



Oracle Database 12c i SQL

Programowanie

Jason Price

Helion 

*Oracle
Press*

Tytuł oryginału: Oracle Database 12c SQL

Tłumaczenie: Andrzej Stefański
na podstawie „Oracle Database 11g i SQL. Programowanie”
w tłumaczeniu: Marcina Rogóza

ISBN: 978-83-246-9922-3

Original edition copyright © 2014 by McGraw-Hill Education (Publisher).
All rights reserved.

Polish edition copyright © 2015 by HELION S.A.
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/ord12p>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/ord12p.zip>

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Wprowadzenie	19
1 Wprowadzenie	23
Czym jest relacyjna baza danych?	23
Wstęp do SQL	24
Używanie SQL*Plus	25
Uruchamianie SQL*Plus	25
Uruchamianie SQL*Plus z wiersza poleceń	26
Wykonywanie instrukcji SELECT za pomocą SQL*Plus	26
SQL Developer	27
Tworzenie schematu bazy danych sklepu	30
Zawartość skryptu	30
Uruchamianie skryptu	31
Instrukcje DDL używane do tworzenia schematu bazy danych sklepu	32
Dodawanie, modyfikowanie i usuwanie wierszy	38
Dodawanie wiersza do tabeli	38
Modyfikowanie istniejącego wiersza w tabeli	39
Usuwanie wiersza z tabeli	40
Łączenie z bazą danych i rozłączanie	40
Kończenie pracy SQL*Plus	40
Wprowadzenie do Oracle PL/SQL	41
Podsumowanie	41
2 Pobieranie informacji z tabel bazy danych	43
Wykonywanie instrukcji SELECT dla jednej tabeli	43
Pobieranie wszystkich kolumn z tabeli	44
Wykorzystanie klauzuli WHERE do wskazywania wierszy do pobrania	44
Identyfikatory wierszy	44
Numery wierszy	45
Wykonywanie działań arytmetycznych	45
Wykonywanie obliczeń na datach	46
Korzystanie z kolumn w obliczeniach	47
Kolejność wykonywania działań	48
Używanie aliasów kolumn	48
Łączenie wartości z kolumn za pomocą konkatencji	49
Wartości null	49
Wyświetlanie unikatowych wierszy	50
Porównywanie wartości	51
Operator <>	51
Operator >	52
Operator <=	52
Operator ANY	52
Operator ALL	52

Korzystanie z operatorów SQL	53
Operator LIKE	53
Operator IN	54
Operator BETWEEN	55
Operatory logiczne	55
Operator AND	55
Operator OR	56
Następstwo operatorów	56
Sortowanie wierszy za pomocą klauzuli ORDER BY	57
Instrukcje SELECT wykorzystujące dwie tabele	58
Używanie aliasów tabel	59
Iloczyny kartezjańskie	60
Instrukcje SELECT wykorzystujące więcej niż dwie tabele	60
Warunki złączenia i typy złączeń	61
Nierównozłączenia	61
Złączenia zewnętrzne	62
Złączenia własne	65
Wykonywanie złączeń za pomocą składni SQL/92	66
Wykonywanie złączeń wewnętrznych dwóch tabel z wykorzystaniem składni SQL/92	66
Upraszczanie złączeń za pomocą słowa kluczowego USING	67
Wykonywanie złączeń wewnętrznych obejmujących więcej niż dwie tabele (SQL/92)	67
Wykonywanie złączeń wewnętrznych z użyciem wielu kolumn (SQL/92)	68
Wykonywanie złączeń zewnętrznych z użyciem składni SQL/92	68
Wykonywanie złączeń własnych z użyciem składni SQL/92	69
Wykonywanie złączeń krzyżowych z użyciem składni SQL/92	70
Podsumowanie	70
3 SQL*Plus	71
Przeglądanie struktury tabeli	71
Edycja instrukcji SQL	72
Zapisywanie, odczytywanie i uruchamianie plików	73
Formatowanie kolumn	76
Ustawianie rozmiaru strony	77
Ustawianie rozmiaru wiersza	78
Czyszczenie formatowania kolumny	78
Używanie zmiennych	79
Zmienne tymczasowe	79
Zmienne zdefiniowane	81
Tworzenie prostych raportów	83
Używanie zmiennych tymczasowych w skrypcie	83
Używanie zmiennych zdefiniowanych w skrypcie	84
Przesyłanie wartości do zmiennej w skrypcie	84
Dodawanie nagłówka i stopki	85
Obliczanie sum pośrednich	86
Uzyskiwanie pomocy od SQL*Plus	87
Automatyczne generowanie instrukcji SQL	88
Kończenie połączenia z bazą danych i pracy SQL*Plus	88
Podsumowanie	89
4 Proste funkcje	91
Typy funkcji	91
Funkcje jednowierszowe	91
Funkcje znakowe	92
Funkcje numeryczne	98

Funkcje konwertujące	103
Funkcje wyrażeń regularnych	112
Funkcje agregujące	117
AVG()	118
COUNT()	119
MAX() i MIN()	119
STDDEV()	120
SUM()	120
VARIANCE()	120
Grupowanie wierszy	120
Grupowanie wierszy za pomocą klauzuli GROUP BY	120
Nieprawidłowe użycie funkcji agregujących	123
Filtrowanie grup wierszy za pomocą klauzuli HAVING	124
Jednoczesne używanie klauzul WHERE i GROUP BY	124
Jednoczesne używanie klauzul WHERE, GROUP BY i HAVING	125
Podsumowanie	125
5 Składowanie oraz przetwarzanie dat i czasu	127
Proste przykłady składowania i pobierania dat	127
Konwertowanie typów DataGodzina za pomocą funkcji TO_CHAR() i TO_DATE()	128
Konwersja daty i czasu na napis za pomocą funkcji TO_CHAR()	128
Konwersja napisu na wyrażenie DataGodzina za pomocą funkcji TO_DATE()	132
Ustawianie domyślnego formatu daty	134
Jak Oracle interpretuje lata dwucyfrowe?	135
Użycie formatu YY	135
Użycie formatu RR	136
Funkcje operujące na datach i godzinach	137
ADD_MONTHS()	138
LAST_DAY()	138
MONTHS_BETWEEN()	138
NEXT_DAY()	139
ROUND()	139
SYSDATE	140
TRUNC()	140
Strefy czasowe	140
Funkcje operujące na strefach czasowych	141
Strefa czasowa bazy danych i strefa czasowa sesji	141
Uzyskiwanie przesunięć strefy czasowej	142
Uzyskiwanie nazw stref czasowych	143
Konwertowanie wyrażenia DataGodzina z jednej strefy czasowej na inną	143
Datowniki (znaczniki czasu)	143
Typy datowników	144
Funkcje operujące na znacznikach czasu	147
Interwały czasowe	151
Typ INTERVAL YEAR TO MONTH	152
Typ INTERVAL DAY TO SECOND	153
Funkcje operujące na interwałach	155
Podsumowanie	156
6 Podzapytania	157
Rodzaje podzapytań	157
Pisanie podzapytań jednowierszowych	157
Podzapytania w klauzuli WHERE	157
Użycie innych operatorów jednowierszowych	158

Podzapytania w klauzuli HAVING	159
Podzapytania w klauzuli FROM (widoki wbudowane)	160
Błędy, które można napotkać	160
Pisanie podzapytań wielowierszowych	161
Użycie operatora IN z podzapytaniem wielowierszowym	161
Użycie operatora ANY z podzapytaniem wielowierszowym	162
Użycie operatora ALL z podzapytaniem wielowierszowym	163
Pisanie podzapytań wielokolumnowych	163
Pisanie podzapytań skorelowanych	163
Przykład podzapytania skorelowanego	163
Użycie operatorów EXISTS i NOT EXISTS z podzapytaniem skorelowanym	164
Pisanie zagnieżdżonych podzapytań	166
Pisanie instrukcji UPDATE i DELETE zawierających podzapytania	167
Pisanie instrukcji UPDATE zawierającej podzapytanie	167
Pisanie instrukcji DELETE zawierającej podzapytanie	168
Przygotowywanie podzapytań	168
Podsumowanie	169

7 Zapytania zaawansowane	171
Operatory zestawu	171
Przykładowe tabele	171
Operator UNION ALL	172
Operator UNION	173
Operator INTERSECT	174
Operator MINUS	174
Łączenie operatorów zestawu	175
Użycie funkcji TRANSLATE()	176
Użycie funkcji DECODE()	177
Użycie wyrażenia CASE	178
Proste wyrażenia CASE	179
Przeszukiwane wyrażenia CASE	179
Zapytania hierarchiczne	181
Przykładowe dane	181
Zastosowanie klauzul CONNECT BY i START WITH	182
Użycie pseudokolumny LEVEL	183
Formatowanie wyników zapytania hierarchicznego	183
Rozpoczynanie od węzła innego niż główny	184
Użycie podzapytania w klauzuli START WITH	185
Poruszanie się po drzewie w górę	185
Eliminowanie węzłów i gałęzi z zapytania hierarchicznego	185
Umieszczanie innych warunków w zapytaniu hierarchicznym	186
Zapytania hierarchiczne wykorzystujące rekurencyjne podzapytania przygotowywane	187
Klauzule ROLLUP i CUBE	190
Przykładowe tabele	190
Użycie klauzuli ROLLUP	192
Klauzula CUBE	194
Funkcja GROUPING()	195
Klauzula GROUPING SETS	197
Użycie funkcji GROUPING_ID()	198
Kilkukrotne użycie kolumny w klauzuli GROUP BY	199
Użycie funkcji GROUP_ID()	200
Użycie CROSS APPLY i OUTER APPLY	201
CROSS APPLY	201
OUTER APPLY	202

LATERAL	202
Podsumowanie	203
8 Analiza danych	205
Funkcje analityczne	205
Przykładowa tabela	205
Użycie funkcji klasyfikujących	206
Użycie odwrotnych funkcji rankingowych	212
Użycie funkcji okna	212
Funkcje raportujące	218
Użycie funkcji LAG() i LEAD()	220
Użycie funkcji FIRST i LAST	221
Użycie funkcji regresji liniowej	221
Użycie funkcji hipotetycznego rankingu i rozkładu	222
Użycie klauzuli MODEL	223
Przykład zastosowania klauzuli MODEL	223
Dostęp do komórek za pomocą zapisu pozycyjnego i symbolicznego	224
Uzyskiwanie dostępu do zakresu komórek za pomocą BETWEEN i AND	225
Sięganie do wszystkich komórek za pomocą ANY i IS ANY	225
Pobieranie bieżącej wartości wymiaru za pomocą funkcji CURRENTV()	226
Uzyskiwanie dostępu do komórek za pomocą pętli FOR	227
Obsługa wartości NULL i brakujących	227
Modyfikowanie istniejących komórek	229
Użycie klauzul PIVOT i UNPIVOT	230
Prosty przykład klauzuli PIVOT	230
Przestawianie w oparciu o wiele kolumn	231
Użycie kilku funkcji agregujących w przestawieniu	232
Użycie klauzuli UNPIVOT	233
Zapytania o określoną liczbę wierszy	234
Użycie klauzuli FETCH FIRST	234
Użycie klauzuli OFFSET	235
Użycie klauzuli PERCENT	236
Użycie klauzuli WITH TIES	236
Odnajdywanie wzorców w danych	237
Odnajdywanie wzorców formacji typu V w danych z tabeli all_sales2	237
Odnajdywanie formacji typu W w danych z tabeli all_sales3	240
Odnajdywanie formacji typu V w tabeli all_sales3	241
Podsumowanie	242
9 Zmienianie zawartości tabeli	243
Wstawianie wierszy za pomocą instrukcji INSERT	243
Pomijanie listy kolumn	244
Określanie wartości NULL dla kolumny	244
Umieszczanie pojedynczych i podwójnych cudzysłówów w wartościach kolumn	245
Kopiowanie wierszy z jednej tabeli do innej	245
Modyfikowanie wierszy za pomocą instrukcji UPDATE	245
Klauzula RETURNING	246
Usuwanie wierszy za pomocą instrukcji DELETE	246
Integralność bazy danych	247
Wymuszanie więzów klucza głównego	247
Wymuszanie więzów kluczy obcych	247
Użycie wartości domyślnych	248
Scalanie wierszy za pomocą instrukcji MERGE	249

Transakcje bazodanowe	251
Zatwierdzanie i wycofywanie transakcji	251
Rozpoczynanie i kończenie transakcji	252
Punkty zachowania	252
ACID — właściwości transakcji	254
Transakcje współbieżne	254
Blokowanie transakcji	255
Poziomy izolacji transakcji	256
Przykład transakcji SERIALIZABLE	256
Zapytania retrospektywne	257
Przyznawanie uprawnień do używania zapytań retrospektywnych	257
Zapytania retrospektywne w oparciu o czas	258
Zapytania retrospektywne z użyciem SCN	259
Podsumowanie	260
10 Użytkownicy, uprawnienia i role	261
Bardzo krótkie wprowadzenie do przechowywania danych	261
Użytkownicy	262
Tworzenie konta użytkownika	262
Zmianie hasła użytkownika	263
Usuwanie konta użytkownika	263
Uprawnienia systemowe	263
Przyznawanie uprawnień systemowych użytkownikowi	263
Sprawdzanie uprawnień systemowych przyznanych użytkownikowi	264
Zastosowanie uprawnień systemowych	265
Odbieranie uprawnień systemowych	265
Uprawnienia obiektowe	266
Przyznawanie użytkownikowi uprawnień obiektowych	266
Sprawdzanie przekazanych uprawnień	267
Sprawdzanie otrzymanych uprawnień obiektowych	268
Zastosowanie uprawnień obiektowych	269
Synonimy	270
Synonimy publiczne	270
Odbieranie uprawnień obiektowych	271
Role	271
Tworzenie ról	271
Przyznawanie uprawnień roli	272
Przyznawanie roli użytkownikowi	272
Sprawdzanie ról przyznanych użytkownikowi	272
Sprawdzanie uprawnień systemowych przyznanych roli	273
Sprawdzanie uprawnień obiektowych przyznanych roli	274
Zastosowanie uprawnień przyznanych roli	275
Aktywacja i deaktywacja ról	276
Odbieranie roli	276
Odbieranie uprawnień roli	276
Usuwanie roli	277
Obserwacja	277
Uprawnienia wymagane do przeprowadzania obserwacji	277
Przykłady obserwacji	277
Perspektywy zapisu obserwacji	279
Podsumowanie	279

11 Tworzenie tabel, sekwencji, indeksów i perspektyw	281
Tabele	281
Tworzenie tabeli	281
Pobieranie informacji o tabelach	282
Uzyskiwanie informacji o kolumnach w tabeli	283
Zmienianie tabeli	284
Zmienianie nazwy tabeli	291
Dodawanie komentarza do tabeli	291
Obcinanie tabeli	292
Usuwanie tabeli	292
Typy <code>BINARY_FLOAT</code> i <code>BINARY_DOUBLE</code>	292
Użycie kolumn <code>DEFAULT ON NULL</code>	293
Kolumny niewidoczne	294
Sekwencje	296
Tworzenie sekwencji	296
Pobieranie informacji o sekwencjach	298
Używanie sekwencji	298
Wypełnianie klucza głównego z użyciem sekwencji	300
Określanie domyślnej wartości kolumny za pomocą sekwencji	300
Kolumny typu <code>IDENTITY</code>	301
Modyfikowanie sekwencji	301
Usuwanie sekwencji	302
Indeksy	302
Tworzenie indeksu typu B-drzewo	303
Tworzenie indeksów opartych na funkcjach	303
Pobieranie informacji o indeksach	304
Pobieranie informacji o indeksach kolumny	304
Modyfikowanie indeksu	305
Usuwanie indeksu	305
Tworzenie indeksu bitmapowego	305
Perspektywy	306
Tworzenie i używanie perspektyw	307
Modyfikowanie perspektywy	313
Usuwanie perspektywy	313
Używanie niewidocznych kolumn w perspektywach	313
Archiwa migawek	314
Podsumowanie	316
12 Wprowadzenie do programowania w PL/SQL	317
Bloki	317
Zmienne i typy	319
Logika warunkowa	319
Pętle	320
Proste pętle	320
Pętle <code>WHILE</code>	321
Pętle <code>FOR</code>	321
Kursory	322
Krok 1. — deklarowanie zmiennych przechowujących wartości kolumn	322
Krok 2. — deklaracja kursora	322
Krok 3. — otwarcie kursora	323
Krok 4. — pobieranie wierszy z kursora	323
Krok 5. — zamknięcie kursora	323
Pełny przykład — <code>product_cursor.sql</code>	324

Kursory i pętle FOR	325
Instrukcja OPEN-FOR	325
Kursory bez ograniczenia	327
Wyjątki	328
Wyjątek ZERO_DIVIDE	330
Wyjątek DUP_VAL_ON_INDEX	330
Wyjątek INVALID_NUMBER	330
Wyjątek OTHERS	331
Procedury	331
Tworzenie procedury	332
Wywoływanie procedury	333
Uzyskiwanie informacji o procedurach	334
Usuwanie procedury	335
Przeglądanie błędów w procedurze	335
Funkcje	335
Tworzenie funkcji	336
Wywoływanie funkcji	336
Uzyskiwanie informacji o funkcjach	337
Usuwanie funkcji	337
Pakiety	337
Tworzenie specyfikacji pakietu	338
Tworzenie treści pakietu	338
Wywoływanie funkcji i procedur z pakietu	339
Uzyskiwanie informacji o funkcjach i procedurach w pakiecie	340
Usuwanie pakietu	340
Wyzwalacze	340
Kiedy uruchamiany jest wyzwalacz	340
Przygotowania do przykładu wyzwalacza	341
Tworzenie wyzwalacza	341
Uruchamianie wyzwalacza	343
Uzyskiwanie informacji o wyzwalaczach	343
Włączanie i wyłączanie wyzwalacza	345
Usuwanie wyzwalacza	345
Rozszerzenia PL/SQL	345
Typ SIMPLE_INTEGER	345
Sekwencje w PL/SQL	346
Generowanie natywnego kodu maszynowego z PL/SQL	347
Klauzula WITH	347
Podsumowanie	348
13 Obiekty bazy danych	349
Wprowadzenie do obiektów	349
Uruchomienie skryptu tworzącego schemat bazy danych object_schema	350
Tworzenie typów obiektowych	350
Uzyskiwanie informacji o typach obiektowych za pomocą DESCRIBE	351
Użycie typów obiektowych w tabelach bazy danych	352
Obiekty kolumnowe	352
Tabele obiektowe	354
Identyfikatory obiektów i odwołania obiektowe	357
Porównywanie wartości obiektów	359
Użycie obiektów w PL/SQL	361
Funkcja get_products()	361
Procedura display_product()	362
Procedura insert_product()	363

Procedura update_product_price()	363
Funkcja get_product()	364
Procedura update_product()	364
Funkcja get_product_ref()	365
Procedura delete_product()	365
Procedura product_lifecycle()	366
Procedura product_lifecycle2()	367
Dziedziczenie typów	368
Uruchamianie skryptu tworzącego schemat bazy danych object_schema2	368
Dziedziczenie atrybutów	369
Użycie podtypu zamiast typu nadrzędnego	370
Przykłady SQL	370
Przykłady PL/SQL	371
Obiekty NOT SUBSTITUTABLE	371
Inne przydatne funkcje obiektów	372
Funkcja IS OF()	372
Funkcja TREAT()	375
Funkcja SYS_TYPEID()	378
Typy obiektowe NOT INSTANTIABLE	378
Konstruktory definiowane przez użytkownika	379
Przesłanianie metod	382
Uogólnione wywoływanie	384
Uruchomienie skryptu tworzącego schemat bazy danych object_schema3	384
Dziedziczenie atrybutów	384
Podsumowanie	385
14 Kolekcje	387
Podstawowe informacje o kolekcjach	387
Uruchomienie skryptu tworzącego schemat bazy danych collection_schema	387
Tworzenie kolekcji	388
Tworzenie typu VARRAY	388
Tworzenie tabeli zagnieżdżonej	388
Użycie kolekcji do definiowania kolumny w tabeli	389
Użycie typu VARRAY do zdefiniowania kolumny w tabeli	389
Użycie typu tabeli zagnieżdżonej do zdefiniowania kolumny w tabeli	389
Uzyskiwanie informacji o kolekcjach	389
Uzyskiwanie informacji o tablicy VARRAY	389
Uzyskiwanie informacji o tabeli zagnieżdżonej	390
Umieszczanie elementów w kolekcji	392
Umieszczanie elementów w tablicy VARRAY	392
Umieszczanie elementów w tabeli zagnieżdżonej	392
Pobieranie elementów z kolekcji	392
Pobieranie elementów z tablicy VARRAY	393
Pobieranie elementów z tabeli zagnieżdżonej	393
Użycie funkcji TABLE() do interpretacji kolekcji jako serii wierszy	394
Użycie funkcji TABLE() z typem VARRAY	394
Użycie funkcji TABLE() z tabelą zagnieżdżoną	395
Modyfikowanie elementów kolekcji	395
Modyfikowanie elementów tablicy VARRAY	396
Modyfikowanie elementów tabeli zagnieżdżonej	396
Użycie metody mapującej do porównywania zawartości tabel zagnieżdżonych	397
Użycie funkcji CAST do konwersji kolekcji z jednego typu na inny	399
Użycie funkcji CAST() do konwersji tablicy VARRAY na tabelę zagnieżdżoną	399
Użycie funkcji CAST() do konwersji tabeli zagnieżdżonej na tablicę VARRAY	400

