SQL do zaawansowanych analiz danych – część 2.

Funkcje analityczne

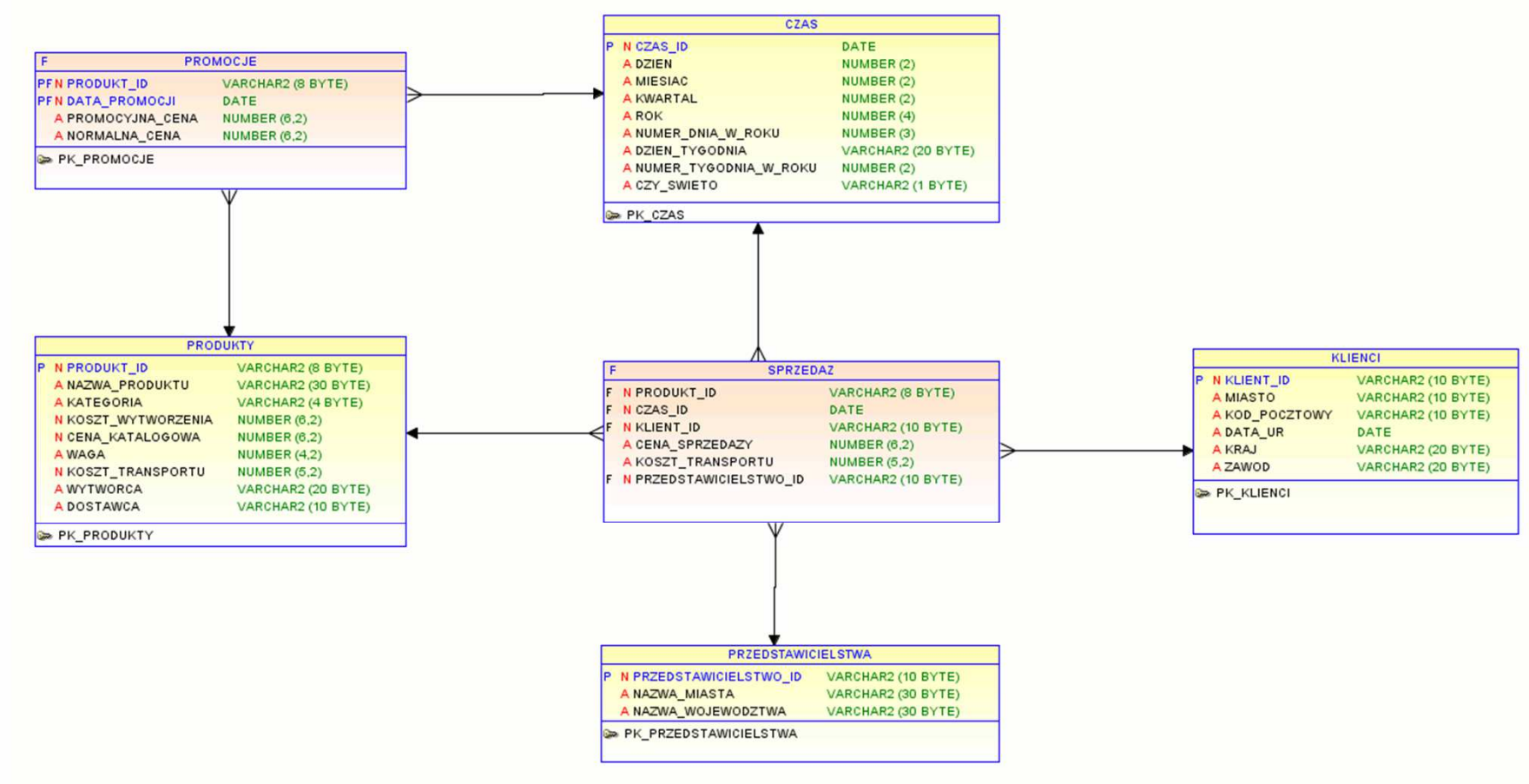
Materiały wykładowe

Bartosz Bębel
Politechnika Poznańska, Instytut Informatyki

Plan wykładu

1. Podstawowe definicje.
2. Sposób działania funkcji analitycznych.
3. Funkcje rankingu.
4. Funkcje okna.
5. Pozostałe funkcje.

Schemat użyty w przykładach



Podstawowe definicje (1)

- Bieżący rekord – rekord, dla którego wyliczana jest wartość funkcji analitycznej.
- Partycja – grupa rekordów, przetwarzanych przez zapytanie; podział realizowany na podstawie wartości jednej/wielu kolumn lub wyrażeń; możliwości:
  - jedna partycja, zawierająca wszystkie rekordy zapytania,
  - wiele partycji, każda z podzbiorem rekordów zapytania.

Podstawowe definicje (2)

- Okno – zbiór rekordów, definiowanych dla bieżącego rekordu partycji:
  - określa, ile rekordów „przed” (początek okna) i ile „za” (koniec okna) bieżącym jest brane pod uwagę przy obliczeniach,
  - możliwości:
    - wszystkie rekordy partycji należą do jednego okna,
    - okno przesuwa się wraz ze zmianą bieżącego rekordu.
  - definicja rozmiaru okna:
    - liczba rekordów,
    - wyrażenie definiujące przedział (np. czas).

Etapy wykonania polecenia SQL

1. Połączenia, klauzule **`WHERE`**, **`GROUP BY`**, **`HAVING`** .
2. Podział zbioru rekordów na partycje.
3. **Zastosowanie funkcji analitycznej dla każdego rekordu partycji.**
4. Wykonanie klauzuli **`ORDER BY`** .

Uwaga! Wynik działania funkcji analitycznej nie może być bezpośrednio używany do filtrowania danych zapytania!

Użycie funkcji analitycznych

- Składnia:

```
nazwa_funkcji(<lista parametrów funkcji>)
      OVER ([<definicja partycji>]
             <porządek sortowania w partycji>
             [<definicja okna>])
```

- Elementy:
  - definicja partycji – określa podział zbioru rekordów na partycje, element opcjonalny,
  - porządek sortowania rekordów w partycji – element obowiązkowy,
  - definicja okna – tylko dla funkcji okna.

