

# **Funkcje analityczne**

# Plan rozdziału

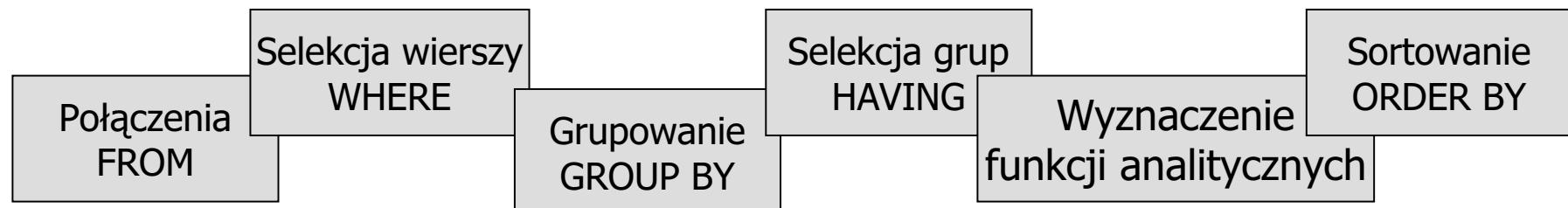
- Wprowadzenie do funkcji analitycznych
- Funkcje rankingu
- Funkcje okna
- Funkcje raportujące
- Funkcje LAG/LEAD
- Funkcje FIRST/LAST
- Odwrotne funkcje percentyli
- Funkcje rankingu hipotetycznego
- Funkcja WIDTH\_BUCKET
- Pozostałe funkcje analityczne

# Wprowadzenie do funkcji analitycznych

- Funkcje analityczne stanowią analityczne rozszerzenie funkcji SQL
- Podstawowy podział funkcji analitycznych jest następujący:
  - **funkcje rankingu** – wykorzystywane do wyznaczania rankingów, podziałów zbiorów wierszy na grupy (n-tki)
  - **funkcje okna** – wyznaczają wartości agregatów dla zbiorów wierszy wyznaczanych przy użyciu definicji okna
  - **funkcje raportujące** – wyznaczają wartości agregatów dla zbiorów wierszy w ramach tzw. partycji
  - **funkcje LAG/LEAD** – znajdują wartości określonych atrybutów w wierszach sąsiednich
  - **funkcje FIRST/LAST** – znajdują początkową lub końcową wartość w uporządkowanym zbiorze
  - **odwrotne funkcje procentyli** – wyznaczają wartość występującą w określonym miejscu w uporządkowanym zbiorze
  - **funkcje rankingu hipotetycznego** – wyznaczają hipotetyczny ranking zadanych wartości w uporządkowanym zbiorze
  - **funkcja WIDTH\_BUCKET** – dzieli uporządkowany zbiór na określoną liczbę przedziałów o zadanej szerokości
  - **funkcje statystyczne** – wyliczają zmiany poziomów i inne statystyki

# Miejsce funkcji analitycznych (1/2)

- Funkcje analityczne różnią się metodą działania i sposobem użycia od tradycyjnych funkcji SQL-owych
- Funkcje analityczne wyznaczając wynik dla bieżącej wiersza z reguły korzystają z informacji znajdujących się w wierszach sąsiednich
- Uwzględniając przetwarzanie polecenia SQL, można powiedzieć, że wartości funkcji analitycznych są wyliczane po wszystkich składowych operacjach (połączeniach, selekcji wierszy, grupowaniu, selekcji grup itd.)
- Po wyznaczeniu wyników funkcji analitycznych wykonywana jest jedynie operacja sortowania



# Miejsce funkcji analitycznych (2/2)

- Ściśle określone miejsca wyznaczania wartości funkcji analitycznych ma swoje znaczące konsekwencje:
  - nie mogą być używane w klauzulach WHERE, GROUP BY, HAVING
  - wykorzystywane są tylko i wyłącznie w klauzuli SELECT lub ORDER BY
  - działają tylko i wyłącznie na wierszach, grupach będących efektem finalnym zapytania
  - wiersze lub grupy odrzucone za pomocą klauzul WHERE lub HAVING nie są uwzględniane przez funkcje analityczne

np. jeśli odrzucimy w zapytaniu wszystkich zarabiających powyżej 1000 zł. to może okazać się, że w rankingu osób wg zarobków pierwsze miejsce zajmuje osoba z kwotą 900 zł.

# Partycje, okna, wiersz bieżący

- Przy używaniu funkcji analitycznych istotna jest znajomość podstawowej terminologii
  - **Partycje** – umożliwiają podział rezultatu zapytania na autonomiczne, niezależne zbiory, w ramach których funkcje analityczne będą mogły wyznaczać oddzielne rankingi, średnie itp.
  - **Okna** – występują tylko w przypadku funkcji okna. Pozwalają na zdefiniowane ruchomego zakresu – określanego indywidualnie dla każdego wiersza – w ramach którego funkcja będzie wyznaczała swoją wartość
  - **Bieżący wiersz** – wiersz, dla którego w danym momencie wyznaczany jest wynik funkcji analitycznej. W szczególności stanowi on punkt odniesienia przy wyznaczaniu zakresu okna

