1 Wprowadzenie

Jednym z trendów rozwojowych dominujących w ostatnich latach w systemach relacyjnych baz danych jest integracja mechanizmów analitycznych (OLAP) z funkcjonalnością systemu zarządzania bazą danych (SZBD). Rozwój ten ukierunkowany jest na osiągnięcie dwóch podstawowych celów. Po pierwsze, możliwość realizacji przetwarzania analitycznego w ramach operacyjnej bazy danych pozwala uniknąć kosztów replikacji jej zawartości do specjalizowanych wielowymiarowych magazynów (hurtowni) danych. Po drugie, systemy relacyjnych baz danych stają się poważną platformą implementacji środowisk magazynów (hurtowni) danych.

Podstawowym składnikiem analitycznego rozszerzenia funkcjonalności SZBD jest rozbudowa implementacji języka zapytań SQL. Nowe elementy tego języka pozwalają programistom aplikacji OLTP na łatwe ich rozszerzanie o moduły wspomagania podejmowania decyzji. Coraz większa liczba producentów SZBD dostarcza funkcje statystyczne, funkcje agregujące z ruchomym oknem, funkcje rankingowe itp.

W artykule przedstawiono analityczne rozszerzenia języka SQL na przykładzie SZBD Oracle8i oraz Oracle9i. Struktura artykułu jest następująca. W rozdziale drugim opisano dodatkowe operatory agregujące, służące do uproszczenia zapisu i wydajności zapytań wykorzystujących klauzulę GROUP BY. W rozdziale trzecim zaprezentowano nowe funkcje SQL, umożliwiające i ułatwiające realizację złożonych zapytań analitycznych. Rozdział czwarty stanowi podsumowanie.

2 Operatory agregujące SQL

2.1 Operator ROLLUP

Operator ROLLUP stanowi rozszerzenie klauzuli GROUP BY zapytania SELECT, pozwalające na wyznaczanie wartości funkcji grupowych na rosnących poziomach agregacji. Dzięki temu, agregaty szczegółowe i agregaty ogólne mogą być wyznaczane przy użyciu jednego zapytania, a nie kilku oddzielnych zapytań, co w efekcie pozwala istotnie polepszyć wydajność aplikacji. Zapytanie wykorzystujące operator ROLLUP jest semantycznie równoważne złożeniu wielu zapytań, stosujących grupowanie według zawężającej się listy kolumn. Poniżej przedstawiono przykład zapytania, które w jednym przebiegu wyznacza sumy sprzedaży w ramach produktów w regionach, w ramach regionów oraz w ramach całej firmy.

```
select region, produkt, sum(kwota)
from sprzedaz
group by rollup(region, produkt)
```

```
REGION         PRODUKT      SUM(KWOTA)
-------------- ------------ ----------
Mazowsze       Oleje             23000
Mazowsze       Opony             97000
Mazowsze       Reflektory        97000
Mazowsze       Tłumiki         1150000
Mazowsze                       1367000
Wielkopolska   Oleje            230000
Wielkopolska   Opony            152000
Wielkopolska   Reflektory        48000
Wielkopolska   Tłumiki          230000
Wielkopolska                    660000
                                2027000
```

Uzyskanie identycznego wyniku bez użycia operatora ROLLUP wymagałoby wykonania trzech zapytań z następującymi klauzulami grupowania: „GROUP BY REGION, PRODUKT”, „GROUP BY REGION” i „GROUP BY NULL”.

2.2 OPERATOR CUBE

Operator CUBE również stanowi rozszerzenie klauzuli GROUP BY zapytania SELECT. Zapytanie wykorzystujące operator CUBE jest semantycznie równoważne złożeniu wielu zapytań, stosujących grupowanie według wszystkich kombinacji podanych kolumn. Poniżej przedstawiono przykład zapytania, które w jednym przebiegu wyznacza sumy sprzedaży w ramach produktów w regionach, w ramach regionów, w ramach produktów oraz w ramach całej firmy.

```
select region, produkt, sum(kwota)
from sprzedaz
group by cube(region, produkt)
```

```
REGION         PRODUKT      SUM(KWOTA)
-------------- ------------ ----------
Mazowsze       Oleje             23000
Mazowsze       Opony             97000
Mazowsze       Reflektory        97000
Mazowsze       Tłumiki         1150000
Mazowsze                       1367000
Wielkopolska   Oleje            230000
Wielkopolska   Opony            152000
Wielkopolska   Reflektory        48000
Wielkopolska   Tłumiki          230000
Wielkopolska                    660000
               Oleje            253000
               Opony            249000
               Reflektory       145000
               Tłumiki         1380000
                               2027000
```

Uzyskanie identycznego wyniku bez użycia operatora CUBE wymagałoby wykonania trzech zapytań z następującymi klauzulami grupowania: „GROUP BY REGION, PRODUKT”, „GROUP BY REGION”, „GROUP BY PRODUKT” i „GROUP BY NULL”.

