

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Oracle8. Programowanie w języku PL/SQL

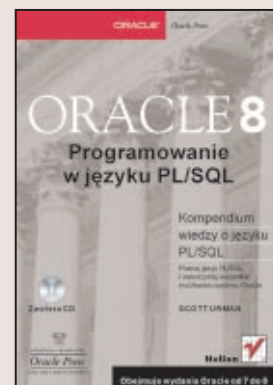
Autor: Scott Urman

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.).



Spis treści

Wstęp	17
Rozdział 1. Wprowadzenie do PL/SQL	23
Dlaczego język PL/SQL?	23
Model klient-serwer	25
Normy	26
Właściwości języka PL/SQL	26
Struktura bloku	26
Zmienne i typy danych	27
Konstrukcje pętlowe	28
Konwencje stosowane w niniejszej książce	30
Wersje języka PL/SQL oraz bazy danych Oracle	30
Dokumentacja Oracle	31
Kod dostępny na płycie CD	32
Przykładowe tabele	32
Podsumowanie	38
Rozdział 2. Podstawy języka PL/SQL	39
Blok PL/SQL	39
Podstawowa struktura bloku	42
Jednostki leksykalne	44
Identyfikatory	44
Ograniczniki	46
Literały	48
Komentarze	49
Deklaracje zmiennych	51
Składnia deklaracji	51
Inicjowanie zmiennych	52
Typy danych w języku PL/SQL	53
Typy skalarne	53
Typy złożone	60
Typy odwołania	60
Typy LOB	60
Wykorzystanie atrybutu %Type	61
Podtypy definiowane przez użytkownika	62
Konwersja pomiędzy typami danych	62
Zakres i widoczność zmiennej	64
Wyrażenia i operatory	65
Przypisanie	66
Wyrażenia	67

Struktury sterowania PL/SQL	69
Instrukcja IF-THEN-ELSE	70
Pętle	73
Instrukcje GOTO oraz etykiety	77
Dyrektywy pragma	80
Styl programowania w języku PL/SQL	80
Wprowadzanie komentarzy	81
Nazywanie zmiennych	82
Stosowanie dużych liter	82
Odstępy w kodzie programu	82
Ogólne uwagi dotyczące stylu programowania	83
Podsumowanie	83
Rozdział 3. Rekordy i tabele	85
Rekordy w języku PL/SQL	85
Przypisanie rekordu	86
Stosowanie operatora %ROWTYPE	88
Tabele	88
Tabele a tablice	89
Atrybuty tabeli	91
Wytyczne stosowania tabel PL/SQL	94
Podsumowanie	95
Rozdział 4. SQL w PL/SQL	97
Instrukcje SQL	97
Wykorzystanie instrukcji SQL w języku PL/SQL	98
Stosowanie instrukcji DML w języku PL/SQL	99
Instrukcja SELECT	101
Instrukcja INSERT	102
Instrukcja UPDATE	104
Instrukcja DELETE	104
Klauzula WHERE	105
Odwołania do tabel	108
Powiązania bazy danych	109
Synonimy	110
Pseudokolumny	110
Pseudokolumny CURRVAL oraz NEXTVAL	110
Pseudokolumna LEVEL	111
Pseudokolumna ROWID	111
Pseudokolumna ROWNUM	112
Instrukcje GRANT i REVOKE. Uprawnienia	112
Uprawnienia obiektowe a uprawnienia systemowe	113
Instrukcje GRANT oraz REVOKE	113
Role	115
Sterowanie transakcjami	116
Instrukcja COMMIT a instrukcja ROLLBACK	116
Punkty zachowania	118
Transakcje a bloki	119
Podsumowanie	120
Rozdział 5. Wbudowane funkcje SQL	121
Wstęp	121
Funkcje znakowe zwracające wartości znakowe	122
CHR	122
CONCAT	122

