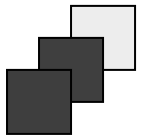
Rozdział 14
Funkcje analityczne

Operatory ROLLUP i CUBE, funkcja GROUPING, funkcje porządkujące (ranking), „okienkowe”, raportujące, statystyczne, funkcje LAG/LAD

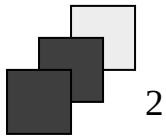


ROLLUP

- **Polecenie ROLLUP jest rozszerzeniem klauzuli GROUP BY, które pozwala wyliczać podsumowania częściowe i ogólne. Polecenie ROLLUP służy do konstruowania pół-kostek danych.**

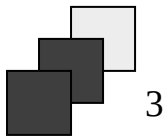
```
SELECT atrybuty grupujące, agregaty
FROM ...
WHERE ...
GROUP BY ROLLUP (atrybuty grupujące);
```

```
select etat, rok, count(*)
from pracownicy
group by rollup(etat, rok)
```



ROLLUP - wynik

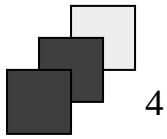
ETAT	ROK	COUNT(*)
ADIUNKT	1977	1
ADIUNKT	1985	1
ADIUNKT		2
ASYSTENT	1992	2
ASYSTENT	1993	2
ASYSTENT		4
DYREKTOR	1968	1
DYREKTOR		1
PROFESOR	1968	1
PROFESOR	1973	1
PROFESOR	1975	1
PROFESOR	1977	1
PROFESOR		4
STAZYSTA	1993	1
STAZYSTA	1994	1
STAZYSTA		2
SEKRETARKA	1985	1
SEKRETARKA		1
		14



Wartości puste w operatorach ROLLUP i CUBE

Puste wartości kolumn w przypadku zapytań z funkcjami ROLLUP i CUBE mogą oznaczać operacje podsumowania wykonywaną na określonym wymiarze lub standardową wartość pustą kolumny, np. we fragmencie poniżej trzeci wiersz z wartością pustą może oznaczać podsumowanie dwóch powyższych jak i np. podsumowanie dla departamentu o pustej nazwie.

ETAT	ROK	COUNT(*)
ADIUNKT	1977	1
ADIUNKT	1985	1
ADIUNKT	NULL	2
ASYSTENT	1992	2
ASYSTENT	1993	2
ASYSTENT	NULL	4
DYREKTOR	1968	1
DYREKTOR	NULL	1
PROFESOR	1968	1
...		

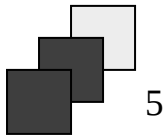


Funkcja GROUPING

Funkcja GROUPING pozwala rozróżnić wiersze z wartościami pustymi od wierszy podsumowań. Jeżeli wartość pusta oznacza podsumowanie GROUPING zwraca wartość 1, w przeciwnym przypadku 0.

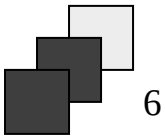
```
SELECT ..., GROUPING(atrybut grupujący)
FROM ...
WHERE ...
GROUP BY { ROLLUP | CUBE } (atrybuty grupujące);
```

```
select   etat, grouping(etat),
         rok, grouping(rok),
         count(*)
from pracownicy
group by rollup(etat, rok)
```



Funkcja GROUPING - wynik

ETAT	GROUPING(ETAT)	ROK	GROUPING(ROK)	COUNT(*)
ADIUNKT	0	1977	0	1
ADIUNKT	0	1985	0	1
ADIUNKT	0		1	2
ASYSTENT	0	1992	0	2
ASYSTENT	0	1993	0	2
ASYSTENT	0		1	4
DYREKTOR	0	1968	0	1
DYREKTOR	0		1	1
PROFESOR	0	1968	0	1
PROFESOR	0	1973	0	1
PROFESOR	0	1975	0	1
PROFESOR	0	1977	0	1
PROFESOR	0		1	4
STAZYSTA	0	1993	0	1
STAZYSTA	0	1994	0	1
STAZYSTA	0		1	2
SEKRETARKA	0	1985	0	1
SEKRETARKA	0		1	1
	1		1	14

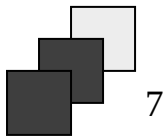


Funkcja GROUPING

rozróżnienie wartości pustych od podsumowań

```
select   decode(grouping(etat),1,'Wszystkie etaty',etat) etat,
         decode(grouping(rok),1,'Wszystkie lata',rok)  rok,
         count(*)
from pracownicy
group by rollup(etat, rok)
```