14 Oracle Database 12c i SQL. Programowanie

Użycie kolekcji w PL/SQL	400
Manipulowanie tablicą VARRAY	400
Manipulowanie tabelą zagnieżdżoną	402
Metody operujące na kolekcjach w PL/SQL	403
Kolekcje wielopoziomowe	411
Uruchomienie skryptu tworzącego schemat bazy danych collection_schema2	412
Korzystanie z kolekcji wielopoziomowych	412
Rozszerzenia kolekcji wprowadzone w Oracle Database 10g	414
Uruchomienie skryptu tworzącego schemat bazy danych collection_schema3	414
Tablice asocjacyjne	415
Zmienianie rozmiaru typu elementu	415
Zwiększanie liczby elementów w tablicy VARRAY	416
Użycie tablic VARRAY w tabelach tymczasowych	416
Użycie innej przestrzeni tabel dla tabeli składującej tabelę zagnieżdżoną	416
Obsługa tabel zagnieżdżonych w standardzie ANSI	417
Podsumowanie	424
15 Duże obiekty	425
Podstawowe informacje o dużych obiektach (LOB)	425
Przykładowe pliki	425
Rodzaje dużych obiektów	426
Tworzenie tabel zawierających duże obiekty	427
Użycie dużych obiektów w SQL	428
Użycie obiektów CLOB i BLOB	428
Użycie obiektów BFILE	430
Użycie dużych obiektów w PL/SQL	431
APPEND()	433
CLOSE()	433
COMPARE()	434
COPY()	435
CREATETEMPORARY()	435
ERASE()	436
FILECLOSE()	436
FILECLOSEALL()	437
FILEEXISTS()	437
FILEGETNAME()	437
FILEISOPEN()	438
FILEOPEN()	438
FREETEMPORARY()	439
GETCHUNKSIZE()	439
GETLENGTH()	439
GET_STORAGE_LIMIT()	440
INSTR()	440
ISOPEN()	441
ISTEMPORARY()	441
LOADFROMFILE()	442
LOADBLOBFROMFILE()	443
LOADCLOBFROMFILE()	443
OPEN()	444
READ()	445
SUBSTR()	445
TRIM()	446

WRITE()	447
WRITEAPPEND()	447
Przykładowe procedury PL/SQL	448
Typy LONG i LONG RAW	462
Przykładowe tabele	462
Wstawianie danych do kolumn typu LONG i LONG RAW	462
Przekształcanie kolumn LONG i LONG RAW w duże obiekty	463
Nowe właściwości dużych obiektów w Oracle Database 10g	463
Niejawna konwersja między obiektami CLOB i NCLOB	464
Użycie atrybutu :new, gdy obiekt LOB jest używany w wyzwalaczu	464
Nowe właściwości dużych obiektów w Oracle Database 11g	465
Szyfrowanie danych LOB	465
Kompresja danych LOB	469
Usuwanie powtarzających się danych LOB	469
Nowe właściwości dużych obiektów w Oracle Database 12c	469
Podsumowanie	470
16 Optymalizacja SQL	471
Podstawowe informacje o optymalizacji SQL	471
Należy filtrować wiersze za pomocą klauzuli WHERE	471
Należy używać złączeń tabel zamiast wielu zapytań	472
Wykonując złączenia, należy używać w pełni kwalifikowanych odwołań do kolumn	473
Należy używać wyrażeń CASE zamiast wielu zapytań	473
Należy dodać indeksy do tabel	474
Kiedy tworzyć indeks typu B-drzewo	475
Kiedy tworzyć indeks bitmapowy	475
Należy stosować klauzulę WHERE zamiast HAVING	475
Należy używać UNION ALL zamiast UNION	476
Należy używać EXISTS zamiast IN	477
Należy używać EXISTS zamiast DISTINCT	477
Należy używać GROUPING SETS zamiast CUBE	478
Należy stosować zmienne dowiązane	478
Nieidentyczne instrukcje SQL	478
Identyczne instrukcje SQL korzystające ze zmiennych dowiązanych	478
Wypisywanie listy i wartości zmiennych dowiązanych	479
Użycie zmiennej dowiązanej do składowania wartości zwróconej przez funkcję PL/SQL	480
Użycie zmiennej dowiązanej do składowania wierszy z REFCURSOR	480
Porównywanie kosztu wykonania zapytań	480
Przeglądanie planów wykonania	481
Porównywanie planów wykonania	485
Przesyłanie wskazówek do optymalizatora	486
Dodatkowe narzędzia optymalizujące	487
Oracle Enterprise Manager	487
Automatic Database Diagnostic Monitor	488
SQL Tuning Advisor	488
SQL Access Advisor	488
SQL Performance Analyzer	488
Database Replay	488
Real-Time SQL Monitoring	488
SQL Plan Management	489
Podsumowanie	489

17 XML i baza danych Oracle	491
Wprowadzenie do XML	491
Generowanie XML z danych relacyjnych	492
XMLELEMENT()	492
XMLATTRIBUTES()	494
XMLFOREST()	494
XMLAGG()	495
XMLCOLATTVAL()	497
XMLCONCAT()	498
XMLPARSE()	498
XMLPI()	499
XMLCOMMENT()	499
XMLSEQUENCE()	500
XMLSERIALIZE()	501
Przykład zapisywania danych XML do pliku w PL/SQL	501
XMLQUERY()	502
Zapisywanie XML w bazie danych	506
Przykładowy plik XML	506
Tworzenie przykładowego schematu XML	506
Pobieranie informacji z przykładowego schematu XML	508
Aktualizowanie informacji w przykładowym schemacie XML	511
Podsumowanie	514
A Typy danych Oracle	515
Typy w Oracle SQL	515
Typy w Oracle PL/SQL	517
Skorowidz	519

Analiza danych

Z tego rozdziału dowiesz się, jak:

- wykorzystywać funkcje analityczne, wykonujące złożone obliczenia,
- wykonywać obliczenia międzywierszowe za pomocą klauzuli `MODEL`,
- wykorzystywać klauzule `PIVOT` i `UNPIVOT`, które przydają się do przeglądania ogólnych trendów w dużych zbiorach danych,
- wykonywać zapytania zwracające pierwszych N lub ostatnich N wierszy wyników,
- odnajdywać wzorce w danych za pomocą klauzuli `MATCH_RECOGNIZE`.

Funkcje analityczne

Baza danych zawiera wiele wbudowanych funkcji analitycznych, umożliwiających wykonywanie złożonych obliczeń, takich jak wyszukanie najlepiej sprzedającego się rodzaju produktów w poszczególnych miesiącach, najlepszych sprzedawców itd. Funkcje analityczne możemy podzielić na następujące kategorie:

- **funkcje klasyfikujące** — umożliwiają obliczanie klasyfikacji, percentyli i n -tyli (tertyli, kwartyli itd.),
- **odwrotne funkcje percentyli** — umożliwiają obliczenie wartości odpowiadającej percentylowi,
- **funkcje okien** — umożliwiają obliczanie agregatów skumulowanych i ruchomych,
- **funkcje raportujące** — umożliwiają obliczanie na przykład udziałów w rynku,
- **funkcje `LAG()` i `LEAD()`** — umożliwiają pobranie wartości z wiersza, który jest oddalony o określoną liczbę wierszy od bieżącego,
- **funkcje `FIRST()` i `LAST()`** — umożliwiają pobranie odpowiednio pierwszej i ostatniej wartości w uporządkowanej grupie,
- **funkcje regresji liniowej** — umożliwiają dopasowanie linii regresji metodą najmniejszych kwadratów do zestawu par liczb,
- **funkcje hipotetycznych klasyfikacji i dystrybucji** — umożliwiają obliczenie klasyfikacji i percentyli, w którym znalazłby się nowy wiersz po wstawieniu go do tabeli.

Te funkcje zostaną opisane wkrótce, zaczniemy jednak od omówienia przykładowej tabeli.

Przykładowa tabela

W kolejnych podrozdziałach będziemy korzystali z tabeli `all_sales`. Składa się ona z sumy wszystkich sprzedaży (w złotych) dla konkretnych lat, miesięcy, rodzajów produktu i pracownika. Tabela `all_sales` jest tworzona przez skrypt `store_schema.sql` za pomocą następującej instrukcji:

```
CREATE TABLE all_sales (  
  year INTEGER NOT NULL,  
  month INTEGER NOT NULL,  
  prd_type_id INTEGER  
    CONSTRAINT all_sales_fk_product_types  
      REFERENCES product_types(product_type_id),  
  emp_id INTEGER  
    CONSTRAINT all_sales_fk_employees2  
      REFERENCES employees2(employee_id),  
  amount NUMBER(8, 2),  
  CONSTRAINT all_sales_pk PRIMARY KEY (  
    year, month, prd_type_id, emp_id  
  )  
);
```

Tabela `all_sales` zawiera pięć kolumn:

- **year**, w której składowany jest rok sprzedaży,
- **month**, w której składowany jest miesiąc sprzedaży (od 1 do 12),
- **prd_type_id**, w której składowany jest `product_type_id` produktu,
- **emp_id**, w której składowany jest `employee_id` pracownika obsługującego sprzedaż,
- **amount**, w której składowana jest wartość sprzedaży w złotych.

Poniższe zapytanie pobiera pierwsze 12 wierszy z tabeli `all_sales`:

```
SELECT *  
FROM all_sales  
WHERE ROWNUM <= 12;
```

YEAR	MONTH	PRD_TYPE_ID	EMP_ID	AMOUNT
2003	1	1	21	10034,84
2003	2	1	21	15144,65
2003	3	1	21	20137,83
2003	4	1	21	25057,45
2003	5	1	21	17214,56
2003	6	1	21	15564,64
2003	7	1	21	12654,84
2003	8	1	21	17434,82
2003	9	1	21	19854,57
2003	10	1	21	21754,19
2003	11	1	21	13029,73
2003	12	1	21	10034,84



Uwaga

Tabela `all_sales` zawiera znacznie więcej wierszy, ze względu jednak na ograniczoną ilość miejsca nie umieszczono całego listingu.

Przejdźmy do omówienia funkcji klasyfikujących.

Użycie funkcji klasyfikujących

Funkcje klasyfikujące służą do obliczania klasyfikacji, percentyli i *n*-tyli. Zostały przedstawione w tabeli 8.1. Zaczniemy od opisu funkcji `RANK()` i `DENSE_RANK()`.

Użycie funkcji `RANK()` i `DENSE_RANK()`

Funkcje `RANK()` i `DENSE_RANK()` służą do klasyfikowania elementów w grupie. Różnica między nimi dotyczy sposobu, w jaki zachowują się, gdy kilka elementów przypada na tę samą pozycję: funkcja `RANK()` pozostawia w takim przypadku lukę w sekwencji, a funkcja `DENSE_RANK()` nie pozostawia luk.

Na przykład gdybyśmy chcieli poklasyfikować sprzedaż według typów produktu, a dwa typy przypadałyby na pierwsze miejsce, funkcja `RANK()` umieściłaby dwa typy na pierwszym miejscu, ale kolejny typ

Tabela 8.1. Funkcje klasyfikujące

Funkcja	Opis
RANK()	Zwraca klasyfikację w grupie. Funkcja RANK() pozostawia lukę w sekwencji rang w przypadku równowagi
DENSE_RANK()	Zwraca klasyfikację w grupie. Funkcja DENSE_RANK() nie pozostawia luki w sekwencji rang w przypadku równowagi
CUME_DIST()	Zwraca pozycję określonej wartości w grupie wartości. CUME_DIST() jest skrótem angielskiej nazwy dystrybucyjnej (ang. <i>cumulative distribution</i>)
PERCENT_RANK()	Zwraca procentową rangę wartości w grupie wartości
NTILE()	Zwraca <i>n</i> -tyle: tertyle, kwartyle itd.
ROW_NUMBER()	Zwraca numer z każdym wierszem z grupy

znajdowałby się na trzeciej pozycji. Funkcja DENSE_RANK() natomiast również umieściłaby dwa rodzaje produktów na pierwszym miejscu, kolejny znajdowałby się jednak na drugiej pozycji.

Poniższe zapytanie obrazuje zastosowanie funkcji RANK() i DENSE_RANK() do pobrania klasyfikacji sprzedaży według rodzajów produktu w 2003 roku. Należy zwrócić uwagę na użycie słowa kluczowego OVER w wywołaniach funkcji RANK() i DENSE_RANK():

```
SELECT
    prd_type_id, SUM(amount),
    RANK() OVER (ORDER BY SUM(amount) DESC) AS rank,
    DENSE_RANK() OVER (ORDER BY SUM(amount) DESC) AS dense_rank
FROM all_sales
WHERE year = 2003
AND amount IS NOT NULL
GROUP BY prd_type_id
ORDER BY prd_type_id;
```

PRD_TYPE_ID	SUM(AMOUNT)	RANK	DENSE_RANK
1	905081,84	1	1
2	186381,22	4	4
3	478270,91	2	2
4	402751,16	3	3

Sprzedaż produktu nr 1 jest klasyfikowana jako 1, sprzedaż produktu nr 2 — jako 4 itd. Ponieważ nie występują konflikty, funkcje RANK() i DENSE_RANK() zwracają takie same rangi.

Kolumna amount tabeli all_sales zawiera wartość NULL dla wszystkich wierszy, w których PRD_TYPE_ID wynosi 5. W poprzednim zapytaniu te wiersze zostały pominięte na skutek umieszczenia w klauzuli WHERE warunku AND amount IS NOT NULL. W kolejnym przykładzie zostały one dołączone w wyniku pominięcia wiersza AND z klauzuli WHERE:

```
SELECT
    prd_type_id, SUM(amount),
    RANK() OVER (ORDER BY SUM(amount) DESC) AS rank,
    DENSE_RANK() OVER (ORDER BY SUM(amount) DESC) AS dense_rank
FROM all_sales
WHERE year = 2003
GROUP BY prd_type_id
ORDER BY prd_type_id;
```

PRD_TYPE_ID	SUM(AMOUNT)	RANK	DENSE_RANK
1	905081,84	2	2
2	186381,22	5	5
3	478270,91	3	3
4	402751,16	4	4
5		1	1

Ostatni wiersz zawiera wartość NULL jako sumę w kolumnie amount i w tym wierszu funkcje RANK() i DENSE_RANK() zwracają 1. Jest tak dlatego, że domyślnie w klasyfikacjach malejących obydwie te funkcje przypisują wartości NULL najwyższą rangę (1) (jeżeli słowo kluczowe DESC zostanie użyte w klauzuli OVER) i najniższą rangę w rankingach rosnących (jeżeli w klauzuli OVER zostanie użyte słowo kluczowe ASC).

**Sterowanie klasyfikowaniem wartości NULL
za pomocą klauzul NULLS FIRST i NULLS LAST**

W funkcjach analitycznych za pomocą klauzul NULLS FIRST i NULLS LAST możemy jawnie określić, czy wartości NULL mają być najwyższe, czy też najniższe w grupie. W poniższym przykładzie użyto klauzuli NULLS LAST do określenia, że wartości NULL są wartościami najniższymi:

```
SELECT
    prd_type_id, SUM(amount),
    RANK() OVER (ORDER BY SUM(amount) DESC NULLS LAST) AS rank,
    DENSE_RANK() OVER (ORDER BY SUM(amount) DESC NULLS LAST) AS dense_rank
FROM all_sales
WHERE year = 2003
GROUP BY prd_type_id
ORDER BY prd_type_id;
```

PRD_TYPE_ID	SUM(AMOUNT)	RANK	DENSE_RANK
1	905081,84	1	1
2	186381,22	4	4
3	478270,91	2	2
4	402751,16	3	3
5		5	5

Użycie klauzuli PARTITION BY w funkcjach analitycznych

Klauzulę PARTITION BY stosujemy w funkcjach analitycznych, jeżeli musimy podzielić grupy na podgrupy. Na przykład jeżeli chcemy podzielić wartości sprzedaży na miesiące, możemy użyć klauzuli PARTITION BY month, tak jak w poniższym zapytaniu:

```
SELECT
    prd_type_id, month, SUM(amount),
    RANK() OVER (PARTITION BY month ORDER BY SUM(amount) DESC) AS rank
FROM all_sales
WHERE year = 2003
AND amount IS NOT NULL
GROUP BY prd_type_id, month
ORDER BY prd_type_id, month;
```

PRD_TYPE_ID	MONTH	SUM(AMOUNT)	RANK
1	1	38909,04	1
1	2	70567,9	1
1	3	91826,98	1
1	4	120344,7	1
1	5	97287,36	1
1	6	57387,84	1
1	7	60929,04	2
1	8	75608,92	1
1	9	85027,42	1
1	10	105305,22	1
1	11	55678,38	1
1	12	46209,04	2
2	1	14309,04	4
2	2	13367,9	4
2	3	16826,98	4
2	4	15664,7	4
2	5	18287,36	4

2	6	14587,84	4
2	7	15689,04	3
2	8	16308,92	4
2	9	19127,42	4
2	10	13525,14	4
2	11	16177,84	4
2	12	12509,04	4
3	1	24909,04	2
3	2	15467,9	3
3	3	20626,98	3
3	4	23844,7	2
3	5	18687,36	3
3	6	19887,84	3
3	7	81589,04	1
3	8	62408,92	2
3	9	46127,42	3
3	10	70325,29	3
3	11	46187,38	2
3	12	48209,04	1
4	1	17398,43	3
4	2	17267,9	2
4	3	31026,98	2
4	4	16144,7	3
4	5	20087,36	2
4	6	33087,84	2
4	7	12089,04	4
4	8	58408,92	3
4	9	49327,42	2
4	10	75325,14	2
4	11	42178,38	3
4	12	30409,05	3

Użycie operatorów ROLLUP, CUBE i GROUPING SETS w funkcjach analitycznych

Z funkcjami analitycznymi mogą być stosowane operatory ROLLUP, CUBE i GROUPING SETS. W poniższym zapytaniu użyto operatora ROLLUP oraz funkcji RANK(), aby uzyskać klasyfikację wartości sprzedaży według identyfikatorów typów produktu:

```
SELECT
    prd_type_id, SUM(amount),
    RANK() OVER (ORDER BY SUM(amount) DESC) AS rank
FROM all_sales
WHERE year = 2003
GROUP BY ROLLUP(prd_type_id)
ORDER BY prd_type_id;
```

PRD_TYPE_ID	SUM(AMOUNT)	RANK
1	905081,84	3
2	186381,22	6
3	478270,91	4
4	402751,16	5
5		1
	1972485,13	2

W kolejnym zapytaniu użyto CUBE i RANK() do uzyskania całej klasyfikacji wartości sprzedaży według identyfikatorów typu produktu oraz identyfikatorów pracowników:

```
SELECT
    prd_type_id, emp_id, SUM(amount),
    RANK() OVER (ORDER BY SUM(amount) DESC) AS rank
FROM all_sales
WHERE year = 2003
GROUP BY CUBE(prd_type_id, emp_id)
ORDER BY prd_type_id, emp_id;
```

PRD_TYPE_ID	EMP_ID	SUM(AMOUNT)	RANK
1	21	197916,96	19
1	22	214216,96	17
1	23	98896,96	26
1	24	207216,96	18
1	25	93416,96	28
1	26	93417,04	27
1		905081,84	9
2	21	20426,96	40
2	22	19826,96	41
2	23	19726,96	42
2	24	43866,96	34
2	25	32266,96	38
2	26	50266,42	31
2		186381,22	21
3	21	140326,96	22
3	22	116826,96	23
3	23	112026,96	24
3	24	34829,96	36
3	25	29129,96	39
3	26	45130,11	33
3		478270,91	10
4	21	108326,96	25
4	22	81426,96	30
4	23	92426,96	29
4	24	47456,96	32
4	25	33156,96	37
4	26	39956,36	35
4		402751,16	13
5	21		1
5	22		1
5	23		1
5	24		1
5	25		1
5	26		1
5			1
	21	466997,84	11
	22	432297,84	12
	23	323077,84	15
	24	333370,84	14
	25	187970,84	20
	26	228769,93	16
		1972485,13	8

W następnym zapytaniu użyto GROUPING SETS i RANK() do pobrania jedynie klasyfikacji częściowych podsumowań wartości sprzedaży:

```
SELECT
  prd_type_id, emp_id, SUM(amount),
  RANK() OVER (ORDER BY SUM(amount) DESC) AS rank
FROM all_sales
WHERE year = 2003
GROUP BY GROUPING SETS(prd_type_id, emp_id)
ORDER BY prd_type_id, emp_id;
```

PRD_TYPE_ID	EMP_ID	SUM(AMOUNT)	RANK
1		905081,84	2
2		186381,22	11
3		478270,91	3
4		402751,16	6
5			1
	21	466997,84	4
	22	432297,84	5

23	323077,84	8
24	333370,84	7
25	187970,84	10
26	228769,93	9

Użycie funkcji CUME_DIST() i PERCENT_RANK()

Funkcja CUME_DIST() służy do wyznaczania pozycji określonej wartości w grupie, a funkcja PERCENT_RANK() — do wyznaczania rangi procentowej określonej wartości względem grupy wartości.