INITCAP.....	123
LOWER.....	123
LPAD.....	124
LTRIM.....	125
NLS_INITCAP.....	125
NLS_LOWER.....	126
NLS_UPPER.....	126
REPLACE.....	127
RPAD.....	128
RTRIM.....	128
SOUNDEX.....	129
SUBSTR.....	130
SUBSTRB.....	131
TRANSLATE.....	131
UPPER.....	132
Funkcje znakowe zwracające wartości liczbowe.....	133
ASCII.....	133
INSTR.....	133
INSTRB.....	134
LENGTH.....	135
LENGTHB.....	135
NLSSORT.....	136
Funkcje numeryczne.....	136
ABS.....	136
ACOS.....	137
ASIN.....	137
ATAN.....	138
ATAN2.....	138
CEIL.....	139
COS.....	139
COSH.....	140
EXP.....	140
FLOOR.....	140
LN.....	141
LOG.....	141
MOD.....	142
POWER.....	142
ROUND.....	143
SIGN.....	143
SIN.....	144
SINH.....	144
SQRT.....	144
TAN.....	145
TANH.....	145
TRUNC.....	146
Funkcje związane z datą.....	146
ADD_MONTHS.....	146
LAST_DAY.....	147
MONTHS_BETWEEN.....	147
NEW_TIME.....	148
NEXT_DAY.....	149
ROUND.....	149
SYSDATE.....	150
TRUNC.....	151

Funkcje dokonujące konwersji.....	152
CHARTOROWID	152
CONVERT	153
HEXTORAW	153
RAWTOHEX	154
ROWIDTOCHAR	154
TO_CHAR(daty)	155
TO_CHAR(etykiety)	155
TO_CHAR(liczba).....	157
TO_DATE	159
TO_LABEL	159
TO_MULTI_BYTE.....	160
TO_NUMBER.....	160
TO_SINGLE_BYTE	161
Funkcje grupowe.....	161
AVG.....	161
COUNT.....	162
GLB	163
LUB	163
MAX	163
MIN.....	164
STDDEV.....	164
SUM.....	165
VARIANCE.....	165
Inne funkcje.....	166
BFILENAME	166
DECODE	166
DUMP.....	167
EMPTY_CLOB/EMPTY_BLOB.....	169
GREATEST.....	169
GREATEST_LB.....	170
LEAST	170
LEAST_UB	170
NVL	171
UID	171
USER	172
USERENV.....	172
VSIZE	173
PL/SQL w działaniu. Drukowanie liczb w postaci tekstowej.....	174
Podsumowanie	180
Rozdział 6. Kursory	181
Czym jest kursor?.....	181
Przetwarzanie kursorów jawnych	182
Przetwarzanie kursorów niejawnych	189
Pętle pobierania danych kursora	191
Pętle proste.....	191
Pętle WHILE	193
Pętle FOR kursora.....	194
Wyjątek NO_DATA_FOUND kontra atrybut %NOTFOUND	195
Kursory z klauzulą FOR UPDATE instrukcji SELECT.....	196
Zmienne kursora	199
Deklaracja zmiennej kursora	200
Przydzielenie obszaru pamięci dla zmiennych kursora	201
Otwieranie zmiennej kursora dla zapytania.....	202

Zamykanie zmiennych kursora	203
Pierwszy przykład zmiennej kursora	203
Drugi przykład zmiennej kursora	205
Ograniczenia użycia zmiennych kursora	206
Podsumowanie	207
Rozdział 7. Podprogramy: procedury i funkcje	209
Tworzenie procedur i funkcji	209
Tworzenie procedury	210
Tworzenie funkcji	221
Wyjątki wywoływane wewnątrz podprogramów	224
Usuwanie procedur i funkcji	226
Położenie podprogramów	226
Składowane podprogramy oraz słownik danych	226
Podprogramy lokalne	228
Zależności dotyczące podprogramów	231
Określanie zależności	233
Uprawnienia i podprogramy składowane	236
Uprawnienie EXECUTE	236
Składowane podprogramy i role	237
Podsumowanie	239
Rozdział 8. Pakiety	241
Pakiety	241
Specyfikacja pakietu	241
Ciało pakietu	243
Pakiety i zakres	245
Przeciążenie podprogramów pakietowych	246
Inicjalizacja pakietu	247
Pakiety i zależności	249
Stosowanie składowanych funkcji w instrukcjach SQL	251
Poziomy czystości	252
Parametry domyślne	256
PL/SQL w działaniu — Eksporter schematów PL/SQL	256
Podsumowanie	264
Rozdział 9. Wyzwalacze	265
Tworzenie wyzwalaczy	265
Komponenty wyzwalacza	267
Wyzwalacze i słownik danych	270
Kolejność uruchamiania wyzwalaczy	272
Stosowanie wartości :old oraz :new w wyzwalaczach na poziomie wiersza	273
Korzystanie z predykatów wyzwalacza: INSERTING, UPDATING oraz DELETING	276
Tabele mutujące	278
Przykład tabeli mutującej	280
Rozwiązanie problemu błędu tabeli mutującej	281
PL/SQL w działaniu — wdrażanie techniki kaskadowego uaktualniania	283
Program narzędziowy kaskadowego uaktualniania	285
Działanie pakietu kaskadowego uaktualniania	288
Podsumowanie	292
Rozdział 10. Obsługa błędów	293
Zdefiniowanie wyjątku	293
Deklarowanie wyjątków	295
Wywoływanie wyjątków	298
Obsługa wyjątków	299