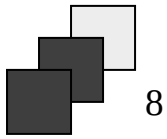
ETAT	ROK	COUNT(*)
ADIUNKT	1977	1
ADIUNKT	1985	1
ADIUNKT	Wszystkie lata	2
ASYSTENT	1992	2
ASYSTENT	1993	2
ASYSTENT	Wszystkie lata	4
DYREKTOR	1968	1
DYREKTOR	Wszystkie lata	1
PROFESOR	1968	1
PROFESOR	1973	1
PROFESOR	1975	1
PROFESOR	1977	1
PROFESOR	Wszystkie lata	4
STAZYSTA	1993	1
STAZYSTA	1994	1
STAZYSTA	Wszystkie lata	2
SEKRETARKA	1985	1
SEKRETARKA	Wszystkie lata	1
Wszystkie etaty	Wszystkie lata	14



Częściowa operacja ROLLUP

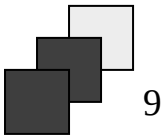
```
SELECT atrybuty grupujące, agregaty
FROM ...
WHERE ...
GROUP BY atrybut, ROLLUP (atrybuty grupujące);
```

```
select    etat,
          decode(grouping(rok),1,'Wszystkie lata',rok)  rok,
          count(*)
from pracownicy
group by etat, rollup(rok)
```



Częściowa operacja ROLLUP - wynik

ETAT	ROK	COUNT(*)
ADIUNKT	1977	1
ADIUNKT	1985	1
ADIUNKT	Wszystkie lata	2
ASYSTENT	1992	2
ASYSTENT	1993	2
ASYSTENT	Wszystkie lata	4
DYREKTOR	1968	1
DYREKTOR	Wszystkie lata	1
PROFESOR	1968	1
PROFESOR	1973	1
PROFESOR	1975	1
PROFESOR	1977	1
PROFESOR	Wszystkie lata	4
STAZYSTA	1993	1
STAZYSTA	1994	1
STAZYSTA	Wszystkie lata	2
SEKRETARKA	1985	1
SEKRETARKA	Wszystkie lata	1

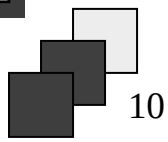


CUBE

Operator CUBE tworzy podsumowania dla wszystkich możliwych kombinacji grupowanych kolumn. W terminologii analiz wielowymiarowych, CUBE generuje podsumowania częściowe i ogólne tabeli faktów dla wszystkich możliwych wymiarów.

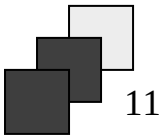
```
SELECT atrybuty grupujące, agregaty
FROM ...
WHERE ...
GROUP BY CUBE (atrybuty grupujące);
```

```
select decode(grouping(etat),1,'Wszystkie etaty',etat) etat,
        decode(grouping(rok),1,'Wszystkie lata',rok) rok,
        count(*)
from pracownicy
group by cube(etat, rok)
```



Operacja CUBE - wynik

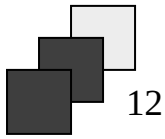
ETAT	ROK	COUNT(*)
Wszystkie etaty	Wszystkie lata	14
Wszystkie etaty	1968	2
Wszystkie etaty	1973	1
Wszystkie etaty	1975	1
Wszystkie etaty	1977	2
Wszystkie etaty	1985	2
Wszystkie etaty	1992	2
Wszystkie etaty	1993	3
Wszystkie etaty	1994	1
ADIUNKT	Wszystkie lata	2
ADIUNKT	1977	1
ADIUNKT	1985	1
ASYSTENT	Wszystkie lata	4
ASYSTENT	1992	2
ASYSTENT	1993	2
DYREKTOR	Wszystkie lata	1
DYREKTOR	1968	1
PROFESOR	Wszystkie lata	4
PROFESOR	1968	1
PROFESOR	1973	1
PROFESOR	1975	1
PROFESOR	1977	1
STAZYSTA	Wszystkie lata	2
STAZYSTA	1993	1
STAZYSTA	1994	1
SEKRETARKA	Wszystkie lata	1
SEKRETARKA	1985	1



Częściowa operacja CUBE

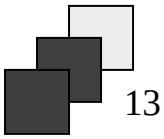
```
SELECT atrybuty grupujące, agregaty
FROM ...
WHERE ...
GROUP BY atrybut, CUBE (atrybuty grupujące);
```

```
select   etat,
         decode(grouping(rok),1,'Wszystkie lata',rok) rok,
         count(*)
from pracownicy
group by etat, cube(rok)
```