Partycjonowanie zbioru rekordów

- Określa podział zbioru rekordów zapytania na oddzielne podzbiory, nazywane **partycjami**.
  - Uwaga! Nie mylić z partycjonowaniem relacji!
- Składnia:

```
PARTITION BY <lista wyrażeń>
```

- Przykłady:

```
PARTITION BY miesiąc
```

```
PARTITION BY kategoria, nazwa_produktu
```

```
PARTITION BY extract(year from czas_id)
```

Porządkowanie zbioru rekordów (1)

- Określa sposób uporządkowania rekordów w zbiorze.
- Jest elementem funkcji analitycznej (w odróżnieniu od klauzuli **ORDER BY** na końcu zapytania) – nie służy do porządkowania wyników zapytania.
- Składnia:

```
ORDER BY <wyrażenie porządkujące>
                        [ASC | DESC]
                        [NULLS FIRST | NULLS LAST][,…]
```

Porządkowanie zbioru rekordów (2)

- Klauzule **`NULLS FIRST`**, **`NULLS LAST`** – wskazują sposób traktowania pustych wartości wyrażenia porządkującego:

Typ	NULLS FIRST	NULLS LAST
porządek rosnący (ASC)	na początku zbioru	na końcu zbioru (domyślnie)
porządek malejący (DESC)	na początku zbioru (domyślnie)	na końcu zbioru

Podział funkcji analitycznych

- Funkcje rankingowe.
- Funkcje rankingu hipotetycznego.
- Funkcje okna.
- Funkcje raportujące.
- Pozostałe funkcje.

Funkcje rankingowe

- Wyliczają wartości na podstawie pozycji bieżącego rekordu w rankingu rekordów w partycji.
- Dostępne funkcje:
  - **`RANK`** i **`DENSE_RANK`** – ranking zwykły,
  - **`CUME_DIST`** – ranking względny,
  - **`PERCENT_RANK`** – ranking procentowy,
  - **`ROW_NUMBER`** – numer rekordu,
  - **`NTILE`** – podział partycji na grupy.

RANK i DENSE_RANK (1)

- Wyliczają pozycję wartości z bieżącego rekordu partycji w rankingu dla partycji.
- Przykłady analiz:
  - Znajdź pięć najlepiej sprzedających się produktów ostatniego kwartału.
  - Podaj nazwiska trzech przedstawicieli handlowych z najgorszymi wynikami sprzedaży.
- Różnice między **RANK** a **DENSE_RANK**:

DOSTAWCA	SUMA	RANK	DENSE_RANK
D-Dell	3060392,88	1	1
BCF	187114,20	2	2
Tefal	181009,08	3	3
Touch Inc	64374,24	4	4
Agfa	55000,00	5	5
CD Inc	55000,00	5	5
Mop Inc	55000,00	5	5
Ricoh	9630,36	8	6

RANK i DENSE_RANK (2)

- Składnia:

```
RANK() | DENSE_RANK()
      OVER ([<definicja partycji>]
             <porządek sortowania>)
```

- Definicja partycji:
  - wiele partycji – jeden ranking dla każdej partycji,
  - brak partycjonowania – jeden ranking w całym zbiorze rekordów zapytania.
- Porządek sortowania – definiuje kryterium ustawiające ranking.

RANK i DENSE_RANK (3)

- Przykład 1. Zbuduj ranking dostawców ze względu na sumę cen sprzedaży produktów przez nich dostarczanych.
  - brak partycjonowania, ranking malejący

```
SELECT dostawca,
       SUM(cena_sprzedazy) AS suma,
       RANK() OVER (
           ORDER BY SUM(cena_sprzedazy) DESC)
           AS ranking
FROM produkty NATURAL JOIN sprzedaz
GROUP BY dostawca
ORDER BY suma DESC, dostawca;
```

DOSTAWCA	SUMA	RANKING
D-Dell	3060392,88	1
BCF	187114,20	2
Tefal	181009,08	3
Touch Inc	64374,24	4
Agfa	57276,44	5
CD Inc	40365,92	6
Mop Inc	37577,64	7
Ricoh	9630,36	8

RANK i DENSE_RANK (4)

- Przykład 2. Zbuduj ranking dostawców ze względu na sumę cen sprzedaży produktów przez nich dostarczanych, z podziałem na kolejne miesiące 1999 roku.
  - partycjonowanie – miesiące 1999 roku, ranking malejący.

RANK i DENSE_RANK (5)

```
SELECT dostawca, miesiac,
  SUM(cena_sprzedazy) AS suma,
  RANK() OVER (
      PARTITION BY miesiac
      ORDER BY SUM(cena_sprzedazy) DESC)
      AS ranking
FROM produkty NATURAL JOIN sprzedaz
              NATURAL JOIN czas
WHERE rok = '1999'
GROUP BY dostawca, miesiac
ORDER BY miesiac, suma DESC, dostawca;
```

DOSTAWCA	MIESIAC	SUMA	RANKING
D-Dell	1	1530196,44	1
BCF	1	93557,1	2
Tefal	1	90504,54	3
Touch Inc	1	32187,12	4
Agfa	1	28638,22	5
CD Inc	1	20182,96	6
Mop Inc	1	18741,32	7
Ricoh	1	4871,2	8
D-Dell	2	1530196,44	1
BCF	2	93557,1	2
Tefal	2	90504,54	3
Touch Inc	2	32187,12	4
Agfa	2	28638,22	5
CD Inc	2	20182,96	6
Mop Inc	2	18741,32	7
Ricoh	2	4759,16	8
Mop Inc	3	84	1
Mop Inc	4	11	1

CUME_DIST (1)

- Oblicza skumulowany rozkład wartości w partycji – pokazuje, ile procent rekordów w partycji poprzedza w rankingu bieżący rekord (tzw. percentyl):
  - wyliczona wartość zawiera się zawsze w przedziale (0; 1>,
  - do wyliczonej wartości wlicza się bieżący rekord.
- Sposób wyliczenia **`CUME_DIST`** dla wartości **`x`** (**`x`** musi wyznaczać ranking!):

```
(liczba rekordów w partycji z wartościami w rankingu „przed”
x razem z rekordami w partycji z wartością x) / liczba
wszystkich rekordów w partycji
```

CUME_DIST (2)

- Składnia:

```
CUME_DIST()
  OVER ([<definicja partycji>] <porządek sortowania>)
```

  - znaczenie poszczególnych elementów analogiczne jak dla **RANK** i **DENSE_RANK**.

