

» Idź do

- Spis treści
- Przykładowy rozdział

» 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

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2008

Oracle Database 11g i SQL. Programowanie

Autor: Jason Price
Tłumaczenie: Marcin Rogóż
ISBN: 978-83-246-1879-8
Tytuł oryginału: [Oracle Database 11g SQL](#)
(Osborne Oracle Press)
Format: B5, stron: 672



Opanuj SQL i PL/SQL w Oracle Database i pisz świetne programy!

- Jak tworzyć obiekty baz danych i kolekcje?
- Jak zoptymalizować instrukcje SQL, aby były wykonywane szybciej?
- Jak pisać programy w PL/SQL?

Doskonała baza danych to jeden z podstawowych elementów sprawnego funkcjonowania współczesnych przedsiębiorstw, instytucji i organizacji. Jednak, aby efektywnie korzystać z jej dobrodziejstw, potrzebujesz specjalnego oprogramowania. Znakomitym systemem zarządzania bazą danych jest Oracle. Natomiast SQL – strukturalny język zapytań – zapewnia dostęp do systemu zarządzania bazą danych, a więc pobieranie, wstawianie i usuwanie z niej wszelkich informacji. PL/SQL (wywodzący się z SQL) umożliwia pisanie programów zawierających instrukcje SQL.

Książka „Oracle Database 11g. Programowanie w języku SQL” zawiera wyczerpujące informacje, dotyczące pracy z bazą danych Oracle za pośrednictwem instrukcji SQL, a także opis najnowszych właściwości i narzędzi tego języka, technik optymalizacyjnych oraz obsługi Javy i XML. Z tego podręcznika dowiesz się między innymi, w jaki sposób Oracle przetwarza oraz przechowuje daty i czas. Nauczysz się wykorzystywać duże obiekty do obsługi plików multimedialnych zawierających obrazy, muzykę i filmy, a także pisać (w języku Java) programy uzyskujące dostęp do bazy danych Oracle za pośrednictwem JDBC.

- Pobieranie informacji z tabel bazy danych
- SQL*Plus
- Funkcje
- Składowanie oraz przetwarzanie dat i czasu
- Zapytania zaawansowane
- Użytkownicy, uprawnienia i role
- Obiekty baz danych
- Kolekcje
- Praca z SQL w Javie
- Zamknięcie obiektu ResultSet
- Optymalizacja SQL
- XML i bazy danych Oracle

Baza Oracle nie będzie miała przed Tobą tajemnic!

Spis treści

O autorze	17
O redaktorze merytorycznym	19
Wprowadzenie	21
Rozdział 1. Wprowadzenie	27
Czym jest relacyjna baza danych?	27
Wstęp do SQL	28
Używanie SQL*Plus	30
Uruchamianie SQL*Plus	30
Uruchamianie SQL*Plus z wiersza poleceń	31
Wykonywanie instrukcji SELECT za pomocą SQL*Plus	32
SQL Developer	33
Tworzenie schematu bazy danych sklepu	34
Uruchamianie skryptu programu SQL*Plus w celu utworzenia schematu bazy danych sklepu	35
Instrukcje DDL używane do tworzenia schematu bazy danych sklepu	36
Dodawanie, modyfikowanie i usuwanie wierszy	44
Dodawanie wiersza do tabeli	44
Modyfikowanie istniejącego wiersza w tabeli	46
Usuwanie wiersza z tabeli	47
Typy BINARY_FLOAT i BINARY_DOUBLE	47
Zalety typów BINARY_FLOAT i BINARY_DOUBLE	47
Użycie typów BINARY_FLOAT i BINARY_DOUBLE w tabeli	48
Wartości specjalne	49
Kończenie pracy SQL*Plus	49
Wprowadzenie do Oracle PL/SQL	50
Podsumowanie	51
Rozdział 2. Pobieranie informacji z tabel bazy danych	53
Wykonywanie instrukcji SELECT dla jednej tabeli	53
Pobieranie wszystkich kolumn z tabeli	54
Wykorzystanie klauzuli WHERE do wskazywania wierszy do pobrania	55
Identyfikatory wierszy	55
Numery wierszy	56
Wykonywanie działań arytmetycznych	56
Wykonywanie obliczeń na datach	57
Korzystanie z kolumn w obliczeniach	58

Używanie aliasów kolumn	59
Łączenie wartości z kolumn za pomocą konkatenacji	60
Wartości null	61
Wyświetlanie odrębnych wierszy	62
Porównywanie wartości	63
Korzystanie z operatorów SQL	65
Operator LIKE	65
Operator IN	67
Operator BETWEEN	67
Operatory logiczne	68
Następstwo operatorów	69
Sortowanie wierszy za pomocą klauzuli ORDER BY	70
Instrukcje SELECT wykorzystujące dwie tabele	71
Używanie aliasów tabel	73
Iloczyny kartezjańskie	74
Instrukcje SELECT wykorzystujące więcej niż dwie tabele	74
Warunki złączenia i typy złączeń	76
Nierównozłączenia	76
Złączenia rozszerzone	77
Złączenia własne	81
Wykonywanie złączeń za pomocą składni SQL/92	82
Wykonywanie złączeń wewnętrznych dwóch tabel z wykorzystaniem składni SQL/92	82
Upraszczenie złączeń za pomocą słowa kluczowego USING	83
Wykonywanie złączeń wewnętrznych obejmujących więcej niż dwie tabele (SQL/92)	84
Wykonywanie złączeń wewnętrznych z użyciem wielu kolumn (SQL/92)	84
Wykonywanie złączeń rozszerzonych z użyciem składni SQL/92	85
Wykonywanie złączeń własnych z użyciem składni SQL/92	86
Wykonywanie złączeń krzyżowych z użyciem składni SQL/92	87
Podsumowanie	87
Rozdział 3. SQL*Plus	89
Przeglądanie struktury tabeli	89
Edycja instrukcji SQL	90
Zapisywanie, odczytywanie i uruchamianie plików	92
Formatowanie kolumn	95
Ustawianie rozmiaru strony	97
Ustawianie rozmiaru wiersza	97
Czyszczenie formatowania kolumny	98
Używanie zmiennych	98
Zmienne tymczasowe	99
Zmienne zdefiniowane	101
Tworzenie prostych raportów	104
Używanie zmiennych tymczasowych w skrypcie	104
Używanie zmiennych zdefiniowanych w skrypcie	105
Przesyłanie wartości do zmiennej w skrypcie	105
Dodawanie nagłówka i stopki	106
Obliczanie sum pośrednich	108
Uzyskiwanie pomocy od SQL*Plus	109
Automatyczne generowanie instrukcji SQL	110
Kończenie połączenia z bazą danych i pracy SQL*Plus	111
Podsumowanie	111

Rozdział 4. Proste funkcje	113
Funkcje jednowierszowe	113
Funkcje znakowe	114
Funkcje numeryczne	121
Funkcje konwertujące	125
Funkcje wyrażeń regularnych	131
Funkcje agregujące	138
AVG()	138
COUNT()	139
MAX() i MIN()	140
STDDEV()	140
SUM()	141
VARIANCE()	141
Grupowanie wierszy	141
Grupowanie wierszy za pomocą klauzuli GROUP BY	142
Nieprawidłowe użycie funkcji agregujących	145
Filtrowanie grup wierszy za pomocą klauzuli HAVING	146
Jednoczesne używanie klauzul WHERE i GROUP BY	147
Jednoczesne używanie klauzul WHERE, GROUP BY i HAVING	147
Podsumowanie	148
Rozdział 5. Składowanie oraz przetwarzanie dat i czasu	149
Proste przykłady składowania i pobierania dat	149
Konwertowanie typów DataGodzina za pomocą funkcji TO_CHAR() i TO_DATE()	151
Konwersja daty i czasu na napis za pomocą funkcji TO_CHAR()	151
Konwersja napisu na wyrażenie DataGodzina za pomocą funkcji TO_DATE()	155
Ustawianie domyślnego formatu daty	158
Jak Oracle interpretuje lata dwucyfrowe?	159
Użycie formatu YY	159
Użycie formatu RR	160
Funkcje operujące na datach i godzinach	161
ADD_MONTHS()	161
LAST_DAY()	163
MONTHS_BETWEEN()	163
NEXT_DAY()	163
ROUND()	164
SYSDATE	164
TRUNC()	165
Strefy czasowe	165
Funkcje operujące na strefach czasowych	166
Strefa czasowa bazy danych i strefa czasowa sesji	167
Uzyskiwanie przesunięć strefy czasowej	168
Uzyskiwanie nazw stref czasowych	168
Konwertowanie wyrażenia DataGodzina z jednej strefy czasowej na inną	169
Datowniki (znaczniki czasu)	169
Typy datowników	169
Funkcje operujące na znacznikach czasu	173
Interwały czasowe	178
Typ INTERVAL YEAR TO MONTH	179
Typ INTERVAL DAY TO SECOND	181
Funkcje operujące na interwałach	183
Podsumowanie	184

Rozdział 6. Podzapytania	187
Rodzaje podzapytań	187
Pisanie podzapytań jednowierszowych	188
Podzapytania w klauzuli WHERE	188
Użycie innych operatorów jednowierszowych	189
Podzapytania w klauzuli HAVING	189
Podzapytania w klauzuli FROM (widoki wbudowane)	191
Błędy, które można napotkać	191
Pisanie podzapytań wielowierszowych	192
Użycie operatora IN z podzapytaniem wielowierszowym	193
Użycie operatora ANY z podzapytaniem wielowierszowym	194
Użycie operatora ALL z podzapytaniem wielowierszowym	194
Pisanie podzapytań wielokolumnowych	195
Pisanie podzapytań skorelowanych	195
Przykład podzapytania skorelowanego	195
Użycie operatorów EXISTS i NOT EXISTS z podzapytaniem skorelowanym	196
Pisanie zagnieźdzonych podzapytań	199
Pisanie instrukcji UPDATE i DELETE zawierających podzapytania	200
Pisanie instrukcji UPDATE zawierającej podzapytanie	200
Pisanie instrukcji DELETE zawierającej podzapytanie	201
Podsumowanie	201
Rozdział 7. Zapytania zaawansowane	203
Operatory zestawu	203
Przykładowe tabele	204
Operator UNION ALL	205
Operator UNION	206
Operator INTERSECT	207
Operator MINUS	207
Łączenie operatorów zestawu	207
Użycie funkcji TRANSLATE()	209
Użycie funkcji DECODE()	210
Użycie wyrażenia CASE	212
Proste wyrażenia CASE	212
Przeszukiwane wyrażenia CASE	213
Zapytania hierarchiczne	215
Przykładowe dane	215
Zastosowanie klauzul CONNECT BY i START WITH	216
Użycie pseudokolumny LEVEL	217
Formatowanie wyników zapytania hierarchicznego	218
Rozpoczynanie od węzła innego niż główny	219
Użycie podzapytania w klauzuli START WITH	219
Poruszanie się po drzewie w górę	220
Eliminowanie węzłów i gałęzi z zapytania hierarchicznego	220
Umieszczanie innych warunków w zapytaniu hierarchicznym	221
Rozszerzone klauzule GROUP BY	222
Przykładowe tabele	222
Użycie klauzuli ROLLUP	224
Klauzula CUBE	226
Funkcja GROUPING()	227
Klauzula GROUPING SETS	230
Użycie funkcji GROUPING_ID()	231
Kilkukrotne użycie kolumny w klauzuli GROUP BY	233
Użycie funkcji GROUP_ID()	233

Funkcje analityczne	235
Przykładowa tabela	235
Użycie funkcji klasyfikujących	236
Użycie odwrotnych funkcji rankingowych	243
Użycie funkcji okna	243
Funkcje raportujące	249
Użycie funkcji LAG() i LEAD()	251
Użycie funkcji FIRST i LAST	252
Użycie funkcji regresji liniowej	252
Użycie funkcji hipotetycznego rankingu i rozkładu	253
Użycie klauzuli MODEL	254
Przykład zastosowania klauzuli MODEL	255
Dostęp do komórek za pomocą zapisu pozycyjnego i symbolicznego	256
Uzyskiwanie dostępu do zakresu komórek za pomocą BETWEEN i AND	257
Sięganie do wszystkich komórek za pomocą ANY i IS ANY	257
Pobieranie bieżącej wartości wymiaru za pomocą funkcji CURRENTV()	258
Uzyskiwanie dostępu do komórek za pomocą pętli FOR	259
Obsługa wartości NULL i brakujących	260
Modyfikowanie istniejących komórek	262
Użycie klauzul PIVOT i UNPIVOT	263
Prosty przykład klauzuli PIVOT	263
Przestawianie w oparciu o wiele kolumn	265
Użycie kilku funkcji agregujących w przestawieniu	266
Użycie klauzuli UNPIVOT	267
Podsumowanie	268
Rozdział 8. Zmienianie zawartości tabeli	269
Wstawianie wierszy za pomocą instrukcji INSERT	269
Pomijanie listy kolumn	270
Określanie wartości NULL dla kolumny	271
Umieszczanie pojedynczych i podwójnych cudzysłowów w wartościach kolumn	271
Kopiowanie wierszy z jednej tabeli do innej	271
Modyfikowanie wierszy za pomocą instrukcji UPDATE	272
Klauzula RETURNING	273
Usuwanie wierszy za pomocą instrukcji DELETE	274
Integralność bazy danych	274
Wymuszanie więzów klucza głównego	274
Wymuszanie więzów kluczy obcych	275
Użycie wartości domyślnych	276
Scalanie wierszy za pomocą instrukcji MERGE	277
Transakcje bazodanowe	279
Zatwierdzanie i wycofywanie transakcji	280
Rozpoczynanie i kończenie transakcji	281
Punkty zachowania	281
ACID — właściwości transakcji	283
Transakcje współbieżne	283
Blokowanie transakcji	284
Poziomy izolacji transakcji	285
Przykład transakcji SERIALIZABLE	286
Zapytania retrospektywne	287
Przyznawanie uprawnień do używania zapytań retrospektywnych	288
Zapytania retrospektywne w oparciu o czas	288
Zapytania retrospektywne z użyciem SCN	290
Podsumowanie	291

Rozdział 9. Użytkownicy, uprawnienia i role	293
Użytkownicy	293
Tworzenie konta użytkownika	294
Zmianie hasła użytkownika	295
Usuwanie konta użytkownika	295
Uprawnienia systemowe	296
Przyznawanie uprawnień systemowych użytkownikowi	296
Sprawdzanie uprawnień systemowych przyznanych użytkownikowi	297
Zastosowanie uprawnień systemowych	298
Odbieranie uprawnień systemowych	298
Uprawnienia obiektowe	299
Przyznawanie użytkownikowi uprawnień obiektowych	299
Sprawdzanie przekazanych uprawnień	300
Sprawdzanie otrzymanych uprawnień obiektowych	301
Zastosowanie uprawnień obiektowych	303
Synonimy	303
Synonimy publiczne	304
Odbieranie uprawnień obiektowych	305
Role	305
Tworzenie ról	306
Przyznawanie uprawnień roli	306
Przyznawanie roli użytkownikowi	307
Sprawdzanie ról przyznanych użytkownikowi	307
Sprawdzanie uprawnień systemowych przyznanych roli	308
Sprawdzanie uprawnień obiektowych przyznanych roli	308
Zastosowanie uprawnień przyznanych roli	310
Role domyślne	310
Odbieranie roli	311
Odbieranie uprawnień roli	311
Usuwanie roli	311
Obserwacja	311
Uprawnienia wymagane do przeprowadzania obserwacji	312
Przykłady obserwacji	312
Perspektywy zapisu obserwacji	314
Podsumowanie	314
Rozdział 10. Tworzenie tabel, sekwencji, indeksów i perspektyw	315
Tabele	315
Tworzenie tabeli	315
Pobieranie informacji o tabelach	317
Uzyskiwanie informacji o kolumnach w tabeli	318
Zmianie tabeli	319
Zmianie nazwy tabeli	328
Dodawanie komentarza do tabeli	328
Obcinanie tabeli	329
Usuwanie tabeli	329
Sekwencje	329
Tworzenie sekwencji	329
Pobieranie informacji o sekwencjach	331
Używanie sekwencji	332
Wypełnianie klucza głównego z użyciem sekwencji	334
Modyfikowanie sekwencji	334
Usuwanie sekwencji	335