Dyrektywa pragma EXCEPTION_INIT	305
Stosowanie funkcji RAISE_APPLICATION_ERROR	305
Propagacja wyjątków	308
Wyjątki wywołane w sekcji wykonania	308
Wyjątki wywołane w sekcji deklaracji	310
Wyjątki wywołane w sekcji wyjątków	312
Wytyczne wyjątków	314
Zakres wyjątków	314
Unikanie nieobsługiwanych wyjątków	315
Maskowanie lokalizacji błędu	315
PL/SQL w działaniu — ogólny program obsługi błędów	316
Podsumowanie	324
Rozdział 11. Obiekty	325
Wprowadzenie	325
Podstawy programowania obiektowego	325
Obiektowo-relacyjne bazy danych	327
Typy obiektów	328
Definiowanie typów obiektowych	328
Deklarowanie i inicjalizacja obiektów	330
Metody	332
Zmiana i usuwanie typów	338
Zależności między obiektami	340
Obiekty w bazie danych	340
Położenie obiektów	341
Obiekty w instrukcjach DML	344
Metody MAP i ORDER	349
Podsumowanie	351
Rozdział 12. Kolekcje	353
Tabele zagnieżdżone	353
Deklarowanie tabeli zagnieżdżonej	353
Zagnieżdżone tabele w bazie danych	356
Tabele zagnieżdżone a tabele indeksowe	361
Tablice o zmiennym rozmiarze	361
Deklarowanie tablicy o zmiennym rozmiarze	362
Tablice o zmiennym rozmiarze w bazie danych	363
Tablice o zmiennym rozmiarze a tabele zagnieżdżone	365
Metody dla kolekcji	366
EXISTS	366
COUNT	367
LIMIT	368
FIRST i LAST	368
NEXT i PRIOR	369
EXTEND	369
TRIM	371
DELETE	373
Podsumowanie	374
Rozdział 13. Środowiska wykonawcze PL/SQL	375
Różne mechanizmy języka PL/SQL	375
Implikacje umieszczenia mechanizmu PL/SQL po stronie klienta	377
Mechanizm PL/SQL po stronie serwera	378
Program SQL*Plus	378
Prekompilatory Oracle	383

