Oracle8. Programowanie w języku PL/SQL

Autor: Scott Urman
Tłumaczenie: Tomasz Pędziwiatr, Grzegorz Stawikowski, Cezary Welsyng
ISBN: 83-7197-533-3
Tytuł oryginału: Oracle8 PL/SQL Programming
Format: B5, stron: 762

Wykorzystanie wbudowanego w system Oracle języka PL/SQL w znaczący sposób powiększa potencjał programisty systemów bazodanowych. PL/SQL łączy w sobie duże możliwości i elastyczność języka czwartej generacji (4GL) SQL z konstrukcjami proceduralnymi języka trzeciej generacji (3GL). Programy napisane w tym języku umożliwiają obsługę danych zarówno w samym systemie Oracle, jak i w zewnętrznych aplikacjach.

Książka „Oracle8. Programowanie w języku PL/SQL" to wyczerpujące omówienie języka PL/SQL. To doskonała pozycja ułatwiająca naukę tego języka, świetnie też sprawdza się jako podręczne kompendium wiedzy o PL/SQL, pomocne w codziennej pracy. Liczne przykłady uzupełniają informacje zawarte w książce pokazując sprawdzone metody rozwiązywania problemów, napotykanych przez programistów.

W książce omówiono między innymi:

- Podstawy języka PL/SQL: struktura programu, zmienne, typy, wyrażenia i operatory oraz instrukcje sterujące
- Korzystanie z rekordów i tabel
- Korzystanie z SQL z poziomu PL/SQL, funkcje SQL dostępne w PL/SQL
- Tworzenie i używanie kursorów
- Bloki w PL/SQL: podprogramy (procedury i funkcje), pakiety i wyzwalacze
- Metody obsługi błędów w PL/SQL
- Obiekty w PL/SQL, kolekcje
- Testowanie i wykrywanie błędów
- Zagadnienia zaawansowane: dynamiczny PL/SQL, komunikacja między sesjami, kolejkowanie, obsługa zadań, procedury zewnętrzne
- Optymalizacja aplikacji PL/SQL i metody zapewnienia maksymalnej wydajności

Książka jest przeznaczona zarówno dla doświadczonych programistów, jak i tych, którzy jeszcze nie poznali innych języków trzeciej generacji. Przydatna, choć niekonieczna, jest ogólna znajomość systemu Oracle (łączenie się i korzystanie z bazy danych, podstawy języka SQL, itp.).

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

Spis treści

Rozdział 4.

SQL w PL/SQL

Strukturalny język zapytań (SQL) określa sposób manipulowania danymi w bazie danych Oracle. Konstrukcje proceduralne, które przedstawiono w rozdziale 2. i 3., stają się znacznie bardziej użyteczne w połączeniu z mocą przetwarzania języka SQL, ponieważ wtedy konstrukcje te pozwalają programom PL/SQL na manipulowanie danymi w bazie danych Oracle. W niniejszym rozdziale zostaną omówione instrukcje SQL, które są dozwolone w języku PL/SQL i instrukcje sterowania transakcjami, które gwarantują utrzymanie spójności danych. W rozdziale 5. Czytelnik zapozna się z wbudowanymi funkcjami SQL.

Instrukcje SQL

Instrukcje SQL można podzielić na sześć kategorii, które wymieniono poniżej. W tabeli 4.1 znajduje się kilka przykładowych instrukcji. Szczegółowy opis wszystkich instrukcji SQL znajduje się w publikacji *Server SQL Reference*.

Tabela 4.1. *Kategorie instrukcji SQL*

Kategoria	Przykładowe instrukcje SQL
Instrukcje języka manipulowania danymi DML (*Data Manipulation Language*)	`SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET TRANSACTION`, `EXPLAIN PLAN`
Instrukcje języka definicji danych DDL (*Data Definition Language*)	`DROP`, `CREATE`, `ALTER`, `GRANT`, `REVOKE`
Instrukcje sterowania transakcją	`COMMIT`, `ROLLBACK`, `SAVEPOINT`
Instrukcje sterowania sesji	`ALTER SESSION`, `SET ROLE`
Instrukcje sterowania systemu	`ALTER SYSTEM`
Wbudowane polecenia SQL	`CONNECT`, `DECLARE CURSOR`, `ALLOCATE`[1]

- *instrukcje języka manipulowania danymi DML* (*Data Manipulation Language*) służą do zmieniania danych w tabelach lub danych zapytań w tabeli bazy danych, ale nie umożliwiają zmiany struktury tabeli lub innych obiektów;

[1] Wbudowane polecenie SQL `ALLOCATE` jest dostępne w wydaniu Oracle 7.2 i wyższym.

- *instrukcje języka definicji danych DDL* (*Data Definition Language*) służą do tworzenia, usuwania lub zmieniania struktury obiektu schematu. Polecenia, które zmieniają uprawnienia do obiektów schematu, są również instrukcjami DDL;
- *instrukcje sterowania transakcji* gwarantują zachowanie spójności danych dzięki zorganizowaniu instrukcji SQL w logiczne transakcje, których wykonywanie jako jednostki kończy się powodzeniem lub niepowodzeniem;
- *instrukcje sterowania sesji* służą do zmieniania ustawień dla pojedynczego połączenia z bazą danych, np. do aktywacji śledzenia sesji SQL;
- *instrukcje sterowania systemu* służą do zmieniania ustawień dla całej bazy danych, np. do aktywowania lub dezaktywowania procesu archiwizacji;
- *wbudowane polecenia SQL* są wykorzystywane w programach prekompilatora Oracle.

Wykorzystanie instrukcji SQL w języku PL/SQL

Jedynymi instrukcjami SQL, które są dozwolone w programie PL/SQL, są instrukcje DML oraz instrukcje sterowania transakcji. W szczególności instrukcje DDL są niedozwolone. Także instrukcja `EXPLAIN PLAN`, mimo że jest klasyfikowana jako instrukcja DML, jest niedozwolona. Aby to wyjaśnić, konieczna jest znajomość założeń projektowych przyjętych dla języka PL/SQL.

Ogólnie język programowania może wiązać zmienne na dwa sposoby: przez wiązanie wczesne lub wiązanie późne. *Wiązanie* zmiennej jest procesem identyfikowania lokacji pamięci skojarzonej z identyfikatorem programu. W języku PL/SQL wiązanie uwzględnia również sprawdzenie istnienia zezwolenia na uzyskanie dostępu do odwoływanego obiektu schematu w bazie danych. W przypadku języka, w którym stosuje się *wiązanie* wczesne, wiązanie zmiennej następuje podczas etapu kompilacji, natomiast w przypadku języka, w którym stosuje się *wiązanie późne*, proces wiązania zmiennej jest odkładany aż do czasu uruchomienia programu. Uwzględnienie procesu wiązania wczesnego oznacza, że etap kompilacji będzie trwać dłużej (ponieważ musi być wykonane wiązanie zmiennych), ale sam program będzie wykonywany szybciej, ponieważ wiązanie będzie już zakończone. Wiązanie późne skraca czas kompilacji, ale wydłuża czas wykonywania programu.

Język PL/SQL celowo zaprojektowano w taki sposób, aby zastosować wiązanie wczesne. Decyzję tę podjęto w celu zapewnienia jak najszybszego wykonywania bloku, ponieważ wszystkie obiekty bazy danych są sprawdzane przez kompilator. Jest to sensowne rozwiązanie, ponieważ bloki PL/SQL mogą być składowane w bazie danych za pomocą procedur, funkcji, pakietów i wyzwalaczy. Obiekty te są składowane w skompilowanej formie, a zatem w razie potrzeby mogą być bezpośrednio ładowane z bazy danych do pamięci i uruchamiane. Więcej informacji na temat obiektów składowanych znajduje się w rozdziałach 7., 8. oraz 9. Konsekwencją podjęcia takiej decyzji projektowej jest zakaz stosowania instrukcji DDL. Instrukcje DDL modyfikują obiekt bazy danych, w więc zachodzi konieczność ponownego sprawdzania zezwoleń. Sprawdzanie zezwoleń wymagałoby ponownego wiązania identyfikatorów, a ten proces jest przeprowadzany podczas kompilacji.

W tym punkcie warto rozważyć następujący, hipotetyczny blok PL/SQL:

```
DECLARE
  CREATE TABLE temp_table (
    num_value NUMBER,
    char_value CHAR(10));
  INSERT INTO temp_table (num_value, char_value)
    VALUES (10, 'Hello');
END;
```

W celu kompilacji tego bloku identyfikator `temp_table` wymaga dowiązania. Podczas tego procesu następuje sprawdzenie faktu istnienia tabeli. Jednak tabela nie może istnieć przed uruchomieniem bloku. Z powyższego wynika, że omawiany blok nie może zostać skompilowany, a zatem nie ma sposobu na uruchomienie go.