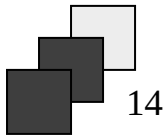
Częściowa operacja CUBE - wynik

ETAT	ROK	COUNT(*)
ADIUNKT	Wszystkie lata	2
ADIUNKT	1977	1
ADIUNKT	1985	1
ASYSTENT	Wszystkie lata	4
ASYSTENT	1992	2
ASYSTENT	1993	2
DYREKTOR	Wszystkie lata	1
DYREKTOR	1968	1
PROFESOR	Wszystkie lata	4
PROFESOR	1968	1
PROFESOR	1973	1
PROFESOR	1975	1
PROFESOR	1977	1
STAZYSTA	Wszystkie lata	2
STAZYSTA	1993	1
STAZYSTA	1994	1
SEKRETARKA	Wszystkie lata	1
SEKRETARKA	1985	1



Podsumowanie CUBE i ROLLUP

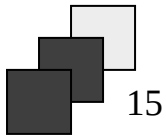
- **Operatory ROLLUP i CUBE działają niezależnie od zdefiniowanych w bazie danych hierarchii**
- **Przy operacjach ROLLUP i CUBE możemy wykorzystać również inne funkcje agregujące: COUNT, AVG, MIN, MAX, STDDEV i VARIANCE**
- **Klauzula HAVING w przypadku użycia CUBE lub ROLLUP odnosi się zarówno do elementów (wierszy) podsumowywanych jak do zwykłych**
- **Klauzula ORDER BY nie rozróżnia wierszy podsumowywanych i zwykłych. Do ich rozróżnienia trzeba użyć funkcji GROUPING**



Funkcje analityczne

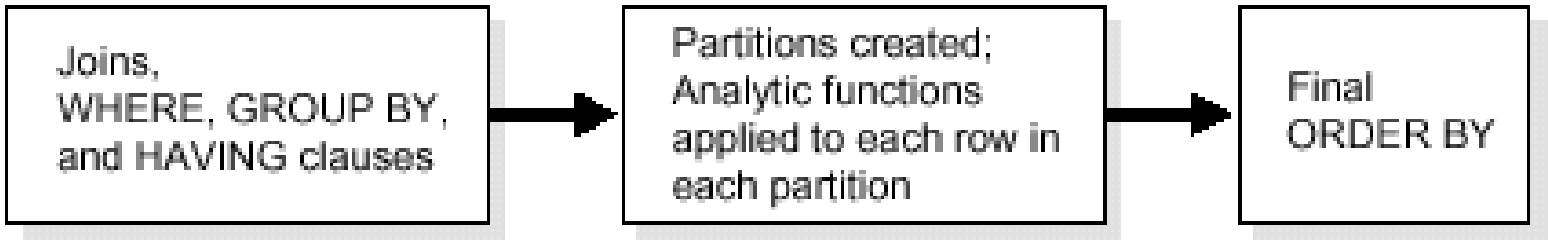
Funkcje analityczne dzielą się na następujące kategorie:

- **porządkujące – wyliczające ranking : RANK, DENSE_RANK, PERCENT_RANK, CUME_DIST, NTILE**
- **Okienkowe (funkcje okna) – wyznaczają wartości agregatów dla zdefiniowanych okien : AVG, SUM, MIN, MAX, COUNT, VARIANCE, STDDEV, FIRST_VALUE, LAST_VALUE**
- **raportujące – wyliczające udziały : AVG, SUM, MIN, MAX, COUNT, VARIANCE, STDDEV**
- **funkcje LAG/LEAD – znajdujące wartości w krotkach o określonej odległości od krotki bieżącej**
- **statystyczne – wyliczające zmiany poziomów, i inne statystyki**



Zagadnienia związane z funkcjami analitycznymi

Kolejność przetwarzania

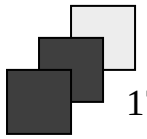


Partycje – funkcje analityczne pozwalają użytkownikom dzielić rezultat zapytania na grupy zwane partycjami. Partycje są tworzone po grupowaniu, dlatego nie są dozwolone dla nich funkcje agregujące. Podział na partycje może być oparty o dowolną kolumnę lub wyrażenie.

Zagadnienia związane z funkcjami analitycznymi

Okno – dla każdej krotki w partycji można zdefiniować ruchome okno danych. Okno takie określa zakres krotek używanych do wykonania obliczeń dla *bieżącej krotki.* Rozmiary okna mogą być oparte zarówno na fizycznej liczbie krotek, jak i na logicznym przedziale, takim jak np. czas.

Bieżąca krotka – każde obliczenie za pomocą funkcji analitycznej jest oparte na bieżącej krotce wewnątrz partycji. Bieżąca krotka służy jako punkt odniesienia określający początek i koniec okna.