Poniższe zapytanie obrazuje użycie funkcji CUME_DIST() i PERCENT_DIST() do pobrania dystrybuanty i klasyfikacji procentowej wartości sprzedaży:

```
SELECT
  prd_type_id, SUM(amount),
  CUME_DIST() OVER (ORDER BY SUM(amount) DESC) AS cume_dist,
  PERCENT_RANK() OVER (ORDER BY SUM(amount) DESC) AS percent_rank
FROM all_sales
WHERE year = 2003
GROUP BY prd_type_id
ORDER BY prd_type_id;
```

PRD_TYPE_ID	SUM(AMOUNT)	CUME_DIST	PERCENT_RANK
1	905081,84	,4	,25
2	186381,22	1	1
3	478270,91	,6	,5
4	402751,16	,8	,75
5		,2	0

Użycie funkcji NTILE()

Funkcja NTILE(*kubelki*) służy do obliczania *n*-tyli (tertyli, kwartyli itd.). Parametr *kubelki* określa liczbę „kubelków”, w których zostaną porozmieszczane grupy wierszy. Na przykład:

- NTILE(2) określa dwa kubelki, w związku z czym wiersze zostaną podzielone na dwie grupy wierszy,
- NTILE(4) dzieli grupy na cztery kubelki, w wyniku czego dzieli wiersze na cztery grupy.

Poniższe zapytanie obrazuje użycie funkcji NTILE(). Przesłano do niej wartość 4, aby podzielić grupy wierszy na cztery kubelki:

```
SELECT
  prd_type_id, SUM(amount),
  NTILE(4) OVER (ORDER BY SUM(amount) DESC) AS ntile
FROM all_sales
WHERE year = 2003
AND amount IS NOT NULL
GROUP BY prd_type_id
ORDER BY prd_type_id;
```

PRD_TYPE_ID	SUM(AMOUNT)	NTILE
1	905081,84	1
2	186381,22	4
3	478270,91	2
4	402751,16	3

Użycie funkcji ROW_NUMBER()

Funkcja ROW_NUMBER() zwraca liczbę (od 1) z każdym wierszem w grupie. Poniższe zapytanie obrazuje jej użycie:

```
SELECT
  prd_type_id, SUM(amount),
  ROW_NUMBER() OVER (ORDER BY SUM(amount) DESC) AS row_number
```

```
FROM all_sales
WHERE year = 2003
GROUP BY prd_type_id
ORDER BY prd_type_id;
```

PRD_TYPE_ID	SUM(AMOUNT)	ROW_NUMBER
1	905081,84	2
2	186381,22	5
3	478270,91	3
4	402751,16	4
5		1

Na tym zakończymy omówienie funkcji rankingowych.

Użycie odwrotnych funkcji rankingowych

Z podrozdziału „Użycie funkcji CUME_DIST() i PERCENT_RANK()” dowiedziałeś się, że funkcja CUME_DIST() służy do obliczania pozycji określonej wartości w grupie wartości. Wiesz też, że za pomocą funkcji PERCENT_RANK() oblicza się rangę procentową wartości w grupie wartości.

Ten podrozdział dotyczy używania odwrotnych funkcji rankingowych działających odwrotnie do funkcji CUME_DIST() i PERCENT_RANK(). Dostępne są dwie odwrotne funkcje rankingowe:

- PERCENTILE_DISC(x) bada wartości dystrybuanty w grupie, aż odnajdzie taką, która jest większa od x lub równa mu,
- PERCENTILE_CONT(x) bada wartości rankingu procentowego aż do wyszukania takiej, która jest większa od x lub równa mu.

W poniższym zapytaniu użyto funkcji PERCENTILE_CONT() i PERCENTILE_DISC() do pobrania sumy wartości sprzedaży, dla której percentyl jest większy od 0,6 lub równy tej wartości:

```
SELECT
  PERCENTILE_CONT(0.6) WITHIN GROUP (ORDER BY SUM(amount) DESC)
  AS percentile_cont,
  PERCENTILE_DISC(0.6) WITHIN GROUP (ORDER BY SUM(amount) DESC)
  AS percentile_disc
FROM all_sales
WHERE year = 2003
GROUP BY prd_type_id;
```

PERCENTILE_CONT	PERCENTILE_DISC
417855,11	402751,16

Jeżeli porównamy powyższe sumy z wynikami zwróconymi w podrozdziale „Użycie funkcji CUME_DIST() i PERCENT_RANK()”, zobaczymy, że odpowiadają one tym wartościom, dla których dystrybanta i ranga procentowa wynoszą odpowiednio 0,6 i 0,75.

Użycie funkcji okna

Funkcje okna służą do obliczania sum kumulacyjnych i średnich kroczących w określonych zakresach wierszy, zakresie wartości czy przedziale czasu.

Zapytanie zwraca zestaw wierszy zwany zestawem wyników. Termin „okno” oznacza podzbiór wierszy zestawu wyników. Ten podzbiór „widziany” przez okno jest przetwarzany przez funkcje okna, które zwracają wartość. Możemy zdefiniować początek i koniec okna.

Okno może być użyte z następującymi funkcjami: SUM(), AVG(), MAX(), MIN(), COUNT(), VARIANCE() i STDDEV(), które zostały opisane w rozdziale 4. Funkcje okna mogą być również wykorzystywane z funkcjami FIRST_VALUE(), LAST_VALUE() oraz NTH_VALUE(), które zwracają pierwszą, ostatnią i n -tą wartość w oknie. Więcej informacji na temat tych funkcji znajduje się w dalszej części tego rozdziału. Z tego podrozdziału dowiesz się, jak obliczyć sumę kumulacyjną, średnią kroczącą i średnią centralną.

Obliczanie sumy kumulacyjnej

Poniższe zapytanie oblicza sumę kumulacyjną wartości sprzedaży w 2003 roku — od stycznia do grudnia. Należy zauważyć, że wartość sprzedaży w każdym miesiącu jest dodawana do wartości kumulacyjnej, która zwiększa się co miesiąc:

```
SELECT
    month, SUM(amount) AS month_amount,
    SUM(SUM(amount)) OVER
        (ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
    AS cumulative_amount
FROM all_sales
WHERE year = 2003
GROUP BY month
ORDER BY month;
```

MONTH	MONTH_AMOUNT	CUMULATIVE_AMOUNT
1	95525,55	95525,55
2	116671,6	212197,15
3	160307,92	372505,07
4	175998,8	548503,87
5	154349,44	702853,31
6	124951,36	827804,67
7	170296,16	998100,83
8	212735,68	1210836,51
9	199609,68	1410446,19
10	264480,79	1674926,98
11	160221,98	1835148,96
12	137336,17	1972485,13

W tym zapytaniu do obliczenia agregatu kumulacyjnego jest wykorzystywane następujące wyrażenie:

```
SUM(SUM(amount)) OVER
    (ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
AS cumulative_amount
```

Omówimy teraz poszczególne części tego wyrażenia:

- `SUM(amount)` oblicza sumę wartości sprzedaży. Zewnętrzna funkcja `SUM()` oblicza wartość skumulowaną.
- `ORDER BY month` porządkuje według miesięcy wiersze odczytywane przez zapytanie.
- `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW` definiuje początek i koniec okna. Początek jest ustawiany przez `UNBOUNDED PRECEDING`, co oznacza, że początek okna jest ustawiony na stałe jako pierwszy wiersz w zestawie wyników zwróconym przez zapytanie. Koniec okna jest ustawiany jako `CURRENT ROW`, czyli bieżący wiersz z przetwarzanego zestawu wyników. Koniec okna zsuwa się o jeden wiersz w dół po wykonaniu obliczeń przez zewnętrzną funkcję `SUM()` i zwraca bieżącą wartość skumulowaną.

Całe zapytanie oblicza i zwraca sumę kumulacyjną wartości sprzedaży, rozpoczynając od pierwszego miesiąca i dodając wartość sprzedaży w drugim miesiącu, później w trzecim itd. aż do ostatniego miesiąca włącznie. Początek okna jest ustawiony na stałe w pierwszym miesiącu, ale okno przesuwa się w dół (po jednym wierszu zestawu wyników) po dodaniu wartości sprzedaży w danym miesiącu do sumy skumulowanej. Ten proces trwa do czasu przetworzenia przez okno i funkcję `SUM()` ostatniego wiersza zestawu wyników.

Nie należy pomylić końca okna z końcem zestawu wyników. W poprzednim przykładzie koniec okna przesuwał się w dół o jeden wiersz w zestawie wyników po przetworzeniu danego wiersza (czyli po dodaniu sumy wartości sprzedaży w danym miesiącu do sumy kumulacyjnej). W tym przykładzie koniec okna na początku znajduje się w pierwszym wierszu, wartość sumy sprzedaży w tym miesiącu jest dodawana do sumy skumulowanej, a następnie koniec okna przesuwa się w dół o jeden wiersz — do drugiego wiersza. W tym momencie okno „widzi” dwa wiersze. Suma wartości sprzedaży w tym miesiącu jest

dodawana do sumy skumulowanej i koniec okna przesuwają się w dół o jeden wiersz, do trzeciego wiersza. W tym momencie okno „widzi” trzy wiersze. Ten proces trwa aż do osiągnięcia dwunastego wiersza. Wówczas okno „widzi” dwanaście wierszy.

W poniższym przykładzie zapytanie oblicza skumulowaną wartość sprzedaży, rozpoczynając od czerwca 2003 roku (szósty miesiąc) i kończąc w grudniu 2003 (na dwunastym miesiącu):

```
SELECT
  month, SUM(amount) AS month_amount,
  SUM(SUM(amount)) OVER
    (ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS
    cumulative_amount
FROM all_sales
WHERE year = 2003
AND month BETWEEN 6 AND 12
GROUP BY month
ORDER BY month;
```

MONTH	MONTH_AMOUNT	CUMULATIVE_AMOUNT
6	124951,36	124951,36
7	170296,16	295247,52
8	212735,68	507983,2
9	199609,68	707592,88
10	264480,79	972073,67
11	160221,98	1132295,65
12	137336,17	1269631,82

Obliczanie średniej kroczącej

Poniższe zapytanie oblicza średnią krocząco wartości sprzedaży między bieżącym miesiącem i poprzednimi trzema:

```
SELECT
  month, SUM(amount) AS month_amount,
  AVG(SUM(amount)) OVER
    (ORDER BY month ROWS BETWEEN 3 PRECEDING AND CURRENT ROW)
    AS moving_average
FROM all_sales
WHERE year = 2003
GROUP BY month
ORDER BY month;
```

MONTH	MONTH_AMOUNT	MOVING_AVERAGE
1	95525,55	95525,55
2	116671,6	106098,575
3	160307,92	124168,357
4	175998,8	137125,968
5	154349,44	151831,94
6	124951,36	153901,88
7	170296,16	156398,94
8	212735,68	165583,16
9	199609,68	176898,22
10	264480,79	211780,578
11	160221,98	209262,033
12	137336,17	190412,155

Należy zauważyć, że do obliczenia średniej kroczącej jest używane poniższe wyrażenie:

```
AVG(SUM(amount)) OVER
  (ORDER BY month ROWS BETWEEN 3 PRECEDING AND CURRENT ROW)
  AS moving_average
```

Przeanalizujmy poszczególne części tego wyrażenia:

- `SUM(amount)` oblicza sumę wartości sprzedaży. Zewnętrzna funkcja `AVG()` oblicza średnią.
- `ORDER BY month` porządkuje według miesięcy wiersze odczytywane przez zapytanie.
- `ROWS BETWEEN 3 PRECEDING AND CURRENT ROW` definiuje początek okna — obejmuje on trzy wiersze poprzedzające bieżący wiersz. Końcem okna jest aktualnie przetwarzany wiersz.

Całe wyrażenie oblicza więc średnią kroczącą wartości sprzedaży między bieżącym miesiącem i poprzednimi trzema. Ponieważ w przypadku pierwszych dwóch miesięcy dostępne są dane z mniej niż trzech miesięcy, średnia krocząca jest obliczana jedynie na podstawie dostępnych informacji.

Zarówno początek, jak i koniec okna na samym początku znajdują się w pierwszym wierszu pobranym przez zapytanie. Koniec okna przesuwa się w dół po przetworzeniu każdego wiersza. Początek okna przesuwa się w dół dopiero po przetworzeniu czwartego wiersza i dalej przesuwa się w dół o jeden wiersz po przetworzeniu każdego wiersza. Ten proces trwa aż do przetworzenia ostatniego wiersza z zestawu wyników.

Obliczanie średniej centralnej

Poniższe zapytanie oblicza średnią kroczącą wartości sprzedaży między bieżącym, poprzednim i następnym miesiącem:

```
SELECT
    month, SUM(amount) AS month_amount,
    AVG(SUM(amount)) OVER
        (ORDER BY month ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
        AS moving_average
FROM all_sales
WHERE year = 2003
GROUP BY month
ORDER BY month;
```

MONTH	MONTH_AMOUNT	MOVING_AVERAGE
1	95525,55	106098,575
2	116671,6	124168,357
3	160307,92	150992,773
4	175998,8	163552,053
5	154349,44	151766,533
6	124951,36	149865,653
7	170296,16	169327,733
8	212735,68	194213,84
9	199609,68	225608,717
10	264480,79	208104,15
11	160221,98	187346,313
12	137336,17	148779,075

Do obliczenia średniej kroczącej w powyższym zapytaniu użyto następującego wyrażenia:

```
AVG(SUM(amount)) OVER
    (ORDER BY month ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
    AS moving_average
```

Składa się ono z następujących części:

- `SUM(amount)` oblicza sumę wartości sprzedaży. Zewnętrzna funkcja `AVG()` oblicza średnią.
- `ORDER BY month` porządkuje według miesięcy wiersze pobrane przez zapytanie.
- `ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING` definiuje początek okna jako zawierający wiersz poprzedzający bieżący. Koniec okna znajduje się w wierszu następującym po bieżącym.

Wyrażenie oblicza średnią kroczącą wartości sprzedaży dla bieżącego, poprzedniego i następnego miesiąca. Ponieważ w przypadku pierwszego i ostatniego miesiąca nie dysponujemy danymi dla pełnych trzech miesięcy, średnia krocząca jest obliczana jedynie na podstawie dostępnych informacji.

Początek okna znajduje się najpierw w pierwszym wierszu, odczytanym przez zapytanie. Koniec okna znajduje się wówczas w drugim wierszu i przesuwa się w dół po przetworzeniu każdego wiersza. Początek okna zaczyna przesuwać się w dół po przetworzeniu drugiego wiersza. Przetwarzanie kończy się w ostatnim wierszu odczytanym przez zapytanie.

Pobieranie pierwszego i ostatniego wiersza za pomocą funkcji FIRST_VALUE() i LAST_VALUE()

Funkcje FIRST_VALUE() i LAST_VALUE() służą do pobierania pierwszego i ostatniego wiersza z okna. W poniższym zapytaniu użyto funkcji FIRST_VALUE i LAST_VALUE do pobrania wartości sprzedaży w poprzednim i kolejnym miesiącu:

```
SELECT
  month, SUM(amount) AS month_amount,
  FIRST_VALUE(SUM(amount)) OVER
    (ORDER BY month ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
    AS previous_month_amount,
  LAST_VALUE(SUM(amount)) OVER
    (ORDER BY month ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
    AS next_month_amount
FROM all_sales
WHERE year = 2003
GROUP BY month
ORDER BY month;
```

MONTH	MONTH_AMOUNT	PREVIOUS_MONTH_AMOUNT	NEXT_MONTH_AMOUNT
1	95525,55	95525,55	116671,6
2	116671,6	95525,55	160307,92
3	160307,92	116671,6	175998,8
4	175998,8	160307,92	154349,44
5	154349,44	175998,8	124951,36
6	124951,36	154349,44	170296,16
7	170296,16	124951,36	212735,68
8	212735,68	170296,16	199609,68
9	199609,68	212735,68	264480,79
10	264480,79	199609,68	160221,98
11	160221,98	264480,79	137336,17
12	137336,17	160221,98	137336,17

W kolejnym zapytaniu wartość sprzedaży w bieżącym miesiącu jest dzielona przez wartość sprzedaży w poprzednim miesiącu (etykieta curr_div_prev) oraz przez wartość sprzedaży w następnym miesiącu (etykieta curr_div_next):

```
SELECT
  month, SUM(amount) AS month_amount,
  SUM(amount)/FIRST_VALUE(SUM(amount)) OVER
    (ORDER BY month ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
    AS curr_div_prev,
  SUM(amount)/LAST_VALUE(SUM(amount)) OVER
    (ORDER BY month ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
    AS curr_div_next
FROM all_sales
WHERE year = 2003
GROUP BY month
ORDER BY month;
```

MONTH	MONTH_AMOUNT	CURR_DIV_PREV	CURR_DIV_NEXT
1	95525,55	1	,818755807
2	116671,6	1,22136538	,727796855
3	160307,92	1,37400978	,910846665
4	175998,8	1,09787963	1,14026199

5	154349,44	,876991434	1,23527619
6	124951,36	,809535558	,733729756
7	170296,16	1,36289961	,800505867
8	212735,68	1,24921008	1,06575833
9	199609,68	,93829902	,754722791
10	264480,79	1,3249898	1,65071478
11	160221,98	,605798175	1,16664081
12	137336,17	,857161858	1

Pobieranie n-tego wiersza za pomocą funkcji NTH_VALUE()

Funkcja `NTH_VALUE()` zwraca *n*-ty wiersz z okna. Ta funkcja została wprowadzona w Oracle Database 11g Release 2. Poniższe zapytanie wykorzystuje `NTH_VALUE()` do pobrania wartości sprzedaży w drugim miesiącu analizowanego okna w konstrukcji `NTH_VALUE(SUM(amount), 2)`:

```
SELECT
  month, SUM(amount) AS month_amount,
  NTH_VALUE(SUM(amount), 2) OVER (
    ORDER BY month ROWS BETWEEN
    UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
  ) nth_value
FROM all_sales
WHERE year = 2003
GROUP BY month
ORDER BY month;
```

MONTH	MONTH_AMOUNT	NTH_VALUE
1	95525,55	116671,6
2	116671,6	116671,6
3	160307,92	116671,6
4	175998,8	116671,6
5	154349,44	116671,6
6	124951,36	116671,6
7	170296,16	116671,6
8	212735,68	116671,6
9	199609,68	116671,6
10	264480,79	116671,6
11	160221,98	116671,6
12	137336,17	116671,6

Kolejne zapytanie wykorzystuje `NTH_VALUE()` do pobrania maksymalnej sprzedaży pracownika nr 24 dla produktów typu 1, 2 i 3. Wartość ta znajduje się na 4. pozycji w oknie i jest pobierana za pomocą wyrażenia `NTH_VALUE(MAX(amount), 4)`.

```
SELECT
  prd_type_id, emp_id, MAX(amount),
  NTH_VALUE(MAX(amount), 4) OVER (
    PARTITION BY prd_type_id ORDER BY emp_id
    ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
  ) nth_value
FROM all_sales
WHERE prd_type_id BETWEEN 1 AND 3
GROUP BY prd_type_id, emp_id
ORDER BY prd_type_id, emp_id;
```

PRD_TYPE_ID	EMP_ID	MAX(AMOUNT)	NTH_VALUE
1	21	25057,45	25214,56
1	22	29057,45	25214,56
1	23	16057,45	25214,56
1	24	25214,56	25214,56
1	25	14057,45	25214,56
1	26	16754,27	25214,56
2	21	2754,19	7314,56

2	22	2657,45	7314,56
2	23	2357,45	7314,56
2	24	7314,56	7314,56
2	25	5364,84	7314,56
2	26	5434,84	7314,56
3	21	32754,19	6337,83
3	22	27264,84	6337,83
3	23	23264,84	6337,83
3	24	6337,83	6337,83
3	25	4364,64	6337,83
3	26	5457,45	6337,83

Na tym zakończymy omówienie funkcji okna.

