modyfikowalnej w przypadku, gdy istnieje potrzeba wykonania dodatkowych działań.

Tabela 11.3. *Modyfikowalne perspektywy ze złączeniami*

Operacja DML	Dozwolona, jeżeli:
`INSERT`	Instrukcja nie odnosi się jawnie lub niejawnie do kolumn tabeli bez zachowania kluczy.
`UPDATE`	Zmodyfikowane kolumny są odwzorowane na kolumny tabeli z zachowaniem kluczy.
`DELETE`	W złączeniu istnieje dokładnie jedna tabela z zachowaniem kluczy.

Tabela 11.3 odnosi się do tabel z zachowaniem kluczy (*key-preserved tables*). *Tabela z zachowaniem kluczy* to taka, w której po złączeniu z inną tabelą, klucze w tej oryginalnej są także kluczami w powstałym złączeniu. Więcej informacji na temat tabel z zachowaniem kluczy znajduje się w *Application Developer's Guide — Fundamentals*.

Przykład wyzwalacza zastępującego

Proszę przeanalizować perspektywę `grupy_pokoje`, którą wprowadzono we wcześniejszej części tego rozdziału:

Dostępne na płycie CD jako część skryptu *Zamiast.sql*.

```
CREATE OR REPLACE VIEW grupy_pokoje AS
  SELECT wydzial, kurs, budynek, pokoj_numer
  FROM pokoje, grupy
  WHERE pokoje.pokoj_id = grupy.pokoj_id;
```

Jak można się było wcześniej przekonać, przeprowadzenie operacji `INSERT` dla tej perspektywy jest niedozwolone. Chociaż wykonywanie operacji `UPDATE` i `DELETE` jest dla niej poprawne, to jednak nie jest pewne, czy operacje te zostaną dobrze wykonane. Przykładowo, instrukcja `DELETE FROM grupy_pokoje` spowoduje usunięcie wierszy z tabeli grupy.

Jakie działania instrukcji DML w odniesieniu do perspektywy grupy_pokoje jest prawidłowe? To zależy od ustanowionych reguł. Warto jednak założyć, że operacje te mają następujące znaczenie:

Operacja	Znaczenie
INSERT	Przypisuje nowo wprowadzoną grupę do nowo wprowadzonego pokoju. Wynikiem tego działania jest uaktualnienie tabeli grupy.
UPDATE	Modyfikacja pokoju przypisanego do grupy. Wynikiem tego działania może być modyfikacja tabeli grupy lub pokoje, w zależności od tego, które kolumny perspektywy grupy_pokoje zmodyfikowano.
DELETE	Wyzerowanie identyfikatora pokoju z usuniętej grupy. Powoduje to uaktualnienie tabeli grupy i ustawienie identyfikatora na wartość NULL.

Wymienione wyżej reguły wymusza wyzwalacz GrupyPokojeZamiast, dzięki czemu operacje DML dla perspektywy grupy_pokoje mogą być wykonane poprawnie. Jest to pełniejsza wersja wyzwalacza InsertGrupyPokoje, który pokazano w pierwszym podrozdziale niniejszego rozdziału:

Dostępne na płycie CD w skrypcie *GrupyPokojeZamiast.sql*.

```
CREATE OR REPLACE TRIGGER GrupyPokojeZamiast
  INSTEAD OF INSERT OR UPDATE OR DELETE ON grupy_pokoje
  FOR EACH ROW
DECLARE
  z_pokojID pokoje.pokoj_id%TYPE;
  z_UaktualnijGrupy BOOLEAN := FALSE;
  z_UaktualnijPokoje BOOLEAN := FALSE;

  -- Funkcja lokalna zwracająca identyfikator pokoju na podstawie numeru budynku
  -- oraz numeru pokoju.  Funkcja zwraca błąd ORA-20000, jeżeli nie znaleziono
  -- numeru budynku lub numeru pokoju.
  FUNCTION pobierzIDpokoju(p_Budynek IN pokoje.budynek%TYPE,
                           p_pokoj IN pokoje.pokoj_numer%TYPE)
    RETURN pokoje.pokoj_id%TYPE IS

    z_pokojID pokoje.pokoj_id%TYPE;
  BEGIN
    SELECT pokoj_id
      INTO z_pokojID
      FROM pokoje
      WHERE budynek = p_Budynek
      AND pokoj_numer = p_pokoj;
    RETURN z_pokojID;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR(-20000, 'Nie znaleziono pokoju');
  END pobierzIDpokoju;

  -- Procedura lokalna sprawdzająca, czy grupa identyfikowana przez
  -- p_Wydzial oraz p_Kurs istnieje.  Jeżeli nie, zgłaszany jest błąd
  -- ORA-20001.
```

```
  PROCEDURE sprawdzGrupe(p_Wydzial IN grupy.wydzial%TYPE,
                         p_Kurs IN grupy.kurs%TYPE) IS
    z_Pomocnicza NUMBER;
  BEGIN
    SELECT 0
      INTO z_Pomocnicza
      FROM grupy
      WHERE wydzial = p_Wydzial
      AND kurs = p_Kurs;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR(-20001,
        p_Wydzial || ' ' || p_Kurs || ' nie istnieje');
  END sprawdzGrupe;

BEGIN
  IF INSERTING THEN
    -- Ten fragment w zasadzie przypisuje grupę do określonego pokoju. Logika tego
    -- fragmentu jest identyczna z przypadkiem "uaktualnij pokoje" poniżej:  Najpierw
    -- określa się identyfikator pokoju:
      z_pokojID := pobierzIDpokoju(:new.budynek, :new.pokoj_numer);

      -- Następnie uaktualnia tabelę grupy, wprowadzając nowy identyfikator.
      UPDATE GRUPY
        SET pokoj_id = z_pokojID
        WHERE wydzial = :new.wydzial
        AND kurs = :new.kurs;

  ELSIF UPDATING THEN
    -- Sprawdzenie, czy uaktualnia się tabelę grupy czy tabelę pokoje.
    z_UaktualnijGrupy := (:new.wydzial != :old.wydzial) OR
                         (:new.kurs != :old.kurs);
    z_UaktualnijPokoje := (:new.budynek != :old.budynek) OR
                       (:new.pokoj_numer != :old.pokoj_numer);

    IF (z_UaktualnijGrupy) THEN
      -- W takim przypadku zmienia się grupę przypisaną do określonego
      -- pokoju.  Najpierw sprawdzając, czy nowa grupa istnieje.
      sprawdzGrupe(:new.wydzial, :new.kurs);

      -- Pobranie identyfikatora pokoju,
      z_pokojID := pobierzIDpokoju(:old.budynek, :old.pokoj_numer);

      -- Wyzerowanie pokoju dla starej grupy,
      UPDATE grupy
        SET pokoj_id = NULL
        WHERE wydzial = :old.wydzial
        AND kurs = :old.kurs;

      -- Wreszcie przypisanie starego pokoju do nowej grupy.
      UPDATE grupy
        SET pokoj_id = z_pokojID
        WHERE wydzial = :new.wydzial
        AND kurs = :new.kurs;
    END IF;

    IF z_UaktualnijPokoje THEN
      -- W tym fragmencie następuje zmiana pokoju dla danej grupy.
```

```
      -- Logika tego fragmentu jest identyczna z przypadkiem "wstawiania" powyżej, poza
      -- tym że następuje uaktualnienie tabeli grupy wartością :old zamiast :new.
      -- Najpierw należy określić identyfikator pokoju.
      z_pokojID := pobierzIDpokoju(:new.budynek, :new.pokoj_numer);

      -- Uaktualnienie tabeli grupy poprzez wprowadzenie nowego identyfikatora.
      UPDATE GRUPY
        SET pokoj_id = z_pokojID
        WHERE wydzial = :old.wydzial
        AND kurs = :old.kurs;
    END IF;

  ELSE
    -- W tym fragmencie następuje wyzerowanie przypisania grupy do pokoju bez
    -- konieczności usuwania wierszy z powiązanych tabel.
    UPDATE grupy
      SET pokoj_id = NULL
      WHERE wydzial = :old.wydzial
      AND kurs = :old.kurs;
  END IF;
END GrupyPokojeZamiast;
```

Klauzula `FOR EACH ROW` **dla wyzwalacza zastępującego jest opcjonalna. Wszystkie wyzwalacze zastępujące są wyzwalaczami na poziomie wiersza, niezależnie od występowania klauzuli.**

Wyzwalacz `GrupyPokojeZamiast` wykorzystuje predykaty wyzwalaczy w celu zidentyfikowania wykonywanej operacji DML i podjęcia odpowiednich działań. Na rysunku 11.1 przedstawiono początkową zawartość tabel `grupy`, `pokoje` oraz perspektywy `grupy_pokoje`.