CUME_DIST (3)

- Przykład. Znajdź względne pozycje poszczególnych dostawców w rankingu ze względu na sumę cen sprzedaży produktów przez nich dostarczanych.
  - brak partycjonowania, ranking malejący.

```
SELECT dostawca,
 SUM(cena_sprzedazy) AS suma,
 CUME_DIST() OVER (
  ORDER BY SUM(cena_sprzedazy) DESC)
  AS cume_dist,
 100*CUME_DIST() OVER (
  ORDER BY SUM(cena_sprzedazy) DESC)
  AS proc
FROM produkty NATURAL JOIN sprzedaz
GROUP BY dostawca
ORDER BY suma DESC, dostawca;
```

```
                DOSTAWCA       SUMA CUME_DIST  PROC
                -------- ---------- --------- -----
(0+1)/8->       D-Dell   3060392,88      ,125  12,5
(1+1)/8->       BCF        187114,2      ,25    25
(2+1)/8->       Tefal     181009,08      ,375  37,5
 …              Touch Inc  64374,24       ,5    50
                Agfa       57276,44      ,625  62,5
                CD Inc     40365,92      ,75    75
                Mop Inc    37577,64      ,875  87,5
(7+1)/8->       Ricoh       9630,36         1   100
```

PERCENT_RANK (1)

- Wylicza procentowy ranking wartości z bieżącego rekordu w stosunku do zbioru wartości w partycji:
  - wyliczona wartość zawiera się zawsze w przedziale <0; 1>,
  - działa podobnie jak **CUME_DIST**, ale pomija bieżący rekord,
- Sposób wyliczenia **PERCENT_RANK** dla rekordu:

```
(pozycja rekordu w rankingu w partycji – 1)/
(liczba rekordów w partycji – 1)
```

- Składnia:

```
PERCENT_RANK()
  OVER ([<definicja partycji>] <porządek sortowania>)
```

PERCENT_RANK (2)

- Przykład. Znajdź pozycję dostawców w rankingu procentowym utworzonym ze względu na sumę cen sprzedaży produktów przez nich dostarczanych.
  - brak partycjonowania, ranking malejący.

```
SELECT dostawca,
  SUM(cena_sprzedazy) AS suma,
  PERCENT_RANK() OVER
   (ORDER BY SUM(cena_sprzedazy) DESC)
   AS perc_rank
FROM produkty NATURAL JOIN sprzedaz
GROUP BY dostawca
ORDER BY suma DESC, dostawca;
```

DOSTAWCA	SUMA	PERC_RANK	
D-Dell	3060392,88	0	<-(1-1)/(8-1)
BCF	187114,2	,142857143	<-(2-1)/(8-1)
Tefal	181009,08	,285714286	<-(3-1)/(8-1)
Touch Inc	64374,24	,428571429	…
Agfa	57276,44	,571428571	
CD Inc	40365,92	,714285714	
Mop Inc	37577,64	,857142857	
Ricoh	9630,36	1	<-(8-1)/(8-1)

ROW_NUMBER (1)

- Przydziela każdemu rekordowi w partycji unikalny numer, wartości generowane sekwencyjnie od 1.
- Składnia:

```
ROW_NUMBER()
   OVER ([<definicja partycji>] <porządek sortowania>)
```

- Uwaga! Funkcja niedeter-ministyczna!
- Przykład: ranking ze względu na sumę cen.

DOSTAWCA	SUMA	ROW_NUMBER
D-Dell	3060392,88	1
BCF	187114,20	2
Tefal	181009,08	3
Touch Inc	64374,24	4
Agfa	57276,44	5
Mop Inc	57276,44	6
CD Inc	57276,44	7
Ricoh	9630,36	8

ROW_NUMBER (2)

- Przykład. Nadaj kolejne numery rekordom w rankingu dostawców, zbudowanym ze względu na sumę cen sprzedaży produktów przez nich dostarczanych, z podziałem na kolejne miesiące 1999 roku.
  - partycjonowanie – miesiące 1999 roku, ranking malejący.

```
SELECT dostawca, miesiac,
  SUM(cena_sprzedazy) AS suma,
  ROW_NUMBER() OVER (
    PARTITION BY miesiac
    ORDER BY SUM(cena_sprzedazy) DESC) AS rn
FROM produkty NATURAL JOIN sprzedaz
              NATURAL JOIN czas
WHERE rok = '1999'
GROUP BY dostawca, miesiac
ORDER BY miesiac, suma DESC;
```

DOSTAWCA	MIESIAC	SUMA	RN
D-Dell	1	1530196,44	1
BCF	1	93557,1	2
Tefal	1	90504,54	3
Touch Inc	1	32187,12	4
Agfa	1	28638,22	5
CD Inc	1	20182,96	6
Mop Inc	1	18741,32	7
Ricoh	1	4871,2	8
D-Dell	2	1530196,44	1
BCF	2	93557,1	2
…			

NTILE (1)

- Dzieli partycję na zadaną liczbę grup:
  - każdy grupa w partycji otrzymuje unikalny numer,
  - liczby rekordów w grupach różnią się o co najwyżej 1 (w sytuacji, gdy liczba rekordów w partycji nie pozwala na umieszczenie w grupach tej samej liczby rekordów), np.:
    - podział 103 rekordów na 5 grup: 21, 21, 21, 20, 20.
- Składnia:

```
NTILE(<wyrażenie>)
  OVER ([<definicja partycji>]<porządek sortowania>)
```

  - wyrażenie – liczba grup w partycji
- Uwaga! Funkcja niedeterministyczna!

NTILE (2)

- Przykład. Dla każdego miesiąca 1999 roku wylicz kwotę sprzedaży, podziel wynik na kwartały.
  - brak partycjonowania.

```
SELECT miesiac,
       SUM(cena_sprzedazy) AS suma,
       NTILE(4) OVER
         (ORDER BY miesiac) AS kwartał
FROM czas NATURAL JOIN sprzedaz
WHERE rok = 1999
GROUP BY miesiac
ORDER BY miesiac;
```

MIESIAC	SUMA	KWARTAŁ
1	1819005,90	1
2	1818966,86	1
3	3234213,90	1
4	874345,21	2
5	1819015,9	2
6	1818954,86	2
7	592300,84	3
8	4974302,42	3
9	3575380,00	3
10	964978,21	4
11	3098762,31	4
12	1298764,10	4

Funkcje rankingu hipotetycznego (1)

- Umożliwiają analizę "co, jeśli", wyznaczając **hipotetyczny** ranking wartości.
- Składnia:

```
RANK | DENSE_RANK | PERCENT_RANK | CUME_DIST (<wyrażenie>)
     WITHIN GROUP (<porządek sortowania>)
```

  - wyrażenie definiuje wartość, braną pod uwagę przy ustalaniu pozycji w rankingu.