Funkcje raportujące

Funkcje raportujące służą do wykonywania obliczeń uwzględniających różne grupy i partycje wewnątrz nich.

Raportowanie może być wykonywane z użyciem następujących funkcji: SUM(), AVG(), MAX(), MIN(), COUNT(), VARIANCE() i STDDEV(). Można ponadto użyć funkcji RATIO_TO_REPORT() do obliczenia stosunku wartości do sumy wartości z zestawu oraz funkcji LISTAGG() do uporządkowania wierszy w grupie i połączenia zestawu grupowanych wartości.

Z tego podrozdziału dowiesz się, jak raportować sumy i stosować funkcje RATIO_TO_REPORT() oraz LISTAGG().

Raportowanie sumy

Poniższe zapytanie raportuje dla pierwszych trzech miesięcy 2003 roku następujące elementy:

- całkowitą sumę wszystkich wartości sprzedaży we wszystkich trzech miesiącach (etykieta total_month_amount),
- całkowitą sumę wszystkich wartości sprzedaży dla wszystkich typów produktów (etykieta total_product_type_amount).

```
SELECT
  month, prd_type_id,
  SUM(SUM(amount)) OVER (PARTITION BY month)
    AS total_month_amount,
  SUM(SUM(amount)) OVER (PARTITION BY prd_type_id)
    AS total_product_type_amount
FROM all_sales
WHERE year = 2003
AND month <= 3
GROUP BY month, prd_type_id
ORDER BY month, prd_type_id;
```

MONTH	PRD_TYPE_ID	TOTAL_MONTH_AMOUNT	TOTAL_PRODUCT_TYPE_AMOUNT
1	1	95525,55	201303,92
1	2	95525,55	44503,92
1	3	95525,55	61003,92
1	4	95525,55	65693,31
1	5	95525,55	
2	1	116671,6	201303,92
2	2	116671,6	44503,92
2	3	116671,6	61003,92
2	4	116671,6	65693,31
2	5	116671,6	
3	1	160307,92	201303,92
3	2	160307,92	44503,92
3	3	160307,92	61003,92
3	4	160307,92	65693,31
3	5	160307,92	

Do raportowania całkowitej sumy sprzedaży we wszystkich trzech miesiącach (etykieta `total_month_amount`) użyto wyrażenia:

```
SUM(SUM(amount)) OVER (PARTITION BY month)
AS total_month_amount
```

Składa się ono z następujących części:

- `SUM(amount)` oblicza średnią wartość sprzedaży. Zewnętrzna funkcja `SUM()` oblicza sumę całkowitą.
- `OVER (PARTITION BY month)` powoduje, że zewnętrzna funkcja `SUM()` oblicza sumę dla każdego miesiąca.

Do raportowania całkowitej sumy sprzedaży wszystkich rodzajów produktów (etykieta `total_product_type_amount`) w powyższym zapytaniu użyto następującego wyrażenia:

```
SUM(SUM(amount)) OVER (PARTITION BY prd_type_id)
AS total_product_type_amount
```

Składają się na nie następujące elementy:

- `SUM(amount)` oblicza średnią wartość sprzedaży. Zewnętrzna funkcja `SUM()` oblicza sumę całkowitą.
- `OVER (PARTITION BY prd_type_id)` powoduje, że zewnętrzna funkcja `SUM()` oblicza sumę dla każdego typu produktu.

Użycie funkcji `RATIO_TO_REPORT()`

Funkcja `RATIO_TO_REPORT()` służy do obliczania stosunku wartości do sumy zestawu wartości.

Poniższe zapytanie raportuje dla pierwszych trzech miesięcy 2003 roku następujące informacje:

- sumę wartości sprzedaży według typów produktu w każdym miesiącu (etykieta `prd_type_amount`),
- stosunek wartości sprzedaży danego rodzaju produktu do całkowitej sprzedaży w miesiącu (etykieta `prd_type_ratio`). Ta wartość jest obliczana za pomocą funkcji `RATIO_TO_REPORT()`.

```
SELECT
    month, prd_type_id,
    SUM(amount) AS prd_type_amount,
    RATIO_TO_REPORT(SUM(amount)) OVER (PARTITION BY month) AS prd_type_ratio
FROM all_sales
WHERE year = 2003
AND month <= 3
GROUP BY month, prd_type_id
ORDER BY month, prd_type_id;
```

MONTH	PRD_TYPE_ID	PRD_TYPE_AMOUNT	PRD_TYPE_RATIO
1	1	38909,04	,40731553
1	2	14309,04	,149792804
1	3	24909,04	,260757881
1	4	17398,43	,182133785
1	5		
2	1	70567,9	,604842138
2	2	13367,9	,114577155
2	3	15467,9	,132576394
2	4	17267,9	,148004313
2	5		
3	1	91826,98	,57281624
3	2	16826,98	,104966617
3	3	20626,98	,128670998
3	4	31026,98	,193546145
3	5		

Do obliczenia wspomnianego stosunku (etykieta `prd_type_ratio`) użyto następującego wyrażenia:

```
RATIO_TO_REPORT(SUM(amount)) OVER (PARTITION BY month) AS prd_type_ratio
```

Jego składowe to:

- SUM(amount) oblicza sumę wartości sprzedaży.
- OVER (PARTITION BY month) powoduje, że zewnętrzna funkcja SUM() oblicza sumę wartości sprzedaży w każdym miesiącu,
- stosunek jest obliczany przez podzielenie sumy wartości sprzedaży poszczególnych typów produktów przez całkowitą wartość (sumę) sprzedaży w danym miesiącu.

Użycie funkcji LISTAGG()

Funkcja LISTAGG() porządkuje wiersze w grupie i łączy zgrupowane wartości. Ta funkcja została wprowadzona w Oracle Database 11g Release 2. Poniższe zapytanie pobiera produkty od 1 do 5 z tabeli products uporządkowane według ceny i nazwy, a następnie zwraca produkty najdroższe:

```
SELECT
  LISTAGG(name, ', ') WITHIN GROUP (ORDER BY price, name) AS "Lista produktów",
  MAX(price) AS "Najdroższy"
FROM products
WHERE product_id <= 5;
```

Lista produktów	Najdroższy

Wojny czołgów, Nauka współczesna, Supernowa, Chemia, Z Files 49,99	

Następne zapytanie pobiera produkty od 1 do 5 z tabeli products i dla każdego produktu używa LISTAGG(), by wyświetlić produkty z taką samą wartością product_type_id:

```
SELECT
  product_id, product_type_id, name,
  LISTAGG(name, ', ')
    WITHIN GROUP (ORDER BY name)
    OVER (PARTITION BY product_type_id) AS "Product List"
FROM products
WHERE product_id <= 5
ORDER BY product_id, product_type_id;
```

PRODUCT_ID	PRODUCT_TYPE_ID	NAME	Product List

1	1	Nauka współczesna	Chemia, Nauka współczesna
2	1	Chemia	Chemia, Nauka współczesna
3	2	Supernowa	Supernowa, Wojny czołgów, Z Files
4	2	Wojny czołgów	Supernowa, Wojny czołgów, Z Files
5	2	Z Files	Supernowa, Wojny czołgów, Z Files

Na tym zakończymy omówienie funkcji raportujących.

Użycie funkcji LAG() i LEAD()

Funkcje LAG() i LEAD() służą do pobierania wartości z wiersza, który jest oddalony o określoną liczbę wierszy od bieżącego. W poniższym zapytaniu użyto tych funkcji do pobrania wartości sprzedaży w poprzednim i kolejnym miesiącu w stosunku do bieżącego:

```
SELECT
  month, SUM(amount) AS month_amount,
  LAG(SUM(amount), 1) OVER (ORDER BY month) AS previous_month_amount,
  LEAD(SUM(amount), 1) OVER (ORDER BY month) AS next_month_amount
FROM all_sales
WHERE year = 2003
GROUP BY month
ORDER BY month;
```

MONTH	MONTH_AMOUNT	PREVIOUS_MONTH_AMOUNT	NEXT_MONTH_AMOUNT

1	95525,55		116671,6
2	116671,6	95525,55	160307,92
3	160307,92	116671,6	175998,8

4	175998,8	160307,92	154349,44
5	154349,44	175998,8	124951,36
6	124951,36	154349,44	170296,16
7	170296,16	124951,36	212735,68
8	212735,68	170296,16	199609,68
9	199609,68	212735,68	264480,79
10	264480,79	199609,68	160221,98
11	160221,98	264480,79	137336,17
12	137336,17	160221,98	

Wartość sprzedaży w tych okresach jest pobierana za pomocą następujących wyrażeń:

```
LAG(SUM(amount), 1) OVER (ORDER BY month) AS previous_month_amount,
LEAD(SUM(amount), 1) OVER (ORDER BY month) AS next_month_amount
```

LAG(SUM(amount), 1) pobiera sumę wartości z poprzedniego wiersza, a LEAD(SUM(amount), 1) — z następnego.

Użycie funkcji FIRST i LAST

Funkcje FIRST i LAST pobierają pierwszą i ostatnią wartość w uporządkowanej grupie. Mogą zostać użyte z następującymi funkcjami: SUM(), AVG(), MAX(), MIN(), COUNT(), STDDEV() i VARIANCE().

W poniższym zapytaniu użyto funkcji FIRST i LAST do pobrania danych o tym, w których miesiącach 2003 roku sprzedaż była najwyższa i najniższa:

```
SELECT
  MIN(month) KEEP (DENSE_RANK FIRST ORDER BY SUM(amount))
  AS highest_sales_month,
  MIN(month) KEEP (DENSE_RANK LAST ORDER BY SUM(amount))
  AS lowest_sales_month
FROM all_sales
WHERE year = 2003
GROUP BY month
ORDER BY month;
```

```
HIGHEST_SALES_MONTH  LOWEST_SALES_MONTH
-----
1                      10
```

Użycie funkcji regresji liniowej

Funkcje regresji liniowej służą do dopasowania metodą najmniejszych kwadratów linii regresji do zestawu par liczb. Mogą być używane jako funkcje agregujące, funkcje okien lub raportujące.

Funkcje regresji liniowej zostały opisane w tabeli 8.2. W składni funkcji *y* jest interpretowane przez funkcję jak zmienna zależna od *x*.

Tabela 8.2. Funkcje regresji liniowej

Funkcja	Opis
REGR_AVGX(<i>y</i> , <i>x</i>)	Zwraca średnią <i>x</i> po usunięciu par <i>x</i> i <i>y</i> , w których jedna (lub obie) wartości to NULL
REGR_AVGY(<i>y</i> , <i>x</i>)	Zwraca średnią <i>y</i> po usunięciu par <i>x</i> i <i>y</i> , w których jedna (lub obie) wartości to NULL
REGR_COUNT(<i>y</i> , <i>x</i>)	Zwraca liczbę par liczbowych, nierównych NULL, które są używane do dopasowania linii regresji
REGR_INTERCEPT(<i>y</i> , <i>x</i>)	Zwraca punkt przecięcia się linii regresji z osią <i>Y</i>
REGR_R2(<i>y</i> , <i>x</i>)	Zwraca współczynnik determinacji (R kwadrat) linii regresji
REGR_SLOPE(<i>y</i> , <i>x</i>)	Zwraca nachylenie linii regresji
REGR_SXX(<i>y</i> , <i>x</i>)	Zwraca REG_COUNT (<i>y</i> , <i>x</i>) * VAR_POP(<i>x</i>)
REGR_SXY(<i>y</i> , <i>x</i>)	Zwraca REG_COUNT (<i>y</i> , <i>x</i>) * COVAR_POP(<i>y</i> , <i>x</i>)
REGR_SYY(<i>y</i> , <i>x</i>)	Zwraca REG_COUNT (<i>y</i> , <i>x</i>) * VAR_POP (<i>y</i>)

Poniższe zapytanie obrazuje użycie funkcji regresji liniowej:

```
SELECT
  prd_type id,
  REGR_AVGX(amount, month) AS avgx,
  REGR_AVGY(amount, month) AS avgy,
  REGR_COUNT(amount, month) AS count,
  REGR_INTERCEPT(amount, month) AS inter,
  REGR_R2(amount, month) AS r2,
  REGR_SLOPE(amount, month) AS slope,
  REGR_SXX(amount, month) AS sxx,
  REGR_SXY(amount, month) AS sxy,
  REGR_SYY(amount, month) AS syy
FROM all_sales
WHERE year = 2003
GROUP BY prd_type_id;
```

PRD_TYPE_ID	AVGX	AVGY	COUNT	INTER	R2
SLOPE	SXX	SXY	SYY		
1	6,5	12570,5811	72	13318,4543	,003746289
-115,05741	858	-98719,26	3031902717		
2	6,5	2588,62806	72	2608,11268	,0000508
-2,997634	858	-2571,97	151767392		
3	6,5	6642,65153	72	2154,23119	,126338815
690,526206	858	592471,485	3238253324		
4	6,5	5593,76611	72	2043,47164	,128930297
546,199149	858	468638,87	1985337488		
5					0

Użycie funkcji hipotetycznego rankingu i rozkładu

Funkcje hipotetycznego rankingu i rozkładu służą do obliczania pozycji i percentylu, które zajęłby nowy wiersz po wstawieniu go do tabeli. Obliczenia hipotetyczne można wykonywać za pomocą następujących funkcji: RANK(), DENSE_RANK(), PERCENT_RANK() i CUME_DIST().

W poniższym zapytaniu użyto funkcji RANK() i PERCENT_RANK() do pobrania pozycji i pozycji procentowej wartości sprzedaży według rodzajów produktów w 2003 roku:

```
SELECT
  prd_type id, SUM(amount),
  RANK() OVER (ORDER BY SUM(amount) DESC) AS rank,
  PERCENT_RANK() OVER (ORDER BY SUM(amount) DESC) AS percent_rank
FROM all_sales
WHERE year = 2003
AND amount IS NOT NULL
GROUP BY prd_type_id
ORDER BY prd_type_id;
```

PRD_TYPE_ID	SUM(AMOUNT)	RANK	PERCENT_RANK
1	905081,84	1	0
2	186381,22	4	1
3	478270,91	2	,333333333
4	402751,16	3	,666666667

Kolejne zapytanie oblicza hipotetyczną pozycję i pozycję procentową wartości sprzedaży wynoszącej 500 000 zł:


```

SELECT
  RANK(500000) WITHIN GROUP (ORDER BY SUM(amount) DESC)
    AS rank,
  PERCENT_RANK(500000) WITHIN GROUP (ORDER BY SUM(amount) DESC)
    AS percent_rank
FROM all_sales
WHERE year = 2003
AND amount IS NOT NULL
GROUP BY prd_type_id
ORDER BY prd_type_id;

```

```

      RANK PERCENT_RANK
-----
      2          ,25

```

Hipotetyczna pozycja i pozycja procentowa wartości sprzedaży równej 500 000 zł wynosi odpowiednio 2 i 0,25.

Na tym zakończymy omówienie funkcji obliczających wartości hipotetyczne.

Użycie klauzuli MODEL

Klauzula MODEL została wprowadzona w Oracle Database 10g i umożliwia wykonywanie obliczeń międzywierszowych. Dzięki niej możliwe jest uzyskanie dostępu do kolumny w wierszu w taki sposób, jakby była to komórka w tablicy. Możemy więc wykonywać obliczenia w sposób przypominający pracę w arkuszu kalkulacyjnym. Na przykład tabela `all_sales` zawiera informacje o sprzedaży w miesiącach 2003 roku. Za pomocą klauzuli MODEL na podstawie wartości sprzedaży w 2003 roku możemy obliczyć wartość sprzedaży w przyszłych miesiącach.

Przykład zastosowania klauzuli MODEL

Sposób działania klauzuli MODEL najłatwiej przedstawić na przykładzie. Poniższe zapytanie pobiera wartości sprzedaży w poszczególnych miesiącach 2003 roku, uzyskane przez pracownika nr 21 dla produktów 1. i 2. rodzaju, a następnie oblicza przewidywaną sprzedaż w styczniu, lutym i marcu 2004 roku na podstawie sprzedaży w 2003 roku:

```

SELECT prd_type_id, year, month, sales_amount
FROM all_sales
WHERE prd_type_id BETWEEN 1 AND 2
AND emp_id = 21
MODEL
PARTITION BY (prd_type_id)
DIMENSION BY (month, year)
MEASURES (amount sales_amount) (
  sales_amount[1, 2004] = sales_amount[1, 2003],
  sales_amount[2, 2004] = sales_amount[2, 2003] + sales_amount[3, 2003],
  sales_amount[3, 2004] = ROUND(sales_amount[3, 2003] * 1.25, 2)
)
ORDER BY prd_type_id, year, month;

```

Zapytanie składa się z następujących elementów:

- `PARTITION BY (prd_type_id)` określa, że wyniki są partycjonowane według `prd_type_id`.
- `DIMENSION BY (month, year)` określa, że wymiarami tablicy są `month` i `year`. To oznacza, że dostęp do komórki tablicy uzyskujemy, podając miesiąc i rok.
- `MEASURES (amount sales_amount)` określa, że każda komórka tablicy zawiera kwotę sprzedaży i że nazwa tablicy to `sales_amount`. Aby wydobyć z tablicy `sales_amount` wartość dla stycznia 2003 roku, używamy `sales_amount[1, 2003]`.
- Po klauzuli `MEASURES` znajdują się trzy wiersze, które obliczają przyszłe kwoty sprzedaży w styczniu, lutym i marcu 2004 roku:

- `sales_amount[1, 2004] = sales_amount[1, 2003]` ustawia jako kwotę sprzedaży w styczniu 2004 roku kwotę sprzedaży w styczniu 2003 roku.
- `sales_amount[2, 2004] = sales_amount[2, 2003] + sales_amount[3, 2003]` ustawia jako kwotę sprzedaży w lutym 2004 roku sumę kwot sprzedaży w lutym i marcu 2003 roku.
- `sales_amount[3, 2004] = ROUND(sales_amount[3, 2003] * 1.25, 2)` ustawia kwotę sprzedaży w marcu 2004 roku jako zaokrąglony iloczyn kwoty sprzedaży w marcu 2003 i liczby 1,25.
- `ORDER BY prd_type_id, year, month` po prostu ustala kolejność wyników zwróconych przez całe zapytanie.

Wynik tego zapytania został przedstawiony poniżej. Należy zauważyć, że zawiera on kwoty sprzedaży typów produktów nr 1 i 2 we wszystkich miesiącach 2003 roku oraz przewidywane kwoty sprzedaży w pierwszych trzech miesiącach 2004 roku (zostały one wyróżnione):

PRD_TYPE_ID	YEAR	MONTH	SALES_AMOUNT
1	2003	1	10034,84
1	2003	2	15144,65
1	2003	3	20137,83
1	2003	4	25057,45
1	2003	5	17214,56
1	2003	6	15564,64
1	2003	7	12654,84
1	2003	8	17434,82
1	2003	9	19854,57
1	2003	10	21754,19
1	2003	11	13029,73
1	2003	12	10034,84
1	2004	1	10034,84
1	2004	2	35282,48
1	2004	3	25172,29
2	2003	1	1034,84
2	2003	2	1544,65
2	2003	3	2037,83
2	2003	4	2557,45
2	2003	5	1714,56
2	2003	6	1564,64
2	2003	7	1264,84
2	2003	8	1734,82
2	2003	9	1854,57
2	2003	10	2754,19
2	2003	11	1329,73
2	2003	12	1034,84
2	2004	1	1034,84
2	2004	2	3582,48
2	2004	3	2547,29

Dostęp do komórek za pomocą zapisu pozycyjnego i symbolicznego

W poprzednim przykładzie uzyskaliśmy dostęp do komórki tablicy za pomocą zapisu: `sales_amount[1, 2004]`, w którym 1 oznacza miesiąc, a 2004 — rok. Jest to zapis pozycyjny, ponieważ znaczenie wymiarów jest określone przez ich pozycję: pierwsza zawiera miesiąc, a druga rok.