OCI	389
Program SQL-Station	392
Mechanizm PL/SQL po stronie klienta.....	396
Przyczyny wykorzystywania mechanizmu PL/SQL po stronie klienta.....	396
Program Oracle Forms.....	397
Program Procedure Builder	399
Wrapper PL/SQL	401
Wykonanie wrappera	401
Pliki wejścia i wyjścia	402
Sprawdzanie syntaktyki i semantyki	402
Wytyczne dla programu wrapper.....	403
Podsumowanie	403
Rozdział 14. Testowanie i wykrywanie błędów.....	405
Diagnostyka problemu	405
Wytyczne wykrywania i usuwania błędów	405
Pakiet Debug służący do wykrywania i usuwania błędów	407
Wstawianie do tabeli testowania.....	407
Problem 1.....	407
Pakiet DBMS_OUTPUT	415
Składniki pakietu DBMS_OUTPUT	416
Problem 2.....	420
Programy PL/SQL służące do wykrywania i usuwania błędów	426
Program Procedure Builder.....	426
Problem 3.....	426
Program SQL-Station.....	432
Problem 4.....	433
Porównanie programów Procedure Builder i SQL-Station.....	437
Metodyka programowania	438
Programowanie modułarne	438
Projektowanie zstępujące.....	439
Abstrakcja danych	440
Podsumowanie	440
Rozdział 15. Dynamiczny PL/SQL	441
Wprowadzenie	441
Instrukcje SQL statyczne a instrukcje dynamiczne	441
Ogólny opis pakietu DBMS_SQL	442
Wykonywanie instrukcji DML oraz DDL nie będących zapytaniami.....	446
Otwieranie kursora.....	447
Parsowanie instrukcji.....	447
Wiązanie każdej zmiennej wyjściowej	448
Wykonanie instrukcji.....	450
Zamykanie kursora	450
Przykład	451
Wykonywanie instrukcji DDL.....	452
Wykonywanie zapytań.....	453
Parsowanie instrukcji.....	454
Zdefiniowanie zmiennych wyjściowych	454
Pobieranie wierszy.....	456
Zwracanie wyników do zmiennych PL/SQL.....	456
Przykład	458
Wykonywanie bloku PL/SQL.....	461
Parsowanie instrukcji.....	461
Pobranie wartości każdej zmiennej wyjściowej	462

Przykład	463
Zastosowanie parametru out_value_size	465
PL/SQL w działaniu — wykonywanie dowolnych procedur składowanych	466
Udoskonalenia pakietu DBMS_SQL w wydaniu PL/SQL 8.0	472
Parsowanie dużych ciągów znaków instrukcji SQL	473
Przetwarzanie tablicowe za pomocą pakietu DBMS_SQL	474
Opisywanie listy instrukcji SELECT	478
Różne procedury	481
Pobieranie danych typu LONG	481
Dodatkowe funkcje obsługi błędów	482
PL/SQL w działaniu — zapisywanie wartości typu LONG do pliku	484
Uprawnienia a pakiet DBMS_SQL	486
Uprawnienia wymagane dla pakietu DBMS_SQL	486
Role a pakiet DBMS_SQL	487
Porównanie pakietu DBMS_SQL z innymi metodami przetwarzania dynamicznego	487
Opisywanie listy instrukcji SELECT	488
Przetwarzanie tablicowe	488
Operacje dzielenia na części danych typu LONG	488
Różnice interfejsów	489
Wskazówki i techniki	489
Ponowne zastosowanie cursorów	489
Zezwolenia	489
Zawieszenia programu związane z operacjami DDL	490
Podsumowanie	490
Rozdział 16. Komunikacja między sesjami	491
Pakiet DBMS_PIPE	491
Wysyłanie komunikatu	495
Odbieranie komunikatu	496
Tworzenie potoków i zarządzanie nimi	498
Uprawnienia i bezpieczeństwo	500
Ustanawianie protokołu komunikacji	501
Przykład	503
Pakiet DBMS_ALERT	509
Wysyłanie ostrzeżenia	509
Odbieranie ostrzeżenia	509
Inne procedury	511
Ostrzeżenia i słownik danych	512
Porównanie pakietów DBMS_PIPE i DBMS_ALERT	514
Podsumowanie	515
Rozdział 17. Zaawansowane kolejkowanie w Oracle	517
Wprowadzenie	517
Elementy systemu zaawansowanego kolejkowania	518
Realizacja zaawansowanego kolejkowania	520
Operacje na kolejkach	520
Typy pomocnicze	521
Operacja ENQUEUE	525
Operacja DEQUEUE	526
Administrowanie kolejką	526
Podprogramy pakietu DBMS_AQADM	526
Uprawnienia do kolejek	533
Kolejki i słownik danych	533
Przykłady	536
Tworzenie kolejek i tabel kolejek	536
Proste wstawianie i odbieranie komunikatów	538