Funkcje rankingu hipotetycznego (2)

- Przykład. Na którym miejscu w rankingu dostawców produktów znalazłby się dostawca, dla którego sprzedaż produktów wynosi 100 000 zł?

```
SELECT RANK(100000) WITHIN GROUP (ORDER BY SUM(cena_sprzedazy) DESC) AS pozycja
FROM produkty NATURAL JOIN sprzedaz GROUP BY dostawca;
```

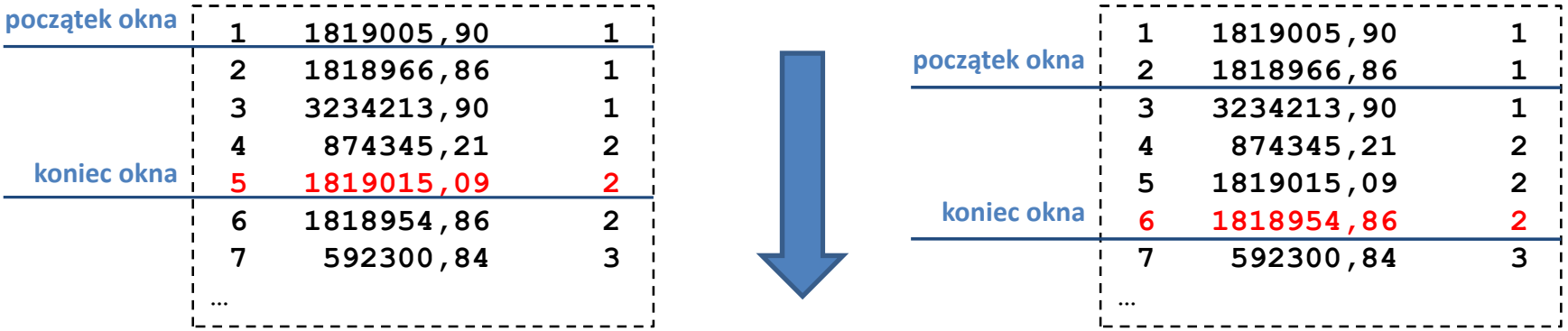
POZYCJA
4

Rzeczywisty ranking:

DOSTAWCA	SUMA	RANKING
D-Dell	3060392,88	1
BCF	187114,20	2
Tefal	181009,08	3
Touch Inc	64374,24	4
Agfa	57276,44	5
CD Inc	40365,92	6
Mop Inc	37577,64	7
Ricoh	9630,36	8

Funkcje okna

- Wyliczają wartość dla bieżącego rekordu, biorąc pod uwagę wartości innych rekordów należących do tego samego **okna** co bieżący rekord.
- Przykład. Wylicz średnią wielkość sprzedaży z bieżącego miesiąca i trzech miesięcy poprzedzających bieżący miesiąc.



Definicja okna – typ okna

- Okno fizyczne – rozmiar wyrażony w liczbie rekordów, słowo kluczowe **`ROWS`**.
- Okno logiczne – rozmiar zdefiniowany przez wyrażenie logiczne, wyliczające przesunięcie względem bieżącego rekordu, słowo kluczowe **`RANGE`**.
  - przykład: „2 miesiące przed i 1 miesiąc po”,
  - typ wartości wyrażenia musi być zgodny z typem wartości wyrażenia porządkującego rekordy w partycji.

Definicja okna – rozmiar okna (1)

- Definiowany:
  - wyrażaniem **`BETWEEN`** … **`AND`** … – umożliwia wskazanie zarówno początku jak i końca okna,
  - przez podanie jedynie początku okna – domyślnie końcem okna jest bieżący wiersz.
- Dostępne wyrażenia:
  - **`UNBOUNDED PRECEDING`** – początkiem okna jest pierwszy rekord partycji (używane tylko przy definiowaniu początku okna),
  - **`UNBOUNDED FOLLOWING`** – końcem okna jest ostatni rekord partycji (używane tylko przy definiowaniu końca okna),

Definicja okna – rozmiar okna (2)

- Dostępne wyrażenia (cd):
  - **`CURRENT ROW`** – początkiem (końcem) okna jest:
    - dla **`ROWS`** – bieżący rekord w partycji,
    - dla **`RANGE`** – wskazana wartość w bieżącym rekordzie partycji.
  - **`<wyrażenie> PRECEDING`**:
    - dla **`ROWS`** – przesunięcie wyrażone w liczbie rekordów od bieżącego rekordu do początku okna,
    - dla **`RANGE`** – przesunięcie logiczne od bieżącego rekordu do początku okna.

Definicja okna – rozmiar okna (3)

- Dostępne wyrażenia (cd):
  - **<wyrażenie> FOLLOWING:**
    - dla **ROWS** – przesunięcie wyrażone w liczbie rekordów od bieżącego rekordu do końca okna,
    - dla **RANGE** – przesunięcie logiczne od bieżącego rekordu do końca okna.

```
{ ROWS | RANGE }
{ BETWEEN { UNBOUNDED PRECEDING | CURRENT ROW |
            <wyrażenie> { PRECEDING | FOLLOWING }}
  AND { UNBOUNDED FOLLOWING | CURRENT ROW |
            <wyrażenie> { PRECEDING | FOLLOWING }}
| { UNBOUNDED PRECEDING | CURRENT ROW | <wyrażenie> PRECEDING }}
```

Definicja okna – przykłady (1)

- Okno fizyczne o rozmiarze 16 rekordów: 5 przed bieżącym rekordem + bieżący rekord + 10 po bieżącym rekordzie:

```
ROWS BETWEEN 5 PRECEDING AND 10 FOLLOWING
```

- Okno fizyczne od początku partycji do bieżącego rekordu:

```
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Definicja okna – przykłady (2)

- Okno logiczne o rozmiarze 10 ubiegłych dni i 5 dni „przyszłych”:

```
RANGE BETWEEN INTERVAL '10' DAY PRECEDING AND
              INTERVAL '5' DAY FOLLOWING
```

- Okno logiczne zawierające rekordy o wartościach większych bądź równych 10 i mniejszych bądź równych 20:

```
RANGE BETWEEN 10 PRECEDING AND 20 FOLLOWING
```

Definicja okna – przykłady (3)

- Okno logiczne obejmujące 6 miesięcy wstecz:

```
RANGE INTERVAL '5' MONTH PRECEDING
```

- Uwaga! Całkowite pominięcie definicji okna powoduje przyjęcie okna o w postaci: wszystkie rekordy w partycji przed rekordem bieżącym + bieżący rekord:

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

w skrócie

```
RANGE UNBOUNDED PRECEDING
```

Funkcje okna – przykłady (1)

- Przykład 1. Dla każdego miesiąca pierwszej połowy 1999 roku wylicz sumaryczną kwotę sprzedaży, podaj również dla każdego miesiąca sumaryczną sprzedaż od początku roku.