Rysunek 11.1. *Początkowa zawartość tabel grupy, pokoje oraz perspektywy grupy_pokoje*

grupy

Wydz.	Kurs	Id pokoju
HIS	101	20000
INF	101	20001
EKN	203	20002
INF	102	20003
HIS	301	20004
MUZ	410	20005
MUZ	100	
EKN	101	20007
ŻYW	307	20008

pokoje

Id pokoju	Budynek	Numer pokoju
20000	Budynek 7	201
20001	Budynek 6	101
20002	Budynek 6	150
20003	Budynek 6	160
20004	Budynek 6	170
20005	Budynek Muzyki	100
20006	Budynek Muzyki	200
20007	Budynek 7	300
20008	Budynek 7	310

grupy_pokoje

Wydział	Kurs	Budynek	Numer pokoju
HIS	101	Budynek 7	201
INF	101	Budynek 6	101
EKN	203	Budynek 6	150
INF	102	Budynek 6	160
HIS	301	Budynek 6	170
MUZ	410	Budynek Muzyki	100
EKN	101	Budynek 7	300
ŻYW	307	Budynek 7	310

Proszę sobie wyobrazić, że wydano następnie taką instrukcję `INSERT`:

Dostępne na płycie CD jako część skryptu *GrupyPokojeZamiast.sql*.

```
INSERT INTO grupy_pokoje
  VALUES ('MUZ', 100, 'Budynek Muzyki', 200);
```

Wyzwalacz powoduje, że tabela `grupy` jest zmieniana zgodnie z nowym przypisaniem grupy do pokoju. Zilustrowano to na rysunku 11.2.

Rysunek 11.2. *Zawartość tabel i perspektywy po wykonaniu instrukcji INSERT*

grupy

Wydz.	Kurs	Id pokoju
HIS	101	20000
INF	101	20001
EKN	203	20002
INF	102	20003
HIS	301	20004
MUZ	410	20005
MUZ	**100**	**20006**
EKN	101	20007
ŻYW	307	20008

pokoje

Id pokoju	Budynek	Numer pokoju
20000	Budynek 7	201
20001	Budynek 6	101
20002	Budynek 6	150
20003	Budynek 6	160
20004	Budynek 6	170
20005	Budynek Muzyki	100
20006	Budynek Muzyki	200
20007	Budynek 7	300
20008	Budynek 7	310

grupy_pokoje

Wydział	Kurs	Budynek	Numer pokoju
HIS	101	Budynek 7	201
INF	101	Budynek 6	101
EKN	203	Budynek 6	150
INF	102	Budynek 6	160
HIS	301	Budynek 6	170
MUZ	410	Budynek Muzyki	100
EKN	101	Budynek 7	300
ŻYW	307	Budynek 7	310
MUZ	**100**	**Budynek Muzyki**	**200**

Następnie wydano następującą instrukcję `UPDATE`:

Dostępne na płycie CD jako część skryptu *GrupyPokojeZamiast.sql*.

```
UPDATE grupy_pokoje
  SET wydzial = 'ŻYW', kurs = 307
  WHERE budynek = 'Budynek 7' AND pokoj_numer = 201;
```

Tabela `grupy` została uaktualniona zgodnie z dokonanymi, nowymi zmianami. Teraz grupa `Historia 101` nie ma przypisanego pokoju, a `Żywienie 307` ma przypisany pokój, który pierwotnie przynależał grupie `Historia 101`. Sytuację tę zilustrowano na rysunku 11.3.

Rysunek 11.3. *Zawartość tabel i perspektywy po wykonaniu instrukcji UPDATE*

grupy

Wydz.	Kurs	Id pokoju
HIS	101	NULL
INF	101	20001
EKN	203	20002
INF	102	20003
HIS	301	20004
MUZ	410	20005
MUZ	100	20006
EKN	101	20007
ŻYW	**307**	**20000**

pokoje

Id pokoju	Budynek	Numer pokoju
20000	Budynek 7	201
20001	Budynek 6	101
20002	Budynek 6	150
20003	Budynek 6	160
20004	Budynek 6	170
20005	Budynek Muzyki	100
20006	Budynek Muzyki	200
20007	Budynek 7	300
20008	Budynek 7	310

grupy_pokoje

Wydział	Kurs	Budynek	Numer pokoju
HIS	101	Budynek 7	201
INF	101	Budynek 6	101
EKN	203	Budynek 6	150
INF	102	Budynek 6	160
MUZ	410	Budynek Muzyki	100
EKN	101	Budynek 7	300
ŻYW	**307**	**Budynek 7**	**201**
MUZ	100	Budynek Muzyki	200

Wreszcie można sobie wyobrazić, że wydano następującą instrukcję `DELETE`:

Dostępne na płycie CD jako część skryptu *GrupyPokojeZamiast.sql*.

```
DELETE FROM grupy_pokoje
  WHERE budynek = 'Budynek 6';
```

Zmiany w tabeli `grupy` polegają teraz na ustawieniu wartości pola `pokoj_ID` na wartość `NULL` dla tych grup, którym pierwotnie przypisano Budynek 6. Sytuację tę pokazano na rysunku 11.4. Proszę zwrócić uwagę, że podczas wykonywania wszystkich tych operacji DML, tabela `pokoje` pozostała niezmieniona, a uaktualniano jedynie tabelę `grupy`.

Rysunek 11.4. *Zawartość tabel i perspektywy po wykonaniu instrukcji DELETE*

grupy

Wydz.	Kurs	Id pokoju
ŻYW	307	20000
INF	101	
EKN	203	
INF	102	
HIS	201	
MUZ	410	20005
MUZ	100	20006
EKN	101	20007
HIS	101	

pokoje

Id pokoju	Budynek	Numer pokoju
20000	Budynek 7	201
20001	Budynek 6	101
20002	Budynek 6	150
20003	Budynek 6	160
20004	Budynek 6	170
20005	Budynek Muzyki	100
20006	Budynek Muzyki	200
20007	Budynek 7	300
20008	Budynek 7	310

grupy_pokoje

Wydział	Kurs	Budynek	Numer pokoju
MUZ	410	Budynek Muzyki	100
EKN	101	Budynek 7	300
ŻYW	307	Budynek 7	310
MUZ	100	Budynek Muzyki	200

Tworzenie wyzwalaczy systemowych

W poprzednim podrozdziale Czytelnik mógł się przekonać, że zarówno wyzwalacze DML, jak i zastępujące uruchamiają się razem ze zdarzeniami DML, dokładniej instrukcjami INSERT, UPDATE lub DELETE (bądź też zamiast nich). Natomiast wyzwalacze systemowe uruchamiają się w przypadku wystąpienia dwóch różnych rodzajów zdarzeń: DDL lub bazy danych. Zdarzenia DDL obejmują instrukcje CREATE, ALTER lub DROP, natomiast zdarzenia bazy danych obejmują uruchomienie (zatrzymanie) serwera, rejestrowanie (wyrejestrowanie) użytkownika oraz błąd serwera. Składnia utworzenia wyzwalacza systemowego jest następująca:

```
CREATE [OR REPLACE] TRIGGER [schemat.]nazwa_wyzwalacza
   {BEFORE|AFTER}
   {lista_zdarzen_ddl| lista_zdarzen_bazy_danych}
   ON {DATABASE | [schemat.]SCHEMA}
   [klauzula_when]
   tresc_wyzwalacza;
```

— *lista_zdarzen_ddl* zawiera jedno zdarzenie DDL lub więcej (rozdzielonych słowem kluczowym OR) natomiast *lista_zdarzen_bazy_danych* zawiera jedno zdarzenie bazy danych lub więcej (rozdzielonych słowem kluczowym OR).

W tabeli 11.4 opisano zdarzenia DDL oraz zdarzenia bazy danych wraz z dozwolonym czasem wykonania (BEFORE lub AFTER). Utworzenie systemowego wyzwalacza zastępującego jest niedozwolone. Instrukcji TRUNCATE nie odpowiada żadne zdarzenie bazy danych.

Aby móc utworzyć wyzwalacz systemowy, trzeba posiadać uprawnienie systemowe ADMINISTER DATABASE TRIGGER. Więcej informacji na ten temat znajduje się w podrozdziale „Uprawnienia dotyczące wyzwalaczy”.

Wyzwalacze bazy danych a wyzwalacze schematu

Wyzwalacz systemowy można zdefiniować na poziomie bazy danych lub na poziomie schematu. Wyzwalacz na poziomie bazy danych uruchomi się za każdym razem, kiedy nastąpi zdarzenie wyzwalające, natomiast wyzwalacz poziomu schematu uruchomi się tylko wtedy, gdy zajdzie zdarzenie wyzwalające dla określonego schematu. Słowa kluczowe DATABASE lub SCHEMA określają poziom dla wybranego wyzwalacza systemowego.