Indeksy	335
Tworzenie indeksu typu B-drzewo	336
Tworzenie indeksów opartych na funkcjach	337
Pobieranie informacji o indeksach	338
Pobieranie informacji o indeksach kolumny	338
Modyfikowanie indeksu	339
Usuwanie indeksu	339
Tworzenie indeksu bitmapowego	339
Perspektywy	340
Tworzenie i używanie perspektyw	341
Modyfikowanie perspektywy	348
Usuwanie perspektywy	349
Archiwa migawek	349
Podsumowanie	352
Rozdział 11. Wprowadzenie do programowania w PL/SQL	353
Bloki	354
Zmienne i typy	355
Logika warunkowa	356
Pętle	356
Proste pętle	357
Pętle WHILE	358
Pętle FOR	358
Kursory	359
Krok 1. — deklarowanie zmiennych przechowujących wartości kolumn	359
Krok 2. — deklaracja kursora	360
Krok 3. — otwarcie kursora	360
Krok 4. — pobieranie wierszy z kursora	360
Krok 5. — zamknięcie kursora	361
Pełny przykład — product_cursor.sql	361
Kursory i pętle FOR	363
Instrukcja OPEN-FOR	363
Kursory bez ograniczenia	365
Wyjątki	367
Wyjątek ZERO_DIVIDE	368
Wyjątek DUP_VAL_ON_INDEX	369
Wyjątek INVALID_NUMBER	370
Wyjątek OTHERS	370
Procedury	371
Tworzenie procedury	371
Wywoływanie procedury	373
Uzyskiwanie informacji o procedurach	374
Usuwanie procedury	375
Przeglądanie błędów w procedurze	375
Funkcje	376
Tworzenie funkcji	376
Wywoływanie funkcji	377
Uzyskiwanie informacji o funkcjach	378
Usuwanie funkcji	378
Pakiety	378
Tworzenie specyfikacji pakietu	379
Tworzenie treści pakietu	379
Wywoływanie funkcji i procedur z pakietu	381
Uzyskiwanie informacji o funkcjach i procedurach w pakiecie	381
Usuwanie pakietu	382

Wyzwalacze	382
Kiedy uruchamiany jest wyzwalacz	382
Przygotowania do przykładu wyzwalacza	382
Tworzenie wyzwalacza	383
Uruchamianie wyzwalacza	385
Uzyskiwanie informacji o wyzwalaczach	386
Włączanie i wyłączanie wyzwalacza	387
Usuwanie wyzwalacza	387
Rozszerzenia PL/SQL wprowadzone w Oracle Database 11g	388
Typ SIMPLE_INTEGER	388
Sekwencje w PL/SQL	389
Generowanie natywnego kodu maszynowego z PL/SQL	390
Podsumowanie	390
Rozdział 12. Obiekty bazy danych	393
Wprowadzenie do obiektów	393
Tworzenie typów obiektowych	394
Uzyskiwanie informacji o typach obiektowych za pomocą DESCRIBE	395
Użycie typów obiektowych w tabelach bazy danych	397
Obiekty kolumnowe	397
Tabele obiektowe	399
Identyfikatory obiektów i odwołania obiektowe	403
Porównywanie wartości obiektów	405
Użycie obiektów w PL/SQL	407
Funkcja get_products()	408
Procedura display_product()	409
Procedura insert_product()	410
Procedura update_product_price()	410
Funkcja get_product()	411
Procedura update_product()	412
Funkcja get_product_ref()	412
Procedura delete_product()	413
Procedura product_lifecycle()	413
Procedura product_lifecycle2()	414
Dziedziczenie typów	416
Użycie podtypu zamiast typu nadrzędnego	418
Przykłady SQL	418
Przykłady PL/SQL	419
Obiekty NOT SUBSTITUTABLE	420
Inne przydatne funkcje obiektów	421
Funkcja IS OF()	421
Funkcja TREAT()	424
Funkcja SYS_TYPEID()	427
Typy obiektowe NOT INSTANTIABLE	428
Konstruktory definiowane przez użytkownika	430
Przesłanianie metod	433
Uogólnione wywoływanie	435
Podsumowanie	437
Rozdział 13. Kolekcje	439
Podstawowe informacje o kolekcjach	439
Tworzenie kolekcji	440
Tworzenie typu VARRAY	440
Tworzenie tabeli zagnieżdżonej	441

Użycie kolekcji do definiowania kolumny w tabeli	441
Użycie typu VARRAY do zdefiniowania kolumny w tabeli	441
Użycie typu tabeli zagnieżdżonej do zdefiniowania kolumny w tabeli	442
Uzyskiwanie informacji o kolekcjach	442
Uzyskiwanie informacji o tablicy VARRAY	442
Uzyskiwanie informacji o tabeli zagnieżdżonej	443
Umieszczanie elementów w kolekcji	445
Umieszczanie elementów w tablicy VARRAY	445
Umieszczanie elementów w tabeli zagnieżdżonej	446
Pobieranie elementów z kolekcji	446
Pobieranie elementów z tablicy VARRAY	446
Pobieranie elementów z tabeli zagnieżdżonej	447
Użycie funkcji TABLE() do interpretacji kolekcji jako serii wierszy	448
Użycie funkcji TABLE() z typem VARRAY	448
Użycie funkcji TABLE() z tabelą zagnieżdżoną	449
Modyfikowanie elementów kolekcji	450
Modyfikowanie elementów tablicy VARRAY	450
Modyfikowanie elementów tabeli zagnieżdżonej	450
Użycie metody mapującej do porównywania zawartości tabel zagnieżdżonych	451
Użycie funkcji CAST do konwersji kolekcji z jednego typu na inny	454
Użycie funkcji CAST() do konwersji tablicy VARRAY na tabelę zagnieżdżoną	454
Użycie funkcji CAST() do konwersji tabeli zagnieżdżonej na tablicę VARRAY	455
Użycie kolekcji w PL/SQL	455
Manipulowanie tablicą VARRAY	456
Manipulowanie tabelą zagnieżdżoną	457
Metody operujące na kolekcjach w PL/SQL	459
Kolekcje wielopoziomowe	469
Rozszerzenia kolekcji wprowadzone w Oracle Database 10g	472
Tablice asocjacyjne	472
Zmianianie rozmiaru typu elementu	473
Zwiększanie liczby elementów w tablicy VARRAY	474
Użycie tablic VARRAY w tabelach tymczasowych	474
Użycie innej przestrzeni tabel dla tabeli składującej tabelę zagnieżdżoną	474
Obsługa tabel zagnieżdżonych w ANSI	475
Podsumowanie	483
Rozdział 14. Duże obiekty	485
Podstawowe informacje o dużych obiektach (LOB)	485
Przykładowe pliki	486
Rodzaje dużych obiektów	486
Tworzenie tabel zawierających duże obiekty	487
Użycie dużych obiektów w SQL	488
Użycie obiektów CLOB i BLOB	488
Użycie obiektów BFILE	490
Użycie dużych obiektów w PL/SQL	492
APPEND()	494
CLOSE()	495
COMPARE()	495
COPY()	496
CREATETEMPORARY()	497
ERASE()	498
FILECLOSE()	499
FILECLOSEALL()	499
FILEEXISTS()	499

FILEGETNAME()	500
FILEISOPEN()	500
FILEOPEN()	501
FREETEMPORARY()	501
GETCHUNKSIZE()	502
GET_STORAGE_LIMIT()	502
GETLENGTH()	502
INSTR()	503
ISOPEN()	504
ISTEMPORARY()	505
LOADFROMFILE()	505
LOADBLOBFROMFILE()	506
LOADCLOBFROMFILE()	507
OPEN()	508
READ()	509
SUBSTR()	510
TRIM()	511
WRITE()	512
WRITEAPPEND()	512
Przykładowe procedury PL/SQL	513
Typy LONG i LONG RAW	529
Przykładowe tabele	530
Wstawianie danych do kolumn typu LONG i LONG RAW	530
Przekształcanie kolumn LONG i LONG RAW w duże obiekty	531
Nowe właściwości dużych obiektów w Oracle Database 10g	531
Niejawna konwersja między obiektami CLOB i NCLOB	532
Użycie atrybutu :new, gdy obiekt LOB jest używany w wyzwalaczu	533
Nowe właściwości dużych obiektów w Oracle Database 11g	533
Szyfrowanie danych LOB	534
Kompresja danych LOB	537
Usuwanie powtarzających się danych LOB	538
Podsumowanie	538
Rozdział 15. Praca z SQL w Javie	541
Zaczynamy	541
Konfigurowanie komputera	542
Ustawianie zmiennej środowiska ORACLE_HOME	542
Ustawianie zmiennej środowiska JAVA_HOME	543
Ustawianie zmiennej środowiska PATH	543
Ustawianie zmiennej środowiska CLASSPATH	544
Ustawianie zmiennej środowiska LD_LIBRARY_PATH	544
Sterowniki Oracle JDBC	545
Sterownik Thin	545
Sterownik OCI	545
Sterownik wewnętrzny po stronie serwera	546
Sterownik Thin po stronie serwera	546
Importowanie pakietów JDBC	546
Rejestrowanie sterowników Oracle JDBC	547
Otwieranie połączenia z bazą danych	547
Połączenie z bazą danych za pomocą getConnection()	547
URL bazy danych	548
Połączenie z bazą danych za pomocą źródła danych Oracle	549
Tworzenie obiektu JDBC Statement	552

Pobieranie wierszy z bazy danych	553
Krok 1: Tworzenie obiektu ResultSet i umieszczanie w nim danych	553
Krok 2: Odczyt wartości kolumn z obiektu ResultSet	554
Krok 3: Zamknięcie obiektu ResultSet	556
Wstawianie wierszy do bazy danych	557
Modyfikowanie wierszy w bazie danych	558
Usuwanie wierszy z bazy danych	558
Obsługa liczb	559
Obsługa wartości NULL z bazy danych	560
Sterowanie transakcjami bazy danych	562
Wykonywanie instrukcji Data Definition Language	563
Obsługa wyjątków	563
Zamykanie obiektów JDBC	565
Przykładowy program: BasicExample1.java	566
Kompilacja BasicExample1	570
Uruchamianie programu BasicExample1	570
Przygotowane instrukcje SQL	572
Przykładowy program: BasicExample2.java	574
Rozszerzenia Oracle JDBC	576
Pakiet oracle.sql	577
Pakiet oracle.jdbc	580
Przykładowy program: BasicExample3.java	584
Podsumowanie	586
Rozdział 16. Optymalizacja SQL	587
Podstawowe informacje o optymalizacji SQL	587
Należy filtrować wiersze za pomocą klauzuli WHERE	587
Należy używać złączeń tabel zamiast wielu zapytań	588
Wykonując złączenia, należy używać w pełni kwalifikowanych odwołań do kolumn	589
Należy używać wyrażeń CASE zamiast wielu zapytań	590
Należy dodać indeksy do tabel	591
Należy stosować klauzulę WHERE zamiast HAVING	592
Należy używać UNION ALL zamiast UNION	593
Należy używać EXISTS zamiast IN	594
Należy używać EXISTS zamiast DISTINCT	595
Należy używać GROUPING SETS zamiast CUBE	596
Należy stosować zmienne dowiązane	596
Nieidentyczne instrukcje SQL	596
Identyczne instrukcje SQL korzystające ze zmiennych dowiązanych	597
Wypisywanie listy i wartości zmiennych dowiązanych	598
Użycie zmiennej dowiązanej do składowania wartości zwróconej przez funkcję PL/SQL	598
Użycie zmiennej dowiązanej do składowania wierszy z REF_CURSOR	598
Porównywanie kosztu wykonania zapytań	599
Przeglądanie planów wykonania	600
Porównywanie planów wykonania	605
Przesyłanie wskazówek do optymalizatora	606
Dodatkowe narzędzia optymalizujące	608
Oracle Enterprise Manager Diagnostics Pack	608
Automatic Database Diagnostic Monitor	608
Podsumowanie	609

Rozdział 17. XML i baza danych Oracle	611
Wprowadzenie do XML	611
Generowanie XML z danych relacyjnych	612
XMLELEMENT()	612
XMLATTRIBUTES()	615
XMLFOREST()	615
XMLAGG()	617
XMLCOLATVAL()	619
XMLCONCAT()	620
XMLPARSE()	620
XMLPI()	621
XMLCOMMENT()	621
XMLSEQUENCE()	622
XMLSERIALIZE()	623
Przykład zapisywania danych XML do pliku w PL/SQL	623
XMLQUERY()	625
Zapisywanie XML w bazie danych	629
Przykładowy plik XML	629
Tworzenie przykładowego schematu XML	630
Pobieranie informacji z przykładowego schematu XML	632
Aktualizowanie informacji w przykładowym schemacie XML	636
Podsumowanie	639
Dodatek A Typy danych Oracle	641
Typy w Oracle SQL	641
Typy w Oracle PL/SQL	643
 Skorowidz	 645

Rozdział 4.

Proste funkcje

W tym rozdziale poznasz kilka wbudowanych funkcji bazy danych Oracle. Funkcja przyjmuje zero lub więcej parametrów i zwraca parametr. W bazie danych Oracle występują dwa główne typy funkcji:

- ◆ **Funkcje jednowierszowe** operują na jednym wierszu i zwracają jeden wiersz wyników dla każdego wiersza na wejściu. Przykładem funkcji jednowierszowej jest `CONCAT(x, y)`, która dołącza y do x i zwraca powstały napis.
- ◆ **Funkcje agregujące** operują na kilku wierszach jednocześnie i zwracają jeden wiersz wyników. Przykładem funkcji agregującej jest `AVG(x)`, która zwraca średnią x , gdzie x może być kolumną lub dowolnym wyrażeniem.

Zacznę od omówienia funkcji jednowierszowych, a następnie przejdziemy do funkcji agregujących. W dalszej części książki zostaną przedstawione bardziej złożone funkcje.