```
SELECT miesiac,
  SUM(cena_sprzedazy) AS miesiecznie,
  SUM(SUM(cena_sprzedazy))
    OVER(ORDER BY miesiac
         ROWS BETWEEN
            UNBOUNDED PRECEDING
            AND CURRENT ROW) AS narast
FROM czas JOIN sprzedaz USING(czas_id)
WHERE rok = 1999 AND miesiac <=6
GROUP BY miesiac
ORDER BY miesiac;
```

MIESIAC	MIESIECZNIE	NARAST
1	1819005,90	1819005,90
2	1818966,86	3637972,76
3	3234213,90	6872186,66
4	874345,21	7746531,87
5	1819015,90	9565547,77
6	1818954,86	11384502,63

```
SUM(SUM(cena_sprzedazy))
 OVER(ORDER BY miesiac
      ROWS UNBOUNDED PRECEDING)
```

albo

```
SUM(SUM(cena_sprzedazy))
   OVER(ORDER BY miesiac)
```

Funkcje okna – przykłady (2)

- Przykład 2. Podaj sumaryczną dzienną sprzedaż produktów w pierwszym kwartale 2001 roku, dodatkowo wylicz procentowy udział sprzedaży dziennej w stosunku do sumarycznej sprzedaży tygodniowej.

```
SELECT czas_id AS data, numer_tygodnia_w_roku AS tyg, sum(cena_sprzedazy) AS suma_dz,
 ROUND(100*SUM(cena_sprzedazy)/SUM(SUM(cena_sprzedazy)) OVER (
   PARTITION BY numer_tygodnia_w_roku ORDER BY czas_id
   RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)) AS "%TYG"
FROM sprzedaz JOIN czas USING(czas_id)
WHERE rok = 2001 AND kwartal = 1
GROUP BY czas_id, numer_tygodnia_w_roku
ORDER BY czas_id;
```

DATA	TYG	SUMA_DZ	%TYG
01/01/01	1	12	5
01/01/02	1	32	13
01/01/03	1	9	4
01/01/04	1	32	13
01/01/05	1	54	21
01/01/06	1	72	28
01/01/07	1	43	17
01/01/08	2	47	15
…			

FIRST_VALUE i LAST_VALUE (1)

- Umożliwiają odczyt wartości wyrażenia z pierwszego/ ostatniego rekordu okna.
- Składnia:

```
FIRST_VALUE | LAST_VALUE(<wyrażenie>)
  [RESPECT NULLS | IGNORE NULLS]
  OVER ([<definicja partycji>] <porządek sortowania>
        [<definicja okna>])
```

  - **`IGNORE NULLS`** – przy wyszukaniu rekordu zostaną pominięte rekordy z pustą wartością wyrażenia (domyślnie **`RESPECT NULLS`**):
    - dla **`FIRST_VALUE`** – pierwsza niepusta wartość w zbiorze lub NULL gdy wszystkie puste,
    - dla **`LAST_VALUE`** – ostatnia niepusta wartość w zbiorze lub NULL gdy wszystkie puste.

FIRST_VALUE i LAST_VALUE (2)

- Przykład: Podaj sumaryczną dzienną sprzedaż produktów w pierwszym kwartale 2001 roku, porównaj sprzedaż dzienną z sumaryczną sprzedażą każdego pierwszego dnia tygodnia.

```
SELECT czas_id AS dzien,
  numer_tygodnia_w_roku AS nr_tyg,
  SUM(cena_sprzedazy) AS suma_dzien,
  SUM(cena_sprzedazy)-
     FIRST_VALUE(SUM(cena_sprzedazy))
       OVER (PARTITION BY numer_tygodnia_w_roku
             ORDER BY czas_id
             RANGE BETWEEN UNBOUNDED PRECEDING
             AND UNBOUNDED FOLLOWING) AS zmiana
FROM sprzedaz JOIN czas USING(czas_id)
WHERE rok = 2001 AND kwartal = 1
GROUP BY czas_id, numer_tygodnia_w_roku
ORDER BY czas_id;
```

DZIEN	NR_TYG	SUMA_DZIEN	ZMIANA
01/01/01	1	12	0
01/01/02	1	32	20
01/01/03	1	9	-3
01/01/04	1	32	20
01/01/05	1	54	42
01/01/06	1	72	60
01/01/07	1	43	31
01/01/08	2	47	0
01/01/09	2	54	7
01/01/10	2	93	46
01/01/11	2	32	-15
…			

NTH_VALUE (1)

- Umożliwia odczyt wartości wyrażenia ze wskazanego rekordu okna.
- Składnia:

```
NTH_VALUE(<wyrażenie>, <n>)
   [FROM FIRST | FROM LAST]
   [RESPECT NULLS | IGNORE NULLS]
   OVER ([<definicja partycji>] <porządek sortowania>
         [<definicja okna>])
```

  - **n** – numer wiersza w oknie,
  - **FROM FIRST**, **FROM LAST** – czy obliczenia rozpoczną się od pierwszego (**FROM FIRST**, domyślnie) czy od ostatniego (**FROM LAST**) wiersza w oknie.

NTH_VALUE (2)

- Przykład: Podaj sumaryczną dzienną sprzedaż produktów w pierwszym kwartale 2001 roku, porównaj sprzedaż dzienną z sumaryczną sprzedażą każdego pierwszego dnia tygodnia.