Tabela 11.4. *Systemowe zdarzenia DDL oraz zdarzenia bazy danych*

Zdarzenie	Dozwolony parametr czasowy	Opis
STARTUP	AFTER	Uruchamia się w momencie uruchomienia bazy danych.
SHUTDOWN	BEFORE	Uruchamia się w momencie zatrzymania bazy danych. Zdarzenie to może nie nastąpić w przypadku zatrzymania bazy danych w trybie awarii (np. anulowanie procesu zatrzymywania).
SERVERERROR	AFTER	Uruchamia się w przypadku wystąpienia błędu.
LOGON	AFTER	Uruchamia się po pomyślnym połączeniu użytkownika z bazą danych.
LOGOFF	BEFORE	Uruchamia się na początku procesu wyrejestrowania użytkownika z bazy danych.
CREATE	BEFORE, AFTER	Uruchamia się przed utworzeniem obiektu schematu lub po.
DROP	BEFORE, AFTER	Uruchamia się przed usunięciem obiektu schematu lub po.
ALTER	BEFORE, AFTER	Uruchamia się przed uaktualnieniem obiektu schematu lub po.

Jeżeli zastosuje się słowo kluczowe SCHEMA, a nie określi schematu, domyślnym będzie ten schemat, który zawiera wyzwalacz. Na przykład, będąc połączonym jako UzytkownikA, utworzono następujący wyzwalacz:

Dostępne na płycie CD jako część skryptu *SchematBazyDanych.sql*.

```
CREATE OR REPLACE TRIGGER ZarejestrujPolaczeniaUzytkA
  AFTER LOGON ON SCHEMA
BEGIN
  INSERT INTO przyklad.tabela_tymcz
    VALUES (1, 'uruchomiono wyzwalacz ZarejestrujPolaczeniaUzytkA!');
END ZarejestrujPolaczeniaUzytkA;
```

Wyzwalacz ZarejestrujPolaczeniaUzytkA każdorazowo zapisze w tabeli tabela_tymcz fakt połączenia użytkownika UzytkownikA. Można stworzyć podobny mechanizm dla użytkownika UzytkownikB, budując poniższy wyzwalacz z pozycji połączonego jako UzytkownikB:

Dostępne na płycie CD jako część skryptu *SchematBazaDanych.sql*.

```
CREATE OR REPLACE TRIGGER ZarejestrujPolaczeniaUzytkB
  AFTER LOGON ON SCHEMA
BEGIN
  INSERT INTO przyklad.tabela_tymcz
    VALUES (1, 'uruchomiono wyzwalacz ZarejestrujPolaczeniaUzytkB!');
END ZarejestrujPolaczeniaUzytkB;
```

Wreszcie można utworzyć następujący wyzwalacz, będąc połączonym jako użytkownik przyklad. Wyzwalacz ZarejestrujPolaczWszystkich zarejestruje wszystkie połączenia z bazą danych, ponieważ jest to wyzwalacz poziomu bazy danych:

Dostępne na płycie CD jako część skryptu *SchematBazaDanych.sql*.

```
CREATE OR REPLACE TRIGGER ZarejestrujPolaczWszystkich
  AFTER LOGON ON DATABASE
BEGIN
  INSERT INTO przyklad.tabela_tymcz
    VALUES (1, 'uruchomiono wyzwalacz ZarejestrujPolaczWszystkich!');
END ZarejestrujPolaczWszystkich;
```

Przed uruchomieniem powyższych przykładów najpierw należy utworzyć użytkowników `UzytkownikA` oraz `UzytkownikB` i nadać im odpowiednie uprawnienia. Więcej informacji na ten temat znajduje się w skrypcie *SchematBazyDanych.sql*.

Analizując następującą sesję programu SQL*Plus, można teraz zaobserwować efekty działania poszczególnych wyzwalaczy:

Dostępne na płycie CD jako część skryptu *SchematBazyDanych.sql*.

```
SQL> Connect UzytkownikA/UzytkownikA
Połączony.

SQL> Connect UzytkownikB/UzytkownikB
Połączony.

SQL> Connect przyklad/przyklad
Połączony.

SQL> SELECT * FROM tabela_tymcz;
KOL_NUM    KOL_ZNAK
---------  ------------------------------------------------------------
          3   uruchomiono wyzwalacz ZarejestrujPolaczWszystkich!
          2   uruchomiono wyzwalacz ZarejestrujPolaczeniaUzytkB!
          3   uruchomiono wyzwalacz ZarejestrujPolaczWszystkich!
          3   uruchomiono wyzwalacz ZarejestrujPolaczWszystkich!
          1   uruchomiono wyzwalacz ZarejestrujPolaczUzytkA!
```

Wyzwalacz `ZarejestrujPolaczWszystkich` uruchomił się trzy razy (po jednym razie dla każdego z połączeń), natomiast wyzwalacze `ZarejestrujPolaczeniaUzytkA` oraz `ZarejestrujPolaczeniaUzytkB`, jak należało się spodziewać, uruchomiły się tylko raz.

Wyzwalacze `STARTUP` oraz `SHUTDOWN` można definiować jedynie na poziomie bazy danych. Tworzenie ich na poziomie schematu nie jest niepoprawne, a wyzwalacze po prostu nie uruchomią się.

Funkoje — atrybuty zdarzeń

Dla wyzwalaczy systemowych istnieje kilka funkcji — atrybutów zdarzeń. Podobnie jak predykaty wyzwalaczy (`INSERTING`, `UPDATING` oraz `DELETING`), pozwalają one uzyskać informacje na temat zdarzenia wyzwalającego w treści wyzwalacza. Chociaż wywołanie tych

funkcji z innych bloków PL/SQL jest poprawne (niekoniecznie w treści wyzwalacza systemowego), to jednak funkcje te nie zawsze zwrócą poprawne wyniki. Funkcje — atrybuty zdarzeń opisano w tabeli 11.5.

Tabela 11.5. *Funkcje — atrybuty zdarzeń*

Funkcja — atrybut	Typ danych	Zdarzenia, dla których funkcja ma zastosowanie	Opis
SYSEVENT	VARCHAR2(20)	Wszystkie zdarzenia.	Zwraca zdarzenie systemowe, które uruchomiło wyzwalacz.
INSTANCE_NUM	NUMBER	Wszystkie zdarzenia.	Zwraca bieżący numer egzemplarza. O ile nie pracuje się z klastrami *Oracle Real Application Clusters*, będzie to zawsze wartość 1.
DATABASE_NAME	VARCHAR2(50)	Wszystkie zdarzenia.	Zwraca nazwę bieżącej bazy danych.
SERVER_ERROR	NUMBER	SERVERERROR	Przyjmuje pojedynczy argument typu NUMBER. Zwraca błąd znajdujący się na stosie błędów na pozycji wskazywanej przez argument. Wartość 1 oznacza wierzchołek stosu.
IS_SERVERERROR	BOOLEAN	SERVERERROR	Pobiera numer błędu jako argument i zwraca wartość TRUE, jeżeli wskazywany błąd systemu Oracle znajduje się na stosie błędów.
LOGIN_USER	VARCHAR2(30)	Wszystkie zdarzenia.	Zwraca identyfikator użytkownika, który uruchomił wyzwalacz.
DICTIONARY_OBJ_TYPE	VARCHAR2(20)	CREATE, DROP, ALTER	Zwraca typ obiektu słownika, dla którego wykonano operację DDL, która spowodowała uruchomienie wyzwalacza.
DICTIONARY_OBJ_NAME	VARCHAR2(30)	CREATE, DROP, ALTER	Zwraca nazwę obiektu słownika, dla którego wykonano operację DDL, która spowodowała uruchomienie wyzwalacza.
DICTIONARY_OBJ_OWNER	VARCHAR2(30)	CREATE, DROP, ALTER	Zwraca właściciela obiektu słownika, dla którego wykonano operację DDL, która spowodowała uruchomienie wyzwalacza.
DES_ENCRYPTED_PASSWORD	VARCHAR2(30)	CREATE lub ALTER USER	Zwraca hasło zaszyfrowane algorytmem DES utworzonego użytkownika lub użytkownika, dla którego zmodyfikowano dane.

Niektóre z funkcji — atrybutów wykorzystano w wyzwalaczu ZarejestrujOperTworzenia, który pojawił się na początku tego rozdziału. Inaczej niż predykaty wyzwalaczy, funkcje — atrybuty są samodzielnymi funkcjami PL/SQL, których właścicielem jest SYS. Nie zdefiniowano dla nich domyślnego synonimu, zatem w celu poprawnej identyfikacji należy poprzedzić je prefiksem SYS:

Dostępne na płycie CD jako część skryptu *ZarejestrujOperTworzenia.sql*.

```
CREATE OR REPLACE TRIGGER ZarejestrujOperTworzenia
  AFTER CREATE ON SCHEMA
```

```
BEGIN
  INSERT INTO tworzenie_ddl (id_uzytkownika, typ_obiektu, nazwa_obiektu,
                             wlasciciel_obiektu, data_utworzenia)
    VALUES (USER, SYS.DICTIONARY_OBJ_TYPE, SYS.DICTIONARY_OBJ_NAME,
            SYS.DICTIONARY_OBJ_OWNER, SYSDATE);
END ZarejestrujOperTworzenia;
```

Wykorzystanie zdarzenia SERVERERROR

Zdarzenie SERVERERROR można zastosować w celu śledzenia błędów powstałych w bazie danych. Kod błędu jest dostępny wewnątrz wyzwalacza za pośrednictwem funkcji — atrybutu SERVER_ERROR. Funkcja ta pozwala na zidentyfikowanie kodu błędów znajdujących się na stosie. Nie można jednak uzyskać komunikatów skojarzonych z tymi kodami.