Instrukcje sterowania transakcji są jedynymi instrukcjami SQL, które nie mają możliwości modyfikowania obiektów schematu lub uprawnień do obiektów schematu i w ten sposób są jedynymi poprawnymi instrukcjami SQL, stosowanymi w języku PL/SQL.

Stosowanie instrukcji DDL

PL/SQL 2.1 i następne

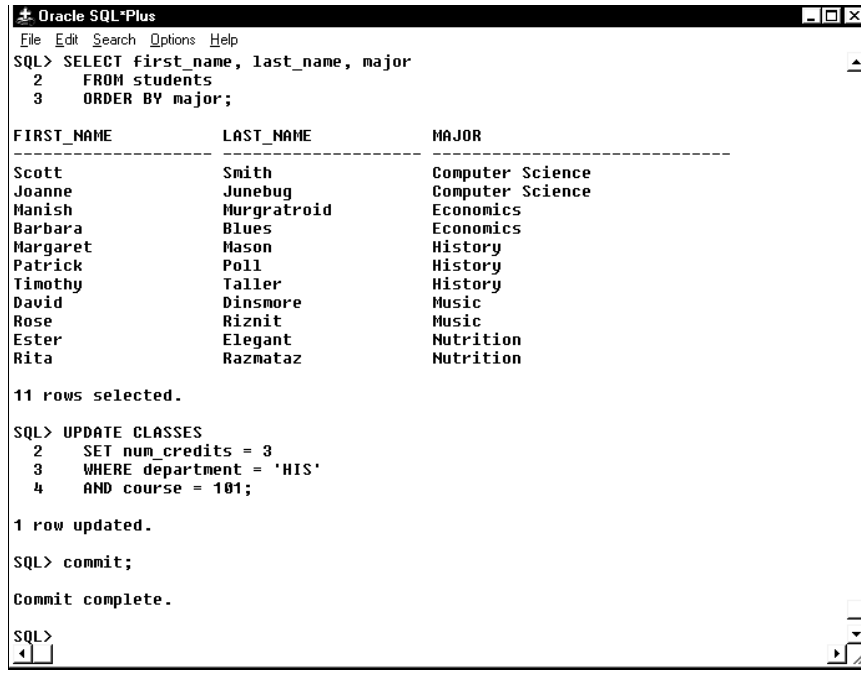
Istnieje alternatywne rozwiązanie problemu przedstawionego w poprzednim podrozdziale. Od wydania PL/SQL 2.1 jest dostępny wbudowany pakiet DBMS_SQL. Jest to pakiet, który umożliwia dynamiczne tworzenie instrukcji SQL podczas 1 tworzona aż do czasu uruchomienia programu, a więc kompilator PL/SQL nie musi wiązać identyfikatorów w instrukcji, co pozwala na kompilowanie bloku. Pakiet DBMS_SQL jest szczegółowo opisany w rozdziale 15. Przykładowo, do wykonania instrukcji CREATE TABLE z poprzedniego bloku można użyć pakietu DBMS_SQL. Jednak nawet wtedy kompilacja instrukcji INSERT nie powiedzie się, ponieważ tabela nie istnieje ć przed uruchomieniem bloku. Rozwiązaniem tego problemu jest zastosowanie pakietu DBMS_SQL również do wykonywania instrukcji INSERT.

Stosowanie instrukcji DML w języku PL/SQL

Dozwolonymi instrukcjami DML w języku PL/SQL są instrukcje: SELECT, INSERT, UPDATE oraz DELETE. Instrukcje te działają w następujący sposób: instrukcja SELECT zwraca te wiersze z tabeli bazy danych, które odpowiadają kryteriom podanym w jej klauzuli WHERE, instrukcja INSERT dodaje wiersze do tabeli bazy danych, instrukcja UPDATE modyfikuje te wiersze w tabeli bazy danych, które odpowiadają klauzuli WHERE, natomiast instrukcja DELETE usuwa wiersze identyfikowane przez klauzulę WHERE. Poza klauzulą WHERE powyższe instrukcje mogą uwzględniać także inne klauzule. Klauzule te opisano w kolejnych podrozdziałach niniejszego rozdziału.

Podczas wykonywania instrukcji SQL w programie *SQL*Plus* wyniki tego wykonania są wyświetlane na ekranie. Przykładowy sposób takiego wyświetlania przedstawiono na rysunku 4.1. W przypadku wykonywania instrukcji UPDATE, INSERT lub DELETE *SQL*Plus* zwraca liczbę przetwarzanych wierszy. W razie wykonywania instrukcji SELECT wiersze, które odpowiadają zapytaniu, są wyświetlane na ekranie.

Rysunek 4.1. *Wyniki wykonania instrukcji SQL w programie SQL*Plus*



Należy zwrócić uwagę na sposób zastosowania instrukcji UPDATE. Wyniki wykonania tej instrukcji pokazano na rysunku 4.1.

```
UPDATE CLASSES
  SET num_credits = 3
  WHERE department = 'HIS'
  AND course = 101;
```

Wszystkie wartości, które są wykorzystane w celu zmiany zawartości tabeli classes, są ustalone — są znane podczas tworzenia instrukcji. Język PL/SQL usuwa to ograniczenie za pomocą zmiennych. Stosowanie zmiennych jest dozwolone wszędzie tam, gdzie w instrukcji SQL jest dozwolone stosowanie wyrażeń. Jeśli zmienne są używane w opisywany sposób, są nazywane *zmiennymi dowiązanymi*. Przykładowo, w poprzedniej instrukcji UPDATE można zastąpić wartość ustaloną liczby zaliczeń (num_credits) zmienną dowiązaną:

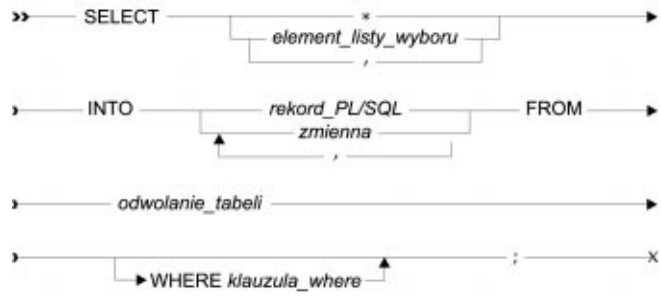
Dostępne na płycie CD w skrypcie *bindvar.sql*

```
DECLARE
  v_NumCredits  classes.num_credits%TYPE;
BEGIN
  /* Przypisz wartosc zmiennej v_NumCredits */
  v_NumCredits := 3;
  UPDATE CLASSES
    SET num_credits = v_NumCredits
    WHERE department = 'HIS'
    AND course = 101;
END;
```

Nie wszystkie elementy w instrukcji SQL mogą być zastępowane zmiennymi — zastępować można tylko wyrażenia. W szczególności muszą być znane nazwy tabel i kolumn. Jest to wymagane ze względu na wiązanie wczesne — nazwy obiektów Oracle muszą być znane w czasie kompilacji. Z definicji wartość zmiennej nie jest znana aż do czasu uruchomienia programu. W celu przezwyciężenia tego ograniczenia można również wykorzystywać pakiet `DBMS_SQL`.

Instrukcja SELECT

Instrukcja `SELECT` pobiera dane z bazy danych do zmiennych PL/SQL. Poniżej przedstawiono składnię instrukcji `SELECT`:



W poniższej tabeli opisano wszystkie elementy.

Tabela 4.2. *Klauzule wyboru instrukcji SELECT*

Klauzula wyboru	Opis
`element_listy_wyboru`	Kolumna (lub wyrażenie) do wybrania. Każdy element listy wyboru jest oddzielony przecinkiem i może być opcjonalnie identyfikowany przez alias (zamiennik). Cały zbiór elementów listy w instrukcji `SELECT` nazywa się *listą wyboru*. Znak * w składni jest skrótem zastępującym cały wiersz. W ten sposób są zwracane poszczególne pola w tabeli w kolejności, w jakiej zdefiniowano pola.
`Zmienna`	Zmienna PL/SQL, do której będzie przekazany element listy wyboru. Każda zmienna powinna być kompatybilna ze swoim skojarzonym elementem listy wyboru. Dlatego elementy listy oraz zmienne wyjściowe powinny istnieć w tej samej liczbie.
`rekord_PL/SQL`	Może być stosowany zamiast listy zmiennych. Rekord powinien zawierać pola, które odpowiadają elementom z listy wyboru, ale również pozwalają na łatwiejszą manipulację zwracanymi danymi. Rekordy łączą powiązane pola w jednej jednostce składniowej. W ten sposób można manipulować tymi polami zarówno jako grupą, jak również indywidualnie. Zagadnienia dotyczące rekordów opisano w dalszej części niniejszego rozdziału. Jeżeli listą wyboru jest znak *, wtedy ten rekord może być zdefiniowany jako `odwolanie_tabeli%ROWTYPE`.
`odwolanie_tabeli`	Identyfikuje tabelę, z której mają być pobrane dane. Może być synonimem lub tabelą w odległej bazie danych, określonej przez powiązanie z bazą danych. Więcej informacji na temat odwołań tabel znajduje się w dalszej części niniejszego rozdziału.
`klauzula_where`	Kryterium dla tego zapytania. Klauzula ta identyfikuje wiersz, który będzie zwrócony przez zapytanie. Kryterium składa się z warunków logicznych (boole'owskich) łączonych operatorami logicznymi. Zagadnienia związane z kryteriami wyboru opisano bardziej szczegółowo w dalszej części niniejszego rozdziału.

