SQL do zaawansowanych analiz danych – część 1.

Mechanizmy języka SQL dla agregacji danych
Rozszerzenia PIVOT i UNPIVOT

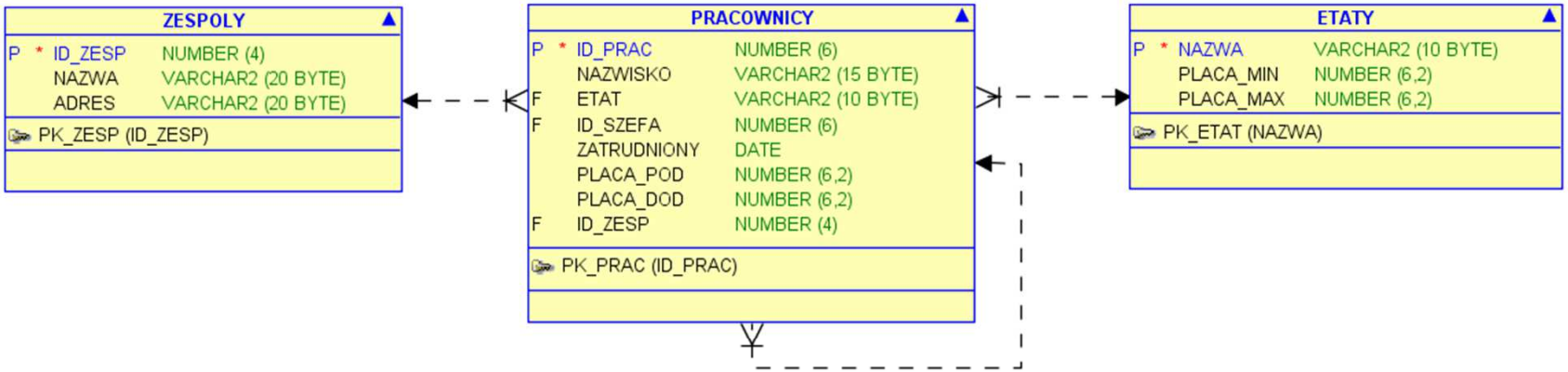
Materiały wykładowe

Bartosz Bębel
Politechnika Poznańska, Instytut Informatyki

Plan wykładu

1. Klauzule: **`GROUP BY`** i **`HAVING`**, funkcje agregujące.
2. Rozszerzenia klauzuli **`GROUP BY`**:
   - **`ROLLUP`**,
   - **`CUBE`**,
   - **`GROUPING SETS`**.
3. Funkcje pomocnicze.
4. Rozszerzenia **`PIVOT`** i **`UNPIVOT`**.

Schemat użyty w przykładach



PRACOWNICY

ID_PRAC	NAZWISKO	ETAT	ID_SZEFA	ZATRUDNIONY	PLACA_POD	PLACA_DOD	ID_ZESP
100	WEGLARZ	DYREKTOR		68/01/01	1730	420,5	10
110	BLAZEWICZ	PROFESOR	100	73/05/01	1350	210	40
120	SLOWINSKI	PROFESOR	100	77/09/01	1070		30
130	BRZEZINSKI	PROFESOR	100	68/07/01	960		20
140	MORZY	PROFESOR	130	75/09/15	830	105	20
150	KROLIKOWSKI	ADIUNKT	130	77/09/01	645,5		20
160	KOSZLAJDA	ADIUNKT	130	85/03/01	590		20
170	JEZIERSKI	ASYSTENT	130	92/10/01	439,7	80,5	20
180	MAREK	SEKRETARKA	100	85/02/20	410,2		10
190	MATYSIAK	ASYSTENT	140	93/09/01	371		20
200	ZAKRZEWICZ	STAZYSTA	140	94/07/15	208		30
210	BIALY	STAZYSTA	130	93/10/15	250	170,6	30
220	KONOPKA	ASYSTENT	110	93/10/01	480		20
230	HAPKE	ASYSTENT	120	92/09/01	480	90	30