Do jawnego określenia znaczenia wymiarów możemy użyć zapisu symbolicznego, na przykład `sales_amount[month=1, year=2004]`. Poniższe zapytanie jest zmodyfikowaną wersją wcześniejszego — użyto w nim zapisu symbolicznego:

```
SELECT prd_type_id, year, month, sales_amount
FROM all_sales
WHERE prd_type_id BETWEEN 1 AND 2
```

```

AND emp_id = 21
MODEL
PARTITION BY (prd_type_id)
DIMENSION BY (month, year)
MEASURES (amount sales_amount) (
  sales_amount[month=1, year=2004] = sales_amount[month=1, year=2003],
  sales_amount[month=2, year=2004] =
    sales_amount[month=2, year=2003] + sales_amount[month=3, year=2003],
  sales_amount[month=3, year=2004] =
    ROUND(sales_amount[month=3, year=2003] * 1.25, 2)
)
ORDER BY prd_type_id, year, month;

```

Gdy używa się zapisu symbolicznego lub pozycyjnego, należy zdawać sobie sprawę, że w różny sposób traktują one wartości NULL w wymiarach. Na przykład `sales_amount[null, 2003]` zwraca kwotę, dla której miesiąc to NULL, a rok to 2003, a `sales_amount[month=null, year=2004]` nie uzyska dostępu do prawidłowej komórki, ponieważ `null=null` zawsze zwraca fałsz.

Uzyskiwanie dostępu do zakresu komórek za pomocą BETWEEN i AND

Za pomocą słów kluczowych BETWEEN i AND możemy uzyskać dostęp do zakresu komórek. Poniższe wyrażenie ustawia kwotę sprzedaży w styczniu 2004 roku jako zaokrągloną średnią kwot sprzedaży między styczniem a marcem 2003 roku:

```

sales_amount[1, 2004] =
  ROUND(AVG(sales_amount)[month BETWEEN 1 AND 3, 2003], 2)

```

Poniższe zapytanie obrazuje użycie tego wyrażenia:

```

SELECT prd_type_id, year, month, sales_amount
FROM all_sales
WHERE prd_type_id BETWEEN 1 AND 2
AND emp_id = 21
MODEL
PARTITION BY (prd_type_id)
DIMENSION BY (month, year)
MEASURES (amount sales_amount) (
  sales_amount[1, 2004] =
    ROUND(AVG(sales_amount)[month BETWEEN 1 AND 3, 2003], 2)
)
ORDER BY prd_type_id, year, month;

```

Sięganie do wszystkich komórek za pomocą ANY i IS ANY

Za pomocą predykatów ANY i IS ANY możemy uzyskać dostęp do wszystkich komórek tablicy. Predykat ANY jest używany z zapisem pozycyjnym, a IS ANY stosuje się z zapisem symbolicznym. Poniższe wyrażenie ustawia kwotę sprzedaży w styczniu 2004 roku jako zaokrągloną sumę kwot sprzedaży we wszystkich miesiącach i latach:

```

sales_amount[1, 2004] =
  ROUND(SUM(sales_amount)[ANY, year IS ANY], 2)

```

Poniższe zapytanie obrazuje użycie tego wyrażenia:

```

SELECT prd_type_id, year, month, sales_amount
FROM all_sales
WHERE prd_type_id BETWEEN 1 AND 2
AND emp_id = 21
MODEL
PARTITION BY (prd_type_id)

```

```
DIMENSION BY (month, year)
MEASURES (amount sales_amount) (
  sales_amount[1, 2004] =
    ROUND(SUM(sales_amount)[ANY, year IS ANY], 2)
)
ORDER BY prd_type_id, year, month;
```

Pobieranie bieżącej wartości wymiaru za pomocą funkcji CURRENTV()

Za pomocą funkcji CURRENTV() możemy pobrać bieżącą wartość wymiaru. Na przykład poniższe wyrażenie ustawia kwotę sprzedaży w pierwszym miesiącu 2004 roku jako iloczyn liczby 1,25 i kwoty sprzedaży w takim samym miesiącu 2003 roku. Funkcję CURRENTV() zastosowano do pobrania bieżącego miesiąca, czyli 1:

```
sales_amount[1, 2004] =
  ROUND(sales_amount[CURRENTV(), 2003] * 1.25, 2)

Poniższe zapytanie obrazuje użycie tego wyrażenia:
```

```
SELECT prd_type_id, year, month, sales_amount
FROM all_sales
WHERE prd_type_id BETWEEN 1 AND 2
AND emp_id = 21
MODEL
PARTITION BY (prd_type_id)
DIMENSION BY (month, year)
MEASURES (amount sales_amount) (
  sales_amount[1, 2004] =
    ROUND(sales_amount[CURRENTV(), 2003] * 1.25, 2)
)
ORDER BY prd_type_id, year, month;
```

Poniżej zostały przedstawione wyniki tego zapytania — wartości dla 2004 roku zostały pogrubione:

PRD_TYPE_ID	YEAR	MONTH	SALES_AMOUNT
1	2003	1	10034,84
1	2003	2	15144,65
1	2003	3	20137,83
1	2003	4	25057,45
1	2003	5	17214,56
1	2003	6	15564,64
1	2003	7	12654,84
1	2003	8	17434,82
1	2003	9	19854,57
1	2003	10	21754,19
1	2003	11	13029,73
1	2003	12	10034,84
1	2004	1	12543,55
2	2003	1	1034,84
2	2003	2	1544,65
2	2003	3	2037,83
2	2003	4	2557,45
2	2003	5	1714,56
2	2003	6	1564,64
2	2003	7	1264,84
2	2003	8	1734,82
2	2003	9	1854,57
2	2003	10	2754,19
2	2003	11	1329,73
2	2003	12	1034,84
2	2004	1	1293,55

Uzyskiwanie dostępu do komórek za pomocą pętli FOR

Dostęp do komórek możemy również uzyskać za pomocą pętli FOR. Na przykład poniższe wyrażenie ustawia kwotę sprzedaży dla pierwszych trzech miesięcy 2004 roku jako iloczyn liczby 1,25 i kwot sprzedaży w odpowiednich miesiącach 2003 roku. Użyto tutaj słowa kluczowego INCREMENT określającego, o ile zostanie zwiększona wartość month przy każdej iteracji pętli:

```
sales_amount[FOR month FROM 1 TO 3 INCREMENT 1, 2004] =  
  ROUND(sales_amount[CURRENTV(), 2003] * 1.25, 2)
```

W poniższym zapytaniu zastosowano to wyrażenie:

```
SELECT prd_type_id, year, month, sales_amount  
FROM all_sales  
WHERE prd_type_id BETWEEN 1 AND 2  
AND emp_id = 21  
MODEL  
PARTITION BY (prd_type_id)  
DIMENSION BY (month, year)  
MEASURES (amount sales_amount) (  
  sales_amount[FOR month FROM 1 TO 3 INCREMENT 1, 2004] =  
    ROUND(sales_amount[CURRENTV(), 2003] * 1.25, 2)  
)  
ORDER BY prd_type_id, year, month;
```

Oto wyniki tego zapytania — wartości dla 2004 roku zostały pogrubione:

PRD_TYPE_ID	YEAR	MONTH	SALES_AMOUNT
1	2003	1	10034,84
1	2003	2	15144,65
1	2003	3	20137,83
1	2003	4	25057,45
1	2003	5	17214,56
1	2003	6	15564,64
1	2003	7	12654,84
1	2003	8	17434,82
1	2003	9	19854,57
1	2003	10	21754,19
1	2003	11	13029,73
1	2003	12	10034,84
1	2004	1	12543,55
1	2004	2	18930,81
1	2004	3	25172,29
2	2003	1	1034,84
2	2003	2	1544,65
2	2003	3	2037,83
2	2003	4	2557,45
2	2003	5	1714,56
2	2003	6	1564,64
2	2003	7	1264,84
2	2003	8	1734,82
2	2003	9	1854,57
2	2003	10	2754,19
2	2003	11	1329,73
2	2003	12	1034,84
2	2004	1	1293,55
2	2004	2	1930,81
2	2004	3	2547,29

Obsługa wartości NULL i brakujących

W tym podrozdziale opisano, jak za pomocą klauzuli MODEL obsłużyć brakujące wartości oraz wartości NULL.

Użycie IS PRESENT

IS PRESENT zwraca wartość prawda, jeżeli wiersz określany przez odwołanie komórki istniał przed wykonaniem klauzuli MODEL. Na przykład:

```
sales_amount[CURRENTV(), 2003] IS PRESENT
```

zwróci prawdę, jeśli sales_amount[CURRENTV(), 2003] istnieje.

Poniższe wyrażenie ustawia kwotę sprzedaży w pierwszych trzech miesiącach 2004 roku jako iloczyn liczby 1,25 i kwoty sprzedaży w tych samych miesiącach 2003 roku:

```
sales_amount[FOR month FROM 1 TO 3 INCREMENT 1, 2004] =
CASE WHEN sales_amount[CURRENTV(), 2003] IS PRESENT THEN
ROUND(sales_amount[CURRENTV(), 2003] * 1.25, 2)
ELSE
0
END
```

Poniższe zapytanie przedstawia zastosowanie tego wyrażenia:

```
SELECT prd_type_id, year, month, sales_amount
FROM all_sales
WHERE prd_type_id BETWEEN 1 AND 2
AND emp_id = 21
MODEL
PARTITION BY (prd_type_id)
DIMENSION BY (month, year)
MEASURES (amount sales_amount) (
sales_amount[FOR month FROM 1 TO 3 INCREMENT 1, 2004] =
CASE WHEN sales_amount[CURRENTV(), 2003] IS PRESENT THEN
ROUND(sales_amount[CURRENTV(), 2003] * 1.25, 2)
ELSE
0
END
)
ORDER BY prd_type_id, year, month;
```

Wynik tego zapytania jest taki sam jak w przykładzie z poprzedniego podrozdziału.

Użycie PRESENTV()

PRESENTV(*komórka*, *wyr1*, *wyr2*) zwraca wyrażenie *wyr1*, jeżeli wiersz określany przez odwołanie *komórka* istniał przed uruchomieniem klauzuli MODEL. Jeśli wiersz nie istnieje, jest zwracane wyrażenie *wyr2*. Na przykład:

```
PRESENTV(sales_amount[CURRENTV(), 2003],
ROUND(sales_amount[CURRENTV(), 2003] * 1.25, 2), 0)
```

zwróci zaokrąglone kwoty sprzedaży, jeżeli sales_amount[CURRENTV(), 2003] istnieje; w przeciwnym razie zostanie zwrócone 0.

Poniższe zapytanie stanowi przykład użycia tego wyrażenia:

```
SELECT prd_type_id, year, month, sales_amount
FROM all_sales
WHERE prd_type_id BETWEEN 1 AND 2
AND emp_id = 21
MODEL
PARTITION BY (prd_type_id)
DIMENSION BY (month, year)
MEASURES (amount sales_amount) (
sales_amount[FOR month FROM 1 TO 3 INCREMENT 1, 2004] =
PRESENTV(sales_amount[CURRENTV(), 2003],
ROUND(sales_amount[CURRENTV(), 2003] * 1.25, 2), 0)
)
ORDER BY prd_type_id, year, month;
```

Użycie PRESENTNNV()

PRESENTNNV(*komórka*, *wyr1*, *wyr2*) zwraca wyrażenie *wyr1*, jeżeli wiersz określany przez odwołanie *komórka* istniał przed wykonaniem klauzuli MODEL, a komórka ma inną wartość niż NULL. Jeśli wiersz nie istnieje lub komórka ma wartość NULL, jest zwracane wyrażenie *wyr2*. Na przykład:

```
PRESENTNNV(sales_amount[CURRENTV()], 2003],  
ROUND(sales_amount[CURRENTV()], 2003] * 1.25, 2), 0)
```

zwróci zaokrąglone kwoty sprzedaży, jeżeli sales_amount[CURRENTV()], 2003] istnieje i jest różne od NULL; w przeciwnym razie zostanie zwrócone 0.

Użycie IGNORE NAV i KEEP NAV

Domyślnie klauzula MODEL traktuje komórkę nie posiadającą wartości tak, jakby miała wartość NULL. Komórka z wartością NULL jest traktowana w ten sam sposób. Można zmienić to domyślne zachowanie, używając IGNORE NAV, które zwraca jedną z poniższych wartości:

- 0 dla brakujących lub równych NULL wartości liczbowych,
- pusty napis dla napisów brakujących lub z wartością NULL,
- 00/01/01 dla brakujących lub równych NULL wartości dat,
- NULL dla brakujących lub równych NULL wartości pozostałych typów danych.

Można też jawnie użyć KEEP NAV, które jest wykorzystywane domyślnie. KEEP NAV zwraca NULL dla brakujących wartości liczbowych lub tych równych NULL.

Poniższe zapytanie obrazuje zastosowanie IGNORE NAV:

```
SELECT prd_type_id, year, month, sales_amount  
FROM all_sales  
WHERE prd_type_id BETWEEN 1 AND 2  
AND emp_id = 21  
MODEL IGNORE NAV  
PARTITION BY (prd_type_id)  
DIMENSION BY (month, year)  
MEASURES (amount sales_amount) (  
  sales_amount[FOR month FROM 1 TO 3 INCREMENT 1, 2004] =  
    ROUND(sales_amount[CURRENTV()], 2003] * 1.25, 2)  
)  
ORDER BY prd_type_id, year, month;
```

Modyfikowanie istniejących komórek

Domyślnie, jeżeli komórka, do której następuje odwołanie po lewej stronie wyrażenia, istnieje, jest modyfikowana. Jeśli nie istnieje, w tablicy tworzony jest nowy wiersz. Za pomocą RULES UPDATE można zmienić to domyślne zachowanie tak, że nowy wiersz nie będzie tworzony.

Poniższe zapytanie jest przykładem zastosowania RULES UPDATE:

```
SELECT prd_type_id, year, month, sales_amount  
FROM all_sales  
WHERE prd_type_id BETWEEN 1 AND 2  
AND emp_id = 21  
MODEL  
PARTITION BY (prd_type_id)  
DIMENSION BY (month, year)  
MEASURES (amount sales_amount)  
RULES UPDATE (  
  sales_amount[FOR month FROM 1 TO 3 INCREMENT 1, 2004] =  
    ROUND(sales_amount[CURRENTV()], 2003] * 1.25, 2)  
)  
ORDER BY prd_type_id, year, month;
```

Ponieważ komórki dla 2004 roku nie istnieją, a użyto `RULES UPDATE`, w tablicy nie zostaną utworzone nowe wiersze dla tego roku, dlatego zapytanie nie zwróci dla niego wierszy. Poniżej przedstawiono wyniki tego zapytania — brakuje w nich wierszy dla 2004 roku:

PRD_TYPE_ID	YEAR	MONTH	SALES_AMOUNT
1	2003	1	10034,84
1	2003	2	15144,65
1	2003	3	20137,83
1	2003	4	25057,45
1	2003	5	17214,56
1	2003	6	15564,64
1	2003	7	12654,84
1	2003	8	17434,82
1	2003	9	19854,57
1	2003	10	21754,19
1	2003	11	13029,73
1	2003	12	10034,84
2	2003	1	1034,84
2	2003	2	1544,65
2	2003	3	2037,83
2	2003	4	2557,45
2	2003	5	1714,56
2	2003	6	1564,64
2	2003	7	1264,84
2	2003	8	1734,82
2	2003	9	1854,57
2	2003	10	2754,19
2	2003	11	1329,73
2	2003	12	1034,84

Użycie klauzul PIVOT i UNPIVOT

Klauzula `PIVOT` została wprowadzona w Oracle Database 11g i umożliwia przestawienie w wynikach zapytania wierszy na miejsce kolumn i jednocześnie wykonanie funkcji agregujących na danych. W Oracle Database 11g jest również dostępna klauzula `UNPIVOT`, która w wynikach zapytania przestawia kolumny w miejsce wierszy.

Klauzule `PIVOT` i `UNPIVOT` przydają się, jeżeli chcemy zobaczyć ogólne trendy w dużej liczbie danych, na przykład trendy sprzedaży w czasie.

Prosty przykład klauzuli PIVOT

Sposób użycia klauzuli `PIVOT` najłatwiej przedstawić na przykładzie. Poniższe zapytanie przedstawia całkowitą kwotę sprzedaży produktów pierwszego, drugiego i trzeciego typu w pierwszych czterech miesiącach 2003 roku. Należy zauważyć, że komórki w wynikach zapytania zawierają sumy kwot sprzedaży każdego produktu w każdym miesiącu:

```
SELECT *
FROM (
  SELECT month, prd_type_id, amount
  FROM all_sales
  WHERE year = 2003
  AND prd_type_id IN (1, 2, 3)
)
PIVOT (
  SUM(amount) FOR month IN (1 AS STY, 2 AS LUT, 3 AS MAR, 4 AS KWI)
)
ORDER BY prd_type_id;
```

PRD_TYPE_ID	STY	LUT	MAR	KWI
-------------	-----	-----	-----	-----

1	38909,04	70567,9	91826,98	120344,7
2	14309,04	13367,9	16826,98	15664,7
3	24909,04	15467,9	20626,98	23844,7

Rozpoczynając od pierwszej wiersza wyników, widzimy, że:

- w styczniu sprzedano produktów pierwszego typu za kwotę 38 909,04 zł,
- w lutym sprzedano produktów tego typu za kwotę 70 567,9 zł,
- ... itd. dla całego wiersza.

W drugim wierszu wyników widać, że:

- w styczniu sprzedano produktów drugiego typu za kwotę 14 309,04 zł,
- w lutym sprzedano produktów tego typu za kwotę 13 367,9 zł,
- ... itd. dla pozostałych wyników.



Uwaga

Klauzula PIVOT jest potężnym narzędziem, które umożliwia przedstawienie trendów sprzedaży różnego rodzaju produktów w poszczególnych miesiącach. Na podstawie takich danych w prawdziwym sklepie można podjąć decyzję o zmianie strategii sprzedażowej lub o przygotowaniu nowych kampanii marketingowych.

Poprzednia instrukcja SELECT ma następującą strukturę:

```
SELECT *
FROM (
    zapytanie_wewnetrzne
)
PIVOT (
    funkcja_agregujaca FOR kolumna_obrotu IN (lista_wartosci)
)
ORDER BY ...;
```

Rozłożmy poprzednie zapytanie na elementy składowe:

- Występuje w nim zapytanie zewnętrzne i wewnętrzne. Wewnętrzne pobiera miesiąc, rodzaj produktu i kwotę z tabeli `all_sales` i przesyła wyniki do zapytania zewnętrznego.
- `SUM(amount) FOR month IN (1 AS STY, 2 AS LUT, 3 AS MAR, 4 AS KWI)` jest linią w klauzuli PIVOT:
 - Funkcja `SUM()` dodaje kwoty sprzedaży poszczególnych rodzajów produktów w pierwszych czterech miesiącach (wymienionych w części `IN`). Miesiące w wynikach nie są określane jako 1, 2, 3 i 4, ponieważ część `AS` zmienia ich nazwy na `STY`, `LUT`, `MAR` i `KWI` w celu zwiększenia czytelności wyników.
 - Kolumna `month` tabeli `all_sales` jest używana jako kolumna przestawienia. To oznacza, że w wynikach miesiące pojawiają się jako kolumny. Na skutek tego wiersze są obrócone — czy też *przestawione* — aby były wyświetlane jako kolumny.
- Na samym końcu przykładu linia `ORDER BY prd_type_id` po prostu porządkuje wyniki według rodzaju produktu.

Przestawianie w oparciu o wiele kolumn

Przestawienie możemy wykonać w oparciu o wiele kolumn, umieszczając ich nazwy w części `FOR` klauzuli PIVOT. W poniższym przykładzie są przestawiane kolumny `month` i `prd_type_id`, do których występuje odwołanie w części `FOR`. Należy zauważyć, że lista wartości w części `IN` klauzuli PIVOT zawiera wartość dla kolumn `month` i `prd_type_id`:

```
SELECT *
FROM (
    SELECT month, prd_type_id, amount
    FROM all_sales
    WHERE year = 2003
```

```

AND prd_type_id IN (1, 2, 3)
)
PIVOT (
  SUM(amount) FOR (month, prd_type_id) IN (
    (1, 2) AS STY_PRDTYPE2,
    (2, 3) AS LUT_PRDTYPE3,
    (3, 1) AS MAR_PRDTYPE1,
    (4, 2) AS KWI_PRDTYPE2
  )
);

```

STY_PRDTYPE2	LUT_PRDTYPE3	MAR_PRDTYPE1	KWI_PRDTYPE2
14309,04	15467,9	91826,98	15664,7

Komórki w wynikach zawierają sumę kwot sprzedaży każdego rodzaju produktu w określonym miesiącu (rodzaj produktu i miesiąc są umieszczone na liście wartości w części IN). Jak widać w wynikach zapytania, kwoty sprzedaży były następujące:

- 14 309,04 zł dla produktu drugiego typu w styczniu,
- 15 467,90 zł dla produktu trzeciego typu w lutym,
- 91 826,98 zł dla produktu pierwszego typu w marcu,
- 15 664,70 zł dla produktu drugiego typu w kwietniu.