„Czyszczenie” kolejek	539
Wstawianie i odbieranie komunikatów z uwzględnieniem priorytetów	540
Wstawianie i odbieranie komunikatów z wykorzystaniem identyfikatora korelacji lub identyfikatora komunikatu	542
Przeglądanie kolejek	544
Stosowanie kolejek wyjątków	546
Usuwanie kolejek	548
Podsumowanie	549
Rozdział 18. Obsługa zadań i plików w bazie danych	551
Zadania w bazie danych	551
Procesy drugoplanowe	551
Uruchamianie zadania	552
Zadania niewykonane	556
Usuwanie zadania	557
Dokonywanie zmian w zadaniu	557
Przeglądanie zadań w słowniku danych	558
Warunki wykonywania zadań	558
Obsługa plików	558
Zabezpieczenia	559
Wyjątki w pakiecie UTL_FILE	560
Otwieranie i zamykanie plików	561
Zapis do pliku	563
Odczyt z pliku	566
Przykłady	566
Podsumowanie	573
Rozdział 19. Serwer WWW Oracle	575
Środowisko serwera WWW Oracle	575
Agent PL/SQL	577
Określanie wartości parametrów w procedurach	578
Narzędzia WWW w PL/SQL	580
Pakiety HTP i HTF	580
Pakiet OWA_UTIL	594
Pakiet OWA_IMAGE	601
Pakiet OWA_COOKIE	604
Tworzenie procedur OWA	607
Procedura OWA_UTIL.SHOWPAGE	607
SQL-Station	608
Podsumowanie	608
Rozdział 20. Procedury zewnętrzne	609
Czym jest procedura zewnętrzna?	609
Wywoływanie procedury zewnętrznej	610
Odwzorowywanie parametrów	617
Funkcje i procedury zewnętrzne w pakietach	624
Połączenie zwrotne z bazą danych	626
Podprogramy usługowe	626
Wykonywanie instrukcji SQL w procedurze zewnętrznej	629
Wskazówki, wytyczne i ograniczenia	630
Wykrywanie błędów w procedurach zewnętrznych	630
Wytyczne	632
Ograniczenia	633
Podsumowanie	634

Rozdział 21. Duże obiekty	635
Czym są duże obiekty?	635
Składowanie dużych obiektów	636
Duże obiekty w instrukcjach DML.....	637
Obiekty typu BFILE.....	639
Katalogi.....	639
Otwieranie i zamykanie plików BFILE	641
Pliki BFILE w instrukcjach DML	641
Pakiet DBMS_LOB	643
Podprogramy pakietu DBMS_LOB	643
Wyjątki zgłaszane przez podprogramy z pakietu DBMS_LOB.....	658
Porównanie interfejsów DBMS_LOB i OCI.....	658
PL/SQL w działaniu: Kopiowanie danych typu LONG do postaci LOB	659
Podsumowanie	661
Rozdział 22. Wydajność i strojenie	663
Obszar wspólny.....	663
Struktura instancji bazy Oracle.....	663
Jak funkcjonuje obszar wspólny?	667
Rozmiar obszaru wspólnego	669
Unieruchamianie obiektów	670
Strojenie instrukcji SQL.....	672
Generowanie planu wykonania.....	672
Wykorzystywanie planu	678
Sieć.....	679
Wykorzystywanie środowiska PL/SQL po stronie klienta.....	679
Unikanie powtórnej analizy składni	679
Przetwarzanie tablicowe	680
Podsumowanie	680
Dodatek A Słowa zastrzeżone w PL/SQL	681
Dodatek B Pakiety dostępne w PL/SQL	683
Tworzenie pakietów	683
Opis pakietów	683
DBMS_ALERT.....	683
DBMS_APPLICATION_INFO	684
DBMS_AQ i DBMS_AQADM.....	686
DBMS_DEFER, DBMS_DEFER_SYS i DBMS_DEFER_QUERY	686
DBMS_DDL.....	686
DBMS_DESCRIBE.....	687
DBMS_JOB.....	688
DBMS_LOB	688
DBMS_LOCK	688
DBMS_OUTPUT	693
DBMS_PIPE.....	693
DBMS_REFRESH i DBMS_SNAPSHOT	693
DBMS_REPCAT, DBMS_REPCAT_AUTH i DBMS_REPCAT_ADMIN	693
DBMS_ROWID	693
DBMS_SESSION.....	694
DBMS_SHARED_POOL.....	695
DBMS_SQL	696
DBMS_TRANSACTION.....	696
DBMS_UTILITY	698
UTL_FILE	700