ETATY

NAZWA	PLACA_MIN	PLACA_MAX
ADIUNKT	510	750
ASYSTENT	300	500
DYREKTOR	1280	2100
PROFESOR	800	1500
SEKRETARKA	270	450
STAZYSTA	150	250

ZESPOLY

ID_ZESP	NAZWA	ADRES
10	ADMINISTRACJA	PIOTROWO 3A
20	SYSTEMY ROZPROSZONE	PIOTROWO 3A
30	SYSTEMY EKSPERCKIE	STRZELECKA 14
40	ALGORYTMY	WLODKOWICA 16
50	BADANIA OPERACYJNE	MIELZYNSKIEGO 30

Proste grupowania

Klauzula GROUP BY (1)

- Dzieli zbiór rekordów zapytania na podzbiory, nazywane **grupami**.
- Rekordy w grupie – ta sama wartość **wyrażenia grupującego**.
- Funkcja grupowa – w każdej grupie wylicza pojedynczą **wartość** na podstawie wartości parametru.
- Funkcje grupowe eliminują wartości puste.

```
SELECT <lista atrybutów> FROM <lista relacji>
WHERE <warunki filtrujące> GROUP BY <lista wyrażeń grupujących>
ORDER BY <wyrażenie porządkujące>;
```

Klauzula GROUP BY (2)

- Problem: znajdź średnią płacę pracowników dla każdej grupy etatowej.

ETAT	PLACA_POD
ADIUNKT	590
ADIUNKT	645,5
ASYSTENT	439,7
ASYSTENT	480
ASYSTENT	371
ASYSTENT	480
PROFESOR	1350
PROFESOR	830
PROFESOR	960
PROFESOR	1070

grupa dla etatu = 'ADIUNKT'

grupa dla etatu = 'ASYSTENT'

grupa dla etatu = 'PROFESOR'

ETAT	SREDNIA
ADIUNKT	1235,5
ASYSTENT	1770,7
PROFESOR	4210

Klauzula GROUP BY (3)

- Funkcje grupowe:
  - **AVG** – średnia,
  - **COUNT** – liczba wystąpień,
  - **MAX** – maksimum,
  - **MIN** – minimum,
  - **SUM** – suma,
  - **VARIANCE** – wariancja,
  - **STDDEV** – odchylenie stand.
- Użycie: **`nazwa_funkcji(all | distinct wyrażenie)`**
- Szczególny przypadek – funkcja **COUNT**:
  - **`COUNT(*)`** – liczba rekordów,
  - **`COUNT([all | distinct] wyrażenie)`** – liczba niepustych wartości wyrażenia.

Zapytania z jedną grupą (1)

- Wszystkie rekordy należą do **jednej** grupy – brak wyrażenia grupującego – pomija się klauzulę **`GROUP BY`** lub stosuje w postaci **`GROUP BY ( )`**.
- Wynik zapytania – co najwyżej **jeden** rekord.
- Problem 1. Znajdź sumaryczną płacę podstawową wszystkich pracowników.

```
SELECT SUM(placa_pod)
FROM pracownicy;
```

```
SELECT SUM(placa_pod)
FROM pracownicy
GROUP BY ();
```

	PLACA_POD
...	200
...	340
...	null
...	456

SUM

Zapytania z jedną grupą (2)

- Problem 2. Znajdź średnią płacę podstawową wśród pracowników zespołu 20.

```
SELECT AVG(placa_pod)
FROM pracownicy
WHERE id_zesp = 20;
```

- Problem 3. Znajdź maksymalną wartość dodatku oraz liczbę pracowników na etacie ASYSTENT.

```
SELECT COUNT(*), MAX(placa_dod)
FROM pracownicy
WHERE etat = 'ASYSTENT';
```

Zapytania z wieloma grupami (1)

- Konieczne zdefiniowanie wyrażenia grupującego – klauzula **`GROUP BY wyrażenie`**.
- Wynik zapytania – **jeden** rekord dla **każdej** grupy.