W części IN możemy umieścić wszystkie interesujące nas wartości. W poniższym przykładzie wartości rodzajów produktów w części IN są wymieszane, aby uzyskać kwoty sprzedaży tych produktów w określonych miesiącach:

```

SELECT *
FROM (
  SELECT month, prd_type_id, amount
  FROM all_sales
  WHERE year = 2003
  AND prd_type_id IN (1, 2, 3)
)
PIVOT (
  SUM(amount) FOR (month, prd_type_id) IN (
    (1, 1) AS STY_PRDTYPE1,
    (2, 2) AS LUT_PRDTYPE2,
    (3, 3) AS MAR_PRDTYPE3,
    (4, 1) AS KWI_PRDTYPE1
  )
);

```

STY_PRDTYPE1	LUT_PRDTYPE2	MAR_PRDTYPE3	KWI_PRDTYPE1
38909,04	13367,9	20626,98	120344,7

Uzyskano następujące kwoty sprzedaży:

- 38 909,04 zł dla produktu pierwszego typu w styczniu,
- 13 367,90 zł dla produktu drugiego typu w lutym,
- 20 626,98 zł dla produktu trzeciego typu w marcu,
- 120 344,70 zł dla produktu pierwszego typu w kwietniu.

Użycie kilku funkcji agregujących w przestawieniu

W przestawieniu możemy użyć wielu funkcji agregujących. Na przykład w poniższym zapytaniu użyto funkcji SUM() do obliczenia całkowitej kwoty sprzedaży wybranych rodzajów produktów w styczniu i lutym oraz funkcji AVG() do obliczenia średnich kwot sprzedaży:

```

SELECT *
FROM (
  SELECT month, prd_type_id, amount
  FROM all_sales
  WHERE year = 2003
  AND prd_type_id IN (1, 2, 3)
)
PIVOT (
  SUM(amount) AS sum_amount,
  AVG(amount) AS avg_amount
  FOR (month) IN (
    1 AS STY, 2 AS LUT
  )
)
ORDER BY prd_type_id;

```

PRD_TYPE_ID	STY_SUM_AMOUNT	STY_AVG_AMOUNT	LUT_SUM_AMOUNT	LUT_AVG_AMOUNT
1	38909,04	6484,84	70567,9	11761,3167
2	14309,04	2384,84	13367,9	2227,98333
3	24909,04	4151,50667	15467,9	2577,98333

Pierwszy wiersz wyników przedstawia kwoty sprzedaży dla produktu pierwszego typu:

- w styczniu całkowita kwota wyniosła 38 909,04 zł przy średniej 6 484,84 zł,
- w lutym całkowita kwota wyniosła 70 567,90 zł przy średniej 11 761,32.

Drugi wiersz wyników przedstawia kwoty sprzedaży dla produktu drugiego typu:

- w styczniu całkowita kwota wyniosła 14 309,04 zł przy średniej 2 384,84 zł,
- w lutym całkowita kwota wyniosła 13 367,90 zł przy średniej 2 227,98.
- ... itd. dla trzeciego wiersza.

Użycie klauzuli UNPIVOT

Klauzula UNPIVOT obraca kolumny w wiersze. Wykonuje ona operację przeciwną do wykonywanej przez klauzulę PIVOT.

W przykładach zawartych w tym podrozdziale korzystano z tabeli `pivot_sales_data` (tworzonej przez skrypt `store_schema.sql`):

```

CREATE TABLE pivot_sales_data AS
SELECT *
FROM (
  SELECT month, prd_type_id, amount
  FROM all_sales
  WHERE year = 2003
  AND prd_type_id IN (1, 2, 3)
)
PIVOT (
  SUM(amount) FOR month IN (1 AS STY, 2 AS LUT, 3 AS MAR, 4 AS KWI)
)
ORDER BY prd_type_id;

```

Dane w tabeli `pivot_sales_data` umieszczane są przez zapytanie zwracające przestawioną wersję danych sprzedaży z tabeli `all_sales`.

Poniższe zapytanie zwraca zawartość tabeli `pivot_sales_data`:

```

SELECT *
FROM pivot_sales_data;

```

PRD_TYPE_ID	STY	LUT	MAR	KWI
1	38909,04	70567,9	91826,98	120344,7
2	14309,04	13367,9	16826,98	15664,7
3	24909,04	15467,9	20626,98	23844,7

W kolejnym zapytaniu użyto klauzuli UNPIVOT do pobrania danych sprzedaży w formie nieprzestawionej:

```
SELECT *
FROM pivot_sales_data
UNPIVOT (
    amount FOR month IN (STY, LUT, MAR, KWI)
)
ORDER BY prd_type_id;
```

PRD_TYPE_ID	MON	AMOUNT
1	STY	38909,04
1	LUT	70567,9
1	MAR	91826,98
1	KWI	120344,7
2	STY	14309,04
2	LUT	13367,9
2	KWI	15664,7
2	MAR	16826,98
3	STY	24909,04
3	MAR	20626,98
3	LUT	15467,9
3	KWI	23844,7

W tych wynikach miesięczne wartości sprzedaży są zaprezentowane pionowo. Warto porównać te wyniki z wynikami wcześniejszego zapytania, gdzie wartości sprzedaży były zaprezentowane horyzontalnie.

Zapytania o określoną liczbę wierszy

Nowością w Oracle Database 12c jest wbudowane wsparcie dla zapytań o określoną liczbę wierszy (ang. *top-N queries*). Zapytanie takie zawiera klauzulę ograniczającą liczbę zwracanych wierszy. Taka klauzula umożliwia ograniczenie ilości pobieranych wierszy przez określenie:

- ilości wierszy do pobrania za pomocą klauzuli `FETCH FIRST`;
- przesunięcia określającego, ile wierszy należy pominąć, zanim rozpocznie się zliczanie wierszy za pomocą klauzuli `OFFSET`;
- części całkowitej ilości wierszy w procentach za pomocą klauzuli `PERCENT`.

Można dodać też kolejne warunki do klauzul ograniczających ilość wierszy:

- `ONLY`, który powoduje zwrócenie dokładnie określonej w klauzuli ilości wierszy;
- `WITH TIES`, który powoduje dołączenie dodatkowych wierszy z taką samą wartością klucza sortowania jak ostatni zwracany wiersz (klucz sortowania to kolumna określona w klauzuli `ORDER BY`).

W kolejnych podrozdziałach pokazane są przykłady.

Użycie klauzuli `FETCH FIRST`

Poniższe zapytanie wykorzystuje `FETCH FIRST` do pobrania pięciu pracowników z najmniejszą wartością identyfikatora `employee_id` z tabeli `more_employees`:

```
SELECT employee_id, first_name, last_name
FROM more_employees
ORDER BY employee_id
FETCH FIRST 5 ROWS ONLY;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	Jan	Kowalski
2	Roman	Joźwierz
3	Fryderyk	Helc
4	Zuzanna	Nowak
5	Robert	Zielony

W tym przykładzie pracownicy uporządkowani są według wartości `employee_id` i zwracane jest pięć wierszy z najniższymi wartościami `employee_id`.

Następne zapytanie pobiera pięciu pracowników z największymi wartościami `employee_id` z tabeli `more_employees`. Dzieje się tak, ponieważ zastosowane zostało porządkowanie malejąco według wartości `employee_id`.

```
SELECT employee_id, first_name, last_name
FROM more_employees
ORDER BY employee_id DESC
FETCH FIRST 5 ROWS ONLY;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
13	Dorota	Prewał
12	Franciszek	Słaby
11	Kamil	Długi
10	Kazimierz	Czarny
9	Henryk	Borny

Kolejne zapytanie korzysta z `FETCH FIRST`, by pobrać cztery produkty z najniższą ceną z tabeli `products`. Produkty są porządkowane według ceny, a następnie zwracane są cztery wiersze z najniższą ceną.

```
SELECT product_id, name, price
FROM products
ORDER BY price
FETCH FIRST 4 ROWS ONLY;
```

PRODUCT_ID	NAME	PRICE
9	Muzyka klasyczna	10,99
8	Z innej planety	12,99
7	Space Force 9	13,49
12	Pierwsza linia	13,49

Użycie klauzuli OFFSET

Za pomocą klauzuli `OFFSET` można określić ilość wierszy, jaką należy pominąć przed rozpoczęciem wyświetlania wyników. Poniższe zapytanie pobiera pracowników z `employee_id` o wartościach od 6 do 10 z tabeli `more_employees`:

```
SELECT employee_id, first_name, last_name
FROM more_employees
ORDER BY employee_id
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
6	Joanna	Brąz
7	Jan	Szary
8	Jadwiga	Niebieska
9	Henryk	Borny
10	Kazimierz	Czarny

Klauzula `OFFSET` w tym przykładzie ma postać:

```
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY
```

Oznacza to, że należy rozpocząć pobieranie wyników po wierszu nr 5 i pobrać pięć kolejnych wierszy po tej pozycji. To powoduje, że zostaną pobrane wiersze z wartościami `employee_id` od 6 do 10.

Następne zapytanie korzysta z klauzuli `offset` do pobrania produktów z tabeli `products`. Zapytanie szereguje produkty według ceny, rozpoczyna po wierszu nr 5 i pobiera kolejne cztery wiersze po tej pozycji.

```
SELECT product_id, name, price
FROM products
```

```
ORDER BY price
OFFSET 5 ROWS FETCH NEXT 4 ROWS ONLY;
```

PRODUCT_ID	NAME	PRICE
6	2412: Powrót	14,95
11	Twórczy wrzask	14,99
10	Pop 3	15,99
1	Nauka współczesna	19,95

Użycie klauzuli PERCENT

Klauzuli PERCENT można użyć do wskazania, jaki procent wszystkich wierszy wyniku zapytania należy zwrócić. Poniższe zapytanie pobiera 20 procent produktów z najwyższą ceną z tabeli products:

```
SELECT product_id, name, price
FROM products
ORDER BY price DESC
FETCH FIRST 20 PERCENT ROWS ONLY;
```

PRODUCT_ID	NAME	PRICE
5	Z Files	49,99
2	Chemia	30
3	Supernowa	25,99

Kolejne zapytanie pobiera 10 procent pracowników z najniższym wynagrodzeniem z tabeli more_employees:

```
SELECT employee_id, first_name, last_name, salary
FROM more_employees
ORDER BY salary
FETCH FIRST 10 PERCENT ROWS ONLY;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
8	Jadwiga	Niebieska	29000
7	Jan	Szary	30000

Użycie klauzuli WITH TIES

Klauzuli WITH TIES można użyć, by dołączyć dodatkowe wiersze z tą samą wartością klucza sortowania jak ostatni z pobieranych wierszy. Klucz sortowania to kolumna wskazana w klauzuli ORDER BY.

Poniższe zapytanie pobiera 10 procent pracowników z najniższym wynagrodzeniem z tabeli more_employees za pomocą WITH TIES. Kluczem sortowania jest kolumna salary:

```
SELECT employee_id, first_name, last_name, salary
FROM more_employees
ORDER BY salary
FETCH FIRST 10 PERCENT ROWS WITH TIES;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
8	Jadwiga	Niebieska	29000
7	Jan	Szary	30000
9	Henryk	Borny	30000

Porównaj te wyniki z wynikami z poprzedniego przykładu. W tych wynikach pojawił się pracownik Henryk Borny, który ma takie samo wynagrodzenie jak pracownik z ostatniego wiersza zwróconego w poprzednim przykładzie.

Odnajdywanie wzorców w danych

Nową funkcjonalnością Oracle Database 12c jest natywne wsparcie poszukiwania wzorca w danych za pomocą klauzuli `MATCH_RECOGNIZE`. Odnajdywanie wzorców w danych jest przydatne w wielu sytuacjach. Na przykład odnajdywanie wzorców w danych jest użyteczne przy poszukiwaniu trendów w sprzedaży, wykrywaniu oszustw w transakcjach dokonywanych za pomocą kart kredytowych czy odkrywaniu włań do sieci.

Przykłady w tym podrozdziale pokazują, jak odnajdywać wzorce formacji typu V i typu W w wielkości sprzedaży produktu w ciągu kilku dni. Te wzorce mogą być użyte na przykład do lepszego wybrania czasu kampanii marketingowych. Przykłady te korzystają z tabel `all_sales2` i `all_sales3` tworzonych przez skrypt `store_schema.sql` za pomocą następujących instrukcji:

```
CREATE TABLE all_sales2 (
  product_id INTEGER REFERENCES products(product_id),
  total_amount NUMBER(8, 2),
  sale_date DATE
);
```

```
CREATE TABLE all_sales3 (
  product_id INTEGER REFERENCES products(product_id),
  total_amount NUMBER(8, 2),
  sale_date DATE
);
```

Obie tabele zawierają takie same kolumny:

- **product_id** to identyfikator sprzedanego produktu,
- **total_amount** to całkowita cena produktu sprzedanego w ciągu dnia,
- **sale_date** to data sprzedaży.

W kolejnych podrozdziałach dowiesz się, jak odnajdować w danych wzorce formacji typu V i typu W.

Odnajdywanie wzorców formacji typu V w danych z tabeli all_sales2

W tym podrozdziale dowiesz się, jak odnaleźć formacje typu V (ang. *V-shaped data patterns*) w tabeli `all_sales2`. Dla uproszczenia w tabeli zapisane są dane o wielkości sprzedaży jednego produktu w ciągu dziesięciu dni.

Poniższe zapytanie pobiera wiersze tabeli `all_sales2`. Wartość `total_amount` osiąga najwyższą wartość 3 czerwca, następnie spada do 7 czerwca, potem rośnie do 9 czerwca. Te daty to odpowiednio: początkowy, najniższy i końcowy punkt formacji typu V w danych sprzedażowych (odpowiednie wiersze pogrubilem w wynikach zapytania).

```
SELECT *
FROM all_sales2;
```

PRODUCT_ID	TOTAL_AMOUNT	SALE_DATE
1	1000	11/06/01
1	1100	11/06/02
1	1200	11/06/03
1	1100	11/06/04
1	1000	11/06/05
1	900	11/06/06
1	800	11/06/07
1	900	11/06/08
1	1000	11/06/09
1	900	11/06/10

Poniższe zapytanie odnajduje wzorzec formacji typu V i zwraca daty wystąpienia początkowego (strt), najniższego (down) i końcowego (end) punktu:

```
SELECT *
FROM all_sales2
MATCH_RECOGNIZE (
  PARTITION BY product_id
  ORDER BY sale_date
  MEASURES
    strt.sale_date AS start_v_date,
    down.sale_date AS low_v_date,
    up.sale_date AS end_v_date
  ONE ROW PER MATCH
  AFTER MATCH SKIP TO LAST up
  PATTERN (strt down+ up+)
  DEFINE
    down AS down.total_amount < PREV(down.total_amount),
    up AS up.total_amount > PREV(up.total_amount)
) my_pattern
ORDER BY my_pattern.product_id, my_pattern.start_v_date;
```

```
PRODUCT_ID START_V_ LOW_V_DA END_V_DA
```

```
-----
1 11/06/03 11/06/07 11/06/09
```

Przeanalizujemy każdą część klauzuli MATCH_RECOGNIZE w tym przykładzie:

- PARTITION BY product_id grupuje wiersze pobrane z tabeli all_sales2 według wartości w kolumnie product_id. Każda grupa wierszy zawiera tę samą wartość product_id. (Tabela all_sales2 opisuje sprzedaż tylko jednego produktu, ale w prawdziwych danych mogą znajdować się dane o sprzedaży tysięcy różnych produktów).
- ORDER BY sale_date sortuje wiersze w każdej grupie według daty sprzedaży zapisanej w kolumnie sale_date.
- MEASURES określa następujące elementy zwracane wśród wyników:
 - strt.sale_date AS start_v_date, która jest datą początkową formacji typu V;
 - down.sale_date AS low_v_date, która jest datą najniższego punktu tej formacji;
 - up.sale_date AS end_v_date, która jest datą końcową formacji.
- ONE ROW PER MATCH generuje jeden wiersz w zestawie wyników dla każdej odnalezionej w danych formacji typu V. W tym przykładzie znajduje się tylko jedna taka formacja i dlatego zapytanie tworzy tylko jeden wiersz wyników. Gdyby było więcej tego typu formacji, w wynikach pojawiłby się oddzielny wiersz dla każdej z nich.
- AFTER MATCH SKIP TO LAST up powoduje, że wyszukiwanie jest kontynuowane od wiersza zawierającego koniec ostatnio znalezionej wzorca. up to zmienna wzorca opisana w klauzuli define.
- PATTERN (strt down+ up+) wskazuje, że wzorcem do wyszukania w tym przypadku jest formacja typu V. Aby wyobrazić sobie ten wzorzec, należy przeanalizować zapisane w klauzuli PATTERN zmienne wzorca: strt, down, up (start, dół, góra). Kolejność tych zmiennych wskazuje kolejność wyszukiwania zmiennych. Symbol plusa (+) umieszczony po zmiennych down i up wskazuje, że przynajmniej jeden wiersz musi być odnaleziony dla każdej zmiennej. W tym przykładzie klauzula PATTERN wyszukuje formacje typu V: początkowy wiersz strt, po którym następują najniższe wartości aż do najniższej (down), a następnie wyższe aż do najwyższej wartości (up).
- DEFINE opisuje następujące zmienne wzorca:
 - down AS down.total_amount < PREV(down.total_amount), która wykorzystuje funkcję PREV() do porównania wartości w kolumnie total_amount bieżącego wiersza z wartością w kolumnie total_amount poprzedniego wiersza; wartość down jest zapisywana, gdy wartość tej kolumny

w bieżącym wierszu jest mniejsza niż w poprzednim wierszu; zmienna ta opisuje najniższy punkt formacji typu V.

- `up AS up.total_amount > PREV(up.total_amount)` opisuje zakończenie formacji typu V.

Poniższe zapytanie pokazuje dodatkowe opcje `MATCH_RECOGNIZE`:

```
SELECT
  sale_date, start_v_date, low_v_date, end_v_date,
  match_variable, up_days, down_days, count_days,
  sales_difference, total_amount
FROM all_sales2
MATCH_RECOGNIZE (
  PARTITION BY product_id
  ORDER BY sale_date
  MEASURES
    strt.sale_date AS start_v_date,
    FINAL LAST(down.sale_date) AS low_v_date,
    FINAL LAST(up.sale_date) AS end_v_date,
    CLASSIFIER() AS match_variable,
    FINAL COUNT(up.sale_date) AS up_days,
    FINAL COUNT(down.sale_date) AS down_days,
    RUNNING COUNT(sale_date) AS count_days,
    total_amount - strt.total_amount AS sales_difference
  ALL ROWS PER MATCH
  AFTER MATCH SKIP TO LAST up
  PATTERN (strt down+ up+)
  DEFINE
    down AS down.total_amount < PREV(down.total_amount),
    up AS up.total_amount > PREV(up.total_amount)
) my_pattern
ORDER BY my_pattern.product_id, my_pattern.start_v_date;
```

SALE_DAT	START_V_	LOW_V_DA	END_V_DA	MATCH_VARIABLE		
	UP_DAYS	DOWN_DAYS	COUNT_DAYS	SALES_DIFFERENCE	TOTAL_AMOUNT	
11/06/03	11/06/03	11/06/07	11/06/09	STRT	0	1200
11/06/04	11/06/03	11/06/07	11/06/09	DOWN	-100	1100
11/06/05	11/06/03	11/06/07	11/06/09	DOWN	-200	1000
11/06/09	11/06/03	11/06/07	11/06/09	UP	-200	1000
11/06/07	11/06/03	11/06/07	11/06/09	DOWN	-400	800
11/06/08	11/06/03	11/06/07	11/06/09	UP	-300	900
11/06/06	11/06/03	11/06/07	11/06/09	DOWN	-300	900

Przeanalizujemy najważniejsze elementy klauzuli `MATCH_RECOGNIZE` z tego przykładu:

- `MEASURES` określa zmienne zwracane w wynikach:
 - `strt.sale_date AS start_v_date`, która opisuje datę początkową formacji typu V.
 - `FINAL LAST(down.sale_date) AS low_v_date`, która opisuje datę w najniższym punkcie formacji typu V. Funkcja `LAST()` zwraca ostatnią wartość. Połączenie `FINAL()` i `LAST()` dla zmiennej

low_v_date powoduje, że wszystkie wiersze opisujące formacje mają taką samą datę najniższego punktu.

- **FINAL LAST**(up.sale_date) AS end_v_date opisuje końcową datę formacji typu V. Połączenie **FINAL** i **LAST()** dla end_v_date powoduje, że wszystkie wiersze opisujące formacje mają taką samą datę punktu końcowego.
- **CLASSIFIER()** AS match_variable, która opisuje zmienną dopasowaną w każdym wierszu. W przykładowym zapytaniu zmienne dopasowane to strt, down i up. W wynikach zmienna dopasowana jest wyświetlana wielkimi literami.
- **FINAL COUNT**(up.sale_date) AS up_days opisuje ilość dni dopasowanych do wzorca opisującego wzrost wartości w każdej z odnalezionych formacji typu V.
- **FINAL COUNT**(down.sale_date) AS down_days opisuje ilość dni dopasowanych do wzorca opisującego spadek wartości w każdej z odnalezionych formacji typu V.
- **RUNNING COUNT**(sale_date) as count_days, która zawiera bieżącą sumę dni każdego z okresów zawierających formację typu V.
- total_amount – strt.total_amount AS sales_difference opisuje różnicę pomiędzy wartością total_amount danego dnia a wartością total_amount pierwszego dnia formacji typu V.
- **ALL ROWS PER MATCH** generuje jeden wiersz wyników dla każdego wiersza formacji typu V. W przykładzie znajduje się siedem wierszy w formacji typu V, co oznacza, że zapytanie wygeneruje siedem wierszy wyników.

Odnajdywanie formacji typu W w danych z tabeli all_sales3

W tym podrozdziale dowiesz się, jak odnaleźć formacje typu W w tabeli all_sales3. Dla uproszczenia w tabeli znajdują się informacje o sprzedaży tylko jednego produktu w ciągu dziesięciu dni.

Poniższe zapytanie pobiera wiersze tabeli all_sales3. Najniższe i najwyższe punkty tworzące formację typu W w kolumnie total_amount są wyróżnione w wynikach zapytania. Formacja typu W rozpoczyna się 2 czerwca i kończy 9 czerwca.