Ogólnie więcej klauzul jest dostępnych dla instrukcji SELECT. Przykładowo, zaliczają się do nich klauzule ORDERED BY oraz GROUP BY. Szczegółowo omówiono je w rozdziale 6. Więcej informacji na ich temat znajduje się w publikacji *Server SQL Reference*.

Instrukcja SELECT według podanej powyżej składni powinna zwracać najwyżej jeden wiersz. Klauzula WHERE jest sprawdzana dla każdego wiersza w tabeli. Jeżeli odpowiada ona więcej niż jednemu wierszowi, PL\SQL zwraca następujący komunikat o błędzie:

```
ORA-1427: Single-row query returns more than one row
        (Zapytanie o jeden wiersz zwraca wiecej niz jeden wiersz)
```

W takim przypadku do pobrania każdego wiersza osobno konieczne jest zastosowanie kursora. Więcej informacji na ten temat znajduje się w rozdziale 6.

W poniższym przykładzie przedstawiono sposób zastosowania dwóch różnych instrukcji SELECT:

Dostępne na płycie CD w skrypcie *select.sql*

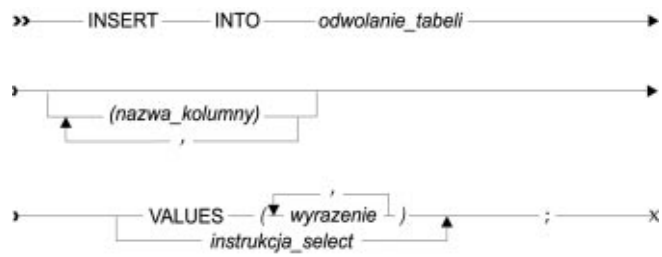
```
DECLARE
  v_StudentRecord  students%ROWTYPE;
  v_Department     classes.department%TYPE;
  v_Course         classes.course%TYPE;
BEGIN
  -- Pobierz jeden rekord z tabeli students i zachowaj w rekordzie
  -- v_StudentRecord. Nalezy zwrocic uwage, ze klauzula WHERE
  -- odpowiada tylko jednemu wierszowi tabeli. Nalezy rowniez zwrocic
  -- uwage, ze zapytanie zwraca wszystkie pola w tabeli students
  -- (poniewaz stosowany jest znak *). W ten sposob ladowany rekord
  -- jest zdefiniowany jako students%ROWTYPE.
  SELECT *
    INTO v_StudentRecord
    FROM students
    WHERE id = 10000;

  -- Pobierz dwa pola z tabeli classes i zachowaj je w zmiennych
  -- v_Department oraz v_Course. Znowu zatem klauzula WHERE odpowiada
  -- tylko jednemu wierszowi tabeli.
  SELECT department, course
    INTO v_Department, v_Course
    FROM classes
    WHERE room_id = 99997;
END;
```

Instrukcja INSERT

Składnię instrukcji INSERT przedstawiono poniżej. Należy zwrócić uwagę, że bezpośrednio w instrukcji nie występuje klauzula WHERE (chociaż może ona występować w podzapytaniu).

Klauzula odwolanie_tabeli odwołuje się do tabeli Oracle, nazwa_kolumny odwołuje się do kolumny w tej tabeli, a wyrazenie jest wyrażeniem SQL lub PL/SQL, co zdefiniowano w poprzednim rozdziale. Odwołania do tabel omówiono bardziej szczegółowo w dalszej



części niniejszego rozdziału. Jeżeli instrukcja `INSERT` zawiera `instrukcja_select`, wtedy elementy listy instrukcji `select` powinny odpowiadać kolumnom, do których mają być wstawiane dane.

Prawidłowe zastosowanie kilku instrukcji `INSERT` przedstawiono w poniższym przykładzie:

Dostępne na płycie CD w skrypcie *insert.sql*

```
DECLARE
  v_StudentID  students.id%TYPE;
BEGIN
  -- Pobierz nowy identyfikator ID studenta
  SELECT student_sequence.NEXTVAL
    INTO v_StudentID
    FROM dual;

  -- Dodaj wiersz do tabeli students.
  INSERT INTO students (id, first_name, last_name)
    VALUES (v_StudentID, 'Timothy', 'Taller');

  -- Dodaj drugi wiersz, ale uzyj numeru sekwencji bezposrednio w
  -- instrukcji INSERT.
  INSERT INTO students (id, first_name, last_name)
    VALUES (student_sequence.NEXTVAL, 'Patrick', 'Poll');
END;
```

W następnym przykładzie przedstawiono nieprawidłowy sposób wykonania polecenia `INSERT`. Elementy listy instrukcji `SELECT` podzapytania nie odpowiadają kolumnom, które mają być wstawiane. Taka instrukcja powoduje zwrócenie błędu Oracle ORA-913: *too many values*.

```
INSERT INTO rooms
  SELECT * FROM classes;
```

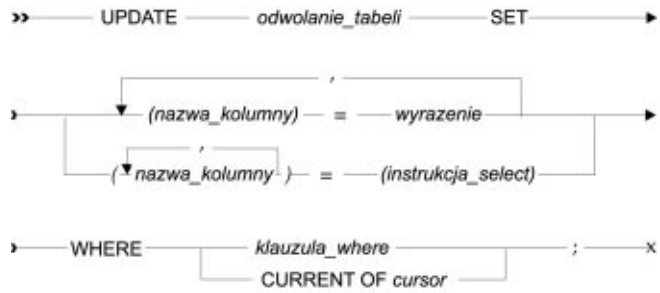
Kolejny przykład przedstawia prawidłowo wydaną instrukcję `INSERT`. Przez wstawienie kopii każdego wiersza następuje podwojenie wielkości tabeli `classes`.

```
INSERT INTO classes
  SELECT * FROM classes;
```

Wersja Oracle8 z opcją obiektów dostarcza dodatkową klauzulę dla instrukcji `INSERT` — klauzuli `REF INTO`. W razie zastosowania tej klauzuli z tabelami obiektów jest zwracane odwołanie do wstawianego obiektu. Więcej informacji na ten temat znajduje się w rozdziale 11.

Instrukcja UPDATE

Poniżej przedstawiono składnię instrukcji UPDATE:



Klauzula odwolanie_tabeli odwołuje się do modyfikowanej tabeli, nazwa_kolumny jest nazwą kolumny, której wartość ma być zmieniona, a wyrazenie jest wyrażeniem SQL, co zdefiniowano w rozdziale 2. Jeżeli instrukcja UPDATE zawiera instrukcja_SELECT, elementy listy wyboru powinny odpowiadać kolumnom w klauzuli SET.

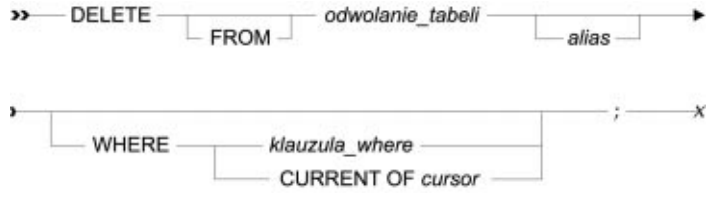
W poniższym przykładzie pokazano zastosowanie instrukcji UPDATE:

Dostępne na płycie CD w skrypcie *update.sql*

```
DECLARE
  v_Major           students.major%TYPE;
  v_CreditIncrease  NUMBER := 3;
BEGIN
  -- Ta instrukcja UPDATE doda 3 do pola current_credits
  -- dla wszystkich studentow, ktorzy studiuja Historie.
  v_Major := 'History';
  UPDATE students
    SET current_credits = current_credits + v_CreditIncrease
    WHERE major = v_Major;
END;
```

Instrukcja DELETE

Instrukcja DELETE usuwa wiersze z tabeli bazy danych. Klauzula WHERE wskazuje, które wiersze mają być usunięte. Poniżej podana jest składnia instrukcji DELETE:



Klauzula odwolanie_tabeli odwołuje się do tabeli z bazy danych Oracle, a klauzula_where definiuje zbiór wierszy, które mają być usunięte. Specjalna składnia CURRENT OF kursor jest używana wraz z definicją kursora i jest omówiona w rozdziale 6. Odwołania do tabeli oraz klauzula WHERE są omawiane szczegółowo w kolejnej części tego rozdziału, w podrozdziale „Klauzula WHERE".