Funkcje jednowierszowe

Funkcja jednowierszowa operuje na jednym wierszu i zwraca jeden wiersz wyników dla każdego wiersza na wejściu. Występuje pięć głównych typów funkcji jednowierszowych:

- ◆ **funkcje znakowe** — manipulują napisami,
- ◆ **funkcje numeryczne** — wykonują obliczenia,
- ◆ **funkcje konwertujące** — konwertują wartość z jednego typu na inny,
- ◆ **funkcje dat** — przetwarzają daty i czas,
- ◆ **funkcje wyrażeń regularnych** — wykorzystują wyrażenia regularne do wyszukiwania danych; zostały wprowadzone w Oracle Database 10g i rozwinięte w 11g.

Rozpoczniemy od omówienia funkcji znakowych, a następnie przejdziemy do numerycznych, konwertujących oraz wyrażeń regularnych. Funkcje dat zostaną opisane w następnym rozdziale.

Funkcje znakowe

Funkcje znakowe przyjmują wejście znakowe, które może pochodzić z kolumny tabeli lub z dowolnego wyrażenia. Dane wejściowe są przetwarzane i jest zwracany wynik. Przykładem funkcji znakowej jest UPPER(), która zwraca napis wejściowy po przekształceniu na wielkie litery. Innym przykładem jest NVL(), która konwertuje wartość NULL na inną. W tabeli 4.1, w której zostały opisane niektóre funkcje znakowe, oraz we wszystkich kolejnych definicjach składni x i y mogą reprezentować kolumny tabeli lub dowolne poprawne wyrażenie.

W kolejnych podrozdziałach zostaną dokładniej opisane funkcje wymienione w tabeli 4.1.

ASCII() i CHAR()

Funkcja ASCII(x) zwraca kod ASCII znaku x . Funkcja CHR(x) zwraca znak o kodzie ASCII x .

Poniższe zapytanie pobiera za pomocą funkcji ASCII() kody ASCII znaków a, A, z, Z, 0 i 9:

```
SELECT ASCII('a'), ASCII('A'), ASCII('z'), ASCII('Z'), ASCII(0), ASCII(9)
FROM dual;

ASCII('A') ASCII('a') ASCII('Z') ASCII('z')  ASCII(0)  ASCII(9)
-----
          97         65         122         90         48         57
```



Uwaga

W tym zapytaniu wykorzystano tabelę dual. Zawiera ona jeden wiersz, za pomocą którego możemy wykonywać zapytania niewykorzystujące żadnej konkretnej tabeli.

Poniższe zapytanie pobiera za pomocą funkcji CHR() znaki o kodach ASCII 97, 65, 122, 90, 48 i 57:

```
SELECT CHR(97), CHR(65), CHR(122), CHR(90), CHR(48), CHR(57)
FROM dual;

C C C C C C
- - - - -
a A z Z 0 9
```

Znaki zwrócone przez funkcję CHR() są tymi samymi, które były przesyłane do funkcji ASCII() w poprzednim zapytaniu. Funkcje CHR() i ASCII() mają zatem przeciwne działanie.

CONCAT()

Funkcja CONCAT(x , y) dołącza y do x i zwraca nowy napis.

Poniższe zapytanie dołącza za pomocą funkcji CONCAT() wartość z kolumny last_name do wartości z kolumny first_name:

```
SELECT CONCAT(first_name, last_name)
FROM customers;

CONCAT(FIRST_NAME, LA
-----
```

Tabela 4.1. Funkcje znakowe

Funkcja	Opis
ASCII(<i>x</i>)	Zwraca kod ASCII znaku <i>x</i>
CHR(<i>x</i>)	Zwraca znak kodzie ASCII <i>x</i>
CONCAT(<i>x</i> , <i>y</i>)	Dołącza <i>y</i> do <i>x</i> i zwraca powstały napis
INITCAP(<i>x</i>)	Przekształca pierwszą literę każdego słowa w <i>x</i> na wielką i zwraca nowy napis
INSTR(<i>x</i> , <i>szukany_napis</i> [, <i>start</i>] [, <i>wystąpienie</i>])	Wyszukuje w <i>x</i> napis <i>szukany_napis</i> i zwraca pozycję, w której on występuje. Można przesłać opcjonalny parametr <i>start</i> , określający pozycję, od której rozpocznie się wyszukiwanie. Ponadto można przesłać opcjonalny parametr <i>wystąpienie</i> , określający, które wystąpienie <i>szukany_napis</i> zostanie zwrócone
LENGTH(<i>x</i>)	Zwraca liczbę znaków w <i>x</i>
LOWER(<i>x</i>)	Przekształca litery w <i>x</i> na małe i zwraca nowy napis
LPAD(<i>x</i> , <i>szerokość</i> , [<i>napis_dopełnienia</i>])	Dopełnia <i>x</i> znakami spacji po lewej stronie, aby uzyskać całkowitą długość napisu równą <i>szerokość</i> . Można przesłać opcjonalny parametr <i>napis_dopełnienia</i> , określający napis, który będzie powtarzany po lewej stronie <i>x</i> w celu wypełnienia dopełnianego obszaru. Zwracany jest dopełniony napis
LTRIM(<i>x</i> [, <i>napis_przycinany</i>])	Usuwa znaki znajdujące się po lewej stronie <i>x</i> . Można przesłać opcjonalny parametr <i>napis_przycinany</i> , określający znaki, które zostaną usunięte. Jeżeli ten parametr nie zostanie przesłany, domyślnie usuwane będą znaki spacji
NANVL(<i>x</i> , <i>wartość</i>)	Zwraca <i>wartość</i> , jeżeli <i>x</i> jest wartością specjalną NAN (nieliczbą). (Ta funkcja została wprowadzona w Oracle Database 10g)
NVL(<i>x</i> , <i>wartość</i>)	Zwraca <i>wartość</i> , jeżeli <i>x</i> to NULL. W przeciwnym razie zwraca <i>x</i>
NVL2(<i>x</i> , <i>wartość1</i> , <i>wartość2</i>)	Zwraca <i>wartość1</i> , jeżeli <i>x</i> to nie NULL. W przeciwnym razie zwraca <i>wartość2</i>
REPLACE(<i>x</i> , <i>szukany_napis</i> , <i>napis_zastępujący</i>)	Wyszukuje w <i>x</i> napis <i>szukany_napis</i> i zastępuje go napisem <i>napis_zastępujący</i>
RPAD(<i>x</i> , <i>szerokość</i> [, <i>napis_dopełnienia</i>])	Działa tak samo jak LPAD(), ale <i>x</i> jest dopełniane po prawej stronie
RTRIM(<i>x</i> , [, <i>napis_przycinany</i>])	Działa tak samo jak LTRIM(), ale <i>x</i> jest przycinane z prawej strony
SOUNDEX(<i>x</i>)	Zwraca napis zawierający fonetyczną reprezentację <i>x</i> . To umożliwi porównywanie słów, które podobnie brzmią w języku angielskim, ale ich pisownia jest inna
SUBSTR(<i>x</i> , <i>start</i> [, <i>długość</i>])	Zwraca podnapis napisu <i>x</i> , rozpoczynający się w pozycji określonej przez <i>start</i> . Można przesłać opcjonalny parametr <i>długość</i> , określający długość podnapisu
TRIM([<i>usuwany_znak</i> FROM] <i>x</i>)	Usuwa znaki po obu stronach <i>x</i> . Można przesłać opcjonalny parametr <i>usuwany_znak</i> , określający znak do usunięcia. Jeżeli parametr ten nie zostanie przesłany, domyślnie zostaną usunięte znaki spacji
UPPER(<i>x</i>)	Zmienia litery w <i>x</i> na wielkie i zwraca nowy napis

JanNikieł
LidiaStal
StefanBraz
GrażynaCynk
JadwigaMosiądz



Uwaga

Działanie funkcji CONCAT() jest takie samo jak operatora ||, który został opisany w rozdziale 2.

INITCAP()

Funkcja INITCAP(*x*) zmienia pierwszą literę każdego słowa w *x* na wielką.

Poniższe zapytanie pobiera kolumny product_id i description z tabeli products, a następnie za pomocą funkcji INITCAP() zmienia pierwszą literę każdego słowa w description na wielką:

```
SELECT product_id, INITCAP(description)
FROM products
WHERE product_id < 4;
```

```
PRODUCT_ID INITCAP(DESCRIPTION)
```

```
-----
1 Opis Współczesnej Nauki
2 Wprowadzenie Do Chemii
3 Eksplozja Gwiazdy
```

INSTR()

Funkcja INSTR(*x*, *szukany_napis* [, *start*] [, *wystąpienie*]) wyszukuje w *x* napis *szukany_napis* i zwraca pozycję, na której się on znajduje. Można przesłać opcjonalny argument *start*, określający pozycję rozpoczęcia wyszukiwania. Można również przesłać opcjonalny parametr *wystąpienie*, określający, które wystąpienie napisu *szukany_napis* zostanie zwrócone.

Poniższe zapytanie pobiera pozycję, na której znajduje się napis współczesna w kolumnie name pierwszego produktu:

```
SELECT name, INSTR(name, 'współczesna')
FROM products
WHERE product_id = 1;
```

```
NAME                                INSTR(NAME, 'WSPÓŁCZESNA')
```

```
-----
```

```
Nauka współczesna                    7
```

Kolejne zapytanie wyświetla pozycję, na której znajduje się drugie wystąpienie znaku a, rozpoczynając od początku nazwy produktu:

```
SELECT name, INSTR(name, 'a', 1, 2)
FROM products
WHERE product_id = 1;
```

```
NAME                                INSTR(NAME, 'A', 1, 2)
```

```
-----
```

```
Nauka współczesna                    5
```

Drugie „e” w tytule Nauka współczesna znajduje się na piątej pozycji.

Funkcje znakowe mogą również operować na datach. Poniższe zapytanie pobiera pozycję, na której znajduje się napis STY w kolumnie dob klienta nr 1:

```
SELECT customer_id, dob, INSTR(dob, 'STY')
FROM customers
WHERE customer_id = 1;
```

```
CUSTOMER_ID DOB          INSTR(DOB,'STY')
-----
1 01-STY-65              4
```

LENGTH()

Funkcja `LENGTH(x)` zwraca liczbę znaków w `x`. Poniższe zapytanie za pomocą tej funkcji pobiera długość napisów w kolumnie `name` tabeli `products`:

```
SELECT name, LENGTH(name)
FROM products;
```

```
NAME                                LENGTH(NAME)
-----
Nauka współczesna                   17
Chemia                               6
Supernowa                            9
Wojny czołgów                       13
Z Files                              7
2412: Powrót                         12
Space Force 9                       13
Z innej planety                      15
Muzyka klasyczna                    16
Pop 3                                 5
Twórczy wrzask                      14
Pierwsza linia                      14
```

Kolejne zapytanie pobiera całkowitą liczbę znaków składających się na cenę produktu (kolumna `price`). Należy zwrócić uwagę, że separator dziesiętny (`.`) jest liczony jako znak w kolumnie `price`:

```
SELECT price, LENGTH(price)
FROM products
WHERE product_id < 3;
```

```
PRICE LENGTH(PRICE)
-----
19,95      5
30         2
```

LOWER() i UPPER()

Funkcja `LOWER(x)` zmienia litery w `x` na małe. Funkcja `UPPER(x)` zmienia natomiast litery w `x` na wielkie.

Poniższe zapytanie zmienia litery w napisach z kolumny `first_name` na wielkie, a litery z napisów z kolumny `last_name` na małe:

```
SELECT UPPER(first_name), LOWER(last_name)
FROM customers;
```

```
UPPER(FIRST_NAME) LOWER(LAST_NAME)
-----
JAN                nikiel
LIDIA              stał
STEFAN            brąz
GRAŻYNA           cynk
JADWIGA           mosiądz
```

LPAD() i RPAD()

Funkcja `LPAD(x, szerokość, [napis_dopełnienia])` dopełnia lewą stronę `x` znakami spacji, aby uzupełnić długość napisu `x` do `szerokość` znaków. Można przesłać opcjonalny parametr `napis_dopełnienia`, który określa napis powtarzany po lewej stronie napisu `x` w celu dopełnienia go. Zwracany jest dopełniony łańcuch. Funkcja `RPAD(x, szerokość, [napis_dopełnienia])` dopełnia prawą stronę napisu `x`.

Poniższe zapytanie pobiera kolumny `name` i `price` z tabeli `products`. Kolumna `name` jest dopełniana po prawej stronie za pomocą funkcji `RPAD()` do długości 30 znaków. Dopełnienie jest wypełniane kropkami. Kolumna `price` jest dopełniana po lewej stronie za pomocą funkcji `LPAD` do długości 8 znaków. Dopełnienie jest wypełniane napisem `*+*`:

```
SELECT RPAD(name, 30, '.') , LPAD(price, 8, '*+')
FROM products
WHERE product_id < 4;

RPAD(NAME,30, '.')                                LPAD(PRICE,8, '*+')
-----
Nauka współczesna.....                          **19,95
Chemia.....                                       ***+*+30
Supernowa.....                                   **25,99
```



Uwaga

Z tego przykładu wynika, że funkcje znakowe mogą operować na liczbach. Kolumna `price` zawiera liczbę, która została dopełniona po lewej stronie przez funkcję `LPAD()`.

LTRIM(), RTRIM() i TRIM()

Funkcja `LTRIM(x [, napis_przycinany])` służy do usuwania znaków z lewej strony `x`. Można przesłać opcjonalny parametr określający, które znaki mają zostać usunięte. Jeżeli parametr ten nie zostanie przesłany, będą domyślnie usuwane znaki spacji. Funkcja `RTRIM()` służy natomiast do usuwania znaków po prawej stronie `x`, `TRIM()` — do usuwania znaków z lewej i prawej strony `x`. Wszystkie trzy funkcje zostały wykorzystane w poniższym zapytaniu:

```
SELECT
  LTRIM(' Cześć Edwardzie Nencki!'),
  RTRIM('Cześć Ryszardzie Spacki!abcabc', 'abc'),
  TRIM('0' FROM '000Cześć Mario Tupska!0000')
FROM dual;

LTRIM('CZEŚĆEDWARDZIENENC RTRIM('CZEŚĆRYSZARDZIESPAC TRIM('0'FROM'000CZEŚĆ
-----
Cześć Edwardzie Nencki! Cześć Ryszardzie Spacki! Cześć Mario Tupska!
```

NVL()

Funkcja NVL() konwertuje wartość NULL na inną. NVL(*x*, *wartość*) zwraca *wartość*, jeżeli *x* wynosi NULL. W przeciwnym razie zwraca *x*.

Poniższe zapytanie pobiera kolumny customer_id i phone z tabeli customers. Wartości NULL w kolumnie phone są przekształcane za pomocą funkcji NVL() na Nieznany numer telefonu:

```
SELECT customer_id, NVL(phone, 'Nieznany numer telefonu')
FROM customers;
```

```
CUSTOMER_ID NVL(PHONE, 'NIEZNANYNUME
-----
1 800-555-1211
2 800-555-1212
3 800-555-1213
4 800-555-1214
5 Nieznany numer telefonu
```

Wartość z kolumny phone dla klienta nr 5 została przekształcona na Nieznany numer telefonu, ponieważ w tym wierszu kolumna phone ma wartość NULL.

NVL2()

Funkcja NVL2(*x*, *wartość1*, *wartość2*) zwraca *wartość1*, jeżeli *x* to nie NULL. W przeciwnym razie zwracana jest *wartość2*.

