ORACLE

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 Rogóża

ISBN: 978-83-246-9922-3

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.

Spis treści

ROZDZIAŁ

8

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** — umożliwiające dopasowanie linii regresji metodą najmniejszych kwadratów do zestawu par liczb,
- **funkcje hipotetycznych klasyfikacji i dystrybucji** — umożliwiają obliczenie klasyfikacji i percentylu, 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`. Składuje ona sumę 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.

Zacznijmy 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 dystrybuanty (ang. *cumulative distribution*)
PERCENT_RANK()	Zwraca procentową rangę wartości w grupie wartości
NTILE()	Zwraca *n*-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 wartościom 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(kubełki)` służy do obliczania *n*-tyli (tertyli, kwartyli itd.). Parametr *kubełki* określa liczbę „kubełków", w których zostaną porozmieszczane grupy wierszy. Na przykład:

- `NTILE(2)` określa dwa kubełki, w związku z czym wiersze zostaną podzielone na dwie grupy wierszy,
- `NTILE(4)` dzieli grupy na cztery kubełki, 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 kubełki:

```
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 dystrybuanta 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 funkcje `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 przesuwa 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 (szóstego miesiąca) 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 kroczącą 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 funkcje jak zmienna zależna od *x*.

Tabela 8.2. *Funkcje regresji liniowej*

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

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_samount[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 jednoczesne 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 pierwszego 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_wewnętrzne
)
PIVOT (
  funkcja_agregująca FOR kolumna_obrotu IN (lista_wartości)
)
ORDER BY …;
```

Rozłóżmy 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      Jóźwierz
          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 employeed_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 włamań 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_data`** 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 pogrubiłem w wynikach zapytania).

```
SELECT *
FROM all_sales2;

PRODUCT_ID TOTAL_AMOUNT SALE_DAT
---------- ------------ --------
         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
```

Przeanalizujmy 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 znalezionego 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
         2          4          1                0         1200

11/06/04 11/06/03 11/06/07 11/06/09 DOWN
         2          4          2             -100         1100

11/06/05 11/06/03 11/06/07 11/06/09 DOWN
         2          4          3             -200         1000

11/06/09 11/06/03 11/06/07 11/06/09 UP
         2          4          7             -200         1000

11/06/07 11/06/03 11/06/07 11/06/09 DOWN
         2          4          5             -400          800

11/06/08 11/06/03 11/06/07 11/06/09 UP
         2          4          6             -300          900

11/06/06 11/06/03 11/06/07 11/06/09 DOWN
         2          4          4             -300          900
```

Przeanalizujmy 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 datą 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
```

Przeanalizujmy 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 wzorzec 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 wzorzec 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 odnalezionego 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 znalezionego 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

G

L

Ł

M

N

O

P

R

S

Ś

T

U

W

X

Z

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
Informatyka w najlepszym wydaniu

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

ISBN 978-83-246-9922-3
9 788324 699223
cena: 89,00 zł

Oracle Press