# Partycje, okna, wiersz bieżący

ID_TRANSAKCJI	NR_KONTA	DATA	KWOTA	TYP
10000	11-11111111	97/01/03	1500	WPLATA
10010	11-11111111	97/01/12	-100	WYPŁATA
10020	11-11111111	97/02/02	750	WPLATA
10030	11-11111111	98/05/22	-150	WYPŁATA
10040	11-11111111	99/11/04	1800	WPLATA
10050	11-11111111	99/12/02	1800	WPLATA
10070	11-11111111	99/12/24	-300	WYPŁATA
10060	11-11111111	99/12/24	-120,5	WYPŁATA
10011	22-22222222	97/01/12	1000	WPLATA
10021	22-22222222	97/09/02	830	WPLATA
10031	22-22222222	98/03/22	-170	WYPŁATA
10001	22-22222222	98/04/03	1650	WPLATA
10071	22-22222222	99/10/02	-330	WYPŁATA
10061	22-22222222	99/10/24	-130,5	WYPŁATA
10081	22-22222222	99/12/19	-110	WYPŁATA
10002	33-33333333	95/10/03	1350	WPLATA
10022	33-33333333	96/07/02	670	WPLATA
10012	33-33333333	97/02/12	-90	WYPŁATA
10052	33-33333333	98/05/02	1710	WPLATA
10042	33-33333333	98/05/04	1620	WPLATA
10032	33-33333333	98/07/22	-130	WYPŁATA
10092	33-33333333	99/08/03	810	WPLATA

Partycje pozwalają na podział wyniku na autonomiczne obszary, w celu wykonania wielu analiz np. rankingów lub obliczenia sumy kumulacyjnej oddzielnie dla każdego z nich

Okno wyznaczone indywidualnie dla każdego wiersza bieżącego pozwala na zdefiniowanie zmiennego zakresu uwzględnianego przez funkcję analityczną

Bieżący wiersz to wiersz, dla którego w danej chwili następuje wyznaczenie wyniku funkcji analitycznej

# Składnia funkcji analitycznych

- Funkcje analityczne znacznie różnią się od standardowych funkcji SQL-owych. Ich ogólna składnia jest następująca

**NAZWA FUNKCJI ANALITYCZNEJ** ( **parametry** –  
różne dla różnego rodzaju funkcji)  
OVER ( **definicja partycji** – opcjonalna  
**definicja porządku wierszy w partycji** – zależna od typu funkcji  
**definicja okna** – tylko w przypadku funkcji okna)

- Przykłady:

**RANK()** OVER ( **partition by numer\_konta**  
**order by kwota**)

**AVG()** OVER ( **partition by numer\_konta**  
**order by data**  
**ROWS UNBOUNDED PRECEDING**)

**LEAD(sum(kwota))** OVER ( **partition by numer\_konta**  
**order by data**)

# Przykładowy schemat

```
SQL> desc transakcje
```

Nazwa	Wartość NULL?	Typ
-----	-----	-----
ID_TRANSAKCJI	NOT NULL	NUMBER(12)
NR_KONTA	NOT NULL	VARCHAR2(30)
DATA	NOT NULL	DATE
KWOTA	NOT NULL	NUMBER(10,2)
TYP		VARCHAR2(10)
KATEGORIA		VARCHAR2(20)

```
SQL> SELECT * FROM transakcje;
```

ID_TRANSAKCJI	NR_KONTA	DATA	KWOTA	TYP	KATEGORIA
10000	11-11111111	97/01/03	1500	WPŁATA	PENSJA
10010	11-11111111	97/01/12	-100	WYPŁATA	WYPŁATA W BANKOMACIE
10020	11-11111111	97/02/02	750	WPŁATA	UMOWA O DZIEŁO
10030	11-11111111	98/05/22	-150	WYPŁATA	RACHUNEK ZA TELEFON
10040	11-11111111	99/11/04	1800	WPŁATA	PENSJA
10050	11-11111111	99/12/02	1900	WPŁATA	PENSJA
10060	11-11111111	99/12/24	-120,5	WYPŁATA	RACHUNEK ZA PRĄD
10070	11-11111111	00/01/02	-300	WYPŁATA	WYPŁATA W BANKOMACIE
10080	11-11111111	00/01/19	-100	WYPŁATA	RACHUNEK ZA TELEFON
10090	11-11111111	00/05/03	900	WPŁATA	UMOWA O DZIEŁO
10100	11-11111111	01/07/16	-600	WYPŁATA	RACHUNEK ZA PRĄD
10110	11-11111111	01/07/24	-150	WYPŁATA	WYPŁATA W BANKOMACIE
10120	11-11111111	01/09/03	1400	WPŁATA	PENSJA
10130	11-11111111	01/09/29	-250	WYPŁATA	RACHUNEK ZA TELEFON
10140	11-11111111	01/12/03	1650	WPŁATA	PENSJA
10150	11-11111111	02/01/05	-500	WYPŁATA	WYPŁATA W BANKOMACIE

. . .

# Funkcje rankingu

- Funkcje rankingu wyznaczają ranking wiersza porównując wartości w nim zawarte z wartościami wierszy znajdujących się w tej samej partycji.
- Partycje pozwalają na wyznaczenie wielu oddzielnych rankingów.
- Porządek wierszy w partycji jest podstawowym elementem definiującym ranking
- Przykłady funkcji rankingu:
  - RANK, DENSE\_RANK
  - CUME\_DIST
  - PERCENT\_RANK
  - NTILE
  - ROW\_NUMBER
- Składnia:

```
nazwa_funkcji() OVER ( PARTITION BY wyrażenie1 - opcjonalnie  
ORDER BY wyrażenie2 - obowiązkowo)
```

# RANK, DENSE\_RANK (1/2)

- Pozwalają na ustalenie rankingu w grupie.
- Funkcja RANK w przypadku dwóch lub większej liczby wierszy znajdujących się na tej samej pozycji pozostawia przerwy w numeracji, funkcja DENSE\_RANK przerw nie zostawia.
- Zadanie:  
Wyznacz rankingi wpłat i wypłat oddzielne dla każdej kategorii.  
Analizę ogranicz tylko do konta 11-11111111

```
SELECT rank() OVER (PARTITION BY kategoria
                    ORDER BY kwota DESC) RANK,
       dense_rank() OVER (PARTITION BY kategoria
                          ORDER BY kwota DESC) dense_rank,
       kategoria, data, kwota
FROM   transakcje
WHERE  nr_konta = '11-11111111'
ORDER BY kategoria, kwota DESC;
```