NAZWISKO	ID_ZESP	PLACA
WEGLARZ	10	1730
SLOWINSKI	30	1070
BRZEZINSKI	20	960
MORZY	20	830
KOSZLAJDA	20	590
ZAKRZEWICZ	30	208
BIALY	30	250

→

NAZWISKO	ID_ZESP	PLACA
WEGLARZ	10	1730
BRZEZINSKI	20	960
MORZY	20	830
KOSZLAJDA	20	590
SLOWINSKI	30	1070
BIALY	30	250
ZAKRZEWICZ	30	208

```
SELECT id_zesp,
       SUM(placa_pod)
FROM pracownicy
GROUP BY id_zesp;
```

→

ID_ZESP	SUM(PLACA_POD)
10	1730
20	2380
30	1528

Zapytania z wieloma grupami (2)

- Problem 1. Dla każdego etatu wylicz najwyższą płacę wśród pracowników zatrudnionych na tym etacie.

```
SELECT etat, MAX(placa_pod)
FROM pracownicy
GROUP BY etat;
```

- Problem 2. W ramach każdego zespołu dla każdego etatu w zespole oblicz najwyższą płacę.

```
SELECT id_zesp, etat, MAX(placa_pod)
FROM pracownicy
GROUP BY id_zesp, etat;
```

Pozostałe operacje (1)

- Filtrowanie grup – klauzula **`HAVING wyr. logiczne.`**
- Porządkowanie grup – klauzula **`ORDER BY wyrażenie`**
- Przykłady:
  - Problem 1. Wyświetl nazwy etatów, których maksymalna płaca podstawowa przekracza 1000 złotych, wynik posortuj wg etatów, wyświetl również sumaryczne płace na etatach:

```
SELECT etat, SUM(placa_pod)FROM pracownicy
GROUP BY etat
HAVING MAX(placa_pod) > 1000
ORDER BY etat;
```

Pozostałe operacje (2)

- Przykłady (cd):
  - Problem 2. Wyświetl nazwy etatów wraz z liczbą osób zatrudnionych na danym etacie, policz tylko te osoby, które pobierają płacę dodatkową, pomiń etaty, dla których wyliczona liczba osób jest mniejsza niż dwie. Wynik posortuj malejąco wg liczby osób na etacie i rosnąco wg nazwy etatu.

```
SELECT etat, COUNT(*) FROM pracownicy
WHERE placa_dod is not null
GROUP BY etat
HAVING COUNT(*) >= 2
ORDER BY COUNT(*) DESC, etat;
```

Specyficzne konstrukcje

- Funkcja agregująca jako parametr innej funkcji agregującej

```
SELECT MAX(SUM(placa_pod))
FROM pracownicy
GROUP BY etat;
```

- Zapytanie z jedną grupą i klauzulą HAVING

```
SELECT MAX(placa_pod)
FROM pracownicy WHERE id_zesp in (10,20)
HAVING COUNT(*) > 12;
```

Najczęściej popełniane błędy (1)

- W klauzuli SELECT zapytania z jedną grupą wyrażenie nie będącego funkcją grupową.

```
SELECT etat, SUM(placa_pod)
FROM pracownicy
WHERE etat = 'PROFESOR';
```

- W klauzuli SELECT zapytania z wieloma grupami wyrażenie nie będącego wyrażeniem grupującym lub funkcją grupową.

```
SELECT id_zesp, nazwisko, SUM(placa_pod)
FROM pracownicy
GROUP BY id_zesp;
```

Najczęściej popełniane błędy (2)

- Funkcja grupowa w warunku w klauzuli WHERE.

```
SELECT id_zesp FROM pracownicy
WHERE COUNT(*) > 3 GROUP BY id_zesp;
```

- Wyrażenie nie będące wyrażeniem grupującym lub funkcją grupową w klauzuli HAVING.

```
SELECT id_zesp, COUNT(distinct placa_dod)
FROM pracownicy
GROUP BY id_zesp
HAVING etat = 'PROFESOR';
```