```
SELECT czas_id AS dzien, numer_tygodnia_w_roku AS nr_tyg,
  SUM(cena_sprzedazy) AS suma_dzien,
  SUM(cena_sprzedazy)-
     NTH_VALUE(SUM(cena_sprzedazy), 1)
       FROM FIRST
       OVER (PARTITION BY numer_tygodnia_w_roku
             ORDER BY czas_id
             RANGE BETWEEN UNBOUNDED PRECEDING
             AND UNBOUNDED FOLLOWING) AS zmiana
FROM sprzedaz JOIN czas USING(czas_id)
WHERE rok = 2001 AND kwartal = 1
GROUP BY czas_id, numer_tygodnia_w_roku
ORDER BY czas_id;
```

DZIEN	NR_TYG	SUMA_DZIEN	ZMIANA
01/01/01	1	12	0
01/01/02	1	32	20
01/01/03	1	9	-3
01/01/04	1	32	20
01/01/05	1	54	42
01/01/06	1	72	60
01/01/07	1	43	31
01/01/08	2	47	0
01/01/09	2	54	7
01/01/10	2	93	46
01/01/11	2	32	-15
…			

Funkcje raportujące (1)

- Wyliczają zagregowane wartości na podstawie wartości wszystkich rekordów w partycji.
- Wyniki są dostępne dla każdego rekordu w partycji.
- Funkcje: **SUM**, **AVG**, **MAX**, **MIN**, **COUNT**, **STDDEV**, **VARIANCE**
- Składnia:

```
<nazwa funkcji>(ALL|DISTINCT <wyrażenie>)
       OVER ([<definicja partycji>])
```

  - przy braku definicji partycji wartość funkcji wyliczana jest na podstawie wszystkich rekordów zapytania.

Funkcje raportujące (2)

- Przykład. Dla każdego produktu podaj jego sumaryczną kwotę sprzedaży oraz sumaryczną kwotę sprzedaży wszystkich produktów, należących do tej samej kategorii, co dany produkt.

```
SELECT nazwa_produktu AS prod,
kategoria AS kat,
 SUM(cena_sprzedazy) AS sprz_prod,
 SUM(SUM(cena_sprzedazy))
   OVER (PARTITION BY kategoria) AS sprz_kat,
 ROUND(100*SUM(cena_sprzedazy)/
  SUM(SUM(cena_sprzedazy))
   OVER (PARTITION BY kategoria),2) AS "UDZ%"
FROM produkty JOIN sprzedaz USING(produkt_id)
GROUP BY nazwa_produktu, kategoria
ORDER BY  kategoria, nazwa_produktu;
```

PROD	KAT	SPRZ_PROD	SPRZ_KAT	UDZ%
…				
Komputer LX8	ELEK	135045,88	3560155,2	3,79
Lampa SX5	ELEK	2299,08	3560155,2	0,06
Lampa LX8	ELEK	2223,08	3560155,2	0,06
Kuchenka LX3	ELEK	29372,11	3560155,2	0,83
Kuchenka SX1	ELEK	6704,04	3560155,2	0,19
Mop SX8	KUCH	1441,64	37577,64	3,84
Mop LX9	KUCH	273,90	37577,64	0,73
Mop TX2	KUCH	4343,99	37577,64	11,56
Mop SX5	KUCH	7435,99	37577,64	19,79
…				

RATIO_TO_REPORT (1)

- Wylicza stosunek wartości wyrażenia do sumy wartości wyrażenia ze wszystkich rekordów partycji
  - dla wyrażenia x: **`x/SUM(x)`**
- Składnia:

```
RATIO_TO_REPORT(<wyrażenie>) OVER ([<definicja partycji>])
```

  - przy braku definicji partycji wartość funkcji wyliczana jest na podstawie wszystkich rekordów zapytania.

RATIO_TO_REPORT (2)

- Przykład. Dla każdego produktu podaj jego sumaryczną kwotę sprzedaży oraz sumaryczną kwotę sprzedaży wszystkich produktów, należących do tej samej kategorii, co dany produkt.

```
SELECT nazwa_produktu AS prod,
kategoria AS kat,
 SUM(cena_sprzedazy) AS sprz_prod,
 SUM(SUM(cena_sprzedazy))
   OVER (PARTITION BY kategoria) AS sprz_kat,
 ROUND(100*
  RATIO_TO_REPORT(SUM(cena_sprzedazy))
    OVER (PARTITION BY kategoria),2) AS "UDZ%"
FROM produkty JOIN sprzedaz USING(produkt_id)
GROUP BY nazwa_produktu, kategoria
ORDER BY  kategoria, nazwa_produktu;
```

```
PROD          KAT  SPRZ_PROD SPRZ_KAT   UDZ%
------------ ---- --------- --------- -----
…
Komputer LX8 ELEK 135045,88 3560155,2  3,79
Lampa SX5    ELEK   2299,08 3560155,2  0,06
Lampa LX8    ELEK   2223,08 3560155,2  0,06
Kuchenka LX3 ELEK  29372,11 3560155,2  0,83
Kuchenka SX1 ELEK   6704,04 3560155,2  0,19
Mop SX8      KUCH   1441,64  37577,64  3,84
Mop LX9      KUCH    273,90  37577,64  0,73
Mop TX2      KUCH   4343,99  37577,64 11,56
Mop SX5      KUCH   7435,99  37577,64 19,79
…
```

Pozostałe funkcje analityczne

- **LAG** i **LEAD**,
- **LISTAGG**,
- **FIRST** i **LAST**,
- **PERCENTILE_DISC** i **PERCENTILE_CONT**.

LAG i LEAD (1)

- Umożliwiają odczyt wartości, znajdujących się w rekordach partycji sąsiadujących z bieżącym rekordem.
- Analogiczne do **`NTH_VALUE`** (ale bez definicji okna).
- Składnia:

```
LAG | LEAD(<wyrażenie> [,przesunięcie] [,wartość domyślna])
  [RESPECT NULLS | IGNORE NULL]
  OVER ([<definicja partycji>] <porządek sortowania>)
```

  - wyrażenie – kolumna, której wartość funkcja ma odczytać,

LAG i LEAD (2)

- Składnia (cd):
  - przesunięcie – określa numer rekordu w stosunku do bieżącego rekordu, z którego wartość ma zostać odczytana (domyślnie 1):
    - **`LAG`** – sięga do tyłu (np. 1 – rekord poprzedni)
    - **`LEAD`** – sięga do przodu (np. 1 – rekord następny)
  - wartość domyślna – wartość, jaką przyjmie funkcja w sytuacji, gdy przesunięcie zaadresuje rekord "poza" zbiorem rekordów partycji lub relacji,
  - **`IGNORE NULLS`** – przy wyszukaniu rekordu zostaną pominięte rekordy z pustą wartością wyrażenia (domyślnie **`RESPECT NULLS`**).

LAG i LEAD (3)

- Przykład. Podaj sumaryczną dzienną sprzedaż produktów w pierwszym kwartale 2001 roku, dla każdego dnia podaj również sumaryczną sprzedaż z dnia poprzedniego (0 jeśli dane o sprzedaży z poprzedniego dnia nie są dostępne).