Można temu zaradzić, posługując się procedurą DBMS_UTILITY.FORMAT_ERROR_STACK. Chociaż sam wyzwalacz nie spowodował błędu, to jednak za pośrednictwem tej procedury stos błędów jest dostępny z poziomu języka PL/SQL. Zilustrowano to w poniższym przykładzie, gdzie błędy są zapisywane w specjalnie do tego celu zaprojektowanej tabeli. Oto jej schemat:

Dostępne na płycie CD jako część skryptu *ZarejestruBledy.sql*.

```
CREATE TABLE rejestracja_bledow (
   datownik             DATE,
   nazwa_uzytkownika    VARCHAR2(30),
   egzemplarz           NUMBER,
   nazwa_bazy_danych    VARCHAR2(50),
   stos_bledow          VARCHAR2(2000)
   );
```

Można teraz utworzyć wyzwalacz, który wstawia dane do powyższej tabeli:

Dostępne na płycie CD jako część skryptu *ZarejestruBledy.sql*.

```
CREATE OR REPLACE TRIGGER ZarejestrujBledy
    AFTER SERVERERROR ON DATABASE
BEGIN
    INSERT INTO rejestracja_bledow
        VALUES (SYSDATE, SYS.LOGIN_USER, SYS.INSTANCE_NUM, SYS.DATABASE_NAME,
                DBMS_UTILITY.FORMAT_ERROR_STACK);
END ZarejestrujBledy;
```

Na koniec wygeneruje się kilka błędów i sprawdzi, czy wyzwalacz ZarejestrujBledy poprawnie je rejestruje. Proszę zwrócić uwagę, że wyzwalacz przechwytuje błędy w instrukcjach SQL, błędy PL/SQL fazy wykonania oraz błędy kompilacji PL/SQL:

Dostępne na płycie CD jako część skryptu *ZarejestruBledy.sql*.

```
SQL> SELECT * FROM nieistniejaca_tabela;
SELECT * FROM nieistniejaca_tabela
              *
BŁĄD w linii 1:
ORA-00942: tabela lub perspektywa nie istnieje
SQL> BEGIN
  2    INSERT INTO nieistniejaca_tabela VALUES ('Witajcie!');
  3  END;
  4  /
  INSERT INTO nieistniejaca_tabela VALUES ('Witajcie!')
              *
BŁĄD w linii 2:
ORA-06550: linia 2, kolumna 15:
PLS-00201: identyfikator 'NIEISTNIEJACA_TABELA' musi być zdeklarowany
ORA-06550: linia 2, kolumna 3:
PL/SQL: Polecenie SQL odrzucone

SQL> BEGIN
  2    -- To jest błąd syntaktyczny!
  3    DELETE FROM studenci
  4  END;
  5   /
END;
*
BŁĄD w linii 4:
ORA-06550: linia 4, kolumna 1:
PLS-00103: Napotkano na symbol "END" tam, gdzie spodziewano się jednego z następujących:
. @ ; RETURNING_<identyfikator>
<identyfikator ujęty w cudzysłów>, część "WHERE"
Dla umożliwienia kontynuacji zamieniono symbol "END" na ";"

SQL> DECLARE
  2    z_ZmLancuchowa VARCHAR2(2);
  3  BEGIN
  4    -- To jest błąd wykonania!
  5    z_ZmLancuchowar := 'abcdef';
  6  END;
  7   /
DECLARE
*
BŁĄD w linii 1:
ORA-06502: PL/SQL: błąd  numeryczny lub błąd wartości: za mały bufor łańcucha znaków.
ORA-06512: w linii 5

SQL> SELECT *
  2   FROM rejestracja_bledow;

DATOWNIK   NAZWA_UZYTKOWNIKA   EGZEMPLARZ   NAZWA_BAZY_DANYCH
---------  ------------------  -----------  ------------------
ERROR_STACK
------------------------------------------------------------
12-OCT-01  PRZYKLAD                     1  V901
ORA-00942: tabela lub perspektywa nie istnieje.

12-OCT-01  PRZYKLAD                     1  V901
ORA-06550: linia 2, kolumna 15:
PLS-00201: identyfikator 'NIEISTNIEJACA_TABELA' musi być zdeklarowany
ORA-06550: linia 2, kolumna 3:
PL/SQL: Polecenie SQL odrzucne
```

```
12-OCT-01  PRZYKLAD                    1  V901
ORA-06550: linia 4, kolumna 1:
PLS-00103: Napotkano na symbol "END" tam, gdzie spodziewano się jednego z następujących:
. @ ; RETURNING_<identyfikator>
<identyfikator ujęty w cudzysłów>, część "WHERE"
Dla umożliwienia kontynuacji zamieniono symbol "END" na ";"


12-OCT-01  PRZYKLAD                    1  V901
ORA-06502: PL/SQL: błąd numeryczny lub wartości: zbyt mały bufor dla ciągu znaków
ORA-06512: w linii 5
```

Wyzwalacze systemowe a transakcje

W zależności od zdarzenia wyzwalającego zmienia się działanie wyzwalacza systemowego w transakcjach. Wyzwalacz systemowy uruchamia się jako oddzielna transakcja, która jest zatwierdzana po pomyślnym wykonaniu wyzwalacza lub uruchamia się jako część transakcji bieżącego użytkownika. Wyzwalacze STARTUP, SHUTDOWN, SERVERERROR oraz LOGON uruchamiają się jako oddzielne transakcje, natomiast wyzwalacze LOGOFF oraz związane z operacjami DDL uruchamiają się w ramach bieżącej transakcji.

Ważne jest, aby zdać sobie sprawę, że działania wykonane przez wyzwalacz będą zatwierdzone niezależnie od innych czynników. W przypadku wyzwalacza DDL bieżąca transakcja (instrukcja CREATE, ALTER lub DROP) jest zatwierdzana automatycznie, co równocześnie zatwierdza działania wykonane przez wyzwalacz. Działanie wyzwalacza LOGOFF również będzie zatwierdzone jako część końcowej transakcji w sesji.

Ponieważ wyzwalacze systemowe w zasadzie są zatwierdzane mimo wszystko, deklarowanie ich jako autonomicznych nie będzie miało żadnego efektu.

Wyzwalacze systemowe a klauzula WHEN

Podobnie jak w przypadku wyzwalaczy DML, w wyzwalaczach systemowych można stosować klauzulę WHEN w celu określenia warunków uruchomienia wyzwalacza. Istnieją jednak ograniczenia rodzaju warunków, które można określić dla każdego rodzaju wyzwalaczy systemowych. Należą do nich:

- Dla wyzwalaczy STARTUP i SHUTDOWN nie można określać żadnych warunków.
- W wyzwalaczach SERVERERROR można także wykorzystywać zmienną ERRNO w celu śledzenia tylko wybranego błędu.
- W wyzwalaczach LOGON i LOGOFF można sprawdzać identyfikator i nazwę użytkownika, posługując się zmiennymi USERID oraz USERNAME.
- W wyzwalaczach DDL można sprawdzać typ i nazwę modyfikowanego obiektu, a także identyfikator i nazwę użytkownika.

Inne zagadnienia związane z wyzwalaczami

W tym podrozdziale będą omówione dodatkowe zagadnienia związane z wyzwalaczami, tj. przestrzeń nazw dla nazw wyzwalaczy, różnorodne ograniczenia związane z wykorzystaniem wyzwalaczy oraz różne rodzaje treści wyzwalaczy. Podrozdział kończy się omówieniem uprawnień związanych z wyzwalaczami.

Nazwy wyzwalaczy

Przestrzeń nazw dla nazw wyzwalaczy różni się od przestrzeni nazw dla nazw podprogramów. *Przestrzeń nazw* jest zbiorem poprawnych identyfikatorów, dostępnych do wykorzystania jako nazwy obiektu. Procedury, pakiety i tabele wspólnie korzystają z tej samej przestrzeni nazw. Oznacza to, że w ramach schematu bazy danych wszystkie obiekty w tej samej przestrzeni nazw muszą mieć niepowtarzalne nazwy. Przykładowo, nieprawidłowe jest nadawanie tej samej nazwy procedurze i pakietowi.