# RANK, DENSE\_RANK (2/2)

Oddzielne rankingi dla każdej partycji

Ten sam porządek – malejący dla wszystkich partycji

	RANK	DENSE_RANK	KATEGORIA	DATA	KWOTA
Różnice dla funkcji RANK i DENSE_RANK	1	1	1 PENSJA	99/11/04	1800
	1		1 PENSJA	99/12/02	1800
	3		2 PENSJA	01/12/03	1650
	4		3 PENSJA	97/01/03	1500
	5		4 PENSJA	01/09/03	1400
	1	2	1 RACHUNEK ZA PRĄD	99/12/24	-120,5
	2		2 RACHUNEK ZA PRĄD	01/07/16	-600
	1	3	1 RACHUNEK ZA TELEFON	98/05/22	-150
	1		1 RACHUNEK ZA TELEFON	00/01/19	-150
	3		2 RACHUNEK ZA TELEFON	01/09/29	-250
	1	2	1 UMOWA O DZIEŁO	00/05/03	900
	2		2 UMOWA O DZIEŁO	97/02/02	750
	1	4	1 WYPŁATA W BANKOMACIE	97/01/12	-100
	2		2 WYPŁATA W BANKOMACIE	01/07/24	-150
	3		3 WYPŁATA W BANKOMACIE	99/12/24	-300
	4		4 WYPŁATA W BANKOMACIE	02/01/05	-500

# Podział wierszy na partycje

- Wiersze dzielone są na partycje za pomocą wyrażenia PARTITION BY
- Podział na partycje może być realizowany za pomocą dowolnego wyrażenia lub zbioru wyrażeń rozdzielonych przecinkami.
- Przykłady:

```
PARTITION BY kategoria
```

```
PARTITION BY konto, kategoria
```

```
PARTITION BY to_char(data, 'YYYY')
```

# Porządek wierszy w partycji

- Większość funkcji analitycznych (poza funkcjami raportującymi) wymaga uporządkowania wierszy w partycji
- Do ustalenia porządku wierszy w partycji wykorzystywane jest wyrażenie ORDER BY
- Sortowanie może odbyć się w oparciu o jedno wyrażenie lub zbiór wyrażeń rozdzielonych przecinkami
- Klauzule NULLS FIRST i NULLS LAST pozwalają precyzyjnie wskazać miejsce wierszy z pustymi wartościami wyrażeń wykorzystywanych do ustalenia porządku. Pominięcie tych klauzul powoduje, że wartości puste są traktowane jako największe w zbiorze
- Porządek wierszy we wszystkich partycjach jest jednakowy
- Przykłady:

```
ORDER BY data
```

```
ORDER BY data, kwota DESC
```

```
ORDER BY to_char(data, 'YYYY'), kwota
```

# CUME\_DIST, PERCENT\_RANK

- Wyniki funkcji CUME\_DIST i PERCENT\_RANK wyznaczone są wg następujących wzorów:

**CUME\_DIST(x) = liczba wierszy w partycji posiadających wartość x "przed" bieżącą wartością x wraz z wierszami posiadającymi wartość x / liczbę wszystkich wierszy w partycji**

**PERCENT\_RANK(x) = ranking bieżącego wiersza - 1 / liczbę wszystkich wierszy w partycji - 1**

gdzie x jest atrybutem, po którym sortujemy lub, w oparciu o który wyznaczamy ranking

```
SQL> SELECT cume_dist() OVER (ORDER BY kwota) cume_dist,
2          percent_rank() OVER (ORDER BY kwota) percent_rank,
3          kwota
4 FROM   transakcje
5 WHERE  nr_konta = '11-11111111'
6 AND    kategoria = 'pensja'
7 ORDER BY kwota;
```

w przypadku braku klauzuli PARTITION BY wszystkie wiersze uzyskane w wyniku stanowią jedną partycję

CUME_DIST	PERCENT_RANK	KWOTA
		1400
,2	0	1400
,4	,25	1500
,6	,5	1650
1	,75	1800
1	,75	1800

$(1+1)/5$

$(4-1)/(5-1)$

Funkcja **CUME\_DIST** wyznacza tzw. procentyle. Informuje na jakim miejscu w uporządkowanym zbiorze, wyrażonym procentowo, znajduje się określona wartość

## NTILE, ROW\_NUMBER (1/2)

- Funkcja NTILE dzieli wiersze w partycji na N grup (ang. 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.
- Funkcja ROW\_NUMBER przypisuje każdemu wierszowi oddzielny numer wynikający z jego porządku w partycji. Podobnie jak funkcja NTILE jest to funkcja niedeterministyczna.
- Niedeterminizm funkcji analitycznych jest uwarunkowany porządkiem wierszy w partycji. Jeśli jest on ściśle określony (sortowanie jest realizowane w oparciu o wyrażenie unikalne) niedeterminizm nie występuje.

# NTILE, ROW\_NUMBER (2/2)

```
SQL> SELECT ntile(4) OVER (ORDER BY kwota) ntile,
2         row_number() OVER (ORDER BY kwota) row_number,
3         kwota, data, kategoria
4 FROM   transakcje
5 WHERE  nr_konta = '11-11111111'
6 AND    kategoria IN ('pensja','rachunek za telefon')
7 ORDER BY kwota;
```

NTILE	ROW_NUMBER	KWOTA	DATA	KATEGORIA
1	1	-250	01/09/29	RACHUNEK ZA TELEFON
1	2	-150	98/05/22	RACHUNEK ZA TELEFON
2	3	-150	00/01/19	RACHUNEK ZA TELEFON
2	4	1400	01/09/03	PENSJA
3	5	1500	97/01/03	PENSJA
3	6	1650	01/12/03	PENSJA
4	7	1800	99/11/04	PENSJA
4	8	1800	99/12/02	PENSJA

Kwota nie jest unikalna, a zatem wyniki funkcji NTILE i ROW\_NUMBER mogą być niedeterministyczne