Poniższe zapytanie pobiera kolumny customer_id i phone z tabeli customers. Wartości inne niż NULL w kolumnie phone są konwertowane na napis Znany, a wartości NULL na napis Nieznany:

```
SELECT customer_id, NVL2(phone, 'Znany', 'Nieznany')
FROM customers;
```

```
CUSTOMER_ID NVL2(PHON
-----
1 Znany
2 Znany
3 Znany
4 Znany
5 Nieznany
```

Wartości kolumny phone zostały przekształcone na Znane w przypadku klientów od 1. do 4., ponieważ w tych wierszach wartości kolumny są różne od NULL. W przypadku klienta nr 5 wartość jest konwertowana na Nieznany, ponieważ w tym wierszu w kolumnie phone występuje wartość NULL.

REPLACE()

Funkcja REPLACE(*x*, *szukany_napis*, *napis_zastępujący*) wyszukuje w *x* napis *szukany_napis* i zastępuje go napisem *napis_zastępujący*.

Poniższy przykład pobiera z tabeli products kolumnę name dla produktu nr 1 (którego nazwa to Nauka współczesna) i zastępuje za pomocą funkcji REPLACE() napis Nauka łańcuchem Fizyka:

```
SELECT REPLACE(name, 'Nauka', 'Fizyka')
FROM products
WHERE product_id = 1;
```

```
REPLACE(NAME, 'NAUKA', 'FIZYKA')
```

```
-----
Fizyka współczesna
```



Uwaga

Funkcja REPLACE() nie modyfikuje zawartości wiersza w bazie danych, a jedynie wiersz zwracany przez funkcję.

SOUNDEX()

Funkcja SOUNDEX(*x*) zwraca napis zawierający fonetyczną reprezentację *x*. To umożliwia porównywanie słów, które brzmią podobnie w języku angielskim, lecz mają inną pisownię.

SUBSTR()

Funkcja SUBSTR(*x*, *start* [, *długość*]) zwraca podnapis napisu *x*, rozpoczynający się w pozycji określonej przez *start*. Można przesłać opcjonalny parametr *długość*, określający długość podnapisu.

Poniższe zapytanie wykorzystuje funkcję SUBSTR() do pobrania 7-znakowego podłańcucha rozpoczynającego się od pozycji 2. w kolumnie name tabeli products:

```
SELECT SUBSTR(name, 2, 7)
FROM products
WHERE product_id < 4;
```

```
SUBSTR(NAME,2,7)
```

```
-----
auka ws
hemia
upernow
```

Używanie wyrażeń z funkcjami

W funkcjach możemy wykorzystywać nie tylko kolumny. Można przesłać dowolne poprawne wyrażenie, które zwraca napis. Poniższe zapytanie wykorzystuje funkcję SUBSTR() do pobrania podnapisu małą z napisu Marysia miała małą owieczkę:

```
SELECT SUBSTR('Marysia miała małą owieczkę', 15, 4)
FROM dual;
```

```
SUBSTR
-----
małą
```

Łączenie funkcji

W instrukcji SQL można zastosować dowolną prawidłową kombinację funkcji. Poniższe zapytanie łączy funkcje UPPER() i SUBSTR(). Wyjście funkcji SUBSTR() jest przesyłane do funkcji UPPER():

```
SELECT name, UPPER(SUBSTR(name, 2, 8))
FROM products
WHERE product_id < 4;
```

NAME	UPPER(SUBSTR(NAME,2,8))
-----	-----
Nauka współczesna	AUKA WSP
Chemia	HEMIA
Supernowa	UPERNOWA



Uwaga

Możliwość łączenia funkcji nie jest ograniczona do funkcji znakowych — można łączyć z sobą funkcje różnego typu.

Funkcje numeryczne

Funkcje numeryczne służą do wykonywania obliczeń. Przyjmują one liczbę pochodzącą z kolumny lub dowolnego wyrażenia, którego wynikiem jest liczba. Następnie są wykonywane obliczenia i jest zwracana liczba. Przykładem funkcji numerycznej jest $\text{SQRT}(x)$, która zwraca pierwiastek kwadratowy x .

W tabeli 4.2 opisano niektóre funkcje numeryczne.

Tabela 4.2. *Funkcje numeryczne*

Funkcja	Opis	Przykłady
$\text{ABS}(x)$	Zwraca wartość absolutną x	$\text{ABS}(10) = 10$ $\text{ABS}(-10) = 10$
$\text{ACOS}(x)$	Zwraca arcus cosinus x	$\text{ACOS}(1) = 0$ $\text{ACOS}(-1) = 3,14159265$
$\text{ASIN}(x)$	Zwraca arcus sinus x	$\text{ASIN}(1) = 1,57079633$ $\text{ASIN}(-1) = -1,57079633$
$\text{ATAN}(x)$	Zwraca arcus tangens x	$\text{ATAN}(1) = 0,785398163$ $\text{ATAN}(-1) = -0,78539816$
$\text{ATAN2}(x, y)$	Zwraca arcus tangens x i y	$\text{ATAN2}(1, -1) = 2,35619449$
$\text{BITAND}(x, y)$	Zwraca wynik bitowego AND dla x i y	$\text{BITAND}(0, 0) = 0$ $\text{BITAND}(0, 1) = 0$ $\text{BITAND}(1, 0) = 0$ $\text{BITAND}(1, 1) = 1$ $\text{NITAND}(1010, 1100) = 64$
$\text{COS}(x)$	Zwraca cosinus x , gdzie x jest kątem wyrażonym w radianach	$\text{COS}(90 * 3,1415926) = 1$ $\text{COS}(45 * 3,1415926) = -1$
$\text{COSH}(x)$	Zwraca cosinus hiperboliczny x	$\text{COSH}(3,1415926) = 11,5919527$
$\text{CEIL}(x)$	Zwraca najmniejszą liczbę całkowitą większą lub równą x	$\text{CEIL}(5.8) = 6$ $\text{CEIL}(-5.2) = -5$
$\text{EXP}(x)$	Zwraca wynik podniesienia liczby e do potęgi x , gdzie e w przybliżeniu wynosi 2,71828183	$\text{EXP}(1) = 2,71828183$ $\text{EXP}(2) = 7,3890561$
$\text{FLOOR}(x)$	Zwraca największą liczbę całkowitą mniejszą lub równą x	$\text{FLOOR}(5.8) = 5$ $\text{FLOOR}(-5.2) = 6$

Tabela 4.2. Funkcje numeryczne — ciąg dalszy

Funkcja	Opis	Przykłady
LOG(x , y)	Zwraca logarytm o podstawie x liczby y	LOG(2, 4) = 2 LOG(2, 5) = 2,32192809
LN(x)	Zwraca logarytm naturalny liczby x	LN(2.71828183) = 1
MOD(x , y)	Zwraca resztę z dzielenia x przez y	MOD(8, 3) = 2 MOD(8, 4) = 0
POWER(x , y)	Zwraca wynik podniesienia liczby x do potęgi y	POWER(2, 1) = 2 POWER(2, 3) = 8
ROUND(x [, y])	Zwraca wynik zaokrąglenia liczby x do opcjonalnej liczby y miejsc po przecinku. Jeżeli y zostanie pominięta, x jest zaokrąglana do 0 miejsc po przecinku. Jeżeli y jest liczbą ujemną, x jest zaokrąglana po lewej stronie separatora dziesiętnego	ROUND(5.75) = 6 ROUND(5.75, 1) = 5,8 ROUND(5.75, -1) = 10
SIGN(x)	Zwraca -1, jeżeli x jest liczbą ujemną, 1, jeżeli jest liczbą dodatnią, lub 0, jeśli x to zero	SIGN(-5) = -1 SIGN(5) = 1 SIGN(0) = 0
SIN(x)	Zwraca sinus liczby x	SIN(0) = 0
SINH(x)	Zwraca sinus hiperboliczny liczby x	SINH(1) = 1,17520119
SQRT(x)	Zwraca pierwiastek kwadratowy liczby x	SQRT(25) = 5 SQRT(5) = 2,23606798
TAN(x)	Zwraca tangens liczby x	TAN(0) = 0
TANH(x)	Zwraca tangens hiperboliczny liczby x	TANH(1) = 0,761594156
TRUNC(x [, y])	Zwraca wynik obcięcia liczby x do opcjonalnych y miejsc dziesiętnych. Jeżeli y nie zostanie określona, x zostanie przycięta do zera miejsc dziesiętnych. Jeżeli y jest liczbą ujemną, x będzie przycinana po lewej stronie separatora dziesiętnego	TRUNC(5.75) = 5 TRUNC(5.75, 1) = 5,7 TRUNC(5.75, -1) = 0

Część z funkcji wymienionych w tabeli 4.2 zostanie opisana dokładniej w kolejnych podrozdziałach.

ABS()

Funkcja ABS(x) oblicza wartość absolutną liczby x . Wartość absolutna liczby jest tą samą liczbą, ale bez żadnego znaku (dodatniego lub ujemnego). Poniższe zapytanie pobiera wartości absolutne liczb 10 i -10:

```
SELECT ABS(10), ABS(-10)
FROM dual;
```

```
ABS(10)  ABS(-10)
-----  -----
10       10
```

Wartość absolutna liczby 10 wynosi 10, a wartość absolutna liczby -10 również wynosi 10.

Parametry przesyłane do funkcji numerycznych nie muszą być literałami liczbowymi. Dane wejściowe mogą również pochodzić z kolumny liczbowej w tabeli lub każdego poprawnego wyrażenia. Poniższe zapytanie pobiera wartości absolutne liczb obliczonych przez odjęcie 30 od wartości kolumny price tabeli products dla pierwszych trzech produktów:

```
SELECT product_id, price, price - 30, ABS(price - 30)
FROM products
WHERE product_id < 4;
```

PRODUCT_ID	PRICE	PRICE-30	ABS(PRICE-30)
1	19,95	-10,05	10,05
2	30	0	0
3	25,99	-4,01	4,01

CEIL()

Funkcja $CEIL(x)$ zwraca najmniejszą liczbę całkowitą równą x lub większą. Poniższe zapytanie oblicza za pomocą funkcji $CEIL()$ sufit (powagę) liczb 5,8 i $-5,2$:

```
SELECT CEIL(5.8), CEIL(-5.2)
FROM dual;
```

CEIL(5.8)	CEIL(-5.2)
6	-5

Sufit liczby 5,8 wynosi 6, ponieważ 6 jest najmniejszą liczbą całkowitą większą od 5,8. Sufit liczby $-5,2$ wynosi -5 , ponieważ $-5,2$ jest liczbą ujemną, a najmniejsza większa liczba całkowita od tej liczby to właśnie -5 .

FLOOR()

Funkcja $FLOOR(x)$ zwraca największą liczbę całkowitą równą x lub mniejszą. Poniższe zapytanie oblicza za pomocą funkcji $FLOOR()$ podłogę (część całkowitą) liczb 5,8 i $-5,2$:

```
SELECT FLOOR(5.8), FLOOR(-5.2)
FROM dual;
```

FLOOR(5.8)	FLOOR(-5.2)
5	-6

Część całkowita liczby 5,8 wynosi 5, ponieważ jest to największa liczba całkowita mniejsza od 5,8. Podłoga liczby $-5,2$ wynosi -6 , ponieważ $-5,2$ jest liczbą ujemną i największa liczba całkowita mniejsza od tej wartości to właśnie -6 .

MOD()

Funkcja $MOD(x, y)$ zwraca resztę z dzielenia liczby x przez y . Poniższe zapytanie oblicza za pomocą funkcji $MOD()$ reszty z dzielenia liczby 8 przez 3 i 4:

```
SELECT MOD(8, 3), MOD(8, 4)
FROM dual;
```



```

MOD(8,3)  MOD(8,4)
-----
      2      0

```

Reszta z dzielenia 8 przez 3 wynosi 2. Liczba 3 „mieści” się dwa razy w liczbie 8, pozostawiając 2 — resztę z dzielenia. Reszta z dzielenia 8 przez 4 wynosi 0. Liczba 4 „mieści” się dwa razy w liczbie 8 bez żadnej reszty.

POWER()

Funkcja `POWER(x, y)` zwraca wynik podniesienia liczby x do potęgi y . Poniższe zapytanie oblicza za pomocą funkcji `POWER()` wynik podniesienia liczby 2 do potęgi 1 i 3:

```

SELECT POWER(2, 1), POWER(2, 3)
FROM dual;

```

```

POWER(2,1) POWER(2,3)
-----
      2      8

```

Podniesienie 2 do potęgi 1 jest równoważne działaniu $2 \cdot 1$, więc w wyniku otrzymujemy 2. Podniesienie liczby 2 do potęgi 3 jest równoważne działaniu $2 \cdot 2 \cdot 2$, więc w wyniku otrzymujemy 8.

ROUND()

Funkcja `ROUND(x, [y])` zwraca wynik zaokrąglenia liczby x do opcjonalnych y miejsc po przecinku. Jeżeli y nie zostanie określone, x zostanie zaokrąglone do zera miejsc po przecinku. Jeżeli y jest liczbą ujemną, x będzie zaokrąglane po lewej stronie separatora dziesiętnego.

Poniższe zapytanie wykorzystuje funkcję `ROUND()` do zaokrąglenia liczby 5,75 do 0, 1 i -1 miejsc po przecinku:

```

SELECT ROUND(5.75), ROUND(5.75, 1), ROUND(5.75, -1)
FROM dual;

```

```

ROUND(5.75) ROUND(5.75,1) ROUND(5.75,-1)
-----
      6      5,8      10

```

Liczba 5,75 zaokrąglona do zera miejsc po przecinku wynosi 6; 5,75 po zaokrągleniu do jednego miejsca po przecinku wynosi 5,8; 5,75 zaokrąglona do jednego miejsca dziesiętnego po lewej stronie separatora dziesiętnego (na co wskazuje znak ujemny) wynosi 10.

SIGN()

Funkcja `SIGN(x)` zwraca znak liczby x . Jeżeli x jest liczbą ujemną, funkcja zwraca -1, jeżeli x jest dodatnia, funkcja zwraca 1. Jeżeli x wynosi 0, funkcja zwraca 0. Poniższe zapytanie pobiera znaki liczb 5, -5 i 0:

```

SELECT SIGN(5), SIGN(-5), SIGN(0)
FROM dual;

```

```

SIGN(5)  SIGN(-5)  SIGN(0)
-----
      1      -1      0

```

Znak -5 to -1 , znak 5 to 1 , znak 0 to 0 .

SQRT()

Funkcja $SQRT(x)$ zwraca pierwiastek kwadratowy liczby x . Poniższe zapytanie oblicza pierwiastki kwadratowe liczby 25 i 5 :

```

SELECT SQRT(25), SQRT(5)
FROM dual;

```

```

SQRT(25)  SQRT(5)
-----
      5  2,23606798

```

Pierwiastek kwadratowy z 25 wynosi 5 , a pierwiastek kwadratowy z 5 wynosi około $2,236$.