W poniższym przykładzie przedstawiono sposób stosowania kilku różnych instrukcji DELETE:

Dostępne na płycie CD w skrypcie *delete.sql*

```
DECLARE
  v_StudentCutoff  NUMBER;
BEGIN
  v_StudentCutoff := 10;
  -- Usun dane dotyczace klas, w ktorych nie ma wystarczajacej liczby
  -- zarejestrowanych studentow.
  DELETE FROM classes
    WHERE current_students < v_StudentCutoff;

  -- Usun dane dotyczace kazdego studenta studiujacego Ekonomie, który
  -- nie posiada jeszcze zadnych zaliczen.
  DELETE FROM students
    WHERE current_credits = 0
    AND   major = 'Economics';
END;
```

Klauzula WHERE

Wraz z instrukcjami SELECT, UPDATE oraz DELETE stosuje się klauzulę WHERE, będącą integralną częścią wykonywanych przez te instrukcje operacji. Klauzula ta definiuje, które instrukcje *tworzą zestaw aktywny* — zestaw wierszy zwracanych przez zapytanie (SELECT) lub na których są wykonywane instrukcje UPDATE oraz DELETE.

Klauzula WHERE składa się z warunków połączonych operatorami logicznymi AND, OR oraz NOT. Warunki zwykle przyjmują formę porównań, tak jak w poniższym przykładzie zastosowania instrukcji DELETE:

```
DECLARE
  v_Department CHAR(3);
BEGIN
  v_Department := 'CS';
  -- Usun wszystkie klasy wydzialu Informatyka
  DELETE FROM classes
    WHERE department = v_Department;
END;
```

Powyższy, przykładowy blok usuwa wszystkie te wiersze z tabeli classes, dla których warunek został oceniony na wartość TRUE (dla których kolumna department = 'CS'). Przy tego typu porównaniach należy zwracać uwagę na właściwe stosowanie nazw zmiennych oraz sposób porównywania znaków.

Nazwy zmiennych

Na potrzeby kolejnego przykładu przyjęto, że w poprzednim bloku programu zmieniono nazwę zmiennej z `v_Department` na `Department`:

```
DECLARE
  Department CHAR(3);
BEGIN
  Department := 'CS';
  -- Usun wszystkie klasy wydzialu Informatyka
  DELETE FROM classes
    WHERE department = Department;
END;
```

Ta prosta zmiana radykalnie wpływa na otrzymane wyniki wykonania tej instrukcji — tak zmodyfikowany blok usunie *wszystkie* wiersze tabeli `classes`, a nie tylko te, dla których zachodzi równość `department = 'CS'`. Wynika to ze sposobu parsowania identyfikatorów w instrukcji SQL. Kiedy mechanizm PL/SQL napotyka na warunek taki jak:

```
1wyrazenie = 2wyrazenie
```

najpierw następuje sprawdzenie identyfikatorów `1wyrazenie` oraz `2wyrazenie` w celu ustalenia, czy odpowiadają one kolumnom tabeli, na których jest wykonywana dana operacja. Następnie identyfikatory te są sprawdzane, czy są zmiennymi w bloku PL/SQL. Język PL/SQL nie rozróżnia małych i dużych liter, zatem w poprzednim bloku obydwa identyfikatory `department` oraz `Department` są skojarzone z kolumną w tabeli `classes`, a nie ze zmienną. Wynikiem sprawdzenia tego warunku dla każdego wiersza tabeli będzie wartość `TRUE` i dlatego wszystkie wiersze zostaną usunięte.

Jeżeli blok posiada etykietę, można w dalszym ciągu użyć tej samej nazwy dla zmiennej jak dla kolumny tabeli — dzięki nadanej etykiecie do odwołania do zmiennej. Przedstawiony poniżej blok daje pożądany efekt, a mianowicie usuwa tylko te wiersze, dla których `department = 'CS'`:

```
<<l_DeleteBlock>>
DECLARE
  Department CHAR(3);
BEGIN
  Department := 'CS';
  -- Usun wszystkie klasy wydzialu Informatyka
  DELETE FROM classes
    WHERE department = l_DeleteBlock.Department;
END;
```

Mimo że ten sposób działania prowadzi do uzyskania pożądanego wyniku, jednak stosowanie tej samej nazwy dla zmiennej PL/SQL i dla kolumny tabeli nie jest cechą dobrego stylu programowania. Te i inne wytyczne dotyczące stylu programowania w języku PL/SQL omówiono w końcowej części rozdziału 2.

Porównania znaków

W poprzednim przykładzie znajduje się fragment kodu służący do porównywania wartości dwóch znaków. W systemie Oracle mogą być zastosowane dwa różne rodzaje porównań: z dopełnieniem odstępu (*blank-padded*) lub bez dopełnienia odstępu (*non-blank-padded*).

Te dwa rodzaje porównań różnią się w sposobie porównywania ciągów znaków o różnych długościach. Przyjęto, że są porównywane dwa ciągi znaków: ciagznakow1 oraz ciagznakow2. Do *porównania z dopełnieniem odstępu* stosuje się następujący algorytm.

1. Jeżeli ciągi znaków ciagznakow1 oraz ciagznakow2 są różnej długości, krótszy ciąg należy dopełnić znakami odstępu (spacjami), tak aby obydwa miały tą samą długość.
2. Następnie porównuje się każdy ciąg znaków, znak po znaku, zaczynając od lewej strony. Przykładowo, w ciągu znaków ciagznakow1 znakiem jest znak1, a w ciągu znaków ciagznakow2 znakiem jest znak2.
3. Jeżeli ASCII(znak1) < ASCII(znak2), to ciagznakow1 < ciagznakow2.
   Jeżeli ASCII(znak1) > ASCII(znak2), to ciagznakow1 > ciagznakow2.
   Jeżeli ASCII(znak1) = ASCII(znak2), to odpowiednio w ciągach znaków ciagznakow1 oraz ciagznakow2 przechodzi się do następnego znaku.
4. Jeżeli możliwe jest osiągnięcie końców ciągów znaków ciagznakow1 oraz ciagznakow2, wtedy te ciągi są sobie równe.

Przy zastosowaniu algorytmu *porównania z dopełnieniem odstępu* wszystkie poniższe warunki zwrócą wartość TRUE:

```
'abc' = 'abc'
'abc   ' = 'abc'  -- Nalezy zwrocic uwage na koncowe znaki odstepu
                  -- w pierwszym ciagu znakow.
'ab' < 'abc'
'abcd' > 'abcc'
```

Algorytm *porównania bez dopełnienia odstępu* jest nieco inny.

1. Należy porównać każdy ciąg znaków, znak po znaku, zaczynając od lewej strony. Przykładowo, w ciągu znaków ciagznakow1 znakiem jest znak1, a w ciągu znaków ciagznakow2 znakiem jest znak2.
2. Jeżeli ASCII(znak1) < ASCII(znak2), to ciagznakow1 < ciagznakow2.
   Jeżeli ASCII(znak1) > ASCII(znak2), to ciagznakow1 > ciagznakow2.
   Jeżeli ASCII(znak1) = ASCII(znak2), to należy przejść do następnego znaku odpowiednio w ciągach znaków ciagznakow1 oraz ciagznakow2.
3. Jeżeli ciąg znaków ciagznakow1 kończy się przed ciągiem znaków ciagznakow2, wtedy ciagznakow1 < ciagznakow2. Jeżeli ciąg znaków ciagznakow2 kończy się przed ciągiem znaków ciagznakow1, wtedy ciagznakow1 > ciagznakow2.

Przy zastosowaniu algorytmu *porównania bez dopełnienia odstępu* poniższe warunki zwrócą wartość TRUE:

```
'abc' = 'abc'
'ab' < 'abc'
'abcd' > 'abcc'
```

Jednak poniższe *porównanie bez dopełnienia odstępu* zwróci wartość FALSE, ponieważ ciągi znaków są różnej długości. Jest to podstawowa różnica pomiędzy powyższymi dwoma metodami porównań.

```
'abc   ' = 'abc'  -- Nalezy zwrocic uwage na koncowe znaki odstepu
                  -- w pierwszym ciagu znakow.
```

Po zdefiniowaniu tych dwóch różnych metod porównań warto się zastanowić, kiedy należy stosować każdą z nich. Język PL/SQL wykorzystuje metody porównania z dopełnieniem odstępu tylko wtedy, gdy obydwa porównywane ciągi znaków są stałej długości. Jeżeli dane ciągi znaków są różnej długości, stosuje się metodę porównywania bez dopełniania odstępu. Typ danych CHAR określa ciąg znaków o stałej długości, a typ danych VARCHAR2 określa ciąg znaków o zmiennej długości. Stałe znakowe (objęte znakami apostrofu) są zawsze uważane za ciągi znaków o stałej długości.