```
SELECT czas_id AS data, SUM(cena_sprzedazy) AS suma_dzien,
       LAG(SUM(cena_sprzedazy),1,0)
          OVER (ORDER BY czas_id) AS suma_pop_dzien
FROM   sprzedaz JOIN czas USING(czas_id)
WHERE  rok = 2001 AND kwartal = 1
GROUP  BY czas_id
ORDER  BY czas_id;
```

DATA	SUMA_DZIEN	SUMA_POP_DZIEN
01/01/01	12	0
01/01/02	32	12
01/01/03	9	32
01/01/04	32	9
01/01/05	54	32
01/01/06	72	54
01/01/07	43	72
01/01/08	47	43
01/01/09	54	47
…		

LISTAGG (1)

- Łączy wartości wskazanego wyrażenia w grupie w jeden ciąg.
- Składnia:

```
LISTAGG(<wyrażenie>, [,separator])
   WITHIN GROUP (<porządek sortowania>)
```

  - wyrażenie – wartości do konkatenacji,
  - separator – znak oddzielający kolejne wartości w ciągu.

LISTAGG (2)

- Przykład. Dla każdego produktu, sprzedanego w pierwszym kwartale 1999 roku, podaj, w formie listy, oddzielone przecinkami identyfikatory klientów, który dany produkt zakupili w danym dniu.

```
SELECT czas_id AS dzien,
   produkt_id AS prod,
   LISTAGG(klient_id,',') WITHIN GROUP(
     ORDER BY klient_id) AS lista_klientow
FROM sprzedaz JOIN czas USING (czas_id)
WHERE rok = '1999' AND kwartal = 1
GROUP BY czas_id, produkt_id
ORDER BY czas_id, produkt_id;
```

```
DZIEN    PROD    LISTA_KLIENTOW
-------- ------- ----------------------
99/01/01  SP1000 AB112,AB322,AB324,…
99/01/01  SP1001 AB4923,AB4214,ZW232,…
99/01/01  SP1003 FY283,HJ123,ZW99,…
99/01/01  SP1031 UY752,ZD289,ZG289,…
99/01/02  SP1000 AB876,ZG289,ZZ921,…
…
```

FIRST i LAST (1)

- Znajdują pierwsze/ostatnie rekordy w rankingu i stosują funkcję agregującą na ich wartościach.
- Są odpowiednikami **`FIRST_VALUE`** i **`LAST_VALUE`** jednak bez definicji okna.
- Typowe zastosowanie: odczytaj zagregowane wartości kolumny A z rekordu/ów na szczycie rankingu ustalonego wg wartości kolumny B.

FIRST i LAST (2)

- Składnia:

```
nazwa_funkcji(<wyrażenie>) KEEP (DENSE_RANK {FIRST | LAST}
<porządek sortowania>) [OVER (<definicja partycji>)]
```

- Działają jako:
  1. zwykłe funkcje agregujące – pomijamy definicję partycji, funkcja działa na całej grupie rekordów, definiowanej przez klauzulę **`GROUP BY`**,
  2. agregujące funkcje raportujące – działają w ramach partycji.

FIRST i LAST (3)

- Przykład 1. (zwykła funkcja agregująca). Podaj sumaryczną dzienną sprzedaż produktów w 2001 roku, dla każdego dnia podaj sumaryczną kwotę sprzedaży oraz identyfikator klienta, który zakupił produkt o najwyższej cenie.

```
SELECT czas_id AS dzien,
  SUM(cena_sprzedazy) AS sprz_dz,
  MAX(klient_id) KEEP (DENSE_RANK FIRST
   ORDER BY cena_sprzedazy DESC) AS maks_kl
FROM sprzedaz JOIN czas USING (czas_id)
WHERE rok = '2001'
GROUP BY czas_id
ORDER BY czas_id;
```

DZIEN	SPRZ_DZ	MAKS_KL
01/01/01	12	AB231
01/01/02	32	BH323
01/01/03	9	AB546
01/01/04	32	ZY454
01/01/05	54	WT754
01/01/06	72	HN872
01/01/07	43	UY609
01/01/08	47	AB231
01/01/09	54	AB231
01/01/10	93	UY742
01/01/11	32	NM631
…		

FIRST i LAST (4)

- Przykład 2. (raportująca funkcja agregująca). Dla każdej transakcji sprzedaży, przeprowadzonej w 1999 roku, podaj jej kwotę oraz różnicę pomiędzy kwotą transakcji a kwotą pierwszej transakcji sprzedaży w miesiącu.

```
SELECT czas_id AS data, cena_sprzedazy,
  cena_sprzedazy -
   MIN(cena_sprzedazy)
     KEEP (DENSE_RANK LAST
           ORDER BY czas_id DESC)
           OVER (PARTITION BY miesiac)
  AS roznica
FROM sprzedaz JOIN czas USING(czas_id)
WHERE rok = 1999
ORDER BY czas_id;
```

DATA	CENA_SPRZEDAZY	ROZNICA
99/01/01	231	0
99/01/02	123	-108
99/01/03	543	312
…		
99/01/31	434	0
99/02/01	213	-221
99/02/02	432	-2
99/02/03	323	-111
…		

Funkcje odwróconych percentyli

- Informują, jaka wartość znajduje się na określonej pozycji w uporządkowanym zbiorze wartości (odwrotność działania **CUME_DIST**).
- Składnia:

```
PERCENTILE_CONT | PERCENTILE_DISC(<wyrażenie>)
  WITHIN GROUP(<porządek sortowania>)
  [OVER (<definicja partycji>)]
```

  - jako zwykłe funkcje agregujące – pomijamy klauzulę **OVER** (brak partycjonowania),
  - jako funkcje raportujące – z definicją partycji w klauzuli **OVER**.

PERCENTILE_DISC (1)

- Funkcja dyskretna.
- Algorytm działania:
  1. funkcja "przegląda" wyniki funkcji **CUME_DIST**,
  2. pierwszy wiersz, w którym wynik **CUME_DIST** jest większy bądź równy od argumentu **PERCENTILE_CONT**, wyznacza poszukiwaną wartość.

PERCENTILE_DISC (2)

- Przykład. Znajdź w rankingu dostawców (ranking ze względu na sumaryczną cenę dostarczanych produktów) wartość dokładnie w środku zbioru (medianę).