2.3 OPERATOR GROUPING SETS

Operator GROUPING SETS umożliwia realizację wielu zadanych schematów grupowania w jednym przebiegu zapytania SELECT. Jego argumentem jest lista wielu zestawów kolumn grupujących. Poniżej przedstawiono przykład zapytania, które wyznacza wartości funkcji grupowej w ramach produktów w regionach oraz w ramach produktów.

```
select region, produkt, sum(kwota)
from sprzedaz
group by grouping sets((region, produkt),(produkt))
```

```
REGION         PRODUKT      SUM(KWOTA)
-------------- ------------ ----------
Mazowsze       Oleje             23000
Wielkopolska   Oleje            230000
               Oleje            253000
Mazowsze       Opony             97000
Wielkopolska   Opony            152000
               Opony            249000
Mazowsze       Reflektory        97000
```

```
Wielkopolska   Reflektory        48000
               Reflektory       145000
Mazowsze       Tłumiki         1150000
Wielkopolska   Tłumiki          230000
               Tłumiki         1380000
```

Uzyskanie identycznego wyniku bez użycia operatora GROUPING SETS wymagałoby wykonania dwóch zapytań z następującymi klauzulami grupowania: „GROUP BY REGION, PRODUKT" i „GROUP BY PRODUKT".

3 Funkcje analityczne SQL

3.1 PARTYCJE OBLICZENIOWE

Tradycyjne funkcje grupowe, jak na przykład SUM(), AVG() i COUNT(), wyliczają swoje wartości dla grup rekordów definiowanych przez klauzulę GROUP BY zapytania SELECT. Dla każdej grupy, funkcja grupowa zwraca pojedynczą wartość wynikową. Jednak dzięki rozszerzeniu składni języka SQL wprowadzonemu w późnych wydaniach wersji 8i SZBD Oracle, możliwe jest wyznaczanie wartości funkcji grupowej oddzielnie dla każdego rekordu grupy (a nie tylko raz dla całej grupy). W celu skorzystania z takiego rozwiązania, klauzula GROUP BY musi zostać zastąpiona wyrażeniem PARTITION osadzonym w następującej strukturze:

funkcja_grupowa() over (partition by *kolumna*)

gdzie:

- *funkcja()* to tradycyjna funkcja grupowa,
- *kolumna()* to kolumna (lub wyrażenie) grupująca rekordy w celu wyliczenia funkcji grupowej.

Wyrażenie PARTITION umożliwia prostszy zapis wielu zapytań analitycznych, które w przeszłości koniecznie wymagały stosowania podzapytań. Poniżej przedstawiono przykład zastosowania wyrażenia PARTITION w celu zwrócenia wartości funkcji grupowej SUM() oddzielnie dla każdego rekordu

```
select region, produkt, kwota,
       sum(kwota) over (partition by region) sum_kwota,
       round(100*kwota/(sum(kwota) over (partition by region))) "UDZIAŁ%"
from sprzedaz
order by region, produkt
```

```
REGION         PRODUKT           KWOTA  SUM_KWOTA    UDZIAŁ%
-------------- ----------- ---------- ---------- ----------
Mazowsze       Oleje            23000    1367000          2
Mazowsze       Opony            97000    1367000          7
Mazowsze       Reflektory       97000    1367000          7
Mazowsze       Tłumiki        1150000    1367000         84
Wielkopolska   Oleje           230000     660000         35
Wielkopolska   Opony           152000     660000         23
Wielkopolska   Reflektory       48000     660000          7
Wielkopolska   Tłumiki         230000     660000         35
```

Z wyniku powyższego zapytania wynika na przykład, że sprzedaż tłumików stanowi na Mazowszu 84% ogólnej sprzedaży.

Z analogicznej składni możemy korzystać także w przypadku nowej funkcji grupowej: RATIO_TO_REPORT(). Funkcja ta zwraca proporcję wartości wybranej kolumny (wyrażenia) w bieżącym rekordzie w stosunku do sumy tej kolumny w całej partycji obliczeniowej. RATIO_TO_REPORT() pozwala zatem w jeszcze prostszy sposób uzyskać rezultat podobny do przedstawionego w poprzednim przykładzie.