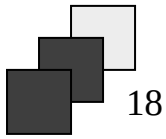


RANK i DENSE_RANK

Funkcje RANK i DENSE_RANK pozwalają uporządkowanie elementów w grupie, np. na znalezienie trzech najlepiej sprzedających się produktów.

```
RANK() OVER (
[ PARTITION BY <atrybut lub wyrażenie> [ , ... ] ]
ORDER BY <atrybut lub wyrażenie> [ , ... ]
[ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ] [ , ... ] )
```

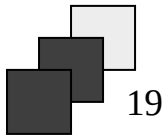
```
DENSE_RANK() OVER (
[ PARTITION BY <atrybut lub wyrażenie> [ , ... ] ]
ORDER BY <atrybut lub wyrażenie> [ , ... ]
[ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ] [ , ... ] )
```



RANK - przykład

```
select   id_zesp, count(*) as lp,
         rank()over(order by count(*)desc) as pozycja
from pracownicy
group by id_zesp
--order by pozycja
```

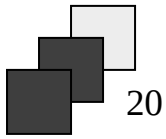
ID_ZESP	LP	POZYCJA
20	7	1
30	7	1
10	2	3
40	1	4



RANK - przykład 1/2

```
select id_zesp, nazwisko, placa_pod, rank()over(partition by id_zesp
order by placa_pod) as pozycja
from pracownicy
order by id_zesp, pozycja
```

ID_ZESP	NAZWISKO	PLACA_POD	POZYCJA
10	MAREK	410,2	1
10	WEGLARZ	1730	2
20	MATYSIAK	371	1
20	JEZIERSKI	439,7	2
20	KONOPKA	480	3
20	KOSZLAJDA	590	4
20	KROLIKOWSKI	645,5	5
20	MORZY	830	6
20	BRZEZINSKI	960	7
30	ZAKRZEWICZ	208	1
30	BIALY	250	2
30	HAPKE	480	3
30	SLOWINSKI	1070	4
40	BLAZEWICZ	1350	1



RANK – przykład 2/2

Przy pomocy funkcji RANK i DENSE_RANK można dokonywać selekcji pozycji wg wyznaczonego przez nie porządku.

```
SELECT * FROM (
select id_zesp, nazwisko, placa_pod,
        rank()over(partition by id_zesp order by placa_pod) as pozycja
from pracownicy
order by id_zesp, pozycja)
WHERE pozycja=1;
```

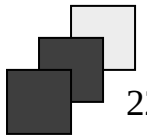
ID_ZESP	NAZWISKO	PLACA_POD	POZYCJA
10	MAREK	410,2	1
20	MATYSIAK	371	1
30	ZAKRZEWICZ	208	1
40	BLAZEWICZ	1350	1

DENSE_RANK - przykład

W przeciwieństwie do funkcji RANK() funkcja DENSE_RANK() nie pozostawia "dziur" w sekwencji rankingu.

```
select   id_zesp, count(*) as lp,
         rank()over(order by count(*)desc) as pozycja,
         dense_rank()over(order by count(*)desc) as pozycja_bez_dziur
from pracownicy
group by id_zesp
--order by pozycja_bez_dziur
```

ID_ZESP	LP	POZYCJA	POZYCJA_BEZ_DZIUR
20	7	1	1
30	4	1	1
10	2	3	2
40	1	4	3



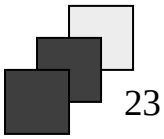
CUME_DIST

Funkcja CUME_DIST wylicza pozycję wyspecyfikowanej wartości w podzbiorze. Kolejność może być rosnąca lub malejąca. Wartości przyjmowane przez CUME_DIST są w zakresie od 0^+ do 1.

Definicja:

CUME_DIST(x) = liczba wartości w zbiorze S występujących przed wartością x (przy określonym porządku wartości) / liczbę wszystkich wartości w zbiorze S

```
CUME_DIST() OVER (
[ PARTITION BY <atrybut lub wyrażenie> [ , ... ] ]
ORDER BY <atrybut lub wyrażenie> [ , ... ]
[ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ] [ , ... ] )
```



CUME_DIST - przykład

```
select   nazwisko, placa_pod,
         cume_dist()over (order by placa_pod) as pozycja
from pracownicy
order by pozycja
```

```
 NAZWISKO          PLACA_POD    POZYCJA
--------------- ---------- ----------
ZAKRZEWICZ              208 ,071428571
BIALY                   250 ,142857143
MATYSIAK                371 ,214285714
MAREK                 410,2 ,285714286
JEZIERSKI             439,7 ,357142857
KONOPKA                 480         ,5
HAPKE                   480         ,5
KOSZLAJDA               590 ,571428571
KROLIKOWSKI           645,5 ,642857143
MORZY                   830 ,714285714
BRZEZINSKI              960 ,785714286
SLOWINSKI              1070 ,857142857
BLAZEWICZ              1350 ,928571429
WEGLARZ                1730           1
```

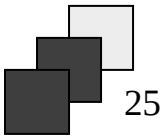
PERCENT_RANK

Funkcja PERCENT_RANK jest bardzo podobna do CUME_DIST. Różnica polega na tym, że PERCENT_RANK bierze pod uwagę pozycję przy określonym porządku a nie wartość. Wartości przyjmowane przez CUME_DIST są w zakresie od 0 do 1.