Dodatek C	Słownik wybranych elementów PL/SQL	701
Dodatek D	Słownik danych	721
	Czym jest słownik danych?.....	721
	Standardy nazewnictwa	721
	Uprawnienia.....	722
	Perspektywy DBA, All i User w słowniku danych.....	722
	Zależności	723
	Kolekcje	724
	Błędy kompilacji.....	724
	Katalogi.....	725
	Zadania	725
	Biblioteki	726
	Duże obiekty (LOB)	726
	Metody obiektów	727
	Parametry metod obiektów	727
	Wartości zwracane przez metody obiektów	728
	Typy obiektowe	729
	Odwołania do obiektów	729
	Atrybuty typów obiektowych	729
	Obiekty w schemacie	730
	Kod źródłowy	730
	Tabele	731
	Kolumny tabeli	732
	Wyzwalacze.....	733
	Kolumny wyzwalaczy	734
	Perspektywy.....	734
	Inne perspektywy słownika danych	735
	dbms_alert_info	735
	dict_columns.....	735
	Skorowidz	737

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 (<i>Data Manipulation Language</i>)	SELECT, INSERT, UPDATE, DELETE, SET TRANSACTION, EXPLAIN PLAN
Instrukcje języka definicji danych DDL (<i>Data Definition Language</i>)	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 trwał 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 transakcją 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 następnie

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 tworzenia 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 przed uruchomieniem bloku. Rozwiązaniem tego problemu jest zastosowanie pakietu `DBMS_SQL` również do wykonywania instrukcji `INSERT`.

Stosowanie instrukcji DML w języku PL/SQL

Dozwołonymi 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 przetworzonych 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

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT first_name, last_name, major
2 FROM students
3 ORDER BY major;

FIRST_NAME      LAST_NAME      MAJOR
-----
Scott           Smith          Computer Science
Joanne          Junebug        Computer Science
Manish          Murgratroid    Economics
Barbara        Blues          Economics
Margaret        Mason          History
Patrick         Poll           History
Timothy         Taller         History
David           Dinsmore       Music
Rose            Riznit         Music
Ester           Elegant        Nutrition
Rita            Razmataz       Nutrition

11 rows selected.

SQL> UPDATE CLASSES
2 SET num_credits = 3
3 WHERE department = 'HIS'
4 AND course = 101;

1 row updated.

SQL> commit;

Commit complete.

SQL>

```

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*

```

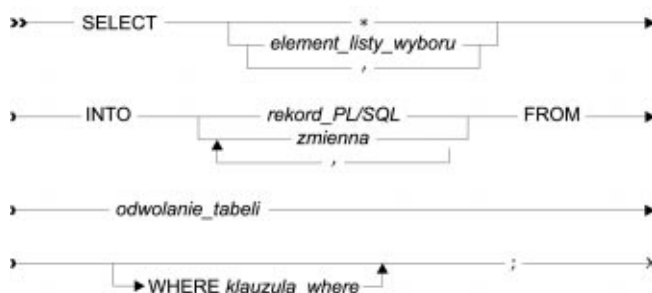
DECLARE
  v_NumCredits classes.num_credits%TYPE;
BEGIN
  /* Przypisz wartosc zmiennej v_NumCredits */
  v_NumCredits := 3;
  UPDATE CLASSES
    SET num_credits = v_NumCredits
    WHERE department = 'HIS'
    AND course = 101;
END;

```


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 <i>listą wyboru</i> . 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 <code>odwołanie_tabeli%ROWTYPE</code> .
odwołanie_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 więcej niż 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*

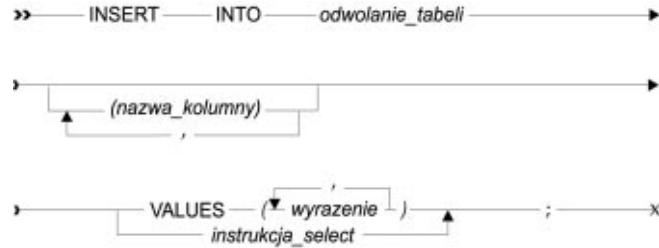
```
DECLARE
  v_StudentRecord  students%ROWTYPE;
  v_Department     classes.department%TYPE;
  v_Course         classes.course%TYPE;
BEGIN
  -- Pobierz jeden rekord z tabeli students i zachowaj w rekordzie
  -- v_StudentRecord. Należy zwrócić uwagę, że klauzula WHERE
  -- odpowiada tylko jednemu wierszowi tabeli. Należy również zwrócić
  -- uwagę, że zapytanie zwraca wszystkie pola w tabeli students
  -- (ponieważ stosowany jest znak *). W ten sposób ładowany 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 (choć może ona występować w podzapytaniu).