```
SELECT *
FROM all_sales3;
```

PRODUCT_ID	TOTAL_AMOUNT	SALE_DAT
1	1000	11/06/01
1	1100	11/06/02
1	900	11/06/03
1	800	11/06/04
1	900	11/06/05
1	1000	11/06/06
1	900	11/06/07
1	800	11/06/08
1	1000	11/06/09
1	900	11/06/10

Poniższe zapytanie odnajduje formacje typu W i zwraca początkową i końcową datę formacji:

```
SELECT *
FROM all_sales3
MATCH_RECOGNIZE (
  PARTITION BY product_id
  ORDER BY sale_date
  MEASURES
    strt.sale_date AS start_w_date,
    up.sale_date AS end_w_date
  ONE ROW PER MATCH
  AFTER MATCH SKIP TO LAST up
  PATTERN (strt down+ up+ down+ up+)
```

```

DEFINE
  down AS down.total_amount < PREV(down.total_amount),
  up AS up.total_amount > PREV(up.total_amount)
) my_pattern
ORDER BY my_pattern.product_id, my_pattern.start_w_date;

```

```

PRODUCT_ID START_W_ END_W_DA
-----

```

```

1 11/06/02 11/06/09

```

Przeanalizujemy odpowiednie części klauzuli MATCH_RECOGNIZE w poniższym przykładzie:

- MEASURES określa następujące elementy do zwrócenia w wynikach zapytania:
 - strt.sale_date AS start_w_date opisuje początkową datę formacji,
 - up.sale_date AS end_w_date opisuje końcową datę formacji.
- PATTERN (strt down+ up+ down+ up+) określa wzorec do wyszukiwania formacji typu W: początkowy wiersz strt, a po nim seria danych z wartościami malejącymi (down), rosnącymi (up), malejącymi (down) i znowu rosnącymi (up).

Odnajdywanie formacji typu V w tabeli all_sales3

W tym podrozdziale zobaczysz, jak można odnaleźć wzorce formacji typu V w tabeli all_sales3 i interpretację tych wyników. W poprzednim podrozdziale pokazane było, że tabela all_sales3 zawiera wzorec formacji typu W. Można przyjąć, że formacja typu W to dwie formacje typu V.

Poniższe zapytanie zwraca daty początkowego, najniższego i końcowego punktu formacji typu V w tabeli all_sales3. Zapytanie wykorzystuje ONE ROW PER MATCH, by wygenerować jeden wiersz wyników dla każdego wzorca typu V odnaniezonego w danych:

```

SELECT *
FROM all_sales3
MATCH_RECOGNIZE (
  PARTITION BY product_id
  ORDER BY sale_date
  MEASURES
    strt.sale_date AS start_v_date,
    down.sale_date AS low_v_date,
    up.sale_date AS end_v_date
  ONE ROW PER MATCH
  AFTER MATCH SKIP TO LAST up
  PATTERN (strt down+ up+)
  DEFINE
    down AS down.total_amount < PREV(down.total_amount),
    up AS up.total_amount > PREV(up.total_amount)
) my_pattern
ORDER BY my_pattern.product_id, my_pattern.start_v_date;

```

```

PRODUCT_ID START_V_ LOW_V_DA END_V_DA
-----

```

```

1 11/06/02 11/06/04 11/06/06

```

```

1 11/06/06 11/06/08 11/06/09

```

Wyniki zawierają dwa wiersze, po jednym dla każdego wzorca formacji typu V znalezioneego w danych.

Poniższe zapytanie wykorzystuje klauzulę ALL ROWS PER MATCH, która generuje jeden wiersz wyników dla każdego wiersza należącego do formacji typu V:

```

SELECT *
FROM all_sales3
MATCH_RECOGNIZE (
  PARTITION BY product_id
  ORDER BY sale_date
  MEASURES
    strt.sale_date AS start_v_date,

```