TRUNC()

Funkcja $TRUNC(x [, y])$ zwraca wynik obcięcia liczby x do opcjonalnych y miejsc dziesiętnych. Jeżeli y nie zostanie określony, x zostanie przycięta do zera miejsc dziesiętnych. Jeżeli y jest liczbą ujemną, x będzie przycinana po lewej stronie separatora dziesiętnego. Poniższe zapytanie przycina liczbę $5,75$ do 0 , 1 i -1 miejsca dziesiętnego:

```

SELECT TRUNC(5.75), TRUNC(5.75, 1), TRUNC(5.75, -1)
FROM dual;

```

```

TRUNC(5.75) TRUNC(5.75,1) TRUNC(5.75,-1)
-----
      5           5,7           0

```

W powyższym przykładzie $5,75$ po przycięciu do zera miejsc dziesiętnych wynosi 5 ; $5,75$ po przycięciu do jednego miejsca dziesiętnego po prawej stronie separatora dziesiętnego wynosi $5,7$; $5,75$ po przycięciu do jednego miejsca dziesiętnego po lewej stronie separatora dziesiętnego (na co wskazuje znak minus) wynosi 0 .

Funkcje konwertujące

Czasami chcemy przekonwertować wartość z jednego typu danych na inny. Możemy chcieć zmienić format ceny produktu, która jest przechowywana jako liczba (na przykład $10346,95$), na napis zawierający symbol waluty i separator tysięcy (na przykład $10\ 346$ zł). Do tego wykorzystujemy funkcje konwertujące, które konwertują wartość z jednego typu danych na inny.

W tabeli 4.3 opisano niektóre funkcje konwertujące.

Funkcje $TO_CHAR()$ i $TO_NUMBER()$ zostaną szczegółowo opisane w kolejnych podrozdziałach. Pozostałe funkcje z tabeli 4.3 zostaną omówione w dalszej części książki. Więcej informacji o zestawach znaków narodowych i systemie Unicode można znaleźć w *Oracle Database Globalization Support Guide* opublikowanym przez Oracle Corporation.

Tabela 4.3. Funkcje konwertujące

Funkcja	Opis
ASCIIISTR(<i>x</i>)	Konwertuje <i>x</i> na napis ASCII, gdzie <i>x</i> może być napisem w dowolnym zestawie znaków
BIN_TO_NUM(<i>x</i>)	Konwertuje liczbę binarną <i>x</i> na typ NUMBER
CAST(<i>x</i> AS <i>typ</i>)	Konwertuje <i>x</i> na kompatybilny typ z bazy danych, określony przez <i>typ</i>
CHARTORWIND(<i>x</i>)	Konwertuje <i>x</i> na ROWID
COMPOSE(<i>x</i>)	Konwertuje <i>x</i> na napis Unicode w jego w pełni znormalizowanej formie, w tym samym zestawie znaków co <i>x</i> . Unicode wykorzystuje 2-bajtowy zestaw znaków i może reprezentować ponad 65 000 znaków, nie tylko angielskich
CONVERT(<i>x</i> , <i>źródłowy_zestaw_znaków</i> , <i>docelowy_zestaw_znaków</i>)	Konwertuje <i>x</i> z zestawu znaków <i>źródłowy_zestaw_znaków</i> na <i>docelowy_zestaw_znaków</i>
DECODE(<i>x</i> , <i>wyszukiwane</i> , <i>wynik</i> , <i>domyślna</i>)	Porównuje <i>x</i> z wartością <i>search</i> . Jeżeli są równe, funkcja zwraca <i>wynik</i> ; w przeciwnym razie zwraca wartość <i>domyślna</i>
DECOMPOSE(<i>x</i>)	Konwertuje <i>x</i> na napis Unicode po dekompozycji napisu do tego samego zestawu znaków co <i>x</i>
HEXTORAW(<i>x</i>)	Konwertuje znak <i>x</i> zawierający szesnastkowe cyfry (o podstawie 16) na liczbę binarną (RAW). Funkcja zwraca liczbę RAW
NUMTODSINTERVAL(<i>x</i>)	Konwertuje liczbę <i>x</i> na INTERVAL DAY TO SECOND (funkcje związane z interwałami daty i czasu zostaną opisane w kolejnym rozdziale)
NUMTOYMINTERVAL(<i>x</i>)	Konwertuje liczbę <i>x</i> na INTERVAL YEAR TO MONTH
RAWTOHEX(<i>x</i>)	Konwertuje liczbę binarną (RAW) <i>x</i> na napis VARCHAR2, zawierający równoważną liczbę szesnastkową
RAWTONHEX(<i>x</i>)	Konwertuje liczbę binarną (RAW) <i>x</i> na napis NVARCHAR2, zawierający równoważną liczbę szesnastkową (NVARCHAR2 składa się z napisu, używając zestawu znaków narodowych)
ROWIDTOCHAR(<i>x</i>)	Konwertuje ROWID <i>x</i> na napis VARCHAR2
ROWIDTONCHAR(<i>x</i>)	Konwertuje ROWID <i>x</i> na napis NVARCHAR2
TO_BINARY_DOUBLE(<i>x</i>)	Konwertuje <i>x</i> na BINARY_DOUBLE (ta funkcja została wprowadzona w Oracle Database 10g)
TO_BINARY_FLOAT(<i>x</i>)	Konwertuje <i>x</i> na BINARY_FLOAT (ta funkcja została wprowadzona w Oracle Database 10g)
TO_BLOB	Konwertuje <i>x</i> na duży obiekt binarny (BLOB). Typ BLOB jest używany do składowania dużych ilości danych binarnych. Więcej informacji na temat dużych obiektów znajduje się w rozdziale 14.
TO_CHAR(<i>x</i> [, <i>format</i>])	Konwertuje <i>x</i> na napis VARCHAR2. Można przesłać opcjonalny parametr <i>format</i> , określający sposób formatowania <i>x</i>
TO_CLOB(<i>x</i>)	Konwertuje <i>x</i> na duży obiekt znakowy (CLOB). Typ CLOB jest używany do przechowywania dużych ilości danych znakowych
TO_DATE(<i>x</i> [, <i>format</i>])	Konwertuje <i>x</i> na typ DATE
TO_DSINTERVAL(<i>x</i>)	Konwertuje napis <i>x</i> na INTERVAL DAY TO SECOND
TO_MULTI_BYTE(<i>x</i>)	Konwertuje jednobajtowe znaki w <i>x</i> na odpowiadające im znaki wielobajtowe. Typ zwracany jest taki sam jak typ <i>x</i>

Tabela 4.3. Funkcje konwertujące — ciąg dalszy

Funkcja	Opis
TO_NCHAR(<i>x</i>)	Konwertuje <i>x</i> z zestawu znaków bazy danych na napis NVARCHAR2
TO_NCLOB(<i>x</i>)	Konwertuje <i>x</i> na duży obiekt NCLOB, używany do przechowywania sporych ilości danych znakowych ze znakami narodowymi
TO_NUMBER(<i>x</i> [, <i>format</i>])	Konwertuje <i>x</i> na typ NUMBER
TO_SINGLE_BYTE(<i>x</i>)	Konwertuje wielobajtowe znaki w <i>x</i> na odpowiadające im znaki jednobajtowe. Typ zwracany jest taki sam jak typ <i>x</i>
TO_TIMESTAMP(<i>x</i>)	Konwertuje napis <i>x</i> na typ TIMESTAMP
TO_TIMESTAMP_TZ(<i>x</i>)	Konwertuje napis <i>x</i> na typ TIMESTAMP WITH TIME ZONE
TO_YMINTERVAL(<i>x</i>)	Konwertuje napis <i>x</i> na typ INTERVAL YEAR TO MONTH
TRANSLATE(<i>x</i> , <i>napis_źródłowy</i> , <i>napis_docełowy</i>)	Konwertuje w <i>x</i> wszystkie wystąpienia <i>napis_źródłowy</i> na <i>napis_docełowy</i>
UNISTR(<i>x</i>)	Konwertuje znaki w <i>x</i> na znak NCHAR. NCHAR składa się z jednego znaku, używając zestawu znaków narodowych

TO_CHAR()

Funkcja TO_CHAR(*x* [, *format*]) konwertuje *x* na napis. Można przesłać opcjonalny parametr *format*, określający sposób formatowania *x*. Struktura parametru *format* zależy od tego, czy *x* jest liczbą, czy datą. Z tego podrozdziału dowiesz się, jak za pomocą funkcji TO_CHAR() konwertować liczby na napisy, a w kolejnym rozdziale opisano, jak konwertować daty na napisy.

Przyjrzyjmy się kilku prostym zapytaniom, konwertującym liczbę na napis za pomocą funkcji TO_CHAR(). Poniższe zapytanie konwertuje na napis liczbę 12345,67:

```
SELECT TO_CHAR(12345.67)
FROM dual;
```

```
TO_CHAR(
-----
12345,67
```

Kolejne zapytanie konwertuje liczbę 12345,67 na napis zgodnie z formatem określonym przez 99G999D99. Przy polskich ustawieniach narodowych zwracany jest łańcuch zawierający znak spacji jako separator tysięcy i przecinek jako separator dziesiętny:

```
SELECT TO_CHAR(12345.67, '99G999D99')
FROM dual;
```

```
TO_CHAR(12
-----
12 345,67
```

Opcjonalny napis *format*, który można przesłać do funkcji TO_CHAR(), posiada wiele parametrów mających wpływ na napis zwracany przez funkcję. Niektóre z tych parametrów zostały opisane w tabeli 4.4.

Tabela 4.4. Parametry formatujące liczby

Parametr	Przykład formatu	Opis
9	999	Zwraca cyfry na określonych pozycjach wraz z początkowym znakiem minus, jeżeli liczba jest ujemna
0	0999 9990	0999 zwraca liczbę poprzedzaną zerami 9990 zwraca liczbę kończoną zerami
.	999.99	Zwraca kropkę jako separator dziesiętny na określonej pozycji
,	999,99	Zwraca przecinek na określonej pozycji (w przypadku polskich ustawień narodowych w takim przypadku separatorem dziesiętnym musi być kropka)
\$	\$999	Poprzedza liczbę znakiem dolara
B	B9.99	Jeżeli całkowita część liczby stałoprzecinkowej jest zerem, zwraca znak spacji zamiast zera
C	999C	Zwraca symbol ISO waluty na określonej pozycji. Symbol pochodzi z parametru NLS_ISO_CURRENCY bazy danych i jest definiowany przez administratora bazy danych
D	9D99	Zwraca symbol separatora dziesiętnego na określonej pozycji. Symbol pochodzi z parametru NLS_NUMERIC_CHARACTER bazy danych (przy polskich ustawieniach narodowych jest to domyślnie przecinek)
EEEE	9.99EEEE	Zwraca liczbę, używając notacji naukowej
FM	FM90.9	Usuwa początkowe i końcowe spacje z liczby
G	9G999	Zwraca symbol separatora grupy na określonej pozycji. Symbol pochodzi z parametru NLS_NUMERIC_CHARACTER bazy danych
L	999L	Zwraca lokalny symbol waluty na określonej pozycji. Symbol pochodzi z parametru NLS_CURRENCY bazy danych
MI	999MI	Zwraca liczbę ujemną ze znakiem minus umieszczonym na końcu. Na końcu liczby dodatniej jest umieszczana spacja
PR	999PR	Zwraca liczbę ujemną w nawiasach ostrokatnych (< >) oraz liczbę dodatnią poprzedzoną i zakończoną znakiem spacji
RN	RN	Zwraca liczbę w zapisie rzymskim. RN zwraca numerały zapisywane wielkimi literami, a rn zwraca numerały zapisywane małymi literami. Liczba musi być liczbą całkowitą z przedziału od 1 do 3999
S	S999 999S	S999 zwraca liczbę ujemną poprzedzoną znakiem minus, a liczbę dodatnią poprzedzoną znakiem plus 999S zwraca liczbę ujemną zakończoną znakiem minus, a liczbę dodatnią zakończoną znakiem plus
TM	TM	Zwraca liczbę z użyciem jak najmniejszej liczby znaków. Domyślnie obowiązuje format TM9, który zwraca liczby, używając zapisu stałoprzecinkowego, chyba że liczba znaków jest większa od 64. W takim przypadku liczba jest zwracana w notacji naukowej
U	U999	Zwraca drugi symbol waluty (na przykład euro) na określonej pozycji. Symbol pochodzi z parametru NLS_DUAL_CURRENCY bazy danych
V	99V99	Zwraca liczbę pomnożoną razy 10^x , gdzie x jest liczbą znaków 9 za znakiem V. Jeżeli jest to konieczne, liczba jest zaokrąglana
X	XXXX	Zwraca liczbę w formacie szesnastkowym. Jeżeli nie jest ona całkowita, jest zaokrąglana do liczby całkowitej

Przyjrzyjmy się kolejnym przykładom konwertowania liczb na napisy za pomocą funkcji `TO_CHAR()`. Tabela 4.5 przedstawia przykłady wywołań funkcji `TO_CHAR()` oraz zwrócone wyniki.

Tabela 4.5. Przykłady zastosowania funkcji `TO_CHAR`

Wywołanie funkcji <code>TO_CHAR()</code>	Wynik
<code>TO_CHAR(12345.67, '99999.99')</code>	12345.67
<code>TO_CHAR(12345.67, '99,999.99')</code>	12,345.67
<code>TO_CHAR(-12345.67, '99,999.99')</code>	-12,345.67
<code>TO_CHAR(12345.67, '099,999.99')</code>	012,345.67
<code>TO_CHAR(12345.67, '99,999.9900')</code>	12,345.6700
<code>TO_CHAR(12345.67, '\$99,999.99')</code>	\$12,345.67
<code>TO_CHAR(0.67, 'B9.99')</code>	.67
<code>TO_CHAR(12345.67, 'C99,999.99')</code>	PLN12345.67
<code>TO_CHAR(12345.67, '99999D99')</code>	12345,67
<code>TO_CHAR(12345.67, '99999.99EEEE')</code>	1.23E+04
<code>TO_CHAR(0012345.6700, 'FM99999.99')</code>	12345.67
<code>TO_CHAR(12345.67, '99999G99')</code>	123 46
<code>TO_CHAR(12345.67, 'L99,999.99')</code>	zł12345.67
<code>TO_CHAR(-12345.67, '99,999.99MI')</code>	12345.67-
<code>TO_CHAR(-12345.67, '99,999.99PR')</code>	<12345.67>
<code>TO_CHAR(2007, 'RN')</code>	MMVII
<code>TO_CHAR(12345.67, 'TM')</code>	12345.67
<code>TO_CHAR(12345.67, 'U99,999.99')</code>	zł12,345.67
<code>TO_CHAR(12345.67, '99999V99')</code>	1234567

Jeżeli spróbujemy sformatować liczbę, która zawiera zbyt wiele cyfr dla przesłanego formatu, funkcja `TO_CHAR()` zwróci ciąg znaków `#`, na przykład:

```
SELECT TO_CHAR(12345678.90, '99,999.99')
FROM dual;
```

```
TO_CHAR(12
-----
#####
```

Funkcja `TO_CHAR()` zwróciła znaki `#`, ponieważ liczba `12345678,90` zawiera więcej cyfr niż limit dopuszczony przez format `99,999.99`.