Należy również wspomnieć, że grupowanie rekordów przy użyciu wyrażenia PARTITION nie wyklucza obecności klauzuli GROUP BY w zapytaniu. Wówczas grupowanie posiada charakter dwustopniowy. W pierwszym kroku następuje grupowanie rekordów według klauzuli GROUP BY oraz wyliczanie wartości tradycyjnych funkcji agregujących. W drugim kroku powstałe grupy podlegają kolejnemu grupowaniu, tym razem zgodnie z wyrażeniem PARTITION, a na wyniku tego grupowania wyliczane są wartości nowych funkcji agregujących.

3.2 RUCHOME OKNO OBLICZENIOWE

Wartości funkcji grupowych mogą być również wyznaczane w oparciu o grupy rekordów przesuwające się wraz z rekordem bieżącym. Grupy takie nazywane są oknami i mogą być definiowane przy pomocy wyrażeń ROWS lub RANGE. Wyrażenie ROWS definiuje tzw. okno fizyczne, którego rozmiar określony jest liczbą rekordów, natomiast wyrażenie RANGE definiuje tzw. okno logiczne, którego rozmiar określony jest warunkiem selekcji rekordów.

Do zdefiniowania ruchomego okna logicznego służy wyrażenie RANGE o następującej składni ogólnej:

funkcja() over ([partition by *kol1*] order by *kol2* [desc] range between *wyr1* and *wyr2*)

gdzie:
- *funkcja()* to funkcja grupowa ,
- *kol1* to kolumna (lub wyrażenie) grupująca rekordy w grupy, wewnątrz których przesuwa się okno; pominięcie *partition* oznacza, że okno przesuwa się przez całą tabelę,
- *kol2* to kolumna (lub wyrażenie) porządkująca rekordy wewnątrz grupy,
- *desc* powoduje porządek przesuwania okna od wartości największej do najmniejszej,
- *wyr1* określa położenie początku okna,
- *wyr2* określa położenie końca okna.

Wyrażenie *wyr1* przyjmuje następującą postać:
- UNBOUNDED PRECEDING – okno rozpoczyna się od pierwszego rekordu grupy,
- CURRENT ROW – okno rozpoczyna się od wartości pobranej z aktualnego rekordu,
- n PRECEDING – okno rozpoczyna się od rekordu posiadającego w kolumnie *kol2* wartość o *n* mniejszą (dla *desc* większą) niż rekord bieżący,
- INTERVAL ‘n’ DAYS | MONTHS | YEARS PRECEDING – okno rozpoczyna się od rekordu posiadającego w kolumnie *kol2* wartość o *n* dni/miesięcy/lat mniejszą (dla *desc* większą) niż rekord bieżący.

Wyrażenie *wyr2* przyjmuje następującą postać:
- UNBOUNDED FOLLOWING – okno kończy się na ostatnim rekordzie grupy,
- CURRENT ROW – okno kończy się na wartości pobranej z aktualnego rekordu,
- n FOLLOWING – okno kończy się na rekordzie posiadającym w kolumnie *kol2* wartość o *n* większą (dla *desc* mniejszą) niż rekord bieżący,
- INTERVAL ‘n’ DAYS | MONTHS | YEARS FOLLOWING – okno kończy się na rekordzie posiadającym w kolumnie *kol2* wartość o *n* dni/miesięcy/lat większą (dla *desc* mniejszą) niż rekord bieżący.

Podobnie definiowane jest okno fizyczne, lecz zamiast słowa kluczowego RANGE stosujemy słowo kluczowe ROWS. Ponadto, okno fizyczne nie wykorzystuje wyrażeń INTERVAL.

Przedstawmy praktyczny przykład użycia ruchomego okna obliczeniowego. Poniższe zapytanie wylicza wartości narastających sum sprzedaży w kolejnych miesiącach w ramach każdego roku.

```
select rok, kwartal, kwota,
       sum(kwota) over (partition by rok order by kwartal
           range between unbounded preceding and current row) as
kwota_kum
from sprzedaz_kwartalna
order by rok, kwartal;
```

ROK	KWARTAL	KWOTA	**KWOTA_KUM**
2000	1	250000	**250000**
2000	2	100000	**350000**
2000	3	130000	**480000**
2000	4	70000	**550000**
2001	1	110000	**110000**
2001	2	130000	**240000**
2001	3	100000	**340000**
2001	4	120000	**460000**

Z wyniku powyższego zapytania wynika na przykład, że w trzecim kwartale roku 2000 kwota rocznej sprzedaży osiągnęła 480000.