Najczęściej popełniane błędy (3)

- Porządkowanie wyników zapytania z wieloma grupami według wartości wyrażenia nie będącego wyrażeniem grupującym lub funkcją grupową.

```
SELECT id_zesp, COUNT(distinct placa_dod)
FROM pracownicy
GROUP BY id_zesp
ORDER BY nazwisko;
```

Przydatne konstrukcje

Zastępowanie wartości pustych (1)

- Funkcja **NVL(wyrażenie_1,wyrażenie_2)**:
  - jeśli **wyrażenie_1** jest puste, zwraca wartość **wyrażenia_2**,
  - jeśli **wyrażenie_1** nie jest puste, zwraca wartość **wyrażenia_1**.

```
SELECT nazwisko, placa_dod
FROM pracownicy ORDER BY nazwisko;
```

NAZWISKO	PLACA_DOD
BIALY	170.6
BLAZEWICZ	210
BRZEZINSKI	
HAPKE	90
JEZIERSKI	80.5
KONOPKA	
…	

```
SELECT nazwisko,
       NVL(TO_CHAR(placa_dod), 'brak płacy')
       AS placa_dod
FROM pracownicy ORDER BY nazwisko;
```

NAZWISKO	PLACA_DOD
BIALY	170.6
BLAZEWICZ	210
BRZEZINSKI	brak płacy
HAPKE	90
JEZIERSKI	80.5
KONOPKA	brak płacy
…	

Zastępowanie wartości pustych (2)

- Funkcja **NVL2(wyrażenie_1,wyrażenie_2,wyrażenie_3)**:
  - jeśli **wyrażenie_1** jest puste, zwraca wartość **wyrażenia_3**,
  - jeśli **wyrażenie_1** nie jest puste, zwraca wartość **wyrażenia_2**.

```
SELECT nazwisko,
       NVL2(placa_dod, 'tak', 'nie') AS czy_dodatek
FROM pracownicy
ORDER BY nazwisko;
```

NAZWISKO	CZY_DODATEK
BIALY	tak
BLAZEWICZ	tak
BRZEZINSKI	nie
HAPKE	tak
JEZIERSKI	tak
KONOPKA	nie
KOSZLAJDA	nie
…	

Wyrażenie CASE (1)

- Symuluje instrukcję warunkową "co-jeśli" w SQL.
- Wersja prosta:

```
CASE wyrażenie WHEN wyrażenie_1 THEN wyrażenie_2
               WHEN wyrażenie_2 THEN wyrażenie_3
               … [ELSE wyrażenie_n] END
```

- Wersja z warunkami logicznymi:

```
CASE WHEN warunek_1 THEN wyrażenie_1
     WHEN warunek_2 THEN wyrażenie_2
     … [ELSE wyrażenie_n] END
```

- Wyrażenie zwraca wartość po znalezieniu pierwszego dopasowania.

Wyrażenie CASE (2)

- Przykład.

```
SELECT nazwisko, etat,
       CASE etat WHEN 'DYREKTOR' THEN '---'
                 WHEN 'PROFESOR' THEN '---'
                 ELSE TO_CHAR(placa_pod) END AS placa_pod
FROM pracownicy ORDER BY nazwisko;
```

```
SELECT nazwisko, etat,
       CASE WHEN etat in ('DYREKTOR','PROFESOR') THEN '---'
            ELSE TO_CHAR(placa_pod) END AS placa_pod
FROM pracownicy ORDER BY nazwisko;
```

NAZWISKO	ETAT	PLACA_POD
BIALY	STAZYSTA	250
BLAZEWICZ	PROFESOR	---
BRZEZINSKI	PROFESOR	---
HAPKE	ASYSTENT	480
JEZIERSKI	ASYSTENT	439.7
KONOPKA	ASYSTENT	480
KOSZLAJDA	ADIUNKT	590

…

Zaawansowane grupowania

Operacja ROLLUP (1)