Jeżeli dana instrukcja nie jest wykonywana na poprawnych wierszach, należy sprawdzić typy danych użyte w klauzuli WHERE. Wykonanie poniższego bloku nie spowoduje usunięcia jakichkolwiek wierszy, ponieważ zmienna v_Department jest typu VARCHAR2, a nie typu CHAR:

```
DECLARE
  v_Department VARCHAR2(3);
BEGIN
  v_Department := 'CS';
  -- Usun wszystkie klasy wydzialu Informatyka
  DELETE FROM classes
    WHERE department = v_Department;
END;
```

Kolumna department tabeli classes ma zdefiniowany typ danych CHAR. Wszystkie klasy informatyki (*computer science*) posiadają wartości 'CS' dla kolumny department — należy zwrócić uwagę na końcowy znak odstępu. Zmienna v_Department = 'CS' nie ma końcowego znaku odstępu i zdefiniowano dla niej typ danych o zmiennej długości, a zatem instrukcja DELETE nie będzie miała żadnego wpływu na wiersze.

Aby klauzula WHERE dała pożądany efekt, dla zmiennych w bloku PL/SQL należy zdefiniować ten sam typ danych, co dla porównywanych kolumn bazy danych. Taki efekt gwarantuje zastosowanie atrybutu %TYPE.

Odwołania do tabel

Wszystkie operacje DML odwołują się do tabeli. Poniżej przedstawiono przykład takiego odwołania:

```
[schemat.]table[@powiazaniebazydanych]
```

gdzie schemat identyfikuje właściciela tabeli, a powiazaniebazydanych identyfikuje tabelę w odległej bazie danych.

W celu nawiązania połączenia z bazą danych konieczne jest podanie nazwy użytkownika i hasła dla odpowiedniego schematu użytkownika. Późniejsze instrukcje SQL, wydawane podczas sesji, będą domyślnie odwoływać się do tego schematu. Jeżeli odwołanie do tabeli jest niesklasyfikowane, jak w poniższym przykładzie:

```
UPDATE students
  SET major = 'Music'
  WHERE id = 10005;
```

wtedy ta nazwa tabeli (w tym przykładzie students) musi być nazwą tabeli w schemacie domyślnym. Jeżeli tak nie jest, wystąpi błąd:

```
ORA-942: table or view does not exist
        (tabela lub perspektywa nie istnieje)
```

```
PLS-201: identifier must be declared
       (identyfikator musi byc zadeklarowany)
```

Schemat domyślny jest schematem, z którym użytkownik jest połączony przed wykonywaniem jakichkolwiek instrukcji. Jeżeli dana tabela znajduje się w innym schemacie, może być kwalifikowana przez nazwę schematu, jak w kolejnym przykładzie:

```
UPDATE example.students
  SET major = 'Music'
  WHERE id = 10005;
```

Instrukcja UPDATE w powyższym przykładzie będzie wykonana, jeżeli zostanie nawiązane połączenie ze schematem tabeli example lub z innym schematem, dla którego przyznano uprawnienie UPDATE na tabeli students.

Powiązania bazy danych

Jeżeli w systemie operacyjnym zainstalowano program SQL*Net, można wykorzystywać powiązania bazy danych. *Powiązanie bazy danych* jest odwołaniem do odległej bazy danych. Odległa baza danych może pracować w zupełnie innym systemie operacyjnym niż lokalna baza danych. Poniżej przedstawiono instrukcję DDL, która tworzy powiązanie bazy danych:

```
CREATE DATABASE LINK nazwa_powiazania
  CONNECT TO nazwauzytkownika IDENTIFIED BY haslo
  USING ciagznakow_sql;
```

Nazwa powiązania bazy danych nazwa_powiazania podlega tym samym regułom składniowym, co identyfikator bazy danych. Schemat w odległej bazie danych jest identyfikowany przez nazwę użytkownika nazwauzytkownika oraz hasło haslo, a ciagznakow_sql jest poprawnym ciągiem znaków połączenia dla odległej bazy danych. Zakładając, że utworzono odpowiednie schematy i że jest zainstalowany program SQL*Net wersja 2., poniższy, przykładowy fragment kodu tworzy powiązanie bazy danych:

```
CREATE DATABASE LINK example_backup
  CONNECT TO example IDENTIFIED BY example
  USING 'backup_database';
```

Więcej informacji dotyczących sposobów instalowania i konfigurowania programu SQL*Net znajduje się w publikacji *SQL*Net User's Guide and Reference*. Kontynuując powyższy przykład, za pomocą utworzonego powiązania można zdalnie modyfikować tabelę students. W tym celu wpisuje się poniższy fragment kodu:

```
UPDATE students@example_backup
  SET major = 'Music'
  WHERE id = 10005;
```

Jeśli powiązanie bazy danych stanowi część transakcji, wtedy ta transakcja jest nazywana transakcją rozproszoną, ponieważ jej rezultatem jest zmodyfikowanie więcej niż jednej bazy danych. Więcej informacji dotyczących transakcji rozproszonych i sposobów zarządzania nimi znajduje się w publikacji *Server SQL Reference*.

Synonimy

Odwołania do tabel mogą być skomplikowane, szczególnie kiedy obejmują operacje związane ze schematem użytkownika i (lub) powiązaniem bazy danych. Aby ułatwić obsługę skomplikowanych odwołań, system Oracle pozwala na tworzenie ich synonimów. Synonim w zasadzie zmienia nazwę odwołania do tabeli, podobnie jak alias dla elementu listy wyboru instrukcji SELECT. Synonim jest obiektem słownika danych i jest tworzony przez instrukcję DDL CREATE SYNONYM:

```
CREATE SYNONYM nazwa_synonimu FOR odwolanie;
```

W celu utworzenia synonimu należy za pomocą powyższego polecenia wstawić w miejsce nazwa_synonimu nazwę synonimu, a w miejsce odwolanie obiekt schematu, do którego ma nastąpić odwołanie. Obiektem tym może być tabela, jak w poniższym przykładzie, ale może to być również procedura, sekwencja lub inny obiekt bazy danych.

```
CREATE SYNONYM example_students
  FOR students@example_backup;
```

Kontynuując powyższy przykład, za pomocą utworzonego synonimu można zapisać rozproszoną instrukcję UPDATE:

```
UPDATE example_students
  SET major = 'Music'
  WHERE id = 10005;
```

Utworzenie synonimu nie powoduje przyznania żadnych uprawnień na odwoływanym obiekcie — po prostu umożliwia wykorzystywanie jego alternatywnej nazwy. Jeżeli obiekt wymaga odwołania z innego schematu, dostęp do tego obiektu powinien być przyznany albo jawnie, albo przez rolę (za pomocą instrukcji GRANT).

Pseudokolumny

Pseudokolumny są dodatkowymi obiektami, które mogą być wywoływane tylko z poziomu instrukcji SQL. Pod względem składniowym pseudokolumny są traktowane jak kolumny w tabeli. Jednak faktycznie nie istnieją w ten sam sposób jak kolumny. Zamiast tego są one określane jako część wykonania instrukcji SQL.

Pseudokolumny CURRVAL oraz NEXTVAL

Pseudokolumny CURRVAL oraz NEXTVAL są używane wraz z sekwencjami. *Sekwencją* jest obiekt Oracle, który jest używany do generowania unikatowych liczb. Sekwencja jest tworzona za pomocą polecenia DDL CREATE SEQUENCE. Po utworzeniu sekwencji można uzyskać do niej dostęp. W tym celu wydaje się polecenie:

```
sekwencja.CURRVAL
```

oraz

```
sekwencja.NEXTVAL
```

gdzie sekwencja jest nazwą sekwencji. Pseudokolumna CURRVAL zwraca bieżącą wartość sekwencji, a pseudokolumna NEXTVAL inkrementuje sekwencję i zwraca nową wartość. Obydwie pseudokolumny CURRVAL oraz NEXTVAL zwracają wartości typu NUMBER.

Wartości sekwencji mogą być używane w liście wyboru zapytania, w klauzuli VALUES instrukcji INSERT oraz w klauzuli SET instrukcji UPDATE. Jednak nie mogą być one zastosowane w klauzuli WHERE lub w instrukcji proceduralnej PL/SQL. Poniżej podano przykłady poprawnego wykorzystania pseudokolumn CURRVAL oraz NEXTVAL:

```
CREATE SEQUENCE student_sequence
  START WITH 10000;

-- W tej instrukcji 10000 zostanie wykorzystane jako wartosc identyfikatora ID
INSERT INTO students (id, first_name, last_name)
  VALUES (student_sequence.NEXTVAL, 'Scott', 'Smith');

-- W tej instrukcji 10001 zostanie wykorzystane jako wartosc identyfikatora ID
INSERT INTO students (id, first_name, last_name)
  VALUES (student_sequence.NEXTVAL, 'Margaret', 'Mason');

SELECT student_sequence.NEXTVAL "Value"
  FROM dual; -- Najpierw inkrementuje liczbe sekwencji
Value
----------
10002

SELECT student_sequence.CURRVAL "Value"
  FROM dual; -- Zwraca biezaca wartosc
Value
----------
10002
```

Pseudokolumna LEVEL

Pseudokolumna LEVEL jest wykorzystywana tylko wewnątrz instrukcji SELECT, która umożliwia poruszanie się po drzewie hierarchii, obejmującym daną tabelę. Podczas tego procesu są stosowane klauzule START WITH oraz CONNECT BY. Pseudokolumna LEVEL zwraca bieżący poziom drzewa jako wartość typu NUMBER. Więcej informacji na ten temat znajduje się w publikacji *Server SQL Reference*.