W kolejnym przykładzie, dla każdego kwartału wyświetlana jest średnia kwota sprzedaży za trzy ostatnie kwartały danego roku (wraz z bieżącym kwartałem). Zauważmy, że dla początkowych kwartałów wartości wynikowe wyliczane są w oparciu o niepełne okno.

```
select rok, kwartal, kwota,
       avg(kwota) over (partition by rok order by kwartal
           range between 3 preceding and current row) as avg_3
from sprzedaz_kwartalna
order by rok, kwartal;
```

ROK	KWARTAL	KWOTA	**AVG_3**
2000	1	250000	**250000**
2000	2	100000	**175000**
2000	3	130000	**160000**
2000	4	70000	**137500**
2001	1	110000	**110000**
2001	2	130000	**120000**
2001	3	100000	**113333**
2001	4	120000	**115000**

Zapytania stosujące ruchome okno obliczeniowe mogą również posługiwać się nowymi funkcjami grupowymi: FIRST_VALUE() i LAST_VALUE(). Funkcje te zwracają wartości wybranej kolumny odpowiednio z pierwszego i ostatniego rekordu okna obliczeniowego.

Podobnie do funkcji przedstawionych powyżej działają funkcje LAG() i LEAD(), które, mimo, iż wewnętrznie korzystają z ruchomego okna obliczeniowego, to jednak nie wymagają definiowania wyrażenia RANGE/ROWS. Funkcja LAG() zwraca wartość wybranej kolumny z rekordu poprzedzającego bieżący rekord o zadaną liczbę rekordów, natomiast funkcja LEAD() zwraca wartość wybranej kolumny z rekordu występującego zadaną liczbę rekordów za bieżącym rekordem. Poniżej przedstawiono przykład zapytania wykorzystującego funkcję LAG() do wyświetlenia kwartalnego wzrostu sprzedaży.

```
select rok, kwartal, kwota,
      lag(kwota,1) over (order by rok, kwartal) as lag,
      kwota - (lag(kwota,1) over (order by rok, kwartal)) as wzrost
from sprzedaz_kwartalna
order by rok, kwartal;

       ROK    KWARTAL      KWOTA        LAG     WZROST
---------- ---------- ---------- ---------- ----------
      2000          1     250000
      2000          2     100000     250000    -150000
      2000          3     130000     100000      30000
      2000          4      70000     130000     -60000
      2001          1     110000      70000      40000
      2001          2     130000     110000      20000
      2001          3     100000     130000     -30000
      2001          4     120000     100000      20000
```

Użyta w powyższym zapytaniu funkcja LAG() otrzymuje dwa argumenty. Pierwszy to nazwa kolumny, której wartość zostanie zwrócona, drugi to odległość rekordu z którego zostanie odczytana ta wartość, licząc od rekordu bieżącego.

3.3 FUNKCJE RANKINGOWE

Funkcje rankingowe umożliwiają wyznaczanie położenia rekordu w odniesieniu do pozostałych rekordów grupy względem wybranej funkcji porządkującej. Implementacja języka SQL w SZBD Oracle9i zawiera sześć funkcji tego typu: RANK(), DENSE_RANK(), CUME_DIST(), PERCENT_RANK(), NTILE() i ROW_NUMBER(). Ogólna składnia użycia funkcji rankingowych jest następująca:

funkcja() over ([partition by *kolumna1*] order by *kolumna2* [desc] [nulls first | last])

gdzie:

- *funkcja()* to nazwa funkcji rankingowej,
- *kolumna1* to kolumna (lub wyrażenie) grupująca rekordy uczestniczące w rankingu; brak wyrażenia *partition* oznacza, że rankingowi podlegają wszystkie rekordy tabeli,
- *kolumna2* to kolumna (lub wyrażenie) porządkująca rekordy wewnątrz grupy,
- *desc* powoduje porządek rankingu od wartości największej do najmniejszej,
- *nulls first* powoduje, że wartości puste trafiają na początek rankingu,
- *nulls last* powoduje, że wartości puste trafiają na koniec rankingu.

Funkcja RANK() zwraca numer pozycji rankingowej rekordu w ramach grupy rekordów, przy czym wystąpienie jednakowej pozycji rankingowej dla wielu rekordów powoduje powstanie przerw w numeracji. Funkcja DENSE_RANK() działa podobnie jak funkcja RANK(), jednak nie powoduje powstawania przerw w numeracji w powyższej sytuacji. Poniżej przedstawiono przykład zapytania wykorzystującego funkcje RANK() i DENSE_RANK() do wyświetlenia numerów pozycji rankingowych produktów według sum kwot ich sprzedaży, w ramach regionów sprzedaży.

```
select region, produkt, kwota,
  rank() over (partition by region order by kwota desc) as rank,
  dense_rank() over (partition by region order by kwota desc) as
dense_rank
from sprzedaz
order by region, produkt
```

```
REGION          PRODUKT            KWOTA       RANK DENSE_RANK
--------------- ---------- ---------- ---------- ----------
Mazowsze        Oleje           23000          4          3
Mazowsze        Opony           97000          2          2
Mazowsze        Reflektory      97000          2          2
Mazowsze        Tłumiki       1150000          1          1
Wielkopolska    Oleje          230000          1          1
Wielkopolska    Opony          152000          3          2
Wielkopolska    Reflektory      48000          4          3
Wielkopolska    Tłumiki        230000          1          1
```

Z wyniku powyższego zapytania wynika na przykład, że najlepiej sprzedającymi się produktami są na Mazowszu *tłumiki*, a w Wielkopolsce równocześnie *oleje* i *tłumiki*. Zwróćmy też uwagę na to, że według funkcji RANK(), *oleje* na Mazowszu znajdują się na pozycji czwartej (były dwie drugie pozycje), a według funkcji DENSE_RANK() na pozycji trzeciej.

Podobnie działają pozostałe funkcje rankingowe. Funkcja CUME_DIST() zwraca wartość pozycji rankingowej wyrażoną liczbą z przedziału (0;1> - liczba ta rozumiana jest jako procent rekordów poprzedzających dany rekord w rankingu z uwzględnieniem bieżącego rekordu. Funkcja PERCENT_RANK() stosuje podobne podejście, lecz nie uwzględnia bieżącego rekordu w określaniu procentu rekordów poprzedzających. Funkcja NTILE() zwraca numer kolejnej n-tki (trójki, piątki, dziesiątki, itp.) rankingu, do której należy bieżący rekord. Jest to funkcja niedeterministyczna (dla identycznych danych może zwracać rózne wyniki). Funkcja ROW_NUMBER() bazuje na rankingu w celu wyznaczenia liczby porządkowej dla każdego rekordu. Jest to również funkcja niedeterministyczna. Poniżej przedstawiono przykład zapytania wykorzystującego omówione funkcje rankingowe.

```
select kwota,
       rank() over (order by kwota desc) as rank,
       cume_dist() over (order by kwota desc) as cume_dist,
       percent_rank() over (order by kwota desc) as percent_rank,
       ntile(3) over (order by kwota desc) as ntile_3,
       row_number() over (order by kwota desc) as row_number
from sprzedaz
order by kwota
```

```
     KWOTA       RANK  CUME_DIST PERCENT_RANK    NTILE_3 ROW_NUMBER
---------- ---------- ---------- ------------ ---------- ----------
     23000          8          1            1          3          8
     48000          7       ,875   ,857142857          3          7
     97000          5        ,75   ,571428571          2          5
     97000          5        ,75   ,571428571          2          6
    152000          4         ,5   ,428571429          2          4
    230000          2       ,375   ,142857143          1          2
    230000          2       ,375   ,142857143          1          3
   1150000          1       ,125            0          1          1
```

Z wyniku powyższego zapytania wynika na przykład, że w rankingu sprzedaży, kwota 15200 jest poprzedzana przez 42% wszystkich kwot z pominięciem jej samej (PERCENT_RANK) oraz że w drugiej trójce sprzedaży znajdują się kwoty 152000 i 97000 (NTILE).

Odrębną grupę stanowią tzw. odwrotne funkcje rankingowe: PERCENTILE_DISC() i PERCENTILE_CONT(). Zwracają one wartość rekordu, który w rankingu reprezentowany jest przez zadany wynik funkcji CUME_DIST(). PERCENTILE_DISC() to funkcja krokowa wykorzystująca wartości dyskretne, a PERCENTILE_CONT() jest interpolacyjną funkcją ciągłą.

Poniżej przedstawiono przykład zapytania wykorzystującego funkcję PERCENTILE_DISC() w celu znalezienia wartości środkowej w kolumnie kwota. Zwróćmy uwagę na drobną zmianę składni wyrażenia: WITHIN GROUP zamiast OVER.

```
select percentile_disc(0.5) within group
       (order by kwota desc) as percentile
from sprzedaz
order by kwota

PERCENTILE
----------
    152000
```

W wyniku powyższego zapytania uzyskano 15200, gdyż właśnie dla tej wartości poprzednie zapytanie zwróciło CUME_DIST()=0.5.

3.4 FUNKCJE STATYSTYCZNE

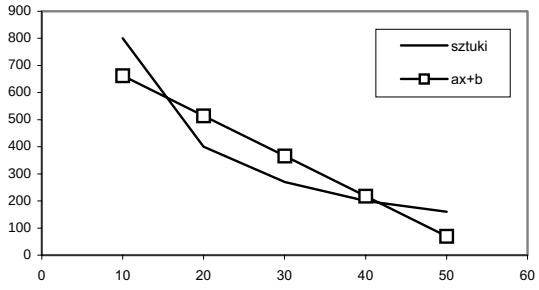
Implementacja języka SQL w SZBD Oracle9i zawiera również funkcje regresji liniowej: REGR_COUNT(), REGR_AVGX(), REGR_AVGY(), REGR_SLOPE(), REGR_INTERCEPT(), REGR_R2(), REGR_SXX(), REGR_SYY(), REGR_SXY(). Funkcja REGR_COUNT() zwraca liczbę punktów danych. Funkcje REGR_AVGY() i REGR_AVGX() zwracają wartości średnie dla, odpowiednio, zmiennej zależnej i zmiennej niezależnej. Funkcje REGR_SLOPE() i REGR_INTERCEPT() wyznaczają, odpowiednio, współczynnik kierunkowy prostej regresji oraz jej przecięcie z osią *y*. Funkcja REGR_R2() zwraca wartość współczynnika dopasowania prostej. Poniżej przedstawiono wynik zapytania wyznaczającego prostą regresji dla danych reprezentujących zależność sprzedaży od ceny produktu. Na jej podstawie analityk może np. szacować sprzedaż produktu oferowanego na wybranym poziomie cenowym.

```
select * from sprzedaz_wg_cen;

PRODUKT           CENA     SZTUKI
---------- ---------- ----------
HDD1525SX           50        160
HDD1525SX           40        200
HDD1525SX           30        270
HDD1525SX           20        400
HDD1525SX           10        800

select regr_slope(sztuki,cena) as
a,
   regr_intercept(sztuki,cena) as b
from sprzedaz_wg_cen

A      B
------- ------
-14,8    810
```



3.5 GRUPOWE FUNKCJE PROGNOSTYCZNE

Funkcje prognostyczne znajdują zastosowania w analizie typu „co by się stało, gdyby...?”. Przykładowo, przy użyciu tego typu funkcji, moglibyśmy zapytać o położenie hipotetycznego rekordu w istniejącym rankingu, gdyby został on dodany do tabeli. Prognostyczne funkcje SQL wykorzystują wyrażenie WITHIN GROUP, zaprezentowane wcześniej na przykładzie odwrotnych

funkcji rankingowych. Poniżej przedstawiono przykład zapytania, które wyświetla numer pozycji rankingowej, na której znalazłaby się kwota 100000, gdyby została dodana do tabeli SPRZEDAZ.

```
select rank(100000) within group (order by kwota desc) as rank
from sprzedaz
order by kwota;

  RANK
------
     5
```

Oprócz funkcji RANK(), charakter prognostyczny mogą posiadać również funkcje DENSE_RANK(), PERCENT_RANK() i CUME_DIST().

4 Podsumowanie

Zaprezentowane rozszerzenia analityczne języka SQL, dostępne w SZBD Oracle9i, umożliwiają programistom zwiększenie czytelności i wydajności kodu analitycznego aplikacji operujących na magazynie (hurtowni) danych opartym na relacyjnej bazie danych. Warto podkreślić znaczenie nowych operatorów agregujących, umożliwiających wyliczanie wartości funkcji grupowych na różnych poziomach grupowania w jednym przebiegu zapytania. Godne uwagi jest też wprowadzenie mechanizmu ruchomego okna obliczeniowego, pozwalającego na daleko idące usprawnienie wydajności zapytań, dotychczas wykorzystujących złożone podzapytania skorelowane.

5 Literatura

[1] Oracle9i R2 Data Warehousing Guide, dokumentacja techniczna