Wyzwalacze posiadają jednak osobną przestrzeń nazw. Oznacza to, że wyzwalacz może mieć tę samą nazwę, którą tabela lub procedura. Jednak wewnątrz jednego schematu dana nazwa może być zastosowana tylko dla jednego wyzwalacza. Można na przykład utworzyć wyzwalacz o nazwie `statystyka_spec` dla tabeli `statystyka_spec`, jednak utworzenie procedury, która także nazywa się `statystyka_spec`, jest nieprawidłowe. Sytuację taką przedstawiono w poniższej sesji programu SQL*Plus:

Dostępne na płycie CD w skrypcie *TeSameNazwy.sql*.

```
SQL> --Poprawne, ponieważ wyzwalacze i tabele wykorzystują różne przestrzenie nazw
SQL> CREATE OR REPLACE TRIGGER spec_stats
  2    BEFORE INSERT ON spec_stats
  3  BEGIN
  4    INSERT INTO tabela_tymcz (kol_znak)
  5      VALUES ('Wyzwalacz uruchomiony!');
  6  END spec_stats;
  7  /
Wyzwalacz został utworzony.

SQL> -- Niepoprawne, ponieważ procedury i tabele wykorzystują różne przestrzenie nazw
SQL> CREATE OR REPLACE PROCEDURE spec_stats
  2  BEGIN
  3    INSERT INTO tabela_tymcz (kol_znak)
  4      VALUES ('Procedurę wywołano!');
  5  END spec_stats;
  6  /
CREATE OR REPLACE PROCEDURE spec_stats AS
*
BŁĄD w linii 1;
ORA-00955: istniejący obiekt używa juz takiej nazwy
```

Mimo że jest możliwe stosowanie takich samych nazw dla wyzwalacza i tabeli, taki sposób programowania nie jest zalecany. Lepszą metodą jest nadawanie każdemu wyzwalaczowi niepowtarzalnej nazwy, identyfikującej jego funkcję oraz tabelę, dla której został zdefiniowany lub stosowanie wspólnego prefiksu dla wyzwalacza, np. `WYZW_`.

Ograniczenia wyzwalaczy

Treść wyzwalacza jest blokiem PL/SQL (w systemie Oracle8i istnieje możliwość wykorzystywania innych rodzajów treści wyzwalaczy — szczegółowe informacje znajdują się w następnym podrozdziale). Każda instrukcja, która jest poprawna w bloku PL/SQL, jest również właściwa w treści wyzwalacza. Wyzwalacze podlegają następującym ograniczeniom:

- W wyzwalaczu nie można wydawać żadnych instrukcji sterowania transakcją — `COMMIT`, `ROLLBACK` lub `SAVEPOINT`. Kompilator języka PL/SQL zezwoli na utworzenie wyzwalacza zawierającego te instrukcje, ale w czasie uruchamiania wyzwalacza uzyska się błąd. Wynika to z faktu, iż wyzwalacz jest uruchamiany jako część wykonania instrukcji wyzwalającej i należy do tej samej transakcji, do której instrukcja wyzwalająca. Kiedy instrukcja wyzwalająca jest zatwierdzana lub wycofana, efekt działania wyzwalacza jest również zatwierdzany lub wycofany (w systemie Oracle8i oraz w systemach wyższych można utworzyć wyzwalacz, który uruchamia się jako transakcja autonomiczna. W takim przypadku działania wykonywane w wyzwalaczu można zatwierdzić lub wycofać niezależnie od stanu instrukcji wyzwalającej. Więcej informacji na temat transakcji autonomicznych znajduje się w rozdziale 4.).
- Podobnie w procedurach lub funkcjach wywołanych w treści wyzwalacza nie można wydawać żadnych instrukcji sterowania transakcją (chyba że one również zostały zadeklarowane jako transakcje autonomiczne w systemie Oracle8i oraz w wersjach wyższych).
- W treści wyzwalacza nie można deklarować żadnych zmiennych typu `LONG` lub `LONG RAW`. Również wartości `:new` oraz `:old` nie mogą odwoływać się do kolumn typu `LONG` lub `LONG RAW` w tabeli, dla której zdefiniowano wyzwalacz.
- W systemie Oracle8 oraz w wersjach wyższych w kodzie w treści wyzwalacza można odwoływać się i wykorzystywać kolumny typu `LOB` (*Large Objects* — obiekty o dużym rozmiarze), ale nie można modyfikować wartości tych kolumn. Dotyczy to również kolumn obiektowych.

Istnieją także pewne ograniczenia uzyskiwania dostępu do tabeli przez treść wyzwalacza. W zależności od typu wyzwalacza i ograniczeń dla tabeli wybrana tabela może stać się tabelą mutującą. Sytuację taką omówiono szczegółowo w dalszej części niniejszego rozdziału, w podrozdziale „Tabele mutujące".

Treść wyzwalaczy

Przed wydaniem systemu Oracle8i treść wyzwalacza musiała być blokiem PL/SQL. W systemie Oracle8i oraz w wersjach wyższych treść wyzwalacza może się składać z instrukcji `CALL`. Wywołana procedura może być podprogramem składowanym PL/SQL lub osłoną (*wrapper*) dla procedury napisanej w języku C lub Java. Dzięki temu można tworzyć wyzwalacze, w których kod funkcji pisany jest w języku Java. Proszę sobie wyobrazić zarejestrowanie połączenia i rozłączenia z bazą danych w następującej tabeli:

Dostępne na płycie CD jako część skryptu *tabele.sql*.

```
CREATE TABLE audyt_polaczen (
    nazwa_uzytkownika       VARCHAR2(30),
    operacja                VARCHAR2(30),
    datownik                DATE);
```

Do rejestrowania połączeń i rozłączeń z bazą danych można zastosować następujący pakiet:

Dostępne na płycie CD jako część skryptu *PakietDziennika1.sql*.

```
CREATE OR REPLACE PACKAGE PakietDziennika AS
  PROCEDURE RejestrujPolaczenia(p_UzytkID IN VARCHAR2);
  PROCEDURE RejestrujRozlaczenia(p_UzytkID IN VARCHAR2);
END PakietDziennika;


CREATE OR REPLACE PACKAGE BODY PakietDziennika AS
  PROCEDURE RejestrujPolaczenia(p_UzytkID IN VARCHAR2) IS
  BEGIN
    INSERT INTO audyt_polaczen (nazwa_uzytkownika, operacja, datownik)
      VALUES (p_UzytkID, 'POLACZENIE', SYSDATE);
  END RejestrujPolaczenia;

  PROCEDURE RejestrujRozlaczenia(p_UzytkID IN VARCHAR2) IS
  BEGIN
    INSERT INTO audyt_polaczen (nazwa_uzytkownika, operacja, datownik)
      VALUES (p_UzytkID, 'ROZLACZENIE', SYSDATE);
  END RejestrujRozlaczenia;
END PakietDziennika;
```

Obie procedury — `PakietDziennika.RejestrujPolaczenia` **oraz** `PakietDziennika.RejestrujRozlaczenia` — **pobierają nazwę użytkownika jako argument i wstawiają wiersz do tabeli** `audyt_polaczen`. **Procedury te można wywołać z wyzwalaczy** `LOGON` **i** `LOGOFF` **w następujący sposób:**

Dostępne na płycie CD jako skrypt *RejestrowaniePolaczen.sql*.

```
CREATE OR REPLACE TRIGGER RejestrowaniePolaczen
   AFTER LOGON ON DATABASE
   CALL PakietDziennika.RejestrujPolaczenia(SYS.LOGIN_USER)
/

CREATE OR REPLACE TRIGGER RejestrowanieRozlaczen
   AFTER LOGON ON DATABASE
   CALL PakietDziennika.RejestrujRozlaczenia(SYS.LOGIN_USER)
/
```

Ponieważ `RejestrowaniePolaczen` **oraz** `RejestrowanieRozlaczen` **to wyzwalacze systemowe na poziomie bazy danych (w odróżnieniu od wyzwalaczy na poziomie schematu), do ich utworzenia jest potrzebne posiadanie uprawnienia systemowego** `ADMINISTER DATABASE TRIGGER`.