Pseudokolumna ROWID

Pseudokolumna ROWID jest wykorzystywana w liście wyboru zapytania. Zwraca ona identyfikator danego wiersza. Formatem pseudokolumny jest 18-znakowy ciąg znaków, co opisano w rozdziale 2. Pseudokolumna ROWID zwraca wartość typu ROWID. W poniższym przykładzie zapytanie zwraca wszystkie identyfikatory wierszy w tabeli rooms (pokoje):

```
SELECT ROWID
  FROM rooms;

ROWID
------------------
00000045.0000.0002
00000045.0001.0002
```

```
00000045.0002.0002
00000045.0003.0002
00000045.0004.0002
```

Identyfikator ROWID w wersji Oracle8 różni się od identyfikatora ROWID w wersji Oracle7. Mimo to format zewnętrzny dla obydwóch wersji jest w dalszym ciągu 18-znakowym ciągiem znaków. Więcej informacji na ten temat znajduje się w rozdziale 2.

Pseudokolumna ROWNUM

Pseudokolumna ROWNUM zwraca bieżący numer wiersza w zapytaniu. Umożliwia to ograniczenie całkowitej liczby wierszy. Pseudokolumna ROWNUM jest wykorzystywana głównie w klauzuli WHERE zapytań oraz w klauzuli SET instrukcji UPDATE. Pseudokolumna ROWNUM zwraca wartość typu NUMBER. Wykonanie poniższego zapytania spowoduje zwrócenie tylko dwóch pierwszych wierszy z tabeli students:

```
SELECT *
  FROM students
  WHERE ROWNUM < 3;
```

Pierwszy wiersz posiada ROWNUM 1, drugi — ROWNUM 2, itd.

Wartość ROWNUM jest przypisywana wierszowi przed wykonaniem operacji sortowania (za pomocą klauzuli ORDER BY). W rezultacie nie można zastosować tej pseudokolumny w celu pobrania wierszy o najniższych wartościach ROWNUM dla określonej kolejności wyszukiwania. Warto rozważyć przykładową, poniższą instrukcję:

```
SELECT first_name, last_name
  FROM students
  WHERE ROWNUM < 3
  ORDER BY first_name;
```

Wprawdzie powyższa instrukcja zwraca dwa wiersze z tabeli students (studenci), posortowane według kolumny first_name (imię), jednak niekoniecznie te wiersze są dwoma pierwszymi wierszami według kolejności w całym sortowaniu. Aby to osiągnąć, najlepiej zadeklarować kursor dla tego zapytania i pobrać tylko dwa pierwsze wiersze. Informacje dotyczące kursorów i sposobów ich stosowania przedstawiono w rozdziale 6.

Instrukcje GRANT i REVOKE. Uprawnienia

Wprawdzie instrukcje DDL, takie jak GRANT i REVOKE, nie mogą być bezpośrednio stosowane w programie PL/SQL, jednak mają one pewien wpływ na poprawność instrukcji SQL. W celu wykonania instrukcji, takiej jak INSERT lub DELETE, na tabeli Oracle jest konieczne posiadanie pewnych uprawnień. Manipulowanie tymi uprawnieniami następuje za pomocą instrukcji SQL: GRANT oraz REVOKE.

Uprawnienia obiektowe a uprawnienia systemowe

Istnieją dwa różne rodzaje uprawnień: obiektowe i systemowe. *Uprawnienie obiektowe* pozwala na wykonywanie operacji na danym obiekcie (takim jak tabela). *Uprawnienie systemowe* pozwala na wykonanie operacji na całej klasie obiektów.

W tabeli 4.3 opisano dostępne uprawnienia obiektowe. Uprawnienia obiektowe DDL (`ALTER`, `INDEX`, `REFERENCES`) nie mogą być zastosowane bezpośrednio w języku PL/SQL (z wyjątkiem pakietu `DBMS_SQL`), ponieważ pozwalają one na przeprowadzenie operacji DDL na rozpatrywanym obiekcie.

Tabela 4.3. *Uprawnienia obiektowe SQL*

Uprawnienie obiektowe	Opis	Typy obiektów schematu
`ALTER`	Pozwala użytkownikowi, któremu przyznano uprawnienie, na wydawanie instrukcji `ALTER` (takiej jak `ALTER TABLE`) dotyczącej obiektu.	Tabele, sekwencje
`DELETE`	Pozwala użytkownikowi, któremu przyznano uprawnienie, na wydawanie instrukcji `DELETE` dotyczącej obiektu.	Tabele, perspektywy
`EXECUTE`	Pozwala użytkownikowi, któremu przyznano uprawnienie, na wykonanie składowanego obiektu PL/SQL (informacje dotyczące obiektów składowanych znajdują się w rozdziałach od 7. do 9.).	Procedury, funkcje, pakiety
`INDEX`	Pozwala użytkownikowi, któremu przyznano uprawnienie, na utworzenie indeksu na tabeli za pomocą polecenia `CREATE INDEX`.	Tabele
`INSERT`	Pozwala użytkownikowi, któremu przyznano uprawnienie, na wydawanie instrukcji `INSERT` w odniesieniu do obiektu.	Tabele, perspektywy
`REFERENCES`	Pozwala użytkownikowi, któremu przyznano uprawnienie, na utworzenie ograniczenia, które odwołuje się do tabeli.	Tabele
`SELECT`	Pozwala użytkownikowi, któremu przyznano uprawnienie, na wydawanie instrukcji `SELECT` dotyczącej obiektu.	Tabele, perspektywy, sekwencje, migawki
`UPDATE`	Pozwala użytkownikowi, któremu przyznano uprawnienie, na wydawanie instrukcji `UPDATE` dotyczącej obiektu.	Tabele, perspektywy

Istnieje wiele uprawnień systemowych, dotyczą one prawie każdej możliwej operacji DDL. Na przykład, uprawnienie systemowe `CREATE TABLE` pozwala użytkownikowi, któremu przyznano to uprawnienie, na tworzenie tabel. Uprawnienie systemowe `CREATE ANY TABLE` pozwala użytkownikowi, któremu przyznano uprawnienie, na tworzenie tabel w innym schemacie użytkownika. Publikacja *Server SQL Reference* dokumentuje wszystkie dostępne uprawnienia systemowe.

Instrukcje GRANT oraz REVOKE

Instrukcja `GRANT` jest używana w celu umożliwienia innemu schematowi uzyskania dostępu do danego uprawnienia, natomiast instrukcja `REVOKE` służy do blokowania dostępu do uprawnienia, uzyskanego za pomocą instrukcji `GRANT`. Obydwie instrukcje mogą być stosowane do zarządzania uprawnieniami systemowymi i obiektowymi.

Instrukcja GRANT

Poniżej przedstawiono składnię instrukcji GRANT dla uprawnień obiektowych:

```
GRANT uprawnienie ON obiekt TO uzytkownik [WITH GRANT OPTION];
```

gdzie uprawnienie jest nadawanym uprawnieniem, obiekt jest obiektem, do którego jest przyznawany dostęp, a uzytkownik jest użytkownikiem, który otrzymuje uprawnienie. Przykładowo, jeśli uzytkownikA jest poprawnym schematem bazy danych, poniższa instrukcja GRANT jest prawidłowa:

```
GRANT SELECT ON classes TO uzytkownikA;
```

W razie określenia opcji WITH GRANT OPTION użytkownik uzytkownikA otrzymuje prawo przyznawania tego uprawnienia innym użytkownikom. W jednej instrukcji GRANT można określać więcej niż jedno uprawnienie obiektowe, przykładowo:

```
GRANT UPDATE, DELETE ON students TO uzytkownikA;
```

Poniżej przedstawiono składnię instrukcji GRANT służącej do przydzielania uprawnień systemowych:

```
GRANT uprawnienie TO uzytkownik [WITH ADMIN OPTION];
```

gdzie uprawnienie jest uprawnieniem systemowym, a uzytkownik jest użytkownikiem, który otrzymuje uprawnienie. W razie określenia opcji WITH ADMIN OPTION użytkownik uzytkownik otrzymuje prawo przyznawania tego uprawnienia innym użytkownikom. Na przykład:

```
GRANT CREATE TABLE, ALTER ANY PROCEDURE to uzytkownikA;
```

Podobnie jak w przypadku instrukcji GRANT dla uprawnień obiektowych, w jednej instrukcji GRANT można określać większą liczbę uprawnień systemowych.