```
SELECT PERCENTILE_DISC(0.5) WITHIN GROUP
     (ORDER BY SUM(cena_sprzedazy) DESC) AS mediana
FROM produkty NATURAL JOIN sprzedaz
GROUP BY dostawca;
```

MEDIANA
64374,24

ranking CUME_DIST:

DOSTAWCA	SUMA	CUME_DIST
D-Dell	3060392,88	,125
BCF	187114,2	,25
Tefal	181009,08	,375
Touch Inc	64374,24	,5
Agfa	57276,44	,625
CD Inc	40365,92	,75
Mop Inc	37577,64	,875
Ricoh	9630,36	1

PERCENTILE_CONT (1)

- Funkcja ciągła.
- Wynik wyznaczany jest przez liniową interpolację wierszy otaczających wskazaną pozycję.
- Algorytm działania (x – argument funkcji):
  1. wyznacz RN = 1+x*(n-1) gdzie n jest liczbą wierszy w grupie,
  2. wyznacz CRN = CEIL(RN) i FRN = FLOOR(RN),
  3. jeśli CRN = FRN wówczas wynikiem jest wartość znajdująca się w wierszu na pozycji RN w rankingu, w przeciwnym przypadku wynikiem jest wartość następującego wyrażenia:

(CRN-RN)*(wartość z wiersza CRN)+(RN-FRN)*(wartość z wiersza FRN)

PERCENTILE_CONT (2)

- Przykład. Znajdź w rankingu dostawców (ranking ze względu na sumaryczną cenę dostarczanych produktów) wartość dokładnie w środku zbioru (medianę).

```
SELECT PERCENTILE_CONT(0.5) WITHIN GROUP
     (ORDER BY SUM(cena_sprzedazy) DESC) AS mediana
FROM produkty NATURAL JOIN sprzedaz
GROUP BY dostawca;
```

```
MEDIANA
--------
60825,34
```

CUME_DIST:

DOSTAWCA	SUMA	CUME_DIST
D-Dell	3060392,88	,125
BCF	187114,2	,25
Tefal	181009,08	,375
Touch Inc	64374,24	,5
Agfa	57276,44	,625
CD Inc	40365,92	,75
Mop Inc	37577,64	,875
Ricoh	9630,36	1

```
RN = 1+x*(n-1) = 4,5
CRN=CEIL(RN) = 5 FRN=FLOOR(RN) = 4
MEDIANA = (CRN-RN)*r(CRN)+(RN-FRN)*r(FRN)
MEDIANA = 0,5 * 57276,44 + 0,5 * 64374,24
MEDIANA = 60825,34
```

WIDTH_BUCKET (1)

- Algorytm działania:
  1. Funkcja buduje histogram z zadaną liczbą przedziałów o identycznej szerokości,
  2. Funkcja dla zadanej wartości podaje numer przedziału, do którego dana wartość należy.
- Składnia:

```
WIDTH_BUCKET(<wyrażenie>,
             <min. wartość zakresu>, <maks. wartość zakresu>,
             <liczba przedziałów>)
```

WIDTH_BUCKET (2)

- Przykład. Na podstawie sumy cen zakupionych przez klientów produktów pogrupuj klientów na cztery kategorie w zależności od sumarycznej kwoty zakupów. Minimalny próg analizy to 0, próg maksymalny to 200 000.

```
SELECT klient_id AS klient, SUM(cena_sprzedazy) AS suma,
  WIDTH_BUCKET(SUM(cena_sprzedazy), 0, 200000, 4) AS kategoria
FROM sprzedaz
GROUP BY klient_id
ORDER BY klient_id;
```

KLIENT	SUMA	KATEGORIA
AB123456	10212	1
AB123457	528	1
AB123458	50232	2
AB123459	185343	4
AB123460	201323	5
AB123461	101232	3
...		

Użycie funkcji analitycznych do filtrowania danych

- Przykład. Podaj nazwy dostawców, którzy w rankingu dostawców ze względu na sumę cen sprzedaży produktów przez nich dostarczanych zajmują trzy pierwsze miejsca.

```
SELECT dostawca, suma, ranking
FROM
  (SELECT dostawca,
      SUM(cena_sprzedazy) AS suma,
      RANK() OVER (
          ORDER BY SUM(cena_sprzedazy) DESC)
          AS ranking
   FROM produkty NATURAL JOIN sprzedaz
   GROUP BY dostawca)
WHERE ranking <= 3
ORDER BY ranking, dostawca;
```

DOSTAWCA	SUMA	RANKING
D-Dell	3060392,88	1
BCF	187114,20	2
Tefal	181009,08	3

Klauzula WITH (1)

- Upraszcza konstrukcję zapytań analitycznych.
- Pozwala na:
  - przydzielenie nazwy do podzapytania,
    - możliwe jest definiowanie wielu podzapytań,
  - następnie odwołanie do podzapytania przez jego nazwę w zapytaniu głównym.

```
WITH nazwa_1[(kolumna_1,…)] AS (SELECT …),
     nazwa_2[(kolumna_2,…)] AS (SELECT …) …
SELECT … FROM nazwa_1 …;
```

Klauzula WITH (2)

- Przykład 1. Podaj numery klientów, którzy zakupili produkty za ponad 80 000.

```
WITH zakupy_klientow AS (
      SELECT klient_id AS klient, SUM(cena_sprzedazy) AS suma
      FROM sprzedaz GROUP BY klient_id)
SELECT klient FROM zakupy_klientow WHERE suma > 80000
ORDER BY suma DESC;
```

```
WITH zakupy_klientow(klient, suma) AS (
      SELECT klient_id, SUM(cena_sprzedazy)
      FROM sprzedaz GROUP BY klient_id)
SELECT klient FROM zakupy_klientow WHERE suma > 80000
ORDER BY suma DESC;
```

Klauzula WITH (3)

- Przykład 2. Podaj nazwy dostawców, którzy w rankingu dostawców ze względu na sumę cen sprzedaży produktów przez nich dostarczanych zajmują trzy pierwsze miejsca.

```
WITH
  dostawcy AS (
    SELECT dostawca, SUM(cena_sprzedazy) AS kwota_dostaw
    FROM produkty NATURAL JOIN sprzedaz GROUP BY dostawca),
  ranking AS (
    SELECT dostawca, kwota_dostaw,
           RANK() OVER (ORDER BY kwota_dostaw DESC) AS ranking
    FROM dostawcy)
SELECT * FROM ranking WHERE ranking <= 3
ORDER BY ranking, dostawca;
```

Bibliografia

1. Dokumentacja techniczna Oracle 11g Release 2 (11.2). www.oracle.com