W przypadku niepodzielności liczby wierszy przez argument funkcji NTILE, początkowe grupy "obdarowywane" są pojedynczymi wierszami wynikającymi z reszty z dzielenia. Dla przykładu licznosci grup partycji składającej się z 15 pierwszy i argumentie funkcji NTILE równym 4 będą następujące 4,4,4,3 wynika to z  $15/4 = 3$  reszta 3

# Ograniczanie wyników w oparciu o wyniki funkcji analitycznych

- Ze względu na kolejność wykonywania operacji w przypadku użycia funkcji analitycznych nie można bezpośrednio zakładać na nich warunków
- Warunki na wartościach wyznaczonych przez funkcje analityczne zakłada się zagnieżdżając zapytanie wykorzystujące funkcję analityczną
- Przykłady:
  - Znajdź trzy największe wpłaty
  - Znajdź pierwszą połowę transakcji wykonanych w roku 1999

```
SELECT kwota, data, nr_konta
FROM (
  SELECT kwota, data, nr_konta,
         rank()
           over (ORDER BY kwota DESC) r
  FROM   transakcje)
WHERE r <= 3;
```

KWOTA	DATA	NR_KONTA
2090	01/07/02	22-22222222
1980	01/05/04	22-22222222
1820	03/04/03	22-22222222

```
SELECT kwota, data, nr_konta
FROM (
  SELECT kwota, data, nr_konta,
         percent_rank()
           over (ORDER BY kwota DESC) pr
  FROM   transakcje
 WHERE  nr_konta = '11-11111111'
 AND    to_char(data, 'YYYY') = '1999')
WHERE pr <= 0.5;
```

KWOTA	DATA	NR_KONTA
1800	99/11/04	11-11111111
1800	99/12/02	11-11111111

# Funkcje okna

- Funkcje okienkowe operują na uporządkowanym zbiorze krotek i dla każdej krotki obliczają wartość agregowaną obejmującą wartości występujące w tzw. oknie.
- Funkcje okna pozwalają na wyznaczanie wyników funkcji agregujących przyrostowych, ruchomych itp. Do określenia typu agregacji wykorzystywana jest definicja okna
- Funkcje okna są bardzo podobne do tradycyjnych funkcji agregujących: MAX, MIN, AVG, SUM, COUNT, STDDEV, VARIANCE. Dodatkowo istnieją dwie funkcje FIRST\_VALUE i LAST\_VALUE wyznaczające odpowiednio pierwszą i ostatnią wartość w partycji. Funkcjami okna mogą również funkcje regresji liniowej.
- Składnia:

```
nazwa_funkcji(wyrażenie1)
OVER ( PARTITION BY wyrażenie2 - opcjonalnie
        ORDER BY wyrażenie3 - obowiązkowo
        definicja okna - opcjonalna)
```

# Definicja okna (1/2)

- Okno może być jednego z dwóch typów od użytych słów kluczowych przy jego definicji
  - ROWS – okno fizyczne – wyrażone w liczbie krotek
  - RANGE – okno logiczne – wyrażone przy użyciu wartości o typie zgodnym z atrybutem porządkującym wiersze w partycji
- Zakres okna może być zdefiniowany za pomocą wyrażeń
  - BETWEEN ... AND ... – pozwala na jawne zdefiniowanie zarówno początku jak i końca okna
  - określenia tylko początku okna, koniec okna w takim przypadku będzie domyślny – bieżący wiersz.
- Słowa kluczowe wykorzystywane przy definiowaniu okna:
  - UNBOUNDED PRECEDING – początkiem okna będzie pierwszy wiersz w partycji, używane tylko przy definiowaniu początku okna
  - UNBOUNDED FOLLOWING – końcem okna będzie końcowy wiersz w partycji, używane tylko przy definiowaniu końca okna
  - CURRENT ROW – bieżący wiersz lub wartość w zależności od typu okna

## Definicja okna (2/2)

- Początek i koniec okna można również zdefiniować za pomocą wyrażeń:
  - wartość FOLLOWING
  - wartość PRECEDING
- Interpretacja wartości jest zależna od typu okna
  - ROWS – wartość jest fizycznym offsetem wyrażonym w liczbie krotek. Musi być albo stałą albo wyrażeniem, którego wartością jest liczba całkowita dodatnia
  - RANGE – wartość jest logicznym offsetem zależnym od wyrażenia wykorzystanego do uporządkowania wierszy w partycji.  
Jeśli wyrażenie to jest liczbą lub datą, wówczas wartość może być liczbą dodatnią.  
Jeśli wyrażenie w klauzuli ORDER BY jest datą wówczas wartość może być zarówno liczbą dodatnią jak i interwałem czasu
- Początek okna nigdy nie może być za końcem okna
- Dla każdego wiersza okno wyznaczane jest indywidualnie
- Pominięcie definicji okna jest równoznaczne z wyrażeniem:

**RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW**

# Definicja okna – przykłady

- Okno obejmujące 5 wierszy: bieżący i po 2 przed i po bieżącym wierszu

```
ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING
```

- Okno rozpoczynające się od bieżącego wiersza a kończące na końcu partycji

```
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
```

- Okno obejmujące dwa ubiegłe miesiące

```
RANGE INTERVAL '2' MONTH PRECEDING
```

- Okno obejmujące wiersze z wartościami atrybutu nie mniejszymi niż 100 i nie większymi niż 200 od wartości w bieżącym wierszu

```
RANGE BETWEEN 100 PRECEDING AND 200 FOLLOWING
```

# Przykłady funkcji okna

- Dla każdej transakcji przedstaw jej datę, kwotę na jaką opiewała oraz:

- saldo po wykonaniu operacji

- średnią kwotę operacji z ostatniego roku

- minimalną kwotę z 3 ostatnich operacji

- ilość operacji wykonanych od 6 miesięcy wstecz do 6 miesięcy po transakcji

	KWOTA	DATA	SALDO	AVG12	MIN3	COUNT6_6
	1500	97/01/03	1500	1500,00	1500	3
	-100	97/01/12	1400	700,00	-100	3
	750	97/02/02	2150	716,67	-100	3
	-150	98/05/22	2000	-150,00	-150	1
	1800	99/11/04	3800	1800,00	-150	6
	1800	99/12/02	5600	1800,00	-150	6
	-300	99/12/24	5300	1100,00	-300	6
	-120,5	99/12/24	5179,5	794,88	-300	6
	-150	00/01/19	5029,5	605,90	-300	6
	900	00/05/03	5929,5	654,92	-300	6
	-600	01/07/16	5329,5	-600,00	-600	6
	-150	01/07/24	5179,5	-375,00	-600	6
	1400	01/09/03	6579,5	216,67	-600	6
	-250	01/09/29	6329,5	100,00	-600	6
	1650	01/12/03	7979,5	410,00	-250	6
	-500	02/01/05	7479,5	258,33	-500	6

```

SELECT kwota, data,
       SUM(kwota) OVER (ORDER BY data) saldo,
       AVG(kwota) OVER (ORDER BY data RANGE INTERVAL '12' month PRECEDING) avg12,
       MIN(kwota) OVER (ORDER BY data rows 3 PRECEDING) min3,
       COUNT(*) OVER (ORDER BY data RANGE BETWEEN interval '6' month PRECEDING
                      AND interval '6' month FOLLOWING) count6_6
FROM   transakcje
WHERE  nr_konta = '11-11111111'
ORDER BY data;

```

# Funkcje raportujące

- Funkcje raportujące podobnie jak funkcje okna pozwalają na wyznaczenie wartości funkcji agregujących w oparciu o zbiór wierszy uzyskanych w wyniku zapytania.
- W odróżnieniu od funkcji okna nie wprowadzają one definicji okna, ani porządku wierszy w partycji.
- Zakresem obejmowanym do wyznaczenia wyniku funkcji agregujących jest zawsze cała partycja
- Funkcje raportujące są bardzo podobne do tradycyjnych funkcji agregujących: MAX, MIN, AVG, SUM, COUNT, STDDEV, VARIANCE, RATIO\_TO\_REPORT. Funkcjami raportującymi mogą również funkcje regresji liniowej.
- Dzięki funkcjom raportującym mamy z poziomu pojedynczych wierszy dostęp do agregatów wyznaczonych na grupie.
- Składnia:

```
nazwa_funkcji(wyrażenie1)  
OVER (PARTITION BY wyrażenie2 - opcjonalnie)
```

# Przykłady funkcji raportujących

- Dla każdej transakcji przedstaw jej kwotę, datę i kategorię, oraz:
  - średnią kwotę operacji wchodzących w skład tej samej kategorii
  - udział kwoty transakcji do wszystkich transakcji z tej samej kategorii

```
SELECT kwota, data, kategoria,
       AVG(kwota) OVER (PARTITION BY kategoria) avg_k,
       kwota/SUM(kwota) OVER (PARTITION BY kategoria) ratio_to_report
FROM   transakcje
WHERE  nr_konta = '11-11111111'
ORDER BY kategoria, data;
```

UWAGA! Użycie klauzuli ORDER BY wewnątrz funkcji raportującej czyni z niej funkcje okna.

Funkcja **RATIO\_TO\_REPORT(x)** jest uproszczeniem dla wyrażenia **x/SUM(x)** gdzie SUM(x) jest raportującą funkcją analityczną

KWOTA	DATA	KATEGORIA	AVG_K	RATIO_TO_
1500	97/01/03	PENSJA	1630,00	0,184
1800	99/11/04	PENSJA		
1800	99/12/02	PENSJA		
1400	01/09/03	PENSJA		
1650	01/12/03	PENSJA		
-120,5	99/12/24	RACHUNEK ZA PRĄD	-360,25	0,167
-600	01/07/16	RACHUNEK ZA PRĄD		
-150	98/05/22	RACHUNEK ZA TELEFON	-183,33	0,273
-150	00/01/19	RACHUNEK ZA TELEFON		
-250	01/09/29	RACHUNEK ZA TELEFON		
750	97/02/02	UMOWA O DZIEŁO	825,00	0,455
900	00/05/03	UMOWA O DZIEŁO		
-100	97/01/12	WYPŁATA W BANKOMACIE	-262,50	0,095
-300	99/12/24	WYPŁATA W BANKOMACIE		
-150	01/07/24	WYPŁATA W BANKOMACIE		
-500	02/01/05	WYPŁATA W BANKOMACIE		

# LAG/LEAD

- Bardzo ciekawymi funkcjami analitycznymi o wielu zastosowaniach są funkcje LAG i LEAD.
- Funkcje te mogą sięgać do wartości określonych atrybutów znajdujących się w sąsiednich wierszach w tej samej partycji
- Przykłady:
  - Pokaż okresy w jakich występował określony stan konta
  - Pokaż zysk z kolejnych lat oraz różnicę w stosunku do zysku z lat poprzednich
- Funkcje LAG i LEAD wymagają określenia kolejności wierszy w partycji

```
nazwa_funkcji(wyrażenie1,o_ile) OVER  
  (PARTITION BY wyrażenie2 - opcjonalnie  
   ORDER BY wyrażenie3 - obowiązkowo)
```

- Składnia:

**wyrażenie1** – jest wskazaniem na atrybut, który leży w kręgu naszych zainteresowań  
**o\_ile** – mówi o ile wierszy wstecz (LAG) lub w przód (LEAD) w wierszach wyników należy sięgnąć aby dobrać się do właściwej wartości

# Przykłady wykorzystania funkcji LAG/LEAD

```
SELECT kwota,
       SUM(kwota) OVER (ORDER BY data) saldo,
       data od_dnia,
       lead(data,1) OVER
         (ORDER BY data) do_dnia
FROM   transakcje
WHERE  nr_konta = '11-11111111'
ORDER BY data;
```