Definicja:

PERCENT_RANK(x) = ranking krotki x w partycji – 1 / liczba krotek w partycji – 1

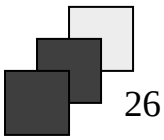
```
PERCENT_RANK() OVER (
[ PARTITION BY <atrybut lub wyrażenie> [ , ... ] ]
ORDER BY <atrybut lub wyrażenie> [ , ... ]
[ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ] [ , ... ] )
```



PERCENT_RANK - przykład

```
select   nazwisko, placa_pod,
         percent_rank()over (order by placa_pod) as pozycja
from pracownicy
order by pozycja
```

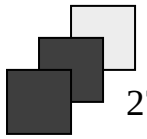
```
NAZWISKO          PLACA_POD    POZYCJA
--------------- ---------- ----------
ZAKRZEWICZ              208          0
BIALY                   250 ,076923077
MATYSIAK                371 ,153846154
MAREK                 410,2 ,230769231
JEZIERSKI             439,7 ,307692308
KONOPKA                 480 ,384615385
HAPKE                   480 ,384615385
KOSZLAJDA               590 ,538461538
KROLIKOWSKI           645,5 ,615384615
MORZY                   830 ,692307692
BRZEZINSKI              960 ,769230769
SLOWINSKI              1070 ,846153846
BLAZEWICZ              1350 ,923076923
WEGLARZ                1730          1
```



NTILE

NTILE dzieli krotki w uporządkowanej partycję na określoną liczbę grup *(bucket*) i przypisuje każdej z nich liczbę porządkową. Liczba krotek między grupami różni się co najwyżej o jedną krotkę. Funkcja NTILE służy przede wszystkim do wyliczania kwantyli, kwartyli i median. NTILE jest funkcją niedeterministyczną.

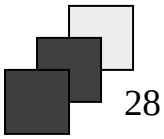
```
NTILE(N) OVER (
[ PARTITION BY <atrybut lub wyrażenie> [ , ... ] ]
ORDER BY <atrybut lub wyrażenie> [ , ... ]
[ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ] [ , ... ] )
```



NTILE - przykład

```
select   nazwisko, placa_pod,
         ntile(3)over (order by rok) as czesc
from pracownicy
order by czesc
```

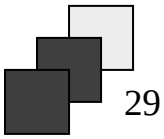
NAZWISKO	PLACA_POD	CZESC
WEGLARZ	1730	1
BRZEZINSKI	960	1
BLAZEWICZ	1350	1
MORZY	830	1
SLOWINSKI	1070	1
KROLIKOWSKI	645,5	2
KOSZLAJDA	590	2
MAREK	410,2	2
JEZIERSKI	439,7	2
HAPKE	480	2
MATYSIAK	371	3
KONOPKA	480	3
BIALY	250	3
ZAKRZEWICZ	208	3



ROW_NUMBER

Funkcja ROW_NUMBER przypisuje unikalny numer każdej krotce w partycji. Podobnie jak funkcja NTILE, ROW_NUMBER jest niedeterministyczna. Aby zapewnić deterministyczny wynik sortowanie musi się odbywać po kluczu unikalnym.

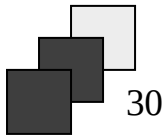
```
ROW_NUMBER() OVER (
[ PARTITION BY <atrybut lub wyrażenie> [ , ... ] ]
ORDER BY <atrybut lub wyrażenie> [ , ... ]
[ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ] [ , ... ] )
```

ROW_NUMBER - przykład

```
SELECT id_zesp, id_prac, nazwisko,
        ROW_NUMBER() OVER (PARTITION BY id_zesp
                ORDER BY id_prac) row_number_in_partition
FROM pracownicy
ORDER by id_zesp, id_prac
```

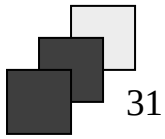
ID_ZESP	ID_PRAC	NAZWISKO	ROW_NUMBER_IN_PARTITION
10	100	WEGLARZ	1
10	180	MAREK	2
20	130	BRZEZINSKI	1
20	140	MORZY	2
20	150	KROLIKOWSKI	3
20	160	KOSZLAJDA	4
20	170	JEZIERSKI	5
20	190	MATYSIAK	6
20	220	KONOPKA	7
30	120	SLOWINSKI	1
....			