GRANT jest instrukcją DDL, a więc jest natychmiast uaktywniana i po jej wykonaniu jest wydawana niejawnie instrukcja potwierdzenia COMMIT.

Instrukcja REVOKE

Poniżej przedstawiono składnię instrukcji REVOKE, która służy do odbierania uprawnień obiektowych:

```
REVOKE uprawnienie ON obiekt FROM uzytkownik [CASCADE CONSTRAINTS];
```

gdzie uprawnienie jest odbieranym uprawnieniem, obiekt jest obiektem, dla którego uprawnienie jest odbierane, a uzytkownik jest użytkownikiem, któremu uprawnienie jest odbierane. Przykładowo, poniżej przedstawiono poprawny sposób zastosowania instrukcji REVOKE:

```
REVOKE SELECT ON classes FROM uzytkownikA;
```

W razie odbierania uprawnienia REFERENCES i uwzględnienia klauzuli CASCADE CONSTRAINTS następuje usunięcie wszystkich więzów integralności referencyjnej, utworzonych przez użytkownika, któremu odbiera się to uprawnienie.

Istnieje możliwość odbierania wielu uprawnień za pomocą jednej instrukcji.

```
REVOKE UPDATE, DELETE, INSERT ON students FROM uzytkownikA;
```

Poniżej przedstawiono składnię instrukcji REVOKE, za pomocą odbiera się uprawnienia systemowe:

```
REVOKE uprawnienie FROM uzytkownik;
```

gdzie uprawnienie jest odbieranym uprawnieniem systemowym, a uzytkownik jest użytkownikiem, któremu jest odbierane dane uprawnienie. Poniżej podano przykładowe, poprawne zastosowanie instrukcji REVOKE:

```
REVOKE ALTER TABLE, EXECUTE ANY PROCEDURE FROM uzytkownikA;
```

Role

W przypadku dużego systemu Oracle, gdzie istnieje wiele różnych kont użytkowników, zarządzanie uprawnieniami może być skomplikowanym zadaniem. W celu ułatwienia zarządzania kontami w systemie Oracle stosuje się role. *Rola* w zasadzie jest zbiorem uprawnień, zarówno systemowych, jak i obiektowych. Warto przeanalizować następujący zestaw instrukcji:

```
CREATE ROLE table_query;
GRANT SELECT ON students TO table_query;
GRANT SELECT ON classes TO table_query;
GRANT SELECT ON rooms TO table_query;
```

Z powyższego wynika, że roli table_query przydzielono uprawnienie SELECT na trzech różnych tabelach. Teraz można przyznać tę rolę użytkownikom:

```
GRANT table_query TO uzytkownikA;
GRANT table_query TO uzytkownikB;
```

W ten sposób użytkownicy: uzytkownikA i uzytkownikB otrzymali uprawnienie SELECT na trzech tabelach. Taki sposób postępowania może ułatwić zarządzanie systemem, ponieważ powyższym działaniem zastąpiono sześć oddzielnych operacji przyznawania uprawnienia SELECT.

Rola PUBLIC jest rolą predefiniowaną w systemie Oracle. Jest ona przyznawana automatycznie każdemu użytkownikowi. A zatem można wykonywać poniższą instrukcję:

```
GRANT uprawnienie TO PUBLIC;
```

W ten sposób dane uprawnienie jest przyznawane każdemu użytkownikowi systemu Oracle.

W systemie Oracle istnieją także inne predefiniowane role. Zawierają one typowe uprawnienia systemowe. Role te wymieniono w tabeli 4.4. Warto zauważyć, ze wszystkie wymienione w tej tabeli role są automatycznie przyznawane użytkownikowi SYSTEM, który jest predefiniowanym użytkownikiem systemu Oracle.

Zwykle role CONNECT oraz RESOURCE są przyznawane użytkownikom bazy danych, którzy mają tworzyć obiekty schematu, a sama rola CONNECT jest przyznawana użytkownikom, którzy wykonują zapytania na obiektach schematu. Użytkownicy, którym przyznano tylko rolę CONNECT, mogą wymagać dodatkowych uprawnień na obiektach schematu, do których muszą mieć dostęp.

Tabela 4.4. *Predefiniowane role systemowe*

Nazwa roli	Przyznane uprawnienia
CONNECT	ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW
RESOURCE	CREATE CLUSTER, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE PROCEDURE
DBA	Wszystkie uprawnienia systemowe (z opcją ADMIN OPTION, więc właściciel roli DBA może je przyznawać z kolei innym użytkownikom), plus EXP_FULL_DATABASE oraz IMP_FULL_DATABASE
EXP_FULL_DATABASE	SELECT ANY TABLE, BACKUP ANY TABLE, plus INSERT, UPDATE, DELETE na tabelach systemowych sys.incexp, sys.incvid oraz sys.incfil
IMP_FULL_DATABASE	BECOME USER

Sterowanie transakcjami

Transakcja jest serią instrukcji SQL, których wykonanie w ramach pewnej jednostki kończy się powodzeniem lub niepowodzeniem. Transakcje są standardowym elementem pracy relacyjnej bazy danych i stanowią zabezpieczenie przed utratą spójności danych. Klasycznym przykładem powyższego jest transakcja bankowa. Warto rozważyć następujące dwie instrukcje SQL, które przeprowadzają transfer kwoty transakcji `kwota_transakcji` pomiędzy dwoma kontami bankowymi, identyfikowanymi jako `konto_nadawcy` i `konto_odbiorcy`.

```
UPDATE konta
  SET saldo = saldo - kwota_transakcji
  nr_konta = konto_nadawcy;
UPDATE konta
  SET saldo = saldo + kwota_transakcji
  nr_konta = konto_odbiorcy;
```

Na potrzeby niniejszego przykładu przyjęto, że pierwsza instrukcja UPDATE została wykonana z powodzeniem, ale wykonanie drugiej instrukcji zakończyło się niepowodzeniem z powodu wystąpienia pewnego błędu (przykładowo, uszkodzenie w bazie danych lub w sieci). W takiej sytuacji dane są niespójne — stan konta `konto_nadawcy` został zmniejszony, ale stan konta `konto_odbiorcy` nie został zwiększony. Jest oczywiste, że nie jest to pożądana sytuacja, zwłaszcza dla właściciela konta `konto_nadawcy`. Przed taką sekwencją zdarzeń można się zabezpieczyć dzięki połączeniu powyższych dwóch instrukcji w jedną transakcję. W ten sposób albo obydwie transakcje zakończą się powodzeniem, albo obydwie transakcje zakończą się niepowodzeniem. Taki sposób postępowania zabezpiecza przed utratą spójności danych.

Transakcja rozpoczyna się od pierwszej instrukcji SQL, wydanej po poprzedniej transakcji, lub pierwszą instrukcją SQL po nawiązaniu połączenia z bazą danych. Transakcja kończy się instrukcją COMMIT lub ROLLBACK.

Instrukcja COMMIT a instrukcja ROLLBACK

Po wydaniu instrukcji COMMIT transakcja przeprowadzana w bazie danych zostanie zakończona. Wystąpią również poniższe zdarzenia:

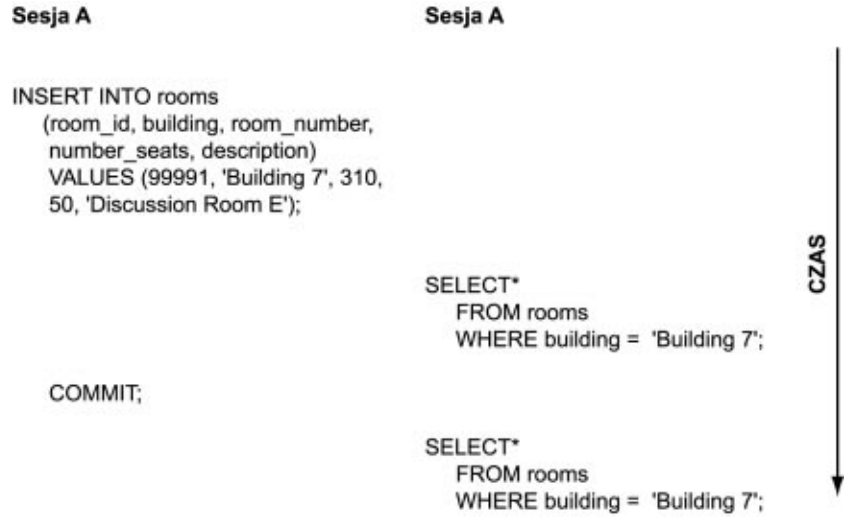
- wyniki pracy wykonane przez transakcję zostaną trwale zachowane;
- zmiany dokonane przez transakcję będą widoczne w innych sesjach;
- wszystkie blokady ustawione przez transakcję zostaną zwolnione.