Za pomocą funkcji `TO_CHAR()` można również konwertować na napisy kolumny zawierające liczby. Na przykład poniższe zapytanie wykorzystuje funkcję `TO_CHAR()` do przeprowadzenia konwersji wartości z kolumny `price` tabeli `products` na napisy:

```
SELECT product_id, 'Cena produktu wynosi' || TO_CHAR(price, '99D99L')
FROM products
WHERE product_id < 5;
```

```

PRODUCT_ID 'CENAPRODUKTUWYNOSI' || TO_CHAR(PRICE,
-----
1 Cena produktu wynosi          19.95zł
2 Cena produktu wynosi          30.00zł
3 Cena produktu wynosi          25.99zł
4 Cena produktu wynosi          13.95zł

```

TO_NUMBER()

Funkcja `TO_NUMBER(x [, format])` konwertuje x na liczbę. Można przesłać opcjonalny napis *format*, określający format x . W napisie *format* mogą znajdować się takie same parametry jak te wymienione w tabeli 4.4.

Poniższe zapytanie konwertuje na liczbę napis 970,13, korzystając z funkcji `TO_NUMBER()`:

```

SELECT TO_NUMBER('970,13')
FROM dual;

```

```

TO_NUMBER('970,13')
-----
          970.13

```

Kolejne zapytanie konwertuje napis 970,13 na liczbę za pomocą funkcji `TO_NUMBER()`, a następnie dodaje do tej liczby 25,5:

```

SELECT TO_NUMBER('970,13') + 25.5
FROM dual;

```

```

TO_NUMBER('970,13')+25.5
-----
          995.63

```

Kolejne zapytanie konwertuje napis -1 234,67zł na liczbę za pomocą funkcji `TO_NUMBER`, przesyłając do niej napis formatujący 9G999D99L:

```

SELECT TO_NUMBER('-1 200.00zł', '9G999D99L')
FROM dual;

```

```

TO_NUMBER('-1234,56zł', '9G999D99L')
-----
          -1234.56

```

CAST()

Funkcja `CAST(x AS typ)` konwertuje x na kompatybilny typ z bazy danych, określany przez parametr *typ*. W tabeli 4.6 przedstawiono dopuszczalne konwersje typów (są oznaczone *X*).

Poniższe zapytanie przedstawia wykorzystanie funkcji `CAST()` do konwersji literałów na określone typy:

```

SELECT
  CAST(12345.67 AS VARCHAR2(10)),
  CAST('9A4F' AS RAW(2)),
  CAST('05-LIP-07' AS DATE),
  CAST(12345.678 AS NUMBER(10,2))
FROM dual;

```

Tabela 4.6. *Dopuszczalne konwersje typów danych*

Na typ	Z typu						
	BINARY_FLOAT BINARY_DOUBLE	CHAR VARCHAR2	NUMBER	DATE TIMESTAMP INTERVAL	RAW	ROWID UROWID	NCHAR NVARCHAR2
BINARY_FLOAT BINARY_DOUBLE	X	X	X				X
CHAR VARCHAR2	X	X	X	X	X	X	
NUMBER	X	X	X				X
DATE TIMESTAMP INTERVAL		X		X			
RAW		X			X		
ROWID UROWID		X				X	
NCHAR NVARCHAR2	X		X	X	X	X	X

```
CAST(12345 CAST CAST('05- CAST(12345.678ASNUMBER(10,2))
-----
12345,67 9A4F 05-LIP-07 12345,68
```

Można również konwertować wartości z kolumn tabeli na inny typ, co obrazuje poniższe zapytanie:

```
SELECT
  CAST(price AS VARCHAR2(10)),
  CAST(price + 2 AS NUMBER(7,2)),
  CAST(price AS BINARY_DOUBLE)
FROM products
WHERE product_id = 1;

CAST(PRICE CAST(PRICE+2ASNUMBER(7,2)) CAST(PRICEASBINARY_DOUBLE)
-----
19,95 21,95 1,995E+001
```

W rozdziale 5. poznasz kolejne przykłady prezentujące wykorzystanie funkcji CAST() do konwertowania dat, czasu i interwałów. Z rozdziału 13. dowiesz się, jak konwertować kolekcje za pomocą funkcji CAST().

Funkcje wyrażeń regularnych

W tym podrozdziale zostały opisane wyrażenia regularne i związane z nimi funkcje bazy danych Oracle, które umożliwiają wyszukiwanie wzorców znaków w napisie. Załóżmy, że dysponujemy poniższą listą lat:

```
1965
1968
```


1971
1970

i chcemy z niej pobrać te z przedziału od 1965 do 1968. Możemy to zrobić za pomocą wyrażenia regularnego:

```
^196[5-8]$
```

Wyrażenie regularne zawiera zbiór *metaznaków*. W tym przykładzie są nimi `^`, `[5-8]` i `$`. `^` oznacza początek napisu, `[5-8]` — przedział znaków od 5 do 8, `$` — pozycję w napisie. `^196` oznacza więc napis rozpoczynający się od 196, a `[5-8]$` — napis kończący się cyfrą 5, 6, 7 lub 8, dlatego warunek `^196[5-8]$` jest spełniany przez 1965, 1966, 1967 i 1968, czyli dokładnie przez te lata, które chcieliśmy pobrać z listy.

W następnym przykładzie został wykorzystany ten napis będący cytatem z *Romea i Julii*:

Lecz cicho! Co za blask strzelił tam z okna!

Załóżmy, że chcemy wyszukać podnapis `blask`. Posłuży do tego poniższe wyrażenie regularne:

```
b[[:alpha:]]{4}
```

W tym wyrażeniu regularnym metaznakami są `[[:alpha:]]` i `{4}`. `[[:alpha:]]` oznacza znak alfanumeryczny od *A* do *Z* i od *a* do *z*; `{4}` powtarza czterokrotnie wcześniejsze dopasowanie. Po połączeniu `b`, `[[:alpha:]]` i `{4}` uzyskujemy wyrażenie spełniane przez sekwencję pięciu liter, rozpoczynającą się literą `b`, dlatego też wyrażenie regularne `b[[:alpha:]]{4}` jest spełniane przez `blask` z napisu.

W tabeli 4.7 opisano niektóre metaznaki możliwe do wykorzystania w wyrażeniach regularnych, a także ich znaczenie i przykłady zastosowania.

Tabela 4.7. Metaznaki w wyrażeniach regularnych

Metaznaki	Znaczenie	Przykłady
<code>\</code>	Spełniany przez znak specjalny lub literał albo wykonuje odwołanie wsteczne	<code>\n</code> oznacza znak nowego wiersza <code>\\</code> oznacza <code>\</code> <code>\(</code> oznacza (<code>\)</code> oznacza)
<code>^</code>	Oznacza początek napisu	<code>^A</code> jest spełniane przez <i>A</i> , jeżeli ta litera jest pierwszym znakiem napisu
<code>\$</code>	Oznacza koniec napisu	<code>\$B</code> jest spełniane przez <i>B</i> , jeżeli ta litera jest ostatnim znakiem napisu
<code>*</code>	Oznacza zero lub więcej wystąpień poprzedzającego znaku	<code>ba*rk</code> jest spełniane przez <code>brk</code> , <code>bark</code> , <code>baark</code> itd.
<code>+</code>	Oznacza co najmniej jedno wystąpienie poprzedzającego znaku	<code>ba+rk</code> jest spełniane przez <code>bark</code> , <code>baark</code> itd., ale nie przez <code>brk</code>
<code>?</code>	Oznacza zero lub jedno wystąpienie poprzedzającego znaku	<code>ba?rk</code> jest spełniane tylko przez <code>brk</code> i <code>bark</code>

Tabela 4.7. Metaznaki w wyrażeniach regularnych — ciąg dalszy

Metaznaki	Znaczenie	Przykłady
{ <i>n</i> }	Oznacza dokładnie <i>n</i> wystąpień znaku. <i>n</i> musi być liczbą całkowitą	hob{2}it jest spełniane przez hobbit
{ <i>n</i> , <i>m</i> }	Oznacza przynajmniej <i>n</i> i maksymalnie <i>m</i> wystąpień znaku, gdzie <i>n</i> i <i>m</i> są liczbami całkowitymi	hob{2..3}it jest spełniane tylko przez hobbit i hobbit
.	Oznacza dowolny jeden znak oprócz NULL	hob.it jest spełniane przez hobait, hobbit itd.
(wzorzec)	Podwyrażenie spełniane przez określony wzorzec. Za pomocą podwyrażeń można tworzyć złożone wyrażenia regularne. Można uzyskać dostęp do poszczególnych wystąpień, zwanych napisami przechwyconymi	telefo(n nia) jest spełnianie przez telefon i telefonia
<i>x</i> <i>y</i>	Jest spełniane przez <i>x</i> lub <i>y</i> , gdzie <i>x</i> i <i>y</i> stanowią co najmniej znak	wojna pokój jest spełniane przez słowo wojna lub pokój
[<i>abc</i>]	Jest spełniane przez każdy wymieniony znak	[ab] bc jest spełniane zarówno przez abc, jak i bbc
[<i>a-z</i>]	Jest spełniane przez każdy znak z określonego zakresu	[a-c]bc jest spełniane przez abc, bbc i cbc
[:]	Określa klasę znaku i jest spełniane przez dowolny znak z tej klasy	[:alphanum:] jest spełniane przez znaki alfanumeryczne 0–9, A–Z i a–z [:alpha:] jest spełniane przez litery A–Z i a–z [:blank:] jest spełniane przez znak spacji lub tabulacji [:digit:] jest spełniane przez cyfry 0–9 [:graph:] jest spełniane przez znak drukowalny [:lower:] jest spełniane przez małe litery alfabetu a–z [:print:] jest podobne do [:graph:], ale uwzględnia spację [:punct:] jest spełniane przez znaki interpunkcyjne . , " ' itd. [:space:] jest spełniane przez znaki odstępu [:upper:] jest spełniane przez wielkie litery alfabetu A–Z [:xdigit:] jest spełniane przez wszystkie znaki dopuszczalne w liczbie szesnastkowej: 0–9, A–F, a–f
[...]	Jest spełniane przez jeden symbol łączony, na przykład w symbolu wieloznakowym	Brak przykładu
[==]	Określa klasy równoważności	Brak przykładu
\ <i>n</i>	Jest to odwołanie wsteczne do wcześniej przechwyconego elementu; <i>n</i> musi być dodatnią liczbą całkowitą	(.)\1 jest spełniane przez dwa identyczne znaki następujące po sobie. (.) przechwytuje każdy znak oprócz NULL, a \1 powtarza przechwycenie, tym samym przechwytyjąc jeszcze raz ten sam znak. Dlatego też wyrażenie jest spełniane przez dwa identyczne znaki następujące po sobie

W Oracle Database 10g Release 2 wprowadzono kilka metaznaków używanych w Perlu. Zostały one opisane w tabeli 4.8.

Tabela 4.8. Metaznaki dodane z języka Perl

Metaznaki	Opis
\d	cyfra
\D	znak niebędący cyfrą
\w	słowo
\W	niesłowo
\s	znak białej spacji
\S	znak inny niż biała spacja
\A	spełniane tylko przez początek napisu lub jego koniec, jeżeli znajduje się przed znakiem nowego wiersza
\Z	spełniane tylko przez koniec napisu
*?	spełniane przez 0 lub więcej wystąpień wcześniejszego elementu wzorca
+?	spełniane przez co najmniej jedno wystąpienie wcześniejszego elementu wzorca
??	spełniane przez 0 lub jedno wystąpienie wcześniejszego elementu wzorca
{n}	spełniane przez dokładnie <i>n</i> wystąpień wcześniejszego elementu wzorca
{n,}	spełniane przez przynajmniej <i>n</i> wystąpień wcześniejszego elementu wzorca
{n,m}	spełniane przez przynajmniej <i>n</i> , ale mniej niż <i>m</i> wystąpień wcześniejszego elementu wzorca

W tabeli 4.9 opisano funkcje operujące na wyrażeniach regularnych. Zostały one wprowadzone w Oracle Database 10g i rozszerzone w wersji 11g, co zostało zaznaczone w tabeli.

W kolejnych podrozdziałach zostaną dokładniej opisane funkcje operujące na wyrażeniach regularnych.

REGEXP_LIKE()

Funkcja `REGEXP_LIKE(x, wzorzec [, opcja_dopasowania])` przeszukuje *x* zgodnie z wyrażeniem regularnym zdefiniowanym przez parametr *wzorzec*. Można również przesłać opcjonalny parametr *opcja_dopasowania*, który może być jednym z poniższych znaków:

- ◆ 'c' określającym, że podczas wyszukiwania wielkość liter będzie miała znaczenie (jest to ustawienie domyślne),
- ◆ 'I' określającym, że podczas wyszukiwania wielkość liter nie będzie miała znaczenia,
- ◆ 'n' umożliwiającym użycie operatora spełnianego przez dowolny znak,
- ◆ 'm' traktującym *x* jak wiele wierszy.

Poniższe zapytanie pobiera za pomocą funkcji `REGEXP_LIKE` informacje o klientach, których data urodzenia zawiera się w przedziale od 1965 do 1968:

```
SELECT customer_id, first_name, last_name, dob
FROM customers
WHERE REGEXP_LIKE(TO_CHAR(dob, 'YYYY'), '^196[5-8]$');
```