```

    down.sale_date AS low_v_date,
    up.sale_date AS end_v_date
ALL ROWS PER MATCH
AFTER MATCH SKIP TO LAST up
PATTERN (strt down+ up+)
DEFINE
    down AS down.total_amount < PREV(down.total_amount),
    up AS up.total_amount > PREV(up.total_amount)
) my_pattern
ORDER BY my_pattern.product_id, my_pattern.start_v_date;

```

PRODUCT_ID	SALE_DAT	START_V_	LOW_V_DA	END_V_DA	TOTAL_AMOUNT
1	11/06/02	11/06/02			1100
1	11/06/03	11/06/02	11/06/03		900
1	11/06/04	11/06/02	11/06/04		800
1	11/06/05	11/06/02	11/06/04	11/06/05	900
1	11/06/06	11/06/02	11/06/04	11/06/06	1000
1	11/06/06	11/06/06			1000
1	11/06/07	11/06/06	11/06/07		900
1	11/06/08	11/06/06	11/06/08		800
1	11/06/09	11/06/06	11/06/08	11/06/09	1000

W wynikach znajduje się w sumie dziewięć wierszy. Na początku pięć wierszy opisujących pierwszą formację typu V. Na końcu cztery wiersze opisujące drugą formację typu V.

W tym podrozdziale pokazany został użyteczny zestaw opcji służących do odnajdywania wzorców w danych. Kompletny zestaw opcji jest zbyt duży, by opisać go w tej książce. Pełny zestaw informacji można znaleźć w dokumencie *Oracle Database Data Warehousing Optimization Guide 12c Release* opublikowanym przez Oracle Corporation.

Podsumowanie

Z tego rozdziału dowiedziałeś się, że:

- funkcje analityczne umożliwiają wykonywanie złożonych obliczeń,
- klauzula MODEL wykonuje obliczenia międzywierszowe i umożliwia traktowanie danych tabeli jako tablicy,
- klauzule PIVOT i UNPIVOT są przydatne do analizy ogólnych trendów w dużej ilości danych,
- można wykonywać zapytania zwracające ograniczoną ilość wierszy wyników,
- klauzula MATCH_RECOGNIZE jest wykorzystywana do odnajdywania wzorców w danych.

W kolejnym rozdziale zajmiemy się zmienianiem zawartości tabeli.

Skorowidz

A

ACID, 254
aktualizowanie informacji, 511
aktywacja ról, 276
aliasy
 kolumn, 48
 tabel, 59
analiza danych, 205
ANSI, 59, 417
archiwa migawek, 314
atrybut :new, 464
Automatic Database Diagnostic
 Monitor, 488
automatyczne generowanie
 instrukcji, 88

B

baza danych
 sklepu, 30
 store, 33
B-drzewo, 302
bloki, 317
blokowanie transakcji, 255
błędy, 160
błędy w procedurze, 335

C

cudzysłowy, 245
czas, 127
czyszczenie formatowania, 78

D

dane LOB, 426
data, 46, 127
Database Replay, 488
datowniki, 143
DCL, Data Control Language, 24
DDL, Data Definition Language,
 24
deaktywacja ról, 276
definiowanie
 kolumny, 389
 konstruktora, 379
 zmiennych, 81
DML, Data Manipulation
 Language, 24
dodawanie
 indeksów, 474
 komentarza, 291
 wierszy, 38
 więzów
 CHECK, 286
 FOREIGN KEY, 287
 NOT NULL, 287
 UNIQUE, 288
dołączane bazy danych, 31
domyślna wartość kolumny, 300
domyślny formatu daty, 134
dostęp do
 danych LOB, 426
 komórek, 224, 227
 zakresu komórek, 225
drzewo, 181, 185
duże obiekty, LOB, 425, 427
działania arytmetyczne, 45

dziedziczenie
 atrybutów, 369, 384
 typów, 368

E

EDIT, 73
edycja instrukcji, 72
edytor, 75
elementy w kolekcji, 392
eliminowanie
 gałęzi, 185
 węzłów, 185

F

filtrowanie wierszy, 124, 199, 471
formacja
 typu V, 238, 241
 typu W, 240
format
 daty i czasu, 133
 RR, 136
 YY, 135
formatowanie
 kolumn, 76, 78
 liczb, 110
 wyników, 183
funkcja, 335
 ABS(), 100
 ADD_MONTHS(), 138
 ASCII(), 93
 ASCIISTR(), 105
 AVG(), 118
 BIN_TO_NUM(), 105
 CARDINALITY(), 420

funkcja

- CAST(), 105, 399
- CEIL(), 100
- CHARTOROWID(), 106
- CHR(), 93
- COLLECT(), 422
- COMPOSE(), 107
- CONCAT(), 93
- CONVERT(), 107
- COUNT(), 119
- CUME_DIST(), 211
- CURRENT_TIMESTAMP, 147
- CURRENTV(), 226
- DECODE(), 177
- DECOMPOSE(), 107
- DENSE_RANK(), 206
- EXTRACT(), 148
- FIRST(), 205, 221
- FIRST_VALUE(), 216
- FLOOR(), 100
- FROM_TZ(), 149
- get_product(), 364
- get_product_ref(), 365
- get_products(), 361
- GREATEST(), 101
- GROUP_ID(), 200
- GROUPING(), 195, 196
- GROUPING_ID(), 198, 199
- HEXTORAW(), 107
- INITCAP(), 94
- INSTR(), 94
- IS OF(), 372
- LAG(), 205, 220
- LAST(), 205, 221
- LAST_DAY(), 138
- LAST_VALUE(), 216
- LEAD(), 205, 220
- LEAST(), 101
- LENGTH(), 95
- LISTAGG(), 220
- LOCALTIMESTAMP, 147
- LOWER(), 95
- LPAD(), 96
- LTRIM(), 96
- MAX(), 119
- MIN(), 119
- MONTHS_BETWEEN(), 138
- NEXT_DAY(), 139
- NTH_VALUE(), 217
- NTILE(), 211
- NUMTODSINTERVAL(), 155
- NUMTOYMINTERVAL(), 155
- NVL(), 50, 96
- NVL2(), 97
- PERCENT_RANK(), 211
- POWER(), 101
- POWERMULTISET(), 423
- POWERMULTISET_BY_CARDINALITY(), 423
- PRESENTNNV(), 229
- PRESENTV(), 228
- RANK(), 206
- RATIO_TO_REPORT(), 219
- RAWTOHEX(), 108
- REGEXP_COUNT(), 117
- REGEXP_INSTR(), 116
- REGEXP_LIKE(), 114
- REGEXP_REPLACE(), 117
- REGEXP_SUBSTR(), 117
- REPLACE(), 97
- ROUND(), 102, 139
- ROW_NUMBER(), 211
- ROWIDTOCHAR(), 108
- RPAD(), 96
- RTRIM(), 96
- SET(), 421
- SIGN(), 102
- SOUNDEX(), 97
- SQRT(), 102
- STDDEV(), 120
- SUBSTR(), 97
- SUM(), 120
- SYS_EXTRACT_UTC(), 149
- SYS_TYPEID(), 378
- SYSDATE, 140
- SYSTIMESTAMP, 147
- TABLE(), 394
- TO_BINARY_DOUBLE(), 108
- TO_BINARY_FLOAT(), 108
- TO_CHAR(), 108, 128, 134
- TO_DATE(), 46, 128, 132, 509
- TO_MULTI_BYTE(), 109
- TO_NUMBER(), 110
- TO_SINGLE_BYTE(), 111
- TO_TIMESTAMP(), 150
- TO_TIMESTAMP_TZ(), 150
- TRANSLATE(), 176
- TREAT(), 375
- TRIM(), 96
- TRUNC(), 103, 140
- UNISTR(), 112
- UPPER(), 95
- VARIANCE(), 120
- XMLAGG(), 495
- XMLATTRIBUTES(), 494
- XMLCOLATTVAL(), 497
- XMLCOMMENT(), 499
- XMLCONCAT(), 498
- XMLELEMENT(), 492
- XMLFOREST(), 494
- XMLPARSE(), 498
- XMLPI(), 499
- XMLQUERY(), 502
- XMLSEQUENCE(), 500
- XMLSERIALIZE(), 501

funkcje

- agregujące, 117, 232
- analityczne, 205
- hipotetycznego rankingu i rozkładu, 222
- hipotetycznych klasyfikacji i dystrybucji, 205
- jednowierszowe, 91
- klasyfikujące, 205, 207
- konwertujące, 103, 128
- numeryczne, 98
- obiektów, 372
- okien, 205
- operujące na
 - datach i godzinach, 137
 - datownikach, 147
 - interwałach, 155
 - strefach czasowych, 141
 - wyrażeniach regularnych, 115
- rankingowe, 212
- raportujące, 205, 218
- regresji liniowej, 205, 221
- wyrażeń regularnych, 112
- znakowe, 92

G

generowanie

- instrukcji, 88
- natywnego kodu maszynowego, 347

planu wykonania, 483
 XML, 492
 grupowanie wierszy, 120
 gwiazdka, 35

H

hurtownia danych, 305

I

identyfikatory
 obiektów, 357
 wierszy, 44
 iloczyn kartezjański, 60
 indeks, 302
 bitmapowy, 305, 475
 oparty na funkcji, 303
 typu B-drzewo, 303, 475
 złożony, 303
 informacje o
 definicjach perspektyw, 310
 dużych obiektach, 425
 funkcjach, 337, 340
 indeksach, 304
 indeksach kolumny, 304
 kolekcjach, 387, 389
 kolumnach, 283
 optymalizacji, 471
 procedurach, 334, 340
 sekwencjach, 298
 tabelach, 282
 tabeli customers, 29
 tabeli zagnieżdżonej, 390
 typach obiektowych, 351
 więzach, 289, 290
 więzach perspektywy, 311
 wyzwalaczach, 343
 instrukcja
 ALTER ROLE, 276
 ALTER TABLE, 287, 315
 CREATE FLASHBACK
 ARCHIVE, 314
 CREATE PROCEDURE, 41
 DELETE, 168, 246
 DESCRIBE, 295
 DISCONNECT, 40
 DROP ROLE, 277
 DROP TABLE, 88

INSERT, 39, 243, 308
 MERGE, 249
 OPEN-FOR, 325
 REVOKE, 276
 ROLLBACK, 251
 SELECT, 26, 39, 43
 UPDATE, 167, 245
 instrukcje
 DCL, 24
 DDL, 24, 32
 DML, 24
 identyczne, 478
 nieidentyczne, 478
 TC, 24
 integralność bazy danych, 247
 interwały czasowe, 127, 151
 DAY TO SECOND, 154
 YEAR TO MONTH, 152
 izolacja transakcji, 256

J

język
 Perl, 114
 PL/SQL, 41, 317
 SQL, 24
 XML, 491

K

klauzula
 CONNECT BY, 182
 CROSS APPLY, 201
 CUBE, 190, 194, 478
 DEFAULT ON NULL, 293
 FETCH FIRST, 234
 FROM, 44, 160
 GROUP BY, 120, 124, 199
 GROUPING SETS, 197, 478
 HAVING, 124, 159, 199, 475
 LATERAL, 202
 MATCH_RECOGNIZE,
 237–239
 MODEL, 223, 227–229
 NOT FINAL, 368
 OFFSET, 235
 ORDER BY, 57, 161
 OUTER APPLY, 201, 202
 PARTITION BY, 208

PERCENT, 236
 PIVOT, 230
 RETURNING, 246
 ROLLUP, 190–193
 START WITH, 182–185
 UNION, 476
 UNION ALL, 476
 UNPIVOT, 230, 233
 WHERE, 44, 124, 157, 471
 WITH, 347
 WITH TIES, 236
 klucz
 główny, 34
 główny złożony, 37
 obcy, 35
 kolejność wykonywania działań,
 48
 kolekcje, 387
 kolekcje wielopoziomowe, 411,
 412
 kolumny, 23
 niewidoczne, 294, 313
 typu IDENTITY, 301
 widoczne, 313
 wirtualne, 284
 komentarze, 291
 kompresja danych LOB, 469
 konfigurowanie połączenia, 28
 konkatenacja, 49
 konstruktor, 352, 379
 konto użytkownika, 32
 konwersja
 kolekcji, 399
 napisu, 150
 niejawna, 464
 tabeli, 400
 typów, 128
 wyrażenia DataGodzina, 143
 kończenie
 połączenia, 88
 transakcji, 252
 kopiowanie
 danych, 452, 455–459
 wierszy, 245
 koszt wykonania zapytań, 480
 kursory, 322, 325
 kursory bez ograniczenia, 327
 kwalifikowane odwołania, 473

L

liść, 182

LOB, 425

BFILE, 426

BLOB, 426

CLOB, 426

NCLOB, 426

logiczna jednostka pracy, 251,
254

logika warunkowa, 319

lokalizator LOB, 426, 448

Ł

łączenie

funkcji, 98

operatorów zestawu, 175

wartości, 49

wywołań funkcji, 134

z bazą danych, 40

M

manipulowanie

tabelą zagnieżdżoną, 402

tablicą VARRAY, 400

metaznaki, 113, 114

metoda

APPEND(), 433

CLOSE(), 433

COMPARE(), 434

COPY(), 435

COUNT(), 404

CREATETEMPORARY(),
435

DELETE(), 406

ERASE(), 436

EXISTS(), 406

EXTEND(), 407

FILECLOSE(), 436

FILECLOSEALL(), 437

FILEEXISTS(), 437

FILEGETNAME(), 437

FILEISOPEN(), 438

FILEOPEN(), 438

FIRST(), 408

FREETEMPORARY(), 439

GET_STORAGE_LIMIT(),
440

GETCHUNKSIZE(), 439

GETLENGTH(), 439

INSTR(), 440

ISOPEN(), 441

ISTEMPORARY(), 441

LAST(), 408

LOADBLOBFROMFILE(),
443

LOADFROMFILE(), 442

NEXT(), 409

OPEN(), 444

PRIOR(), 410

READ(), 445

SUBSTR(), 445

TRIM(), 411, 446

WRITE(), 447

WRITEAPPEND(), 447

metody

mapujące, 397

operujące na kolekcjach, 403

pakietu DBMS_LOB, 431–433

miejsce na tabelę, 30

modyfikowanie

danych, 429

elementów

kolekcji, 395

tabeli zagnieżdżonej, 396

tablicy VARRAY, 396

indeksu, 305

istniejących komórek, 229

kolumny, 285

perspektywy, 313

sekwencji, 301

wierszy, 38, 245

N

nagłówek, 85

napisy

formatujące datę, 132

formatujące godzinę, 132

narzędzia optymalizujące, 487

narzędzie

Automatic Database

Diagnostic Monitor, 488

Database Replay, 488

Oracle Enterprise Manager,
487Real-Time SQL Monitoring,
488

SQL Access Advisor, 488

SQL Performance Analyzer,
488

SQL Plan Management, 489

SQL Tuning Advisor, 488

następstwo operatorów, 56

nazwy stref czasowych, 143

niejawna konwersja, 464

nierównołączenia, 61

numery wierszy, 45

O

obcinanie tabeli, 292

obiekty

bazy danych, 349

BFILE, 430

BLOB, 428

CLOB, 428

dołączanie danych, 450

kolumnowe, 352

kopiowanie danych, 452

NOT SUBSTITUTABLE, 371

odczyt danych, 449

tymczasowe, 452

usuwanie danych, 453

wyszukiwanie danych, 454

obliczanie

sum pośrednich, 86

sumy kumulacyjnej, 213

średniej centralnej, 215

średniej kroczącej, 214

wektora bitowego, 198

obliczenia na datach, 46

obserwowanie operacji

ALL STATEMENTS, 279

BY ACCESS, 278

BY SESSION, 278

IN SESSION CURRENT, 279

uprawnienia, 277

USER_AUDIT_OBJECT, 279

USER_AUDIT_SESSION,
279USER_AUDIT_STATEMENT,
279

USER_AUDIT_TRAIL, 279

WHENEVER NOT

SUCCESSFUL, 278

WHENEVER SUCCESSFUL,
278

obsługa
 tabel zagnieżdżonych, 417
 wartości brakujących, 227
 wartości NULL, 227

odbieranie
 roli, 276
 uprawnień, 271
 uprawnień roli, 276
 uprawnień systemowych, 265

odczytywanie
 danych z CLOB, 449
 plików, 73

odnajdywanie
 formacji, 240, 241
 wzorców, 237

odpytywanie
 perspektyw, 308
 tabeli planu, 483

odwołania
 do kolumn, 473
 obiektowe, 357

odwrotne funkcje percentyli, 205

ograniczenia połączeń
 zewnętrznych, 65

okno SQL Developer, 28

określanie formatu
 czasu, 133
 daty, 133

operacje w planie wykonania, 485

operand, 45

operator
 <=, 52
 <>, 51
 >, 52
 ALL, 52, 163
 AND, 55
 ANY, 52, 162
 BETWEEN, 55
 CUBE, 209
 DISTINCT, 477
 EXISTS, 164, 477
 GROUPING SETS, 209
 IN, 54, 161, 418, 477
 INTERSECT, 174
 IS A SET, 422
 IS EMPTY, 422
 LIKE, 53
 MINUS, 174
 MULTISSET, 419

NOT EXISTS, 164, 165
 NOT IN, 165, 418
 OR, 56
 ROLLUP, 209
 SUBMULTISET, 419
 UNION, 173
 UNION ALL, 172

operatory
 ANSI, 399
 arytmetyczne, 46
 do porównywania, 51
 jednowierszowe, 158
 logiczne, 55
 nierówności, 417
 równości, 417
 SQL, 53
 zestawu, 171

optymalizacja, 480

optymalizacja SQL, 471

optymalizator, 486

Oracle Enterprise Manager, 487

Oracle PL/SQL, 41

P

pakiet DBMS_LOB, 431

pakiety, 337

parametry formatujące
 daty i godziny, 129–131
 liczby, 110

perspektywa user_varrays, 390

perspektywy, 306
 zapisu obserwacji, 279
 złożone, 311

pętla
 FOR, 227, 321, 325
 WHILE, 321

PL/SQL, Procedural
 Language/SQL, 317
 bloki, 317
 duże obiekty, 431
 funkcje, 335
 instrukcja OPEN-FOR, 325
 kod maszynowy, 347
 kursory, 322, 325
 logika warunkowa, 319
 metody, 403
 obiekty, 361
 obsługa błędów, 328

pętla, 320

predefiniowane wyjątki, 328

procedury, 331

rozszerzenia, 345

sekwencje, 346

typy, 319

użycie kolekcji, 400

zmienne, 319

plan wykonania, 480, 483
 dla połączeń tabel, 484

plik
 binaryContent.doc, 425, 426
 textContent.txt, 426

pliki XML, 506

pobieranie
 bieżącej daty, 142
 danych z CLOB, 428
 dat, 127
 elementów, 392, 393
 informacji, 43
 lokalizatora LOB, 448
 n-tego wiersza, 217
 wierszy, 322
 wszystkich kolumn, 44

podstawianie nazw, 80

podtyp, 369

podzapytania
 jednowierszowe, 157
 rekurencyjne, 187
 skorelowane, 157, 163, 164
 w klauzuli FROM, 160
 w klauzuli HAVING, 159
 w klauzuli WHERE, 157
 wielokolumnowe, 157, 163
 wielowierszowe, 157, 161
 zagnieżdżone, 157, 166

polecenie
 ACCEPT, 82
 ALTER SESSION, 134
 APPEND, 72
 BREAK ON, 86
 BTITLE, 86
 CHANGE, 72
 CLEAR, 76
 CLEAR] BUFFER, 72
 COLUMN, 77
 COMPUTE, 86
 DEFINE, 81
 DEL, 72

polecenie
 DESCRIBE, 351
 FORMAT, 76
 GET, 73
 HEADING, 76
 HELP, 87
 JUSTIFY, 76
 LIST, 72
 RUN, 72
 SAVE, 73
 SET DEFINE, 80
 SET DESCRIBE DEPTH, 352
 SET VERIFY, 80
 SPOOL, 73
 START, 73
 TTITLE, 86
 WORD_WRAPPED, 76
 WRAPPED, 76
 połączenie z bazą danych, 28, 88
 pomoc SQL*Plus, 87
 porównywanie
 planów wykonania, 485
 wartości, 51
 wartości obiektów, 359
 poziomy izolacji, 256
 predykat IS ANY, 225
 procedura, 331
 delete_product(), 365
 display_product(), 362
 insert_product(), 363
 nclob_example(), 464
 product_lifecycle(), 366
 update_product(), 364
 update_product_price(), 363
 procedury PL/SQL, 448
 program SQL*Plus, 25, 71
 programowanie, 317
 przeciążanie metod, 381
 przeglądanie
 planów wykonania, 481
 struktury tabeli, 71
 przekształcanie kolumn, 463
 przesłanianie metod, 382
 przedstawianie, 231
 przestrzeń tabel, 261, 416
 przesunięcie strefy czasowej, 142
 przesyłanie wskazówek, 486
 przygotowywanie podzapytań,
 168

przyznawanie
 roli, 272
 uprawnień, 257, 266
 uprawnień roli, 272
 pseudokolumna LEVEL, 183
 punkty zachowania, 252

R

raport, 83
 raportowanie sumy, 218
 Real-Time SQL Monitoring, 488
 rekurencyjne podzapytania
 przygotowywane, 187
 relacja nadrzędny/podrzędny, 36
 relacyjna baza danych, 23
 rodzaje
 dużych obiektów, 426
 podzapytań, 157
 role, 261, 271
 aktywacja, 276
 deaktywacja, 276
 odbieranie uprawnień, 276
 przyznawanie uprawnień, 272
 sprawdzanie, 272
 uprawnienia obiektowe, 274
 uprawnienia systemowe, 273
 usuwanie, 277
 zastosowanie uprawnień, 275
 rozłączanie z bazą danych, 40
 rozmiar
 strony, 77
 typu elementu, 415
 wiersza, 78
 rozpoczynanie transakcji, 252
 rozszerzenia
 kolekcji, 414
 PL/SQL, 345
 równozłączenia, 61

S

skalanie wierszy, 249
 schemat, 23, 30, 33
 schemat XML, 506, 508, 511
 sekwencje, 296
 składowanie
 dat, 127
 wierszy, 480

skrypt, 30, 31
 collection_schema.sql, 387
 collection_schema2.sql, 412
 collection_schema3.sql, 414
 object_schema.sql, 350
 object_schema2.sql, 368
 object_schema3.sql, 384
 store_schema.sql, 34–37
 słowe kluczowe
 AND, 225
 BETWEEN, 225
 PRIMARY KEY, 34
 USING, 67
 specyfikacja pakietu, 338
 sprawdzanie
 ról przyznanych, 272
 uprawnień, 264–268, 273
 SQL, Structured Query Language,
 24
 SQL Access Advisor, 488
 SQL Developer, 27
 SQL Performance Analyzer, 488
 SQL Plan Management, 489
 SQL Tuning Advisor, 488
 SQL*Plus, 25, 71
 SQL/92, 66
 standard ANSI, 59, 417
 statystyki tabeli, 485
 stopka, 85
 strefa czasowa, 140
 bazy danych, 141
 sesji, 141, 142
 struktura tabeli, 71
 strukturalny język zapytań, 24
 suma kumulacyjna, 213
 sumy pośrednie, 86
 synonimy, 270
 synonimy publiczne, 270
 system zarządzania, 24
 szyfrowanie danych
 kolumny, 468
 LOB, 465, 466

Ś

średnia
 centralna, 215
 krocząca, 214
 średnik, 71

T

tabela, 23, 281
 customers, 29
 planu, 481, 483
 tabele
 dodawanie komentarza, 291
 kolumny, 285
 kolumny wirtualne, 284
 nadrzędne, 36, 247
 obcinanie, 292
 obiektove, 354
 pobieranie informacji, 282
 podrzędne, 35, 247
 tymczasowe, 416
 usuwanie, 292
 więzy, 286
 więzy odroczone, 289
 zagnieżdżone, 387–390, 402
 zmienianie, 284
 zmienianie nazwy, 291
 tablica VARRAY, 389, 393, 400
 liczba elementów, 416
 manipulowanie, 400
 tablice asocjacyjne, 387, 415
 TC, Transaction Control, 24
 transakcja SERIALIZABLE, 256
 transakcje bazodanowe
 ACID, 254
 blokowanie, 255
 kończenie, 252
 poziomy izolacji, 256
 punkty zachowania, 252
 rozpoczynanie, 252
 współbieżne, 255
 wycofywanie, 251
 zatwierdzanie, 251
 tworzenie
 archiwów migawek, 314
 funkcji, 336
 indeksu, 281
 bitmapowego, 305
 opartego na funkcji, 303
 typu B-drzewo, 303
 kolekcji, 388
 konta użytkownika, 32, 262
 obiektu katalogu, 430
 perspektyw, 281, 307, 309
 perspektyw złożonych, 311
 portfela, 465
 procedury, 332
 raportów, 83
 ról, 271
 schematu, 30
 schematu bazy danych, 368
 schematu XML, 506
 sekwencji, 281, 296
 specyfikacji pakietu, 338
 synonimów, 270
 tabeli, 281
 tabeli planu, 482
 tabeli zagnieżdżonej, 388
 treści pakietu, 338
 typów obiektowych, 350
 typu VARRAY, 388
 użytkownika, 30
 wyzwalacza, 341
 typ danych
 BFILE, 516
 BINARY_DOUBLE, 292, 515
 BINARY_FLOAT, 292, 515
 BINARY_INTEGER, 517
 BLOB, 516
 BOOLEAN, 517
 CHAR, 33, 515
 CLOB, 516
 DATE, 33, 516
 DEC, 516
 DECIMAL, 516
 DOUBLE PRECISION, 516
 FLOAT, 516
 INT, 516
 INTEGER, 33, 516
 INTERVAL DAY, 153, 516
 INTERVAL YEAR, 152, 516
 LONG, 462, 517
 LONG RAW, 426, 462, 517
 NATURAL, 517
 NATURALN, 517
 NCHAR, 515
 NCLOB, 516
 NUMBER, 33, 515, 516
 NUMERIC, 515, 516
 NVARCHAR2, 515
 PLS_INTEGER, 517
 POSITIVE, 517
 POSITIVEN, 517
 RAW, 517
 REAL, 516

RECORD, 518
 REF, 517
 REF CURSOR, 518
 ROWID, 517
 SIGNTYPE, 517
 SIMPLE_INTEGER, 345, 517
 SMALLINT, 516
 STRING, 518
 TIMESTAMP, 144, 516
 TIMESTAMP WITH LOCAL
 TIME_ZONE, 145, 150
 TIMESTAMP WITH TIME
 ZONE, 145
 TO MONTH, 516
 TO SECOND, 516
 UROWID, 517
 VARCHAR2, 33, 515
 VARRAY, 387, 517
 XMLType, 517

typy
 datowników, 144
 funkcji, 91
 interwałów czasowych, 151
 LOB, 425
 obiektove NOT
 INSTANTIABLE, 378
 nadrzędne, 369
 złączeń, 61

U

ukośnik, 26
 umieszczanie elementów, 392
 unikatowe wiersze, 50
 uogólnione wywoływanie, 383,
 384
 upraszczanie złączeń, 67
 uprawnienia, 23, 261
 dla obserwacji, 277
 do tworzenia perspektyw, 307
 obiektove, 266–269
 obiektove roli, 274
 systemowe, 263–271
 systemowe roli, 273
 uruchamianie
 plików, 73
 skryptu, 31
 SQL*Plus, 25, 26
 wyzwalacza, 343
 ustanowienie połączenia, 30

ustawianie rozmiaru

strony, 77

wiersza, 78

usuwanie

danych z obiektu, 453

funkcji, 337

indeksu, 305

konta użytkownika, 263

pakietu, 340

perspektywy, 313

procedury, 335

roli, 277

sekwencji, 302

tabeli, 292

użytkownika, 30

wierszy, 38, 40, 246

więzów, 288

wyzwalacza, 345

użycie

aliasów kolumn, 48

aliasów tabel, 59

atrybutu

new, 464

dołączanych baz danych, 31

dużych obiektów, 428, 431

funkcji

agregujących, 121, 123

CAST(), 399, 400

CUME_DIST(), 211

CURRENTV(), 226

DECODE(), 177

FIRST(), 221

GROUP_ID(), 200

GROUPING(), 195, 197

GROUPING_ID(), 198

hipotetycznego rankingu

i rozkładu, 222

LAG(), 220

LAST(), 221

LEAD(), 220

LISTAGG(), 220

NTILE(), 211

okna, 212

PERCENT_RANK(), 211

RANK(), 206

DENSE_RANK(), 206

RATIO_TO_REPORT(),
219

regresji liniowej, 221

ROW_NUMBER(), 211

TABLE(), 394, 395

TRANSLATE(), 176

IGNORE NAV, 229

KEEP NAV, 229

IS PRESENT, 228

klauzuli

DEFAULT ON NULL, 293

FETCH FIRST, 234

MODEL, 223

OFFSET, 235

PARTITION BY, 208

PERCENT, 236

PIVOT, 230

ROLLUP, 192

UNPIVOT, 230, 233

WITH TIES, 236

kolekcji, 389, 400

metody mapującej, 397

obiektów, 361

BFILE, 430

BLOB, 428

CLOB, 428

odwrotnych funkcji

rankingowych, 212

operatora

ALL, 163

ANY, 162

CUBE, 209

EXISTS, 164

GROUPING SETS, 209

IN, 161

NOT EXISTS, 164, 165

ROLLUP, 209

perspektyw, 307

perspektyw złożonych, 311

podtypu, 370

pseudokolumny LEVEL, 183

SCN, 259

sekwencji, 298, 300

SQL*Plus, 25

tymczasowych obiektów, 452

typów obiektowych, 352

typu

BINARY_DOUBLE, 293

BINARY_FLOAT, 293

TIMESTAMP, 144, 145

VARRAY, 389

wartości domyślnych, 248

wyrażeń CASE, 178, 196, 473

wyrażeń z funkcjami, 98

zapytań retrospektywnych,
257

złączeń tabel, 472

zmiennych, 79

dowiązanych, 480

tymczasowych, 83

zdefiniowanych, 84

użytkownicy, 261

hasło, 263

konto, 262

odbieranie uprawnień

obiektowych, 271

role, 272

sprawdzanie uprawnień, 264

uprawnienia obiektowe, 266

uprawnienia systemowe, 263

usuwanie konta, 263

W

wartości domyślne, 248

wartości obiektów, 359

wartość NULL, 34, 49, 208, 227

wartość wymiaru, 226

węzeł

główny, 181

nadrzędny, 181

podrzędny, 182

węzły równorzędne, 182

widok wbudowany LATERAL,
203

wiersze, 38

więzy, 286

CHECK, 286

CHECK OPTION, 309

FOREIGN KEY, 287

NOT NULL, 287

odroczone, 289

READ ONLY, 310

UNIQUE, 288

właściwości

dużych obiektów, 463, 465,
469

transakcji, 254

włączanie

więzów, 288

wyzwalacza, 345

wskazywanie wierszy, 44
 wskaźnik do pliku, 430
 współbieżne transakcje, 255
 wstawianie

danych, 462

wierszy, 243

wycofywanie transakcji, 251

wyjątek, 328

DUP_VAL_ON_INDEX, 330

INVALID_NUMBER, 330

OTHERS, 331

ZERO_DIVIDE, 330

wyjątki predefiniowane, 328

wykonywanie złączeń, 66

krzyżowych, 70

wewnętrznych, 66–68

własnych, 69

zewnętrznych, 68

lewostronnych, 68

pełnych, 69

prawostronnych, 69

wyłączanie więzów, 288

wymuszanie więzów

klucza głównego, 247

kluczy obcych, 247

wypisywanie zmiennych, 81

wrażenia

CASE, 178, 196, 473

regularne, 113, 115

wyszukiwanie danych, 454

wyświetlanie unikatowych

wierszy, 50

wywoływanie

funkcji, 336, 339

procedur, 333, 339

wyzwalacze, 340

tworzenie, 341

uruchamianie, 343

usuwanie, 345

włączanie, 345

wyłączanie, 345

wzorce, 237

X

XML, Extensible Markup

Language, 491

Z

zapis

mieszany, 334

nazywany, 334

pozycyjny, 224

symboliczny, 224

zapisywanie

danych XML, 501, 506

do obiektu CLOB, 450

plików, 73

zapytania, 24

hierarchiczne, 181, 183, 187

jednowierszowe, 160

o liczbę wierszy, 234

retrospektywne

uprawnienia, 257

użycie SCN, 259

w oparciu o czas, 258

zaawansowane, 171

zastosowanie uprawnień

obiektowych, 269

roli, 275

systemowych, 265

zatwierdzanie transakcji, 251

zbieranie statystyk tabeli, 485

zestaw wyników, 44

złączenia, 61, 66

krzyżowe, 70

wewnętrzne, 61, 66–68

własne, 61, 65, 69

zewnętrzne, 61, 68

lewostronne, 63

prawostronne, 63

złączenie tabel, 58

złożony klucz główny, 37

zmienianie

edytora, 75

hasła, 263

nazwy tabeli, 291

pozycji kolumn, 193

rozmiaru typu elementu, 415

tabeli, 284

zawartości tabeli, 243

znaku, 80

zmienna środowiskowa

NLS_LANG, 27

zmiennie, 79

dowiązane, 478

tymczasowe, 79

zdefiniowane, 81

znacznik czasu, 127, 143

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Kompendium wiedzy na temat bazy danych Oracle!

Baza danych Oracle to jeden z najpopularniejszych systemów bazodanowych, używany w wielu firmach różnej wielkości. Popularność tej bazy sprawiła, że na całym świecie w jej tabelach znajdują się gigantyczne ilości danych – o kluczowym znaczeniu dla działania przedsiębiorstw. Jeżeli chcesz w pełni wykorzystać potencjał bazy danych Oracle, to trafiłeś na doskonałą książkę.

Sięgnij po nią i poznaj możliwości zapytań SQL oraz programów PL/SQL. Dzięki lekturze kolejnych rozdziałów nauczysz się budować zapytania oraz podzapytania SQL, tworzyć tabele, sekwencje, indeksy oraz widoki, a ponadto korzystać z funkcji wbudowanych w język. Potem przejdziesz do zaawansowanych tematów związanych z analizą danych oraz uprawnieniami. W książce znajdziesz też dokładny opis języka PL/SQL, poznasz jego składnię oraz możliwości. Zwróć uwagę na ostatnie rozdziały, poświęcone optymalizacji SQL oraz korzystaniu z XML. To doskonała lektura dla wszystkich użytkowników bazy danych oraz osób przygotowujących się do egzaminów związanych z bazą danych Oracle.

Dzięki tej książce:

- podłączysz się do bazy danych oraz pobierzesz dane
- zbudujesz proste i zaawansowane zapytania SQL
- wykorzystasz możliwości podzapytań i perspektyw
- poznasz mechanizm uprawnień w bazie Oracle
- zaznajomisz się z językiem PL/SQL

Helion

27249 numer katalogowy

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nowosci>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Informatyka w najlepszym wydaniu

sięgnij po WIECEJ



KOD KORZYŚCI

ISBN 978-83-246-9922-3



cena: 89,00 zł

**Oracle
Press**