Treść obu wyzwalaczy — `RejestrowaniePolaczen` oraz `RejestrowanieRozlaczen` — to po prostu instrukcja `CALL` wywołująca odpowiednią procedurę. Jako argument tej procedury jest przekazywany bieżący użytkownik. W powyższym przykładzie obiektem instrukcji `CALL` jest standardowa procedura PL/SQL wchodząca w skład pakietu. Równie dobrze może to być jednak osłona dla zewnętrznej procedury napisanej w języku C lub Java. Przykładowo załadowano do bazy danych następującą klasę Java:

Dostępne na płycie CD jako skrypt *Rejestrator.java*.

```
import java.sql.*;
import oracle.jdbc.driver.*;

public class Rejestrator {
  public static void RejestrowaniePolaczen(String uzytkID)
    throws SQLException {
    // Uzyskanie domyślnego połączenia JDBC
    Connection polaczenie = new OracleDriver().defaultConnection();

    String CiagInsert =
      "INSERT INTO audyt_polaczen (nazwa_uzytkownika, operacja, datownik)" +
      "  VALUES (?, 'POLACZENIE', SYSDATE)";

   // Przygotowanie i wykonanie instrukcji  wprowadzającej dane do bazy
    PreparedStatement InstrukcjaInsert =
      polaczenie.prepareStatement(CiagInsert);
    InstrukcjaInsert.setString(1, uzytkID);
    InstrukcjaInsert.execute();
  }

  public static void RejestrowanieRozlaczen(String uzytkID)
    throws SQLException {
    // Uzyskanie domyślnego połączenia JDBC
    Connection polaczenie = new OracleDriver().defaultConnection();

    String CiagInsert =
      "INSERT INTO audyt_polaczen (nazwa_uzytkownika, operacja, datownik)" +
      "  VALUES (?, 'ROZLACZENIE', SYSDATE)";

   // Przygotowanie i wykonanie instrukcji  wprowadzającej dane do bazy
    PreparedStatement InstrukcjaInsert =
      polaczenie.prepareStatement(CiagInsert);
    InstrukcjaInsert.setString(1, uzytkID);
    InstrukcjaInsert.execute();
  }
}
```

Następnie można utworzyć pakiet *PakietDziennika* jako osłonę dla tej klasy:

Dostępne na płycie CD jako część skryptu *PakietDziennika2.sql*.

```
CREATE OR REPLACE PACKAGE PakietDziennika AS
  PROCEDURE RejestrujPolaczenia(p_UzytkID IN VARCHAR2);
  PROCEDURE RejestrujRozlaczenia(p_UzytkID IN VARCHAR2);
END PakietDziennika;
```

```
CREATE OR REPLACE PACKAGE BODY PakietDziennika AS
  PROCEDURE RejestrujPolaczenia(p_UzytkID IN VARCHAR2) IS
     LANGUAGE JAVA
     NAME 'Rejestrator.RejestrowaniePolaczen(java.lang.String)';
  PROCEDURE RejestrujRozlaczenia(p_UzytkID IN VARCHAR2) IS
  LANGUAGE JAVA
     NAME 'Rejestrator.RejestrowanieRozlaczen(java.lang.String)';
END PakietDziennika;
```

W celu uzyskania pożądanego efektu można wykorzystać te same wyzwalacze. Więcej informacji na temat procedur zewnętrznych znajduje się w rozdziale 12.

Predykaty wyzwalaczy, takie jak INSERTING, UPDATING **oraz** DELETING **oraz identyfikatory korelacji** :old **i** :new **(a także** :parent**) można zastosować tylko wtedy, gdy treść wyzwalacza w całości jest blokiem PL/SQL, a nie instrukcją** CALL.

Uprawnienia dotyczące wyzwalaczy

Istnieje pięć uprawnień systemowych dotyczących wyzwalaczy. Uprawnienia te opisano w tabeli 11.6. Oprócz nich właściciel wyzwalacza musi posiadać odpowiednie uprawnienia obiektowe do obiektów, do których odwołuje się wyzwalacz. Wyzwalacz jest skompilowanym obiektem, a zatem uprawnienia te muszą być nadane bezpośrednio, a nie za pośrednictwem roli.

Tabela 11.6. *Uprawnienia systemowe dotyczące wyzwalaczy*

Uprawnienie systemowe	Opis
CREATE TRIGGER	Umożliwia użytkownikowi posiadającemu to uprawnienie tworzenie wyzwalacza w swoim schemacie.
CREATE ANY TRIGGER	Umożliwia użytkownikowi posiadającemu to uprawnienie tworzenie wyzwalaczy w dowolnym schemacie, z wyjątkiem schematu SYS. Tworzenie wyzwalaczy dla tabel słownika danych nie jest zalecane.
ALTER ANY TRIGGER	Umożliwia użytkownikowi posiadającemu to uprawnienie włączanie, wyłączanie lub kompilowanie wyzwalaczy bazy danych w dowolnym schemacie z wyjątkiem schematu SYS. Proszę zwrócić uwagę, że jeżeli posiadacz tego uprawnienia nie ma uprawnienia CREATE ANY TRIGGER, to nie może modyfikować kodu wyzwalacza.
DROP ANY TRIGGER	Umożliwia użytkownikowi posiadającemu to uprawnienie usuwanie wyzwalaczy w dowolnym schemacie, z wyjątkiem schematu SYS.
ADMINISTER DATABASE TRIGGER	Umożliwia użytkownikowi posiadającemu to uprawnienie tworzenie lub modyfikację wyzwalacza systemowego na poziomie bazy danych (w odróżnieniu od bieżącego schematu). Posiadacz tego uprawnienia musi również posiadać uprawnienie CREATE TRIGGER lub CREATE ANY TRIGGER.

Wyzwalacze a słownik danych

Niektóre perspektywy słownika danych zawierają informacje dotyczące wyzwalaczy i ich statusu, podobnie jak w przypadku podprogramów składowanych. Perspektywy te zmienia się za każdym razem, kiedy tworzy się lub usuwa wyzwalacz.

Perspektywy słownika danych

Po utworzeniu wyzwalacza jego kod źródłowy jest składowany w perspektywie słownika danych `user_triggers` (wyzwalacze użytkownika). W perspektywie tej znajduje się kod wyzwalacza, dane określające klauzulę `WHEN`, tabelę wyzwalającą oraz typ wyzwalacza. Poniżej znajduje się przykładowy kod zapytania, które zwraca informację dotyczącą wyzwalacza `UaktualnijSpecStat`:

```
SQL> SELECT trigger_type, table_name, triggering_event
  2    FROM user_triggers
  3    WHERE trigger_name = 'UAKTUALNIJSPECSTAT';

TRIGGER_TYPE     TABLE_NAME  TRIGGERING_EVENT
---------------  ----------  -------------------------
AFTER STATEMENT  STUDENCI    INSERT OR UPDATE OR DELETE
```

Perspektywa `user_triggers` zawiera informacje dotyczące wyzwalaczy należących do bieżącego użytkownika. Ponadto istnieją dwie dodatkowe perspektywy: perspektywa `all_triggers` (zawiera informacje na temat wyzwalaczy dostępnych dla bieżącego użytkownika — które mogą należeć do innego użytkownika) i perspektywa `dba_triggers` (zawiera informacje na temat wyzwalaczy w bazie danych). Więcej informacji dotyczących perspektyw słownika danych znajduje się w dodatku C.

Usuwanie i dezaktywacja wyzwalaczy

Podobnie jak procedury i pakiety, tak i wyzwalacze mogą być usuwane. Poniżej przedstawiono składnię odpowiedniego polecenia:

```
DROP TRIGGER nazwa_wyzwalacza;
```

gdzie *nazwa_wyzwalacza* jest nazwą usuwanego wyzwalacza. Wykonanie tego polecenia powoduje trwałe usunięcie danego wyzwalacza ze słownika danych. Podobnie jak w przypadku podprogramów, w instrukcji tworzenia wyzwalacza `CREATE` można określić klauzulę `OR REPLACE`. W takim przypadku po wydaniu omawianego polecenia najpierw następuje usunięcie wyzwalacza, jeżeli taki istnieje.

Jednak w odróżnieniu od procedur i pakietów wyzwalacz może być dezaktywowany bez konieczności jego usuwania. Jeśli wyzwalacz jest nieaktywny, to w dalszym ciągu istnieje w słowniku danych, ale nie jest uruchamiany. W celu dezaktywowania wyzwalacza należy użyć instrukcji `ALTER TRIGGER`.

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

— *nazwa_wyzwalacza* jest nazwą wyzwalacza. Wszystkie wyzwalacze są uaktywniane domyślnie po ich utworzeniu. Instrukcja `ALTER TRIGGER` może służyć do dezaktywowania i następnie do ponownego aktywowania wyzwalaczy. Na przykład, poniższy kod najpierw dezaktywuje, a następnie ponownie aktywuje wyzwalacz `UaktualnijSpecStat`:

```
SQL> ALTER TRIGGER UaktualnijSpecStat DISABLE
Wyzwalacz został zmieniony.

SQL> ALTER TRIGGER UaktualnijSpecStat ENABLE;
Wyzwalacz został zmieniony.
```

Wszystkie wyzwalacze dla poszczególnych tabel mogą być aktywowane lub dezaktywowane za pomocą polecenia ALTER TABLE. Można również zastosować klauzulę ENABLE ALL TRIGGERS (aktywacja wszystkichwyzwalaczy) lub DISABLE ALL TRIGGERS (dezaktywacja wszystkich wyzwalaczy). Poniżej znajduje się odpowiedni przykład:

```
SQL> ALTER TABLE studenci
  2    ENABLE ALL TRIGGERS;
Tabela została zmieniona.

SQL> ALTER TABLE studenci
  2    DISABLE ALL TRIGGERS;
Tabela została zmieniona.
```

Kolumna status (status) perspektywy user_triggers zawiera wartość 'ENABLED' (aktywny) albo wartość 'DISABLED' (nieaktywny), wskazując bieżący status wyzwalacza. Dezaktywacja wyzwalacza nie usuwa go ze słownika danych, jak w przypadku usunięcia wyzwalacza.