KWOTA	SALDO	OD_DNIA	DO_DNIA
1500	1500	97/01/03	97/01/12
-100	1400	97/01/12	97/02/02
750	2150	97/02/02	98/05/22
-150	2000	98/05/22	99/11/04
1800	3800	99/11/04	99/12/02
1800	5600	99/12/02	99/12/24
.	.	.	.
1400	6579,5	01/09/03	01/09/29
-250	6329,5	01/09/29	01/12/03
1650	7979,5	01/12/03	02/01/05
-500	7479,5	02/01/05	

```
SELECT nr_konta, to_char(data, 'YYYY') rok,
       SUM(kwota) przychod,
       SUM(kwota) -
       lag(sum(kwota), 1)
         OVER (PARTITION BY nr_konta
              ORDER BY to_char(data, 'YYYY'))
       roznica
FROM   transakcje
GROUP BY nr_konta, to_char(data, 'YYYY');
```

NR_KONTA	ROK	PRZYCHOD	ROZNICA
11-11111111	1997	2150,00	
11-11111111	1998	-150,00	-2300
11-11111111	1999	3179,50	3329,5
11-11111111	2000	750,00	-2429,5
11-11111111	2001	2050,00	1300
11-11111111	2002	-500,00	-2550
22-22222222	1997	1830,00	
22-22222222	1998	1480,00	-350
22-22222222	1999	-570,50	-2050,5
22-22222222	2001	3400,00	3970,5
22-22222222	2002	1540,00	-1860
22-22222222	2003	1820,00	280
33-33333333	1995	1350,00	
33-33333333	1996	670,00	-680
33-33333333	1997	-90,00	-760
33-33333333	1998	3200,00	3290
.	.	.	.

# FIRST/LAST

- Funkcje FIRST/LAST są odpowiednikami funkcji okna FIRST\_VALUE i LAST\_VALUE. Działają one jednakże na całej partycji i nie ma w nich definicji okna.
- Funkcje FIRST/LAST mogą działać w dwóch trybach
  - jako funkcje agregujące – bez użycia klauzuli OVER – w tym przypadku "partycją" jest grupa wierszy (GROUP BY)
  - jako zwykłe funkcje raportujące – z wykorzystaniem klauzuli OVER
- Składnia: 

```
nazwa_funkcji(wyrażenie1)
KEEP ( DENSE_RANK LAST|FIRST ORDER BY wyrażenie2)
[OVER (PARTITION BY wyrażenie3)]
```
- Funkcje FIRST/LAST mimo iż posiadają wiele postaci MIN, MAX, SUM, AVG, COUNT, VARIANCE, STDDEV ostatecznie koncentrują się na przetwarzaniu jednej wartości występującej na początku lub na końcu grupy wierszy

# Przykłady wykorzystania funkcji FIRST/LAST

- Dla każdego roku wyświetl kwotę oraz datę największego przychodu (funkcja FIRST jako funkcja agregująca)

```
SELECT TO_CHAR(data, 'yyyy') rok,
       MAX(kwota) przychod,
       MIN(data) keep (dense_rank first
                      order by kwota desc) data1,
       MAX(data) keep (dense_rank last
                      order by kwota) data2
FROM   transakcje
WHERE  nr_konta = '11-11111111'
GROUP BY TO_CHAR(data, 'YYYY');
```

ROK	PRZYCHOD	DATA1	DATA2
1997	1500	97/01/03	97/01/03
1998	-150	98/05/22	98/05/22
1999	1800	99/11/04	99/12/02
2000	900	00/05/03	00/05/03
2001	1650	01/12/03	01/12/03
2002	-500	02/01/05	02/01/05

- Dla każdej transakcji wyświetl różnicę kwoty transakcji i kwoty pierwszej transakcji wykonanej tego samego roku

```
SELECT data, kwota,
       kwota-
       AVG(kwota) KEEP (DENSE_RANK LAST
                       ORDER BY data DESC)
       OVER (PARTITION BY to_char(data, 'YYYY')) ROZ
from   transakcje
where  NR_KONTA = '11-11111111'
and to_char(DATA, 'YYYY') between '1997' and '2000'
order by DATA
```

DATA	KWOTA	ROZ
97/01/03	1500	0
97/01/12	-100	-1600
97/02/02	750	-750
98/05/22	-150	0
99/11/04	1800	0
99/12/02	1800	0
99/12/24	-300	-2100
99/12/24	-120,5	-1920,5
00/01/19	-150	0
00/05/03	900	1050

# Funkcje PERCENTILE\_DISC i PERCENTILE\_CONT

- Funkcja CUME\_DIST może służyć do wyznaczenia tzw. procentyli czyli określenia na jakim miejscu w uporządkowanym zbiorze, wyrażonym procentowo, znajduje się określona wartość
- Funkcje PERCENTILE\_DISC i PERCENTILE\_CONT służą do operacji odwrotnej. Informują jaka wartość znajduje się na określonej pozycji w uporządkowanym zbiorze wartości.
- PERCENTILE\_DISC wyznaczane jest przez przeglądanie wyników uzyskanych w wyniku funkcji CUME\_DIST. Pierwszy wiersz, w którym wynik funkcji CUME\_DIST jest większy od argumentu funkcji PERCENTILE\_DISC wyznacza poszukiwaną wartość.
- Wynik funkcji PERCENTILE\_CONT jest wyznaczany przez liniową interpolację wierszy otaczających wskazaną pozycję. Algorytm do wyznaczenia wartości PERCENTILE\_CONT(x) jest następujący:
  - $RN = (1+x*(n-1))$  gdzie n jest liczbą wierszy w grupie
  - $CRN = CEIL(RN)$ ;  $FRN = FLOOR(RN)$
  - Jeśli  $CRN = FRN$  wówczas wynikiem jest wartość znajdująca się w wierszu RN w przeciwnym przypadku wynikiem jest wartość wyrażenia  $(CRN-RN)*(wartość\ z\ wiersza\ CRN)+(RN-FRN)*(wartość\ z\ wiersza\ FRN)$
- Funkcje mają dwie postacie: funkcji agregujących i raportujących
- Składnia:

```
nazwa_funkcji (stała1)
                WITHIN GROUP (ORDER BY wyrażenie2)
                [OVER (PARTITION BY wyrażenie3)]
```