Klauzula odwołanie_tabeli odwołuje się do tabeli Oracle, nazwa_kolumny odwołuje się do kolumny w tej tabeli, a wyrażenie 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 użyj numeru sekwencji bezpośrednio 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.

```

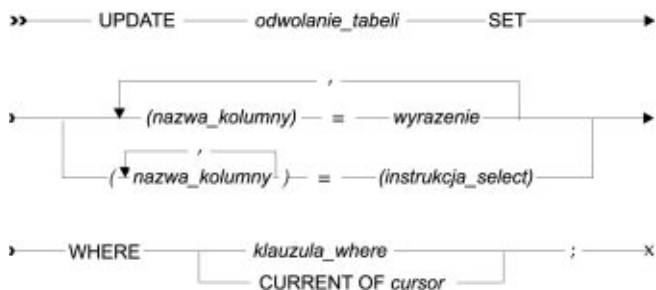
INSERT INTO classes
  SELECT * FROM classes;

```

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 `odwołanie_tabeli` odwołuje się do modyfikowanej tabeli, `nazwa_kolumny` jest nazwą kolumny, której wartość ma być zmieniona, a `wyrażenie` 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:



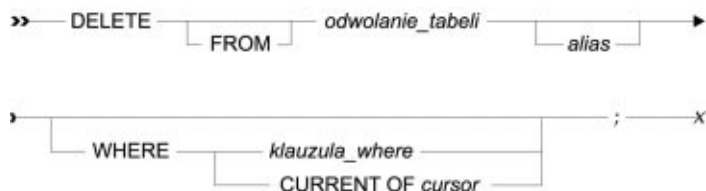
Dostępne na płycie CD w skrypcie `update.sql`

```

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 `odwołanie_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 Ekonomii, 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 wydziału 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:

lwyrażenie = 2wyrażenie

najpierw następuje sprawdzenie identyfikatorów `lwyrażenie` oraz `2wyrażenie` 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 obydwie 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 etykietce do odwołania do zmiennej. Przedstawiony poniżej blok daje pożądaną efekt, a mianowicie usuwa tylko te wiersze, dla których `department = 'CS'`:

```
<<l_DeleteBlock>>
DECLARE
  Department CHAR(3);
BEGIN
  Department := 'CS';
  -- Usun wszystkie klasy wydziału 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: `ciągznakow1` oraz `ciągznakow2`. Do *porównania z dopełnieniem odstepu* stosuje się następujący algorytm.

1. Jeżeli ciągi znaków `ciągznakow1` oraz `ciągznakow2` są różnej długości, krótszy ciąg należy dopełnić znakami odstepu (spacjami), tak aby obydwie miały tę samą długość.
2. Następnie porównuje się każdy ciąg znaków, znak po znaku, zaczynając od lewej strony. Przykładowo, w ciągu znaków `ciągznakow1` znakiem jest `znak1`, a w ciągu znaków `ciągznakow2` znakiem jest `znak2`.
3. Jeżeli $\text{ASCII}(\text{znak1}) < \text{ASCII}(\text{znak2})$, to `ciągznakow1 < ciągznakow2`.
Jeżeli $\text{ASCII}(\text{znak1}) > \text{ASCII}(\text{znak2})$, to `ciągznakow1 > ciągznakow2`.
Jeżeli $\text{ASCII}(\text{znak1}) = \text{ASCII}(\text{znak2})$, to odpowiednio w ciągach znaków `ciągznakow1` oraz `ciągznakow2` przechodzi się do następnego znaku.
4. Jeżeli możliwe jest osiągnięcie końców ciągów znaków `ciągznakow1` oraz `ciągznakow2`, wtedy te ciągi są sobie równe.