Skompilowana forma wyzwalacza p-kod

Kiedy pakiet lub podprogram jest składowany w słowniku danych, oprócz kodu źródłowego danego obiektu jest również składowana jego skompilowana forma — p-kod. Dla wyzwalaczy jest podobnie. Oznacza to, że można wywołać wyzwalacze bez konieczności ich ponownej kompilacji oraz są zapisywane informacje o zależnościach. Dzięki temu wyzwalacze mogą być automatycznie unieważniane w taki sam sposób jak pakiety i podprogramy. Jeśli wyzwalacz zostanie unieważniony, będzie ponownie skompilowany przy następnym uruchomieniu.

Tabele mutujące

Istnieją pewne ograniczenia tabel i kolumn, do których może mieć dostęp treść wyzwalacza. W celu zdefiniowania tych ograniczeń konieczne jest zrozumienie pojęć tabeli wiążącej i mutującej. *Tabela mutująca* jest tabelą, która w danej chwili jest modyfikowana przez instrukcję DML. W przypadku wyzwalacza jest to tabela, dla której zdefiniowano wyzwalacz. Tabelami mutującymi mogą być także tabele, które mogą wymagać uaktualnienia w wyniku kaskadowego usuwania danych (DELETE CASCADE) przy występujących więzach integralności referencyjnej (więcej informacji dotyczących więzów integralności referencyjnej znajduje się w dokumentacji *Oracle Server Reference*). *Tabela wiążąca* jest tabelą, która może wymagać odczytu ze względu na więzy integralności referencyjnej. Dla zilustrowania tych definicji proszę przeanalizować przykładową tabelę zarejestrowani_studenci, która jest tworzona przez podany niżej kod:

Dostępne na płycie CD jako część skryptu *tabele.sql*.

```
CREATE TABLE zarejestrowani_studenci(
  student_id   NUMBER(5) NOT NULL,
  wydzial      CHAR(3)   NOT NULL,
  kurs         NUMBER(3) NOT NULL,
  ocena        CHAR(1),
```

```
  CONSTRAINT zs_ocena
    CHECK (ocena IN ('A', 'B', 'C', 'D', 'E')),
  CONSTRAINT zs_student_id
    FOREIGN KEY (student_id) REFERENCES studenci (id),
  CONSTRAINT zs_wydzial_kurs
    FOREIGN KEY (wydzial, kurs)
  REFERENCES grupy  (wydzial, kurs)
    );
```

Tabela `zarejestrowani_studenci` zawiera dwie deklaracje więzów integralności referencyjnej. Zarówno tabela `studenci`, jak i tabela `grupy` są tabelami wiążącymi dla tabeli `zarejestrowani_studenci`. Sama tabela `zarejestrowani_studenci` podlega mutacji podczas wykonywania dla niej instrukcji DML. Z powodu istniejących więzów tabela `grupy` i tabela `studenci` również wymagają modyfikacji i (lub) wykonania zapytań przez instrukcję DML.

Instrukcje SQL znajdujące się w treści wyzwalacza nie mogą wykonywać następujących operacji:

- Odczytywać lub modyfikować tabeli mutującej wskazanej w instrukcji wyzwalającej. Zasada ta dotyczy samej tabeli wyzwalającej.
- Odczytywać lub modyfikować kolumny klucza głównego, unikatowego lub obcego tabeli wiążącej dla tabeli wyzwalającej. Jednak w razie potrzeby instrukcje SQL mogą przeprowadzać modyfikacje innych kolumn.

Powyższe ograniczenia dotyczą wszystkich wyzwalaczy na poziomie wiersza. W przypadku wyzwalaczy na poziomie instrukcji ograniczenia te są prawdziwe tylko wtedy, gdy wyzwalacz na poziomie instrukcji jest uruchomiony w wyniku operacji kaskadowego usuwania (`DELETE CASCADE`).

Jeżeli instrukcja `INSERT` dotyczy tylko jednego wiersza, to wyzwalacze `BEFORE` oraz `AFTER` dla tego wiersza nie traktują tabeli wyzwalającej jako mutującej. Jest to jedyny przypadek, gdy wyzwalacz na poziomie wiersza może odczytywać lub modyfikować tabelę wyzwalającą. Instrukcje takie jak:

```
INSERT INTO tabela SELECT ...
```

zawsze traktują tabele wyzwalającą jako mutującą, nawet jeżeli podzapytanie zwraca tylko jeden wiersz.

Proszę przeanalizować poniższy wyzwalacz `KaskadaZSInsert`. Mimo że omawiany wyzwalacz modyfikuje zarówno tabelę `studenci`, jak i tabelę `grupy`, to jest on prawidłowy, ponieważ modyfikowane kolumny w tabelach `studenci` oraz `grupy` nie są kluczami (w dalszej części niniejszego rozdziału podano przykład wyzwalacza nieprawidłowego):

Dostępne na płycie CD w skrypcie *KaskadaZSinsert.sql*.

```
CREATE OR REPLACE TRIGGER KaskadaZSInsert
  /* Utrzymuje w synchronizacji tabele zarejestrowani_studenci,
     studenci i grupy. */
  BEFORE INSERT ON zarejestrowani_studenci
  FOR EACH ROW
```

```
DECLARE
  z_Zaliczenia grupy.liczba_zaliczen%TYPE;
BEGIN
  -- Sprawdzenie liczby zaliczeń dla tej grupy.
  SELECT liczba_zaliczen
    INTO z_Zaliczenia
    FROM grupy
    WHERE wydzial = :new.wydzial
    AND kurs = :new.kurs;

  -- Modyfikacja bieżących zaliczeń dla tego studenta.
  UPDATE studenci
    SET biezace_zaliczenia = biezace_zaliczenia + z_Zaliczenia
    WHERE ID = :new.student_id;

  -- Dodaj jeden do liczby studentów w grupie.
  UPDATE grupy
    SET biez_l_studentow = biez_l_studentow + 1
    WHERE wydzial = :new.wydzial
    AND kurs = :new.kurs;
END KaskadaZSInsert;
```

Przykład tabeli mutującej

Można założyć, że dla każdej specjalności ograniczono liczbę studentów do 5. W takim przypadku przydatne byłoby utworzenie wyzwalacza `BEFORE INSERT` lub `UPDATE` na poziomie wiersza dla tabeli `studenci`, tak jak pokazano poniżej:

Dostępne na płycie CD w skrypcie *OgraniczSpec.sql*.

```
CREATE OR REPLACE TRIGGER OgraniczSpec
/* Dla każdej specjalności ogranicza liczbę studentów do 5. Jeżeli limit zostanie
   przekroczony, wywołany zostanie błąd za pośrednictwem procedury
   raise_application_error. */
  BEFORE INSERT OR UPDATE OF specjalnosc ON studenci
  FOR EACH ROW
DECLARE
  z_Maks_l_Studentow CONSTANT NUMBER := 5;
  z_Biez_l_Studentow NUMBER;
BEGIN
  -- Określenie bieżącej liczby studentów dla tej specjalności.
SELECT COUNT(*)
    INTO z_Biez_l_Studentow
    FROM studenci
    WHERE specjalnosc = :new.specjalnosc;

  -- Jeżeli nie ma wolnych miejsc, wywołanie błędu.
  IF z_Biez_l_Studentow + 1 > z_Maks_l_Studentow THEN
    RAISE_APPLICATION_ERROR(-20000,
      'Za dużo studentów dla specjalności ' || :new.specjalnosc);
  END IF;
END OgraniczSpec;
```

Wydawałoby się, że dzięki temu wyzwalaczowi można uzyskać pożądany wynik. Jednak jeżeli uaktualni się tabelę `studenci` i uruchomi powyższy wyzwalacz, to wystąpią następujące błędy:

Dostępne na płycie CD jako część skryptu *OgraniczSpec.sql*.

```
SQL> UPDATE studenci
  2    SET specjalnosc = 'Historia'
  3    WHERE ID = 10003;
UPDATE studenci
 *
BŁĄD w linii 1:
ORA-04091: tabela PRZYKLAD.STUDENTS mutuje, co może być niewidoczne
           dla wyzwalacza lub funkcji
ORA-06512: w "PRZYKLAD.OGRANICZSPEC", linia 7
ORA-04088: błąd podczas wykonywania wyzwalacza 'PRZYKLAD.OGRANICZSPEC'
```

Wystąpienie błędu `ORA-04091` jest spowodowane wykonaniem przez wyzwalacz `OgraniczSpec` zapytania na swojej własnej tabeli wyzwalającej, będącej w trakcie mutowania. Błąd `ORA-04091` jest wywołany podczas uruchamiania wyzwalacza, a nie jego tworzenia.