# Przykłady wykorzystania funkcji PERCENTILE\_DISC i PERCENTILE\_CONT

- Jakie kwoty były w poszczególnych latach kwotami znajdującymi się w środku zbioru transakcji uporządkowanego wg wielkości kwoty – medianę
- Jaki był procentowy udział kwoty każdej transakcji w medianie

```
SELECT TO_CHAR(data, 'yyyy') rok,
       percentile_disc(0.5) WITHIN GROUP (ORDER BY kwota) disc,
       percentile_cont(0.5) WITHIN GROUP (ORDER BY kwota) cont
FROM   transakcje
WHERE  nr_konta = '11-11111111'
AND    to_char(data, 'YYYY') BETWEEN '1997' AND '2000'
GROUP BY to_char(data, 'YYYY')
```

ROK	DISC	CONT
1997	750	750
1998	-150	-150
1999	-120,5	<b>839,75</b>
2000	-150	375

$RN = (1+x*(n-1)) = (1+0,5*(4-1)) = 1+1,5 = 2,5$ ;  $CRN = 3$ ;  $FRN = 2$ ;  
 $CONT = (CRN-RN)*x(CRN)+(RN-FRN)*x(FRN) =$   
 $0,5 * -120,5 + 0,5 * 1800 = -60,25 + 900 = \mathbf{839,75}$

```
SELECT to_char(data, 'yyyy') rok, kwota,
       cume_dist()
       OVER (PARTITION BY to_char(data, 'YYYY')
            ORDER BY kwota ) cume_dist,
       kwota/percentile_disc(0.5)
       WITHIN GROUP (ORDER BY kwota)
       OVER (PARTITION BY to_char(data, 'YYYY')) proc
FROM   transakcje
. . .
```

ROK	KWOTA	CUME_DIST	PROC
1997	-100	0,33	-0,13
1997	750	0,67	1,00
1997	1500	1,00	2,00
1998	-150	1,00	1,00
1999	-300	0,25	2,49
1999	<u>-120,5</u>	<u>0,50</u>	<u>1,00</u>
1999	1800	1,00	-14,94
1999	1800	1,00	-14,94
2000	-150	0,50	1,00
2000	900	1,00	-6,00

# Funkcje rankingu hipotetycznego

- Funkcje rankingu mają swoje odpowiedniki wyznaczające hipotetyczny ranking zadanych wartości
- Mogą być wykorzystywane do badania sytuacji "co by było gdyby"
- Przykłady:
  - na jakim miejscu znajdowałby klient z rocznym przychodem 3000 zł
  - na jaki miejscu, wyrażonym procentowo, w przychodzie z roku 1999 określonego klienta znalazłaby się wpłata 1000 zł
- Do funkcji rankingu hipotetycznego zaliczamy RANK, DENSE\_RANK, PERCENT\_RANK i CUME\_DIST. Argumentem tych funkcji jest wartość dla której poszukujemy wyniku rankingu
- Funkcje rankingu hipotetycznego są funkcjami agregującymi i działają na zbiorze wierszy znajdujących się w grupie
- Liczba i typy argumentów funkcji rankingu hipotetycznego muszą odpowiadać konstrukcji klauzuli ORDER BY
- Składnia:

```
nazwa_funkcji(stała1,...) WITHIN GROUP (ORDER BY wyrażenie2 - obowiązkowo)
```

# Przykłady wykorzystania funkcji rankingu hipotetycznego

```
select RANK(3000) WITHIN GROUP
      (ORDER BY SUM(KWOTA) DESC) GDZIE_3000
from   transakcje
group by NR_KONTA, to_char(DATA, 'YYYY');
```

```
GDZIE_3000
-----
                4
```

## Stan rzeczywisty:

NR_KONTA	ROK	PRZYCHOD	RANK
22-22222222	2001	3400	1
33-33333333	1998	3200	2
11-11111111	1999	3179,5	3
33-33333333	2000	2269,5	4
11-11111111	1997	2150	5
11-11111111	2001	2050	6
22-22222222	1997	1830	7
22-22222222	2003	1820	8

```
select PERCENT_RANK(1000)
      WITHIN GROUP (ORDER BY KWOTA) HIP_PRANK
from   transakcje
where  NR_KONTA = '11-11111111'
and to_char(DATA, 'YYYY') = '1999';
```

```
HIP_PRANK
-----
                ,5
```

## Stan rzeczywisty:

NR_KONTA	DATA	KWOTA	PRANK
11-11111111	99/12/24	-300	0,00
11-11111111	99/12/24	-120,5	0,33
11-11111111	99/11/04	1800	0,67
11-11111111	99/12/02	1800	0,67

# Funkcja WIDTH\_BUCKET

- Interesującą funkcją zaliczaną do funkcji analitycznych jest funkcja WIDTH\_BUCKET. Pozwala ona na zdefiniowanie zakresu i szerokości przedziałów oraz wyznaczanie numerów przedziałów, w których mieszczą się określone wartości w poszczególnych wierszach.
- W przeciwieństwie do funkcji NTILE która dzieli grupę wierszy na równoliczne przedziały, funkcja WIDTH\_BUCKET dzieli grupę wierszy na przedziały o równej szerokości
- Przykłady:
  - Wyznacz wyniki egzaminu zakładając, że maksymalna liczba punktów to 100 a liczba punktów potrzebnych do zaliczenia to 50
  - Na podstawie sumy posiadanych aktywów pogrupuj klientów na dwie kategorie "bogatych" i "biednych" zakładając, że minimalny analizowany próg to 5000, maksymalny to 10000 a podział ma być pośrodku