Poniżej przedstawiono składnię instrukcji COMMIT:

```
COMMIT [WORK]
```

Opcjonalne słowo kluczowe WORK udostępniono w celu zwiększenia zgodności ze standardem SQL. Dopóki transakcja nie zostanie potwierdzona za pomocą instrukcji COMMIT, zmiany dokonane przez tę transakcję są widoczne tylko w sesji, w której dana transakcja jest wykonywana. Taką sytuację pokazano na rysunku 4.2. Najpierw w sesji A jest wydawana instrukcja INSERT. W sesji B jest wykonywane zapytanie na tabeli rooms, jednak w sesji B nie jest widoczne wykonanie instrukcji INSERT przeprowadzanej w sesji A, ponieważ nie zostało one potwierdzone. Następnie w sesji A następuje wydanie instrukcji potwierdzenia COMMIT, a druga instrukcja SELECT w sesji B pokaże nowo wstawiony wiersz.

Rysunek 4.2. *Dwie sesje*



Po wydaniu instrukcji ROLLBACK transakcja przeprowadzana w bazie danych zostaje zakończona oraz zachodzą następujące zdarzenia:

- wszystkie wyniki pracy wykonanej przez transakcję zostają wycofane, jak gdyby transakcja nie była przeprowadzana;
- wszystkie blokady ustawiane przez transakcję zostają zwolnione.

Poniżej przedstawiono składnię instrukcji ROLLBACK:

```
ROLLBACK [WORK]
```

Podobnie jak w instrukcji COMMIT, opcjonalne słowo kluczowe WORK jest dostępne dla zwiększenia zgodności ze standardem SQL. Niejawna instrukcja ROLLBACK jest często wykonywana w razie wykrycia w programie błędu, który uniemożliwia dalszą pracę. W razie nagłego zakończenia sesji (na przykład w razie zerwania połączenia z bazą danych) bez zakończenia

przeprowadzanej transakcji za pomocą instrukcji COMMIT lub ROLLBACK, transakcja jest automatycznie wycofywana z bazy danych.

Program *SQL*Plus* automatycznie wydaje instrukcję COMMIT przy zakańczaniu pracy programu. Także uaktywnienie opcji autocommit powoduje wydawanie instrukcji COMMIT po przeprowadzeniu każdej instrukcji SQL. Jednak nie ma to wpływu na sposób zachowania instrukcji SQL zawartych wewnątrz bloku PL/SQL, ponieważ program *SQL*Plus* nie ma kontroli nad tymi instrukcjami przed zakończeniem działania bloku PL/SQL.

Punkty zachowania

Z poprzedniego podrozdziału wynika, że wydanie instrukcji ROLLBACK powoduje wycofanie całej transakcji. Warto jednak wiedzieć, że po zastosowaniu instrukcji SAVEPOINT tylko część transakcji wymaga wycofania. Poniżej przedstawiono składnię instrukcji SAVEPOINT:

```
SAVEPOINT nazwa;
```

gdzie nazwa jest nazwą punktu zachowania. Nazwy punktów zachowania podlegają tym samym zasadom składniowym, co identyfikatory SQL (patrz rozdział 2.). Należy zwrócić uwagę, że punkty zachowania nie są deklarowane w sekcji deklaracji, ponieważ dla danej transakcji mają one znaczenie globalne i wykonywanie transakcji może być kontynuowane aż do zakończenia bloku. Zatem jeśli zdefiniowano punkt zachowania, wykonywanie programu może być cofnięte do danego punktu zachowania. W tym celu wydaje się polecenie o następującej składni:

```
ROLLBACK[WORK] TO SAVEPOINT nazwa;
```

Po wydaniu polecenia ROLLBACK TO SAVEPOINT zachodzą następujące zdarzenia:

- każde działanie wykonane od punktu zachowania jest wycofywane. Jednak sam punkt wycofania pozostaje aktywny. W razie potrzeby punkt zachowania może być wykorzystany do ponownego wycofania części transakcji;
- blokady i zasoby nabyte dzięki instrukcjom SQL od punktu zachowania zostają zwolnione;
- transakcja nie ulega zakończeniu, ponieważ wykonywanie instrukcji SQL w dalszym ciągu jest zawieszone.

Warto rozważyć następujący fragment bloku PL/SQL:

```
BEGIN
  INSERT INTO temp_table (char_col) VALUES ('Wstaw jeden');
  SAVEPOINT A;
  INSERT INTO temp_table (char_col) VALUES ('Wstaw dwa');
  SAVEPOINT B;
  INSERT INTO temp_table (char_col) VALUES ('Wstaw trzy');
  SAVEPOINT C;
  /* Tutaj instrukcje brakujace*/
  COMMIT;
END;
```

Jeżeli do instrukcji brakujących zostanie wprowadzona instrukcja:

```
ROLLBACK TO B;
```

wtedy trzecia instrukcja INSERT oraz punkt zachowania C zostaną wycofane. Jednak dwie pierwsze instrukcje INSERT będą przetwarzane. Z drugiej strony, jeżeli do instrukcji brakujących zostanie wprowadzona instrukcja:

```
ROLLBACK TO A;
```

wtedy druga i trzecia instrukcja INSERT zostanie wycofana i do przetwarzania pozostanie tylko pierwsza instrukcja INSERT.

Instrukcja SAVEPOINT jest często używana przed rozpoczęciem skomplikowanej sekcji transakcji. Jeżeli część transakcji nie powiedzie się, można ją wycofać, a następnie kontynuować jej wcześniejszą część.

Transakcje a bloki

Należy zwrócić uwagę na pewne różnice między transakcjami a blokami PL/SQL. Rozpoczęcie wykonywania bloku nie oznacza rozpoczęcia wykonywania transakcji. Podobnie rozpoczęcie wykonywania transakcji nie musi wiązać się z rozpoczęciem wykonywania bloku. Przykładowo, warto rozważyć sytuację, w której z wiersza poleceń programu *SQL*Plus* wydano następujące instrukcje:

Dostępne na płycie CD w skrypcie *1trans.sql*

```
INSERT INTO classes
   (department, course, description, max_students,
    current_students, num_credits, room_id)
  VALUES ('CS', 101, 'Computer Science 101', 50, 10, 4, 99998);
BEGIN
  UPDATE rooms
    SET room_id = room_id - 1000;
  ROLLBACK WORK;
END;
```

Z powyższego fragmentu kodu wynika, ze wydano instrukcję INSERT, a następnie anonimowy blok PL/SQL. W bloku tym znajduje się instrukcja UPDATE i następnie instrukcja ROLLBACK. Instrukcja ROLLBACK wycofuje nie tylko instrukcję UPDATE, ale również wcześniejszą instrukcję INSERT. Zarówno instrukcja INSERT, jak i blok są częścią tej samej sesji bazy danych, a w ten sposób także tej samej transakcji.

W podobny sposób jeden blok może zawierać w sobie wiele transakcji. Przykładowo:

Dostępne na płycie CD w skrypcie *1block.sql*

```
DECLARE
  v_NumIterations   NUMBER;
BEGIN
  -- Wykonaj petle od 1 do 500, wstawiajac te wartosci do tabeli
  -- temp_table. Wykonuj instrukcje COMMIT po kazdych 50 wierszach.
  FOR v_LoopCounter IN 1..500 LOOP
    INSERT INTO temp_table (num_col) VALUES (v_LoopCounter);
    v_NumIterations := v_NumIterations + 1;
```

```
    IF v_NumIterations = 50 THEN
      COMMIT;
      v_NumIterations := 0;
    END IF;
  END LOOP;
END;
```

Powyższy blok powoduje wstawianie liczb o wartościach od 1 do 500 do tabeli `temp_table` i potwierdza wstawienie tych liczb po każdych 50 wierszach. Zatem podczas wykonania jednego bloku zostanie przeprowadzonych 10 transakcji.

Podsumowanie

W niniejszym rozdziale ogólnie omówiono zagadnienia dotyczące instrukcji SQL, DML oraz instrukcji sterowania transakcjami, dozwolonymi w szczególności w języku PL/SQL. Podano również informacje związane z uprawnieniami i rolami, a także przedstawiono sposób zabezpieczania danych przed utratą spójności za pomocą mechanizmu transakcji. W następnym rozdziale zostaną opisane wbudowane funkcje SQL. W rozdziale 6. zostaną omówione kursory, które są stosowane dla zapytań wielowierszowych. W celu prawidłowego zrozumienia prezentowanych tam zagadnień należy przeanalizować koncepcje przedstawione w niniejszym oraz w kolejnym rozdziale.