- Rozszerza funkcjonalność klauzuli **GROUP BY** wyliczając **półkostkę danych**.
- Tworzy grupy na podstawie wartości pierwszych n, n-1, n-2, …, 0 wyrażeń grupujących wymienionych w klauzuli **GROUP BY ROLLUP**, dla każdej z grup zwraca jeden rekord podsumowania.

```
SELECT <lista atrybutów> FROM <lista relacji>
...
GROUP BY ROLLUP(<lista wyrażeń grupujących>)
...;
```

Operacja ROLLUP (2)

```
SELECT etat, id_zesp,
       SUM(placa_pod)
FROM pracownicy
GROUP BY ROLLUP (etat, id_zesp);
```

=

```
SELECT etat, id_zesp,
       SUM(placa_pod)
FROM pracownicy
GROUP BY etat, id_zesp;
```

+

```
SELECT etat, SUM(placa_pod)
FROM pracownicy
GROUP BY etat;
```

+

```
SELECT SUM(placa_pod)
FROM pracownicy;
```

ETAT	ID_ZESP	SUM(PLACA_POD)
ADIUNKT	20	1235,5
ADIUNKT		1235,5
ASYSTENT	20	1290,7
ASYSTENT	30	480
ASYSTENT		1770,7
DYREKTOR	10	1730
DYREKTOR		1730
PROFESOR	20	1790
PROFESOR	30	1070
PROFESOR	40	1350
PROFESOR		4210
STAZYSTA	30	458
STAZYSTA		458
SEKRETARKA	10	410,2
SEKRETARKA		410,2
		9814,4

Częściowa operacja ROLLUP (1)

- Wylicza półkostkę danych tylko na **podzbiorze** wyrażeń grupujących klauzuli **`GROUP BY ROLLUP`**.

```
SELECT <lista atrybutów>
FROM <lista relacji>
...
GROUP BY <lista wyrażeń grupujących 1>,
         ROLLUP(<lista wyrażeń grupujących 2>)
...;
```

Częściowa operacja ROLLUP (2)

```
SELECT etat, id_zesp,
       SUM(placa_pod)
FROM pracownicy
GROUP BY etat, ROLLUP (id_zesp);
```

=

```
SELECT etat, id_zesp,
       SUM(placa_pod)
FROM pracownicy
GROUP BY etat, id_zesp;
```

+

```
SELECT etat, SUM(placa_pod)
FROM pracownicy
GROUP BY etat;
```

ETAT	ID_ZESP	SUM(PLACA_POD)
ADIUNKT	20	1235,5
ADIUNKT		1235,5
ASYSTENT	20	1290,7
ASYSTENT	30	480
ASYSTENT		1770,7
DYREKTOR	10	1730
DYREKTOR		1730
PROFESOR	20	1790
PROFESOR	30	1070
PROFESOR	40	1350
PROFESOR		4210
STAZYSTA	30	458
STAZYSTA		458
SEKRETARKA	10	410,2
SEKRETARKA		410,2

Operacja CUBE (1)

- Rozszerza funkcjonalność klauzuli **GROUP BY** wyliczając **kostkę danych**.

❑ Tworzy grupy na podstawie **wszystkich** kombinacji wartości wyrażeń wymienionych w klauzuli **GROUP BY CUBE** (2^n kombinacji dla n wyrażeń), dla każdej z grup zwraca jeden rekord podsumowania.

```
SELECT <lista atrybutów> FROM <lista relacji>
...
GROUP BY CUBE(<lista wyrażeń grupujących>)
...;
```

Operacja CUBE (2)

```
SELECT etat, id_zesp, SUM(placa_pod)
FROM pracownicy
GROUP BY CUBE (etat, id_zesp);
```

=

```
SELECT etat, id_zesp, SUM(placa_pod)
FROM pracownicy
GROUP BY etat, id_zesp;
```

+

```
SELECT etat, SUM(placa_pod)
FROM pracownicy GROUP BY etat;
```

+