Funkcje okienkowe

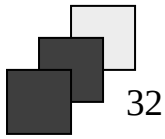
Funkcje okienkowe operują na uporządkowanym zbiorze krotek i dla każdej krotki obliczają wartość agregowaną dla „okna”, którego środkiem jest bieżąca krotka. Rozmiary okna mogą być nieograniczone lub ograniczone. Podobnie jak w przypadku funkcji rankingowych zbiór danych może być najpierw podzielony na partycje.

Rodzina funkcji okienkowych to w rzeczywistości rozszerzona składnia i funkcjonalność klasycznych funkcji agregujących SUM, AVG, MIN, MAX, STDDEV, VARIANCE, COUNT, FIRST_VALUE i LAST_VALUE.



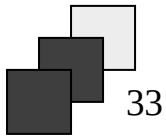
Cechy funkcji okienkowych

- **Oprócz funkcji standardowych SUM, AVG, MIN, ... Jako funkcje okienkowe można wykorzystać funkcje analizy regresji VAR_SAMP, VAR_POP, STDDEV_SAMP, STDDEV_POP, COVAR_SAMP, COVAR_POP, REGR_SLOPE, REGR_INTERCEPT, REGR_R2, REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_SXX, REGR_SXY, REGR_SYY**
- **Każda funkcja może mieć klauzulę definiującą rozmiar okna (w przeciwnym wypadku okno jest nieograniczone – obejmuje wiersze od początku partycji do bieżącego wiersza)**
- **Każda funkcja ma własną klauzulę sortującą**
- **Funkcje agregujące w oknach są resetowane na granicy partycji**
- **Dla każdego wyrażenia występującego w funkcji można określać porządek wzrastający i malejący**
- **Wartości puste mogą być umieszczane na początku bądź końcu (niezależnie od porządku sortowania)**



Określanie rozmiarów okien

- **Fizyczne: wyrażone w liczbie krotek, dodatkowo słowa kluczowe**
  - **CURRENT ROW: okno rozpoczyna się lub kończy w bieżącym wierszu**
  - **UNBOUNDED PRECEDING: okno rozpoczyna się na pierwszej krotce partycji**
  - **UNBOUNDED FOLLOWING: okno kończy się na ostatniej krotce partycji**
- **Czasowe: wyrażone interwałem czasowym**
- **Zakresy wartości: wyrażone różnicą między wartością w bieżącej krotce i wartością poprzedzającą**

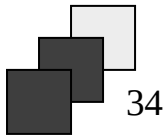


Funkcja okienkowa ze stałym oknem

Okno dla każdej krotki zaczyna się zawsze na początku bieżącej partycji – stałe jest osadzenie początku okna

```
select   nazwisko, placa_pod,
         sum(placa_pod)over(order by nazwisko
                  ROWS UNBOUNDED PRECEDING ) kumulacja
from pracownicy
order by nazwisko
```

NAZWISKO	PLACA_POD	KUMULACJA
BIALY	250	250
BLAZEWICZ	1350	1600
BRZEZINSKI	960	2560
HAPKE	480	3040
JEZIERSKI	439,7	3479,7
KONOPKA	480	3959,7
...		

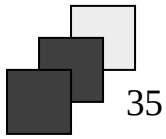


Funkcja okienkowa ze zmiennym oknem 1/2

Okno dla bieżącej krotki jest wyrażone jako odległość 365 dni wstecz od daty zatrudnienia danego pracownika

```
select   zatrudniony, nazwisko, placa_pod,
         avg(placa_pod)over(order by zatrudniony
                  RANGE 365 PRECEDING )  as srednia
from pracownicy
order by zatrudniony
```

```
ZATRUDNI NAZWISKO          PLACA_POD    SREDNIA
-------- --------------- ---------- ----------
68/01/01 WEGLARZ                1730       1730
68/07/01 BRZEZINSKI              960       1345
73/05/01 BLAZEWICZ              1350       1350
75/09/15 MORZY                   830        830
77/09/01 SLOWINSKI              1070     857,75
77/09/01 KROLIKOWSKI           645,5     857,75
...
```