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

```
'abc' = 'abc'
'abc  ' = 'abc'  -- Należy zwrócić uwagę na końcowe znaki odstepu
                  -- w pierwszym ciągu znaków.
'ab' < 'abc'
'abcd' > 'abcc'
```

Algorytm *porównania bez dopełnienia odstepu* jest nieco inny.

1. Należy porównać każdy ciąg znaków, znak po znaku, zaczynając od lewej strony. Przykładowo, w ciągu znaków `ciągznakow1` znakiem jest `znak1`, a w ciągu znaków `ciągznakow2` znakiem jest `znak2`.
2. Jeżeli $\text{ASCII}(\text{znak1}) < \text{ASCII}(\text{znak2})$, to `ciągznakow1 < ciągznakow2`.
Jeżeli $\text{ASCII}(\text{znak1}) > \text{ASCII}(\text{znak2})$, to `ciągznakow1 > ciągznakow2`.
Jeżeli $\text{ASCII}(\text{znak1}) = \text{ASCII}(\text{znak2})$, to należy przejść do następnego znaku odpowiednio w ciągach znaków `ciągznakow1` oraz `ciągznakow2`.
3. Jeżeli ciąg znaków `ciągznakow1` kończy się przed ciągiem znaków `ciągznakow2`, wtedy `ciągznakow1 < ciągznakow2`. Jeżeli ciąg znaków `ciągznakow2` kończy się przed ciągiem znaków `ciągznakow1`, wtedy `ciągznakow1 > ciągznakow2`.

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

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

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

```
'abc  ' = 'abc'  -- Należy zwrócić uwagę na końcowe znaki odstepu
                  -- w pierwszym ciągu znaków.
```

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 wydziału 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 pożądaną 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 być 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 odwołanie;
```

W celu utworzenia synonimu należy za pomocą powyższego polecenia wstawić w miejsce nazwa_synonimu nazwę synonimu, a w miejsce odwołanie 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 *uzytkownik* *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 *uzytkownik* *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, że 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.incxp, sys.incvd 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 między dwoma kontami bankowymi, identyfikowanymi jako konto_nadawcy i konto_odbiornicy.

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

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_odbiornicy 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.

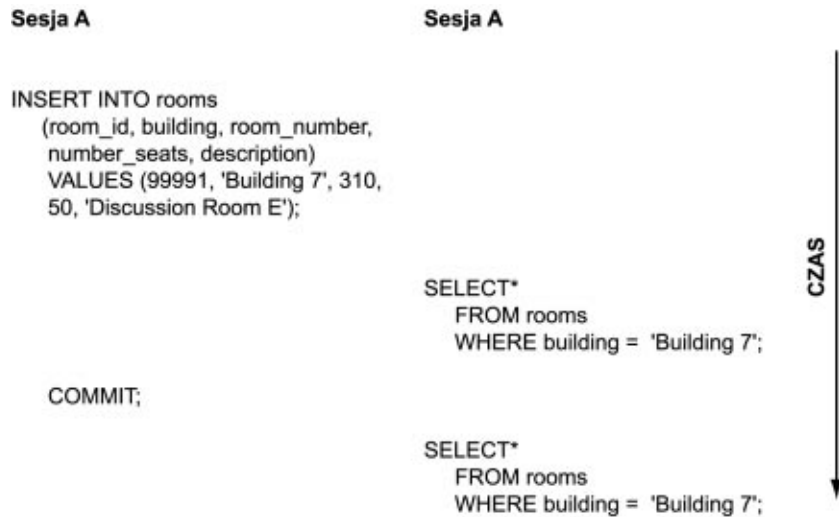
Poniżej przedstawiono składnię instrukcji COMMIT:

```
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 zawieszona.

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 brakujące*/
  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, że 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.