Rozwiązanie problemu błędu tabeli mutującej

Tabela `studenci` mutuje tylko dla wyzwalacza na poziomie wiersza. Oznacza to, że na tej tabeli nie można wykonać zapytania w wyzwalaczu na poziomie wiersza, ale można wykonać zapytanie w wyzwalaczu na poziomie instrukcji. Jednak nie można po prostu zamienić wyzwalacza na poziomie wiersza (`OgraniczSpec`) na wyzwalacz na poziomie instrukcji, ponieważ w treści wyzwalacza konieczne jest użycie wartości `:new.specjalnosc`. Rozwiązaniem tej sytuacji jest utworzenie dwóch wyzwalaczy — na poziomie wiersza i na poziomie instrukcji. W wyzwalaczu na poziomie wiersza można zapisać wartość `:new.specjalnosc`, ale nie można wykonać zapytania na tabeli `studenci`. Zapytanie jest wykonywane w wyzwalaczu na poziomie instrukcji i w ten sposób jest wykorzystana wartość zapisana w wyzwalaczu na poziomie wiersza.

W celu zapisania tej wartości najlepszym sposobem jest wykorzystanie tabeli PL/SQL wewnątrz pakietu. Dzięki temu można zapisać wiele wartości w ciągu jednej operacji uaktualniania tabeli. Ponadto każda sesja tworzy własny egzemplarz zmiennych pakietowych, zatem problem jednoczesnego uaktualniania danych w różnych sesjach jest rozwiązany. Taką metodę zastosowano w pakiecie *DaneStudenta* oraz w wyzwalaczach `WOgraniczSpec` oraz `IOgraniczSpec`:

Dostępne na płycie CD jako część skryptu *Mutujace.sql*.

```
CREATE OR REPLACE PACKAGE DaneStudenta AS
  TYPE t_Specjalnosci IS TABLE OF studenci.specjalnosc%TYPE
    INDEX BY BINARY_INTEGER;
  TYPE t_ID IS TABLE OF studenci.ID%TYPE
    INDEX BY BINARY_INTEGER;
```

```
  z_SpecjalnosciStudentow t_Specjalnosci;
  z_IDStudentow    t_ID;
  z_LiczbaZapisow  BINARY_INTEGER := 0;
END DaneStudenta;

CREATE OR REPLACE TRIGGER WOgraniczSpec
  BEFORE INSERT OR UPDATE OF specjalnosc ON studenci
  FOR EACH ROW
BEGIN
  /* Zarejestrowanie nowych danych w pakiecie DaneStudenta. Nie dokonuje się zmian
     w tabeli studenci w celu uniknięcia błędu ORA-4091. */
  DaneStudenta.z_LiczbaZapisow := DaneStudenta.z_LiczbaZapisow + 1;
  DaneStudenta.z_SpecjalnosciStudentow(DaneStudenta.z_LiczbaZapisow) :=
    :new.specjalnosc;
  DaneStudenta.z_IDStudentow(DaneStudenta.z_LiczbaZapisow) := :new.id;
END WOgraniczSpec;


CREATE OR REPLACE TRIGGER IOgraniczSpec
  AFTER INSERT OR UPDATE OF specjalnosc ON studenci
DECLARE
  z_Maks_l_Studentow   CONSTANT NUMBER := 5;
  z_Biez_l_Studentow   NUMBER;
  z_StudentID          studenci.ID%TYPE;
  z_Specjalnosc        studenci.specjalnosc%TYPE;
BEGIN
  /* Pętla dla każdego studenta, którego dane wprowadzono do bazy lub je uaktualniono
     i sprawdzenie, czy nie przekroczono limitu. */
  FOR z_IndeksPetli IN 1..DaneStudenta.z_LiczbaZapisow LOOP
    z_StudentID := DaneStudenta.z_IDStudentow(z_IndeksPetli);
    z_Specjalnosc := DaneStudenta.z_SpecjalnosciStudentow(z_IndeksPetli);

    -- Określenie bieżącej liczby studentów w tej specjalności.
    SELECT COUNT(*)
      INTO z_Biez_l_Studentow
      FROM studenci
      WHERE specjalnosc = z_Specjalnosc;

    -- Jeżeli nie ma miejsc - zgłoszenie błędu.
    IF z_Biez_l_Studentow > z_Maks_l_Studentow THEN
      RAISE_APPLICATION_ERROR(-20000,
        'Za dużo studentów w specjalności ' || z_Specjalnosc ||
        ' z powodu studenta ' || z_StudentID);
    END IF;
  END LOOP;

  -- Wyzerowanie licznika tak, aby przy następnym wykonaniu wykorzystywano nowe dane.
  DaneStudenta.z_LiczbaZapisow := 0;
END IOgraniczSpec;
```

Należy pamiętać, aby przed uruchomieniem powyższego skryptu usunąć nieprawidłowy wyzwalacz `OgraniczSpec`.

Można teraz przetestować prawidłowość działania powyższych wyzwalaczy poprzez uaktualnianie tabeli `studenci` do momentu, kiedy będzie zbyt dużo studentów studiujących przedmiot `Historia`:

```
SQL> UPDATE studenci
  2    SET specjalnosc = 'Historia'
  3    WHERE ID = 10003;
1 wiersz został zmodyfikowany.

SQL> UPDATE studenci
  2    SET specjalnosc = 'Historia'
  3    WHERE ID = 10002;
1 wiersz został zmodyfikowany.


SQL> UPDATE studenci
  2    SET specjalnosc = 'Historia'
  3    WHERE ID = 10009;
UPDATE studenci
 *
BŁĄD w linii 1:
ORA-20000: Za dużo studentów na specjalności Historia z powodu studenta 10009
ORA-06512: w "PRZYKLAD.IOGRANICZSPEC", linia 19
ORA-04088: błąd podczas wykonywania wyzwalacza 'PRZYKŁAD.IOGRANICZSPEC'
```

Zatem działanie powyższego kodu jest zgodne z założeniami i prawidłowe. Technika ta jest przydatna przy występowaniu błędu `ORA-4091`, kiedy wyzwalacz na poziomie wiersza ma odczytywać lub modyfikować tabelę mutującą. Zamiast przeprowadzania tego nieprawidłowego przetwarzania w wyzwalaczu na poziomie wiersza, można je przekazać do wyzwalacza `AFTER` na poziomie instrukcji, gdzie będzie zachodziło prawidłowo. Tabele pakietowe PL/SQL są używane do składowania zmodyfikowanych wierszy.

Stosując opisywaną technikę, należy pamiętać o kilku istotnych zagadnieniach:

- Tabele PL/SQL znajdują się w pakiecie, tak że będą widoczne dla wyzwalacza zarówno na poziomie wiersza, jak i na poziomie instrukcji. Jedynym sposobem zapewnienia globalności tych zmiennych jest umieszczenie ich w pakiecie.
- Wykorzystano zmienną licznika `DaneStudenta.z_LiczbaZapisow`. Podczas tworzenia pakietu zmienną zainicjowano wartością 0. Zmienna licznika jest inkrementowana przez wyzwalacz na poziomie wiersza. Wyzwalacz na poziomie instrukcji wywołuje ją i następnie zeruje jej wartość po przetworzeniu. Jest to konieczne, aby następna instrukcja w tej sesji miała poprawną wartość.
- Sposób sprawdzania maksymalnej liczby studentów w wyzwalaczu `IOgraniczSpec` należy trochę zmienić. Obecnie jest to wyzwalacz `AFTER` na poziomie instrukcji, a zatem zmienna `z_Biez_l_Studentow` zawiera liczbę studentów danej specjalności po wstawieniu lub uaktualnieniu danych, a nie przed tym zdarzeniem. Tak więc sprawdzenie dla `z_Biez_l_Studentow +1`, przeprowadzane w wyzwalaczu `OgraniczSpec`, jest zastąpione przez zmienną `z_Biez_l_Studentow`.
- Zamiast tabel PL/SQL można zastosować tabelę bazy danych. Nie zaleca się stosowania tej techniki, ponieważ wydawanie instrukcji `UPDATE` w odbywających się równocześnie sesjach powodowałoby powstawanie kolizji (w systemie Oracle8i oraz w wersjach wyższych można zastosować tabelę tymczasową). Tabele PL/SQL są niepowtarzalne między sesjami, dzięki czemu unika się tego problemu.

Podsumowanie

Jak można się było przekonać, wyzwalacze stanowią cenne uzupełnienie języka PL/SQL i bazy danych Oracle. Mogą być stosowane w celu wymuszania ograniczeń danych o wiele bardziej złożonych niż zwykłe więzy integralności referencyjnej. W systemie Oracle8i rozszerzono właściwości wyzwalaczy o zdarzenia inne niż operacje DML dla tabel lub perspektyw. Przedstawienie wyzwalaczy zamyka omawiane we wcześniejszych trzech rozdziałach zagadnienia dotyczące nazwanych bloków PL/SQL.

W ostatnim rozdziale omówiono kilka zaawansowanych możliwości języka PL/SQL.