Tabela 4.9. Funkcje operujące na wyrażeniach regularnych

Funkcja	Opis
REGEXP_LIKE(<i>x</i> , <i>wzorzec</i> [, <i>opcja_dopasowania</i>])	<p>Przeszukuje <i>x</i> zgodnie z wyrażeniem regularnym zdefiniowanym przez parametr <i>wzorzec</i>. Można również przesłać opcjonalny parametr <i>opcja_dopasowania</i>, który może mieć jedną z poniższych wartości:</p> <ul style="list-style-type: none"> ♦ 'c' określa, że podczas wyszukiwania wielkość liter będzie miała znaczenie (jest to opcja domyślna) ♦ 'I' określa, że podczas wyszukiwania wielkość liter nie będzie miała znaczenia ♦ 'n' umożliwia użycie operatora spełnianego przez dowolny znak ♦ 'm' powoduje traktowanie <i>x</i> jako wielu linii
REGEXP_INSTR(<i>x</i> , <i>wzorzec</i> [, <i>start</i> [, <i>wystąpienie</i> [, <i>opcja_zwracania</i> [, <i>opcja_dopasowania</i> [, <i>opcja_podwyrażenia</i>]]]])	<p>Przeszukuje <i>x</i> zgodnie z wyrażeniem regularnym <i>wzorzec</i> i zwraca pozycję, na której występuje <i>wzorzec</i>. Można przesłać opcjonalne parametry:</p> <ul style="list-style-type: none"> ♦ <i>start</i> określa pozycję, od której zostanie rozpoczęte przeszukiwanie. Domyślną wartością jest 1, czyli pierwszy znak w <i>x</i> ♦ <i>wystąpienie</i> określa, które wystąpienie <i>wzorzec</i> powinno zostać zwrócone. Domyślną wartością jest 1, co oznacza, że funkcja zwróci pozycję pierwszego wystąpienia <i>wzorzec</i> ♦ <i>opcja_zwracania</i> określa, jaka liczba całkowita zostanie zwrócona. 0 określa, że zwrócona liczba całkowita będzie oznaczała pozycję pierwszego znaku w <i>x</i>. 1 oznacza, że zwrócona liczba całkowita będzie oznaczała pozycję znaku w <i>x</i> po wystąpieniu <i>wzorzec</i> ♦ <i>opcja_dopasowania</i> zmienia domyślny sposób dopasowywania do wzorca. Opcje są takie same jak w przypadku funkcji REGEXP_LIKE() ♦ <i>opcja_podwyrażenia</i> (nowość w Oracle Database 11g) ma następujące działanie: w przypadku wzorca z podwyrażeniami <i>opcja_podwyrażenia</i> jest nieujemną liczbą całkowitą od 0 do 9, określającą, które podwyrażenie we <i>wzorzec</i> jest celem funkcji. Na przykład wyrażenie 0123((abc)(de)f)ghi45(678) zawiera pięć podwyrażeń: abcdefghi, abcdef, abc, de oraz 678 <p>Jeżeli <i>opcja_podwyrażenia</i> będzie równa 0, zostanie zwrócona pozycja całego wyrażenia <i>wzorzec</i>. Jeżeli <i>wzorzec</i> nie zawiera prawidłowej liczby podwyrażeń, funkcja zwróci 0. Jeżeli <i>opcja_podwyrażenia</i> ma wartość NULL, funkcja zwróci NULL. Domyślną wartością <i>opcja_podwyrażenia</i> jest 0</p>
REGEXP_REPLACE(<i>x</i> , <i>wzorzec</i> [, <i>napis_zastępujący</i> [, <i>start</i> [, <i>wystąpienie</i> [, <i>opcja_dopasowania</i>]]]])	<p>Wyszukuje <i>wzorzec</i> w <i>x</i> i zastępuje go napisem <i>napis_zastępujący</i>. Znaczenie pozostałych opcji zostało opisane powyżej</p>
REGEXP_SUBSTR(<i>x</i> , <i>wzorzec</i> [, <i>start</i> [, <i>wystąpienie</i> [, <i>opcja_dopasowania</i> [, <i>opcja_podwyrażenia</i>]]]])	<p>Zwraca podnapis <i>x</i> zgodny z <i>wzorzec</i>. Wyszukiwanie rozpoczyna się od pozycji określonej przez <i>start</i>. Znaczenie pozostałych opcji zostało opisane powyżej. Znaczenie <i>opcja_podwyrażenia</i> (nowej w Oracle Database 11g) jest takie samo jak w przypadku funkcji REGEXP_INSTR()</p>
REGEXP_COUNT(<i>x</i> , <i>wzorzec</i> [, <i>start</i> [, <i>opcja_dopasowania</i>]])	<p>Nowość w Oracle Database 11g. Wyszukuje <i>wzorzec</i> w <i>x</i> i zwraca liczbę wystąpień <i>wzorzec</i>. Można przesłać poniższe opcjonalne parametry:</p> <ul style="list-style-type: none"> ♦ <i>start</i> określa pozycję, od której rozpocznie się wyszukiwanie. Domyślną wartością jest 1, co oznacza pierwszy znak w napisie <i>x</i> ♦ <i>opcja_dopasowania</i> zmienia domyślny sposób dopasowywania. Ma takie samo znaczenie jak w przypadku funkcji REGEXP_LIKE()

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB
1	Jan	Nikieł	65/01/01
2	Lidia	Stal	68/02/05

Kolejne zapytanie pobiera informacje o klientach, których imię rozpoczyna się literą j lub J. Należy zwrócić uwagę, że do funkcji REGEXP_LIKE() jest przesyłane wyrażenie regularne ^j, a opcja dopasowywania jest ustawiona na i (i oznacza, że w wyszukiwaniu nie będzie brana pod uwagę wielkość liter, więc w tym przykładzie ^j jest spełniane przez j i J):

```
SELECT customer_id, first_name, last_name, dob
FROM customers
WHERE REGEXP_LIKE(first_name, '^j', 'i');
```

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB
1	Jan	Nikieł	65/01/01
5	Jadwiga	Mosiądz	70/05/20

REGEXP_INSTR()

Funkcja REGEXP_INSTR(*x*, *wzorzec* [, *start* [, *wystąpienie* [, *opcja_zwracania* [, *opcja_dopasowania* [, *opcja_podwyrażenia*]]]]) wyszukuje *wzorzec* w *x*. Funkcja zwraca pozycję, na której *wzorzec* występuje w *x* (pozycje rozpoczynają się od 1).

Poniższe zapytanie zwraca pozycję spełniającą wyrażenie regularne b[[:alpha:]]{4}, korzystając z funkcji REGEXP_INSTR():

```
SELECT REGEXP_INSTR('Lecz cicho! Co za blask strzelił tam z okna!', 'b[[:alpha:]]{4}')
AS wynik
FROM dual;
```

```
WYNIK
-----
19
```

Została zwrócona liczba 19, która określa pozycję litery b ze słowa blask w całym napisie.

Następne zapytanie zwraca pozycję drugiego wystąpienia spełniającego wzorzec r[[:alpha:]](2), rozpoczynając od pozycji 1:

```
SELECT REGEXP_INSTR('Idzie rak, nieborak.', 'r[[:alpha:]]{2}', 1, 2) AS wynik
FROM dual;
```

```
WYNIK
-----
17
```

Kolejne zapytanie zwraca pozycję drugiego wystąpienia litery o, rozpoczynając wyszukiwanie od pozycji 10:

```
SELECT REGEXP_INSTR('Lecz cicho! Co za blask strzelił tam z okna!', 'o', 10, 2) AS wynik
FROM dual;
```

```
WYNIK
-----
14
```

REGEXP_REPLACE()

Funkcja `REGEXP_REPLACE(x, wzorzec [, napis_zastępujący [, start [, wystąpienie [, opcja_dopasowania]]])` wyszukuje `wzorzec` w `x` i zastępuje go napisem `napis_zastępujący`.

Poniższe zapytanie za pomocą funkcji `REGEXP_REPLACE()` zastępuje podnapis zgodny z wyrażeniem regularnym `o[[:alpha:]]{3}` napisem `szafy`:

```
SELECT REGEXP_REPLACE('Lecz cicho! Co za blask strzelił tam z okna!', 'o[[:alpha:]]{3}',
↳ 'szafy') AS wynik
FROM dual;
```

WYNIK

```
-----
Lecz cicho! Co za blask strzelił tam z szafy!
```

Słowo `okna` zostało zastąpione słowem `szafy`.

REGEXP_SUBSTR()

Funkcja `REGEXP_SUBSTR(x, wzorzec [, start [, wystąpienie [, opcja_dopasowania [, opcja_podwyrażenia]]])` wyszukuje w `x` podnapis zgodny z `wzorzec`. Przeszukiwanie jest rozpoczynane od pozycji określonej przez `start`.

Poniższe zapytanie zwraca podnapis zgodny z wyrażeniem regularnym `b[[:alpha:]]{3}`, korzystając z funkcji `REGEXP_SUBSTR()`:

```
SELECT REGEXP_SUBSTR('Lecz cicho! Co za blask strzelił tam z okna!', 'b[[:alpha:]]{4}')
↳ AS wynik
FROM dual;
```

WYNIK

```
-----
blask
```

REGEXP_COUNT()

Funkcja `REGEXP_COUNT()` została wprowadzona w Oracle Database 11g. Funkcja `REGEXP_COUNT(x, wzorzec [, start [, opcja_dopasowania])` wyszukuje `wzorzec` w `x` i zwraca liczbę jego wystąpień. Można przesłać opcjonalny parametr `start`, określający znak w `x`, od którego rozpocznie się wyszukiwanie, oraz opcjonalny parametr `opcja_dopasowania`, definiujący opcje dopasowania.

Poniższe zapytanie za pomocą funkcji `REGEXP_COUNT()` zwraca liczbę wystąpień wyrażenia regularnego `r[[:alpha:]]{2}` w napisie:

```
SELECT REGEXP_COUNT('Idzie rak, nieborak', 'r[[:alpha:]]{2}') AS wynik
FROM dual;
```

WYNIK

```
-----
2
```

Została zwrócona liczba 2, co oznacza, że w napisie wystąpiły dwa dopasowania do wyrażenia regularnego.

Funkcje agregujące

Funkcje prezentowane dotychczas operują na pojedynczych wierszach i zwracają jeden wiersz wyników dla każdego wiersza wejściowego. W tym podrozdziale poznamy funkcje agregujące, które operują na grupie wierszy i zwracają jeden wiersz wyników.



Uwaga

Funkcje agregujące są czasem nazywane grupującymi, ponieważ operują na grupach wierszy.

W tabeli 4.10 opisano niektóre funkcje agregujące, z których wszystkie zwracają typ NUMBER. Oto kilka właściwości funkcji agregujących, o których warto pamiętać podczas używania ich:

- ◆ Funkcje agregujące mogą być używane z dowolnymi, prawidłowymi wyrażeniami. Na przykład funkcje COUNT(), MAX() i MIN() mogą być używane z liczbami, napisami i datami.
- ◆ Wartość NULL jest ignorowana przez funkcje agregujące, ponieważ wskazuje, że wartość jest nieznana i z tego powodu nie może zostać użyta w funkcji.
- ◆ Wraz z funkcją agregującą można użyć słowa kluczowego DISTINCT, aby wykluczyć z obliczeń powtarzające się wpisy.

Tabela 4.10. *Funkcje agregujące*

Funkcja	Opis
AVG(<i>x</i>)	Zwraca średnią wartość <i>x</i>
COUNT(<i>x</i>)	Zwraca liczbę wierszy zawierających <i>x</i> , zwróconych przez zapytanie
MAX(<i>x</i>)	Zwraca maksymalną wartość <i>x</i>
MEDIAN(<i>x</i>)	Zwraca medianę <i>x</i>
MIN(<i>x</i>)	Zwraca minimalną wartość <i>x</i>
STDDEV(<i>x</i>)	Zwraca odchylenie standardowe <i>x</i>
SUM(<i>x</i>)	Zwraca sumę <i>x</i>
VARIANCE(<i>x</i>)	Zwraca wariancję <i>x</i>

Funkcje agregujące przedstawione w tabeli 4.10 zostaną szerzej opisane w kolejnych podrozdziałach. Z rozdziałów 7. i 8. dowiesz się, jak używać ich w połączeniu z klauzulami ROLLUP i RETURNING instrukcji SELECT. Klauzula ROLLUP umożliwi obliczenie częściowych podsumowań dla grup wierszy, klauzula RETURNING — zapisanie w zmiennej wartości zwróconej przez funkcję agregującą.

AVG()

Funkcja AVG(*x*) oblicza średnią wartość *x*. Poniższe zapytanie zwraca średnią cenę produktów. Należy zwrócić uwagę, że do funkcji AVG() jest przesyłana kolumna price z tabeli products:

```
SELECT AVG(price)
FROM products;
```

```
AVG(PRICE)
-----
19,7308333
```

Funkcje agregujące mogą być używane z dowolnymi prawidłowymi wyrażeniami. Na przykład poniższe zapytanie przesyła do funkcji `AVG()` wyrażenie `price + 2`. Na skutek tego do wartości `price` w każdym wierszu jest dodawane 2, a następnie jest obliczana średnia wyników:

```
SELECT AVG(price + 2)
FROM products;
```

```
AVG(PRICE+2)
-----
21,7308333
```

W celu wyłączenia z obliczeń identycznych wartości można użyć słowa kluczowego `DISTINCT`. Na przykład w poniższym zapytaniu użyto go do wyłączenia identycznych wartości z kolumny `price` podczas obliczania średniej za pomocą funkcji `AVG()`:

```
SELECT AVG(DISTINCT price)
FROM products;
```

```
AVG(DISTINCTPRICE)
-----
20,2981818
```

Należy zauważyć, że w tym przypadku średnia jest nieco wyższa niż wartość zwrócona przez pierwsze zapytanie prezentowane w tym podrozdziale. Jest tak dlatego, ponieważ wartość kolumny `price` dla produktu nr 2 (13,49) jest taka sama jak dla produktu nr 7. Jest uznawana za duplikat i wyłączana z obliczeń wykonywanych przez funkcję `AVG()`, dlatego średnia w tym przykładzie jest nieco wyższa.

COUNT()

Funkcja `COUNT(x)` oblicza liczbę wierszy zwróconych przez zapytanie. Poniższe zapytanie zwraca liczbę wierszy w tabeli `products`, korzystając z funkcji `COUNT()`:

```
SELECT COUNT(product_id)
FROM products;
```

```
COUNT(PRODUCT_ID)
-----
12
```



Należy unikać stosowania gwiazdki (*) jako argumentu funkcji `COUNT()`, ponieważ obliczenie wyniku może zająć więcej czasu. Zamiast tego należy przesłać nazwę kolumny z tabeli lub użyć pseudokolumny `ROWID`. (Jak wiesz z rozdziału 2., kolumna `ROWID` zawiera wewnętrzną lokalizację wiersza w bazie danych Oracle).

Poniższe zapytanie przesyła `ROWID` do funkcji `COUNT()` i zwraca liczbę wierszy w tabeli `products`:

```
SELECT COUNT(ROWID)
FROM products;
```



```

COUNT(ROWID)
-----
                12

```

MAX() i MIN()

Funkcje MAX(*x*) i MIN(*x*) zwracają maksymalną i minimalną wartość *x*. Poniższe zapytanie zwraca maksymalną i minimalną wartość z kolumny price tabeli products, korzystając z funkcji MAX() i MIN():

```

SELECT MAX(price), MIN(price)
FROM products;

```

```

MAX(PRICE) MIN(PRICE)
-----
    49,99      10,99

```

Funkcje MAX() i MIN() mogą być używane ze wszystkimi typami danych, włącznie z napisami i datami. Gdy używamy MAX() z napisami, są one porządkowane alfabetycznie, z „maksymalnym” napisem umieszczanym na dole listy i „minimalnym” napisem umieszczanym na górze listy. Na przykład na takiej liście napis Albert znajdzie się przed napisem Zenon. Poniższy przykład pobiera „maksymalny” i „minimalny” napis z kolumny name tabeli products, korzystając z funkcji MAX() i MIN():

```

SELECT MAX(name), MIN(name)
FROM products;

```

```

MAX(NAME)                MIN(NAME)
-----
Z innej planety          2412: Powrót

```

W przypadku dat, „maksymalną” datą jest najpóźniejszy moment, „minimalną” — najwcześniejszy. Poniższe zapytanie pobiera maksymalną i minimalną wartość z kolumny dob tabeli customers, korzystając z funkcji MAX() i MIN():

```

SELECT MAX(dob), MIN(dob)
FROM customers;

```

```

MAX(DOB) MIN(DOB)
-----
16-MAR-71 01-STY-65

```

STDDEV()

Funkcja STDDEV(*x*) oblicza odchylenie standardowe *x*. Jest ono funkcją statystyczną i jest definiowane jako pierwiastek kwadratowy wariancji (pojęcie wariancji zostanie opisane za chwilę).

Poniższe zapytanie oblicza odchylenie standardowe wartości w kolumnie price tabeli products, korzystając z funkcji STDDEV():

```

SELECT STDDEV(price)
FROM products;

```

```
STDDEV(PRICE)
-----
11,0896303
```

SUM()

Funkcja SUM() dodaje wszystkie wartości w x i zwraca wynik. Poniższe zapytanie zwraca sumę wartości z kolumny price tabeli products, korzystając z funkcji SUM():

```
SELECT SUM(price)
FROM products;

SUM(PRICE)
-----
236,77
```

VARIANCE()

Funkcja VARIANCE() oblicza wariancję x . Wariancja jest funkcją statystyczną i jest definiowana jako rozpiętość czy zróżnicowanie grupy liczb w próbce. Jest równa kwadratowi odchylenia standardowego.