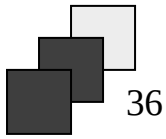


Funkcja okienkowa ze zmiennym oknem 2/2

Okno dla bieżącej krotki jest wyrażone jako dwie krotki wstecz od bieżącej krotki

```
select   zatrudniony, nazwisko, placa_pod,
         avg(placa_pod)over(order by zatrudniony
                  ROWS BETWEEN 2 PRECEDING
                  AND 0 FOLLOWING)  as srednia
from pracownicy
order by zatrudniony
```

```
ZATRUDNI NAZWISKO          PLACA_POD    SREDNIA
-------- --------------- ---------- ----------
68/01/01 WEGLARZ                1730       1730
68/07/01 BRZEZINSKI              960       1345
73/05/01 BLAZEWICZ              1350 1346,66667
75/09/15 MORZY                   830 1046,66667
77/09/01 SLOWINSKI              1070 1083,33333
77/09/01 KROLIKOWSKI           645,5      848,5
...
```

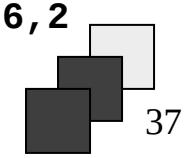


Funkcje FIRST_VALUE i LAST_VALUE

Funkcje FIRST_VALUE i LAST_VALUE pozwalają użytkownikowi odczytywać wartości z pierwszej i ostatniej krotki w oknie. Pozwalają one na analizę porównawczą z wartościami bazowymi dla danego okna.

```
select id_zesp, nazwisko, placa_pod,
        first_value(nazwisko)over(partition by id_zesp order by nazwisko) as fv,
        last_value(nazwisko)over(partition by id_zesp order by nazwisko) as lv,
        sum(placa_pod)over(partition by id_zesp order by nazwisko) kumulacja
from pracownicy
order by id_zesp, nazwisko
```

ID_ZESP	NAZWISKO	PLACA_POD	FV	LV	KUMULACJA
10	MAREK	410,2	MAREK	MAREK	410,2
10	WEGLARZ	1730	MAREK	WEGLARZ	2140,2
20	BRZEZINSKI	960	BRZEZINSKI	BRZEZINSKI	960
20	JEZIERSKI	439,7	BRZEZINSKI	JEZIERSKI	1399,7
20	KONOPKA	480	BRZEZINSKI	KONOPKA	1879,7
20	KOSZLAJDA	590	BRZEZINSKI	KOSZLAJDA	2469,7
20	KROLIKOWSKI	645,5	BRZEZINSKI	KROLIKOWSKI	3115,2
20	MATYSIAK	371	BRZEZINSKI	MATYSIAK	3486,2
...					

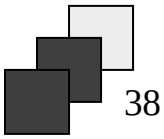


Funkcje raportujące

Podczas analizy często zachodzi konieczność porównywania wartości na różnych poziomach agregacji. Funkcje raportujące umożliwiają włączanie do krotek wartości pochodzących z różnych poziomów agregacji. Funkcje raportujące działają na poziomie okien i zwracają albo wartość agregatu dla całego okna albo wartość agregatu dla okna z wyłączeniem bieżącego okna.

Składnia

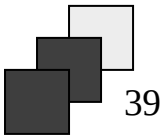
```
{SUM | AVG | MAX | MIN | COUNT | STDDEV | VARIANCE}
( [ ALL | DISTINCT ] { <value expression1> | * } )
OVER ( [ PARTITION BY <value expression2> [ ,... ] ] )
```



Przykłady funkcji raportujących

```
select nazwisko, placa_pod, id_zesp,
        avg(placa_pod)over(partition by id_zesp) as srednia_zespolu
from pracownicy
order by nazwisko
```

NAZWISKO	PLACA_POD	ID_ZESP	SREDNIA_ZESPOLU
BIALY	250	30	502
BLAZEWICZ	1350	40	1350
BRZEZINSKI	960	20	616,6
HAPKE	480	30	502
JEZIERSKI	439,7	20	616,6
KONOPKA	480	20	616,6
KOSZLAJDA	590	20	616,6
KROLIKOWSKI	645,5	20	616,6
MAREK	410,2	10	1070,1
MATYSIAK	371	20	616,6
MORZY	830	20	616,6
SLOWINSKI	1070	30	502
WEGLARZ	1730	10	1070,1
ZAKRZEWICZ	208	30	502

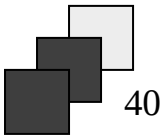


RATIO_TO_REPORT

Funkcja RATIO_TO_REPORT służy do wyliczania stosunku danej wartości do sumy zbioru wartości w oknie. Jeśli wartość jest pusta, to wynikiem funkcji jest NULL. Jeśli klauzula PARTITION BY zostanie pominięta, to funkcja jest wyliczana na podstawie całego wyniku zapytania.