- Składnia:

```
width_bucket(  
    wyrażenie1,          dolna wartość graniczna,  
    górna wartość graniczna,  liczba przedziałów)
```

# Przykłady wykorzystania funkcji WIDTH\_BUCKET

```
SQL> SELECT indeks, punkty,
2          2.5+width_bucket(punkty,50,100,5)*0.5 ocena
3 FROM egzamin
4 WHERE punkty>=50;
```

INDEKS	PUNKTY	OCENA
59786	93	5
59787	76	4
59796	80	4,5
59797	75	4
59798	70	4
59800	82	4,5
59802	69	3,5
59803	66	3,5

```
SQL> SELECT SUM(kwota), nr_konta,
2          width_bucket(SUM(kwota),5000,10000,2) dobry_zly
3 FROM transakcje
4 GROUP BY nr_konta;
```

SUM(KWOTA)	NR_KONTA	DOBRY_ZLY
7479,5	11-11111111	1
9499,5	22-22222222	2
6869,5	33-33333333	1

# Pozostałe funkcje analityczne

- Wśród pozostałych funkcji analitycznych możemy wyróżnić:
  - Funkcje regresji liniowej
    - REGR\_COUNT
    - REGR\_AVGY, REGR\_AVGX
    - REGR\_SLOPE, REGR\_INTERCEPT
    - REGR\_R2
    - REGR\_SXX, REGR\_SYY i REGR\_SXY
  - Funkcje statystyczne wchodzące w skład pakietu DBMS\_STAT\_FUNCS (10g)
    - Weryfikujące jak dobrze dane w tabeli spełniają
      - EXPONENTIAL\_DIST\_FIT – rozkład wykładniczy
      - NORMAL\_DIST\_FIT – rozkład normalny
      - POISSON\_DIST\_FIT – rozkład Poissona
      - UNIFORM\_DIST\_FIT – rozkład normalny
      - WEIBULL\_DIST\_FIT – rozkład Weibulla
    - SUMMARY – podsumowuje kolumny numeryczne w tabeli (wyznacza średnie, minima, maksima, mediany, kwantyle itp.)
  - Do funkcji analitycznych zalicza się często także funkcję CASE

# Funkcje analityczne a wydajność (1/2)

- Funkcje analityczne oprócz swojej dużej funkcjonalności częstokroć niemożliwej do osiągnięcia w tradycyjny sposób, posiadają również bardzo dobrą wydajność
- Wydajność funkcji analitycznych w znaczący sposób przewyższa większość alternatywnych standardowych rozwiązań
- Przykład 1: Wyznacz trzy największe transakcje

```
SQL> SELECT kwota, data, nr_konta
 2 FROM ( SELECT kwota, data, nr_konta,
 3           rank() OVER (ORDER BY kwota DESC) r
 4         FROM   transakcje)
 5 WHERE r <= 3;
```

Całkowity: 00:00:00.08

Plan wykonywania

```
-----
 0      SELECT STATEMENT Optimizer=CHOOSE (Cost=176 Card=24571 Bytes=1277692)
 1    0      VIEW (Cost=176 Card=24571 Bytes=1277692)
 2    1      WINDOW (SORT PUSHED RANK) (Cost=176 Card=24571 Bytes=540562)
 3    2      TABLE ACCESS (FULL) OF 'TRANSAKCJE' (Cost=43 Card=24571 Bytes=540562)
```

# Funkcje analityczne a wydajność (2/2)

```
SELECT kwota, data, nr_konta
FROM   transakcje T1
WHERE 3 >= ( SELECT COUNT(*)
            FROM   transakcje
            WHERE kwota >= T1.kwota);

Całkowity: 00:10:36.01
```

## Plan wykonywania

```
-----
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=52890 ...)
1  0  FILTER
2  1  TABLE ACCESS (FULL) OF 'TRANSAKCJE' (Cost=43 ...)
3  1  SORT (AGGREGATE)
4  3  TABLE ACCESS (FULL) OF 'TRANSAKCJE' (Cost=43 ...)
```

- Przykład 2. Podaj największą transakcję z każdego roku

```
SELECT to_char(data,'YYYY') rok, MAX(kwota) przychod,
       MIN(data) KEEP (DENSE_RANK FIRST ORDER BY kwota DESC) data
FROM   transakcje
WHERE  nr_konta = '11-11111111'
GROUP BY TO_CHAR(data,'YYYY')
```

## Plan wykonywania

```
-----
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=71 ...)
1  0  SORT (GROUP BY) (Cost=71 Card=48 Bytes=1056)
2  1  TABLE ACCESS (FULL) OF 'TRANSAKCJE' (Cost=43 ...)
```

```
SELECT to_char(data,'YYYY') rok,
       kwota przychod,
       data
FROM   transakcje T1
WHERE  nr_konta = '11-11111111'
AND    (to_char(data,'YYYY'),kwota) IN
      ( SELECT to_char(data,'YYYY'), max(kwota)
        FROM   transakcje
        WHERE  nr_konta = '11-11111111'
        GROUP BY to_char(data,'YYYY'))
```

## Plan wykonywania

```
-----
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=116 ...)
1  0  HASH JOIN (Cost=116 Card=1 Bytes=39)
2  1  VIEW OF 'VW_NSO_1' (Cost=71 ...)
3  2  SORT (GROUP BY) (Cost=71 Card=48 Bytes=1056)
4  3  TABLE ACCESS (FULL) OF 'TRANSAKCJE' (Cost= ...)
5  1  TABLE ACCESS (FULL) OF 'TRANSAKCJE' (Cost=43 ...)
```