Poniższe zapytanie oblicza wariancję wartości w kolumnie price tabeli products, korzystając z funkcji VARIANCE():

```
SELECT VARIANCE(price)
FROM products;

VARIANCE(PRICE)
-----
122,979899
```

Grupowanie wierszy

Czasami chcemy pogrupować wiersze tabeli i uzyskać jakieś informacje na temat tych grup wierszy. Na przykład możemy chcieć uzyskać średnie ceny różnych typów produktów z tabeli products. Zaczniemy od trudniejszego sposobu, by potem przejść do łatwiejszego, który wykorzystuje klauzulę GROUP BY w celu grupowania podobnych wierszy.

Trudniejszy sposób polega na ograniczeniu wierszy przesyłanych do funkcji AVG() za pomocą klauzuli WHERE. Na przykład poniższe zapytanie pobiera średnie ceny książek z tabeli products (książki mają product_type_id równy 1):

```
SELECT AVG(price)
FROM products
WHERE product_id = 1;

AVG(PRICE)
-----
19,95
```

Aby uzyskać średnią cenę innych typów produktów, musielibyśmy wykonywać dodatkowe zapytania z użyciem różnych wartości `product_type_id` w klauzuli `WHERE`. Jak można sobie wyobrazić, jest to dosyć żmudna praca. Pocięszająca jest wiadomość, że istnieje łatwiejszy sposób, wykorzystujący do grupowania klauzulę `GROUP BY`.

Grupowanie wierszy za pomocą klauzuli `GROUP BY`

Klauzula `GROUP BY` grupuje wiersze w bloki ze wspólną wartością jakiejś kolumny. Na przykład poniższe zapytanie grupuje wiersze z tabeli `products` w bloki z tą samą wartością `product_type_id`:

```
SELECT product_type_id
FROM products
GROUP BY product_type_id;
```

```
PRODUCT_TYPE_ID
-----
                1
                2
                4
                3
```

Należy zauważyć, że w zestawie wyników znajduje się tylko jeden wiersz dla każdego bloku wierszy z tą samą wartością `product_type_id`, a także, że między 1. i 2. występuje luka (wkrótce dowiemy się, dlaczego się tam znajduje). W zestawie wyników jest jeden wiersz dla produktów, dla których `product_type_id` jest równe 1, kolejny dla produktów, dla których `product_type_id` jest równe 2 itd. W tabeli `products` znajdują się dwa wiersze, dla których `product_type_id` jest równe 1, cztery wiersze, dla których `product_type_id` jest równe 2 itd. Te wiersze są grupowane w osobne bloki za pomocą klauzuli `GROUP BY` — każdy blok zawiera wszystkie wiersze z tą samą wartością `product_type_id`. Pierwszy zawiera dwa wiersze, drugi zawiera cztery wiersze itd.

Luka między wierszami 1. i 2. jest spowodowana tym, że w tabeli `products` występuje wiersz, w którym `product_type_id` ma wartość `NULL`. Ten wiersz jest przedstawiony w poniższym przykładzie:

```
SELECT product_id, name, price
FROM products
WHERE product_type_id IS NULL;
```

```
PRODUCT_ID NAME                                PRICE
-----
          12 Pierwsza linia                    13,49
```

Ponieważ wartość `product_type_id` w tym wierszu wynosi `NULL`, klauzula `GROUP BY` w poprzednim zapytaniu grupuje te wiersze w osobnym bloku. Wiersz w zestawie wyników jest pusty, ponieważ wartość `product_type_id` dla tego bloku wynosi `NULL` — stąd luka między wierszami 1. i 2.

Używanie wielu kolumn w grupie

W klauzuli `GROUP BY` można określić kilka kolumn. Na przykład poniższe zapytanie zawiera w klauzuli `GROUP BY` kolumny `product_id` i `customer_id` z tabeli `purchases`:

```
SELECT product_id, customer_id
FROM purchases
GROUP BY product_id, customer_id;
```

```
PRODUCT_ID CUSTOMER_ID
-----
          1             1
          1             2
          1             3
          1             4
          2             1
          2             2
          2             3
          2             4
          3             3
```

Używanie funkcji agregujących z grupami wierszy

Do funkcji agregującej można przesyłać bloki wierszy. Wykona ona obliczenia na grupie wierszy z każdego bloku i zwróci jedną wartość dla każdego bloku. Na przykład aby uzyskać liczbę wierszy z tą samą wartością `product_type_id` w tabeli `products`, musimy:

- ♦ pogrupować wiersze w bloki z tą samą wartością `product_type_id` za pomocą klauzuli `GROUP BY`,
- ♦ zliczyć wiersze w każdym bloku za pomocą funkcji `COUNT(ROWID)`.

Demonstruje to poniższe zapytanie:

```
SELECT product_type_id, COUNT(ROWID)
FROM products
GROUP BY product_type_id
ORDER BY product_type_id;
```

```
PRODUCT_TYPE_ID COUNT(ROWID)
-----
                1             2
                2             4
                3             2
                4             3
                1             1
```

Należy zauważyć, że w zestawie wyników znajduje się pięć wierszy, z których każdy odpowiada jednemu lub kilku wierszom z tabeli `products`, które zostały pogrupowane według wartości `product_type_id`. W zestawie wyników widzimy, że w dwóch wierszach `product_type_id` ma wartość 1, cztery wiersze mają wartość `product_type_id` równą 2 itd. Ostatni wiersz zestawu wyników wskazuje, że występuje jeden wiersz, w którym `product_type_id` ma wartość `NULL` (jest to wspomniany wcześniej wiersz Pierwsza linia).

Przejdźmy do innego przykładu. Aby uzyskać średnią cenę różnych typów produktów z tabeli products, musimy:

- ◆ za pomocą klauzuli GROUP BY pogrupować wiersze w bloki z tą samą wartością product_type_id,
- ◆ za pomocą funkcji AVG(price) obliczyć średnią cenę w każdym bloku wierszy.

Demonstruje to poniższe zapytanie:

```
SELECT product_type_id, AVG(price)
FROM products
GROUP BY product_type_id
ORDER BY product_type_id;
```

```
PRODUCT_TYPE_ID  AVG(PRICE)
-----
1                24,975
2                26,22
3                13,24
4                13,99
                13,49
```

Każda grupa wierszy z tą samą wartością product_type_id jest przesyłana do funkcji AVG(). Następnie funkcja ta oblicza średnią cenę w każdej grupie. Jak widzimy w zestawie wyników, średnia cena w grupie produktów z product_type_id równym 1 wynosi 24,975, a średnia cena w grupie produktów z product_type_id równym 2 wynosi 26,22. Należy zauważyć, że w ostatnim wierszu zestawu wyników jest wyświetlana średnia cena równa 13,49. Jest to po prostu cena produktu „Pierwsza linia”, czyli jedynego wiersza, w którym product_type_id ma wartość NULL.

Z klauzulą GROUP BY możemy używać dowolnych funkcji agregujących. Na przykład kolejne zapytanie pobiera wariancję cen produktów dla każdego product_type_id:

```
SELECT product_type_id, VARIANCE(price)
FROM products
GROUP BY product_type_id
ORDER BY product_type_id;
```

```
PRODUCT_TYPE_ID  VARIANCE(PRICE)
-----
1                50,50125
2                280,8772
3                ,125
4                7
                0
```

Warto pamiętać, że nie musimy umieszczać kolumn wykorzystywanych w klauzuli GROUP BY bezpośrednio za instrukcją SELECT. Na przykład poniższe zapytanie ma takie samo znaczenie jak poprzednie, ale product_type_id zostało pominięte w klauzuli SELECT:

```
SELECT VARIANCE(price)
FROM products
GROUP BY product_type_id
ORDER BY product_type_id;
```

```
VARIANCE(PRICE)
-----
      50,50125
      280,8772
           ,125
            7
             0
```

Wywołanie funkcji agregującej można również umieścić w klauzuli ORDER BY, co pokazuje poniższe zapytanie:

```
SELECT VARIANCE(price)
FROM products
GROUP BY product_type_id
ORDER BY VARIANCE(price);
```

```
VARIANCE(PRICE)
-----
              0
           ,125
            7
      50,50125
      280,8772
```

Nieprawidłowe użycie funkcji agregujących

Jeżeli zapytanie zawiera funkcję agregującą i pobiera kolumny nieujęte w niej, należy je umieścić w klauzuli GROUP BY. Jeśli o tym zapomnimy, zostanie wyświetlony komunikat o błędzie: ORA-00937: to nie jest jednogrupowa funkcja grupowa. Na przykład poniższe zapytanie próbuje pobrać dane z kolumny product_type_id oraz obliczyć AVG(price), pominięto w nim jednak klauzulę GROUP BY dla product_type_id:

```
SQL> SELECT product_type_id, AVG(price)
      2 FROM products;
SELECT product_type_id, AVG(price)
      *
```

BŁĄD w linii 1:

ORA-00937: to nie jest jednogrupowa funkcja grupowa

Błąd występuje, ponieważ baza danych nie wie, co zrobić z kolumną product_type_id. Zastanówmy się nad tym: zapytanie próbuje użyć funkcji agregującej AVG(), która operuje na wielu wierszach, ale próbuje również pobrać wartości product_type_id dla pojedynczych wierszy. Nie można zrobić tego jednocześnie. Należy zastosować klauzulę GROUP BY, aby wiersze z tą samą wartością product_type_id zostały zgrupowane. Wówczas baza danych prześle te grupy wierszy do funkcji AVG().



Ostrzeżenie

Jeżeli zapytanie zawiera funkcję agregującą i pobiera kolumny, które nie zostały w niej ujęte, należy je umieścić w klauzuli GROUP BY.

Poza tym nie można używać funkcji agregujących do ograniczania wierszy za pomocą klauzuli WHERE. W przeciwnym razie zostanie wyświetlony komunikat o błędzie: ORA-00934: funkcja grupowa nie jest tutaj dozwolona:

```
SQL> SELECT product_type_id, AVG(price)
2 FROM products
3 WHERE AVG(price) > 20
4 GROUP BY product_type_id;
WHERE AVG(price) > 20
*
```

Błąd w linii 3:

ORA-00934: funkcja grupowa nie jest tutaj dozwolona

Błąd występuje, ponieważ klauzula WHERE służy jedynie do filtrowania *pojedynczych* wierszy, a nie grup, do czego służy klauzula HAVING, opisana poniżej.

Filtrowanie grup wierszy za pomocą klauzuli HAVING

Klauzula HAVING służy do filtrowania grup wierszy. Umieszcza się ją za klauzulą GROUP BY:

```
SELECT ...
FROM ...
WHERE
GROUP BY ...
HAVING ...
ORDER BY ...;
```



Uwaga

Klauzula GROUP BY może być używana bez klauzuli HAVING, ale klauzula HAVING musi być używana z klauzulą GROUP BY.

Załóżmy, że chcemy przejrzeć typy produktów, których średnia cena jest większa niż 20 zł. W tym celu musimy:

- ◆ za pomocą klauzuli GROUP BY pogrupować wiersze w bloki o tej samej wartości product_type_id,
- ◆ za pomocą klauzuli HAVING ograniczyć zwrócone wyniki jedynie do tych, w których średnia cena jest większa od 20 zł.

Demonstruje to poniższe zapytanie:

```
SELECT product_type_id, AVG(price)
FROM products
GROUP BY product_type_id
HAVING AVG(price) > 20;
```

```
PRODUCT_TYPE_ID AVG(PRICE)
-----
1          24,975
2          26,22
```

Jak widzimy, zostały wyświetlone jedynie wiersze, w których średnia cena jest większa niż 20 zł.

Jednoczesne używanie klauzul WHERE i GROUP BY

Klauzule WHERE i GROUP BY mogą być użyte w tym samym zapytaniu. Wówczas klauzula WHERE najpierw filtruje zwracane wiersze, a następnie klauzula GROUP BY grupuje pozostałe w bloki. Na przykład w poniższym zapytaniu:

- ♦ Klauzula WHERE filtruje wiersze tabeli products, wybierając jedynie te, w których wartość price jest mniejsza od 15.
- ♦ Klauzula GROUP BY grupuje pozostałe wiersze według wartości kolumny product_type_id.

```
SELECT product_type_id, AVG(price)
FROM products
WHERE price < 15
GROUP BY product_type_id
ORDER BY product_type_id;
```

```
PRODUCT_TYPE_ID  AVG(PRICE)
-----
                2      14.45
                3      13.24
                4      12.99
                   13.49
```

Jednoczesne używanie klauzul WHERE, GROUP BY i HAVING

Klauzule WHERE, GROUP BY i HAVING mogą zostać użyte w tym samym zapytaniu. Wówczas klauzula WHERE najpierw filtruje zwracane wiersze, a następnie klauzula GROUP BY grupuje pozostałe wiersze w bloki, po czym klauzula HAVING filtruje grupy wierszy. Na przykład w poniższym zapytaniu:

- ♦ Klauzula WHERE filtruje wiersze tabeli products, wybierając jedynie te, w których wartość price jest mniejsza od 15.
- ♦ Klauzula GROUP BY grupuje pozostałe wiersze według wartości kolumny product_type_id.
- ♦ Klauzula HAVING filtruje grupy wierszy, wybierając jedynie te, w których średnia cena jest wyższa niż 13.

```
SELECT product_type_id, AVG(price)
FROM products
WHERE price < 15
GROUP BY product_type_id
HAVING AVG(price) > 13
ORDER BY product_type_id;
```

```
PRODUCT_TYPE_ID  AVG(PRICE)
-----
                2      14.45
                3      13.24
                   13.49
```


Porównajmy te wyniki z poprzednim przykładem: po filtracji została usunięta grupa wierszy, w których `product_type_id` ma wartość 4, a to dlatego, że w tej grupie wierszy średnia cena jest mniejsza od 13.

Ostatnie zapytanie wykorzystuje klauzulę `ORDER BY AVG(price)` w celu uporządkowania wyników według średniej ceny:

```
SELECT product_type_id, AVG(price)
FROM products WHERE price < 15
GROUP BY product_type_id
HAVING AVG(price) > 13
ORDER BY AVG(price);
```

```
PRODUCT_TYPE_ID  AVG(PRICE)
-----
3                13.24
                 13.49
2                14.45
```

Podsumowanie

Z tego rozdziału dowiedziałeś się, że:

- ◆ W bazie danych Oracle występują dwie główne grupy funkcji: jednowierszowe i agregujące.
- ◆ Funkcje jednowierszowe operują na pojedynczych wierszach i zwracają jeden wiersz wyników dla każdego wiersza wejściowego. Występuje pięć głównych typów funkcji jednowierszowych: znakowe, numeryczne, konwertujące, dat i wyrażeń regularnych.
- ◆ Funkcje agregujące operują na wielu wierszach i zwracają jeden wiersz wyników.
- ◆ Bloki wierszy mogą być grupowane za pomocą klauzuli `GROUP BY`.
- ◆ Grupy wierszy mogą być filtrowane za pomocą klauzuli `HAVING`.

W następnym rozdziale zawarto szczegółowe informacje o datach i czasie.