Składnia:

```
RATIO_TO_REPORT (<value expression1>) OVER
( [ PARTITION BY <value expression2> [ ,... ] ] )
```

RATIO_TO_REPORT - przykład 1/3

```
select nazwisko, placa_pod,
        ratio_to_report(placa_pod)over() as ratio_to_report
from pracownicy
order by nazwisko
```

```
NAZWISKO          PLACA_POD RATIO_TO_REPORT
--------------- ---------- ---------------
BIALY                   250       ,025472775
BLAZEWICZ              1350       ,137552983
BRZEZINSKI              960       ,097815455
HAPKE                   480       ,048907727
JEZIERSKI              439,7      ,044801516
KONOPKA                 480       ,048907727
KOSZLAJDA               590       ,060115748
KROLIKOWSKI            645,5      ,065770704
MAREK                  410,2      ,041795729
MATYSIAK                371       ,037801598
MORZY                   830       ,084569612
SLOWINSKI              1070       ,109023476
WEGLARZ                1730       ,176271601
ZAKRZEWICZ              208       ,021193349
```

RATIO_TO_REPORT - przykład 2/3

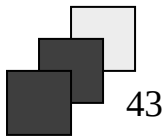
```
select nazwisko, placa_pod, id_zesp,
        ratio_to_report(placa_pod)over(partition by id_zesp) as
ratio_to_report
from pracownicy
order by nazwisko
```

NAZWISKO	PLACA_POD	ID_ZESP	RATIO_TO_REPORT
BIALY	250	30	,124501992
BLAZEWICZ	1350	40	1
BRZEZINSKI	960	20	,222417868
HAPKE	480	30	,239043825
JEZIERSKI	439,7	20	,101872017
KONOPKA	480	20	,111208934
KOSZLAJDA	590	20	,136694314
KROLIKOWSKI	645,5	20	,149552847
MAREK	410,2	10	,19166433
MATYSIAK	371	20	,085955238
MORZY	830	20	,192298781
SLOWINSKI	1070	30	,532868526
WEGLARZ	1730	10	,80833567
ZAKRZEWICZ	208	30	,103585657

RATIO_TO_REPORT - przykład 3/3

```
select   etat, count(*) lp,
         ratio_to_report(count(*))over() ratio_to_report
from pracownicy
group by etat
```

```
ETAT                LP RATIO_TO_REPORT
---------- ---------- ---------------
ADIUNKT              2       ,142857143
ASYSTENT             4       ,285714286
DYREKTOR             1       ,071428571
PROFESOR             4       ,285714286
SEKRETARKA           1       ,071428571
STAZYSTA             2       ,142857143
```

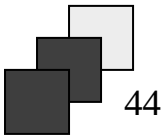


Funkcje LAG i LEAD

Funkcje LAG i LEAD pozwalają na dostęp i porównanie wielu różnych krotek jednocześnie, bez konieczności wykonywania połączenia zwrotnego. Funkcje LAG i LEAD działają na podstawie podanego przez użytkownika przesunięcia (offset) względem bieżącej krotki.

Składnia:

```
{LAG | LEAD}
( <value expression1>, [ <offset> [ , <default> ] ] )
OVER ( [ PARTITION BY <value expression2> [ ,... ] ]
ORDER BY <value expression3>
[ ASC | DESC ] [ NULLS FIRST | NULLS LAST ] [ ,... ] )
```



Funkcje LAG i LEAD - przykład

```
select   zatrudniony, nazwisko, placa_pod,
         lag(placa_pod, 1)over(order by zatrudniony) placa_poprz,
         lead(placa_pod, 1)over(order by zatrudniony) placa_nast
from pracownicy
order by zatrudniony
```

ZATRUDNI	NAZWISKO	PLACA_POD	PLACA_POPRZ	PLACA_NAST
68/01/01	WEGLARZ	1730		960
68/07/01	BRZEZINSKI	960	1730	1350
73/05/01	BLAZEWICZ	1350	960	830
75/09/15	MORZY	830	1350	1070
77/09/01	SLOWINSKI	1070	830	645,5
77/09/01	KROLIKOWSKI	645,5	1070	410,2
85/02/20	MAREK	410,2	645,5	590
85/03/01	KOSZLAJDA	590	410,2	480
92/09/01	HAPKE	480	590	439,7
92/10/01	JEZIERSKI	439,7	480	371
93/09/01	MATYSIAK	371	439,7	480
93/10/01	KONOPKA	480	371	250
93/10/15	BIALY	250	480	208
94/07/15	ZAKRZEWICZ	208	250	

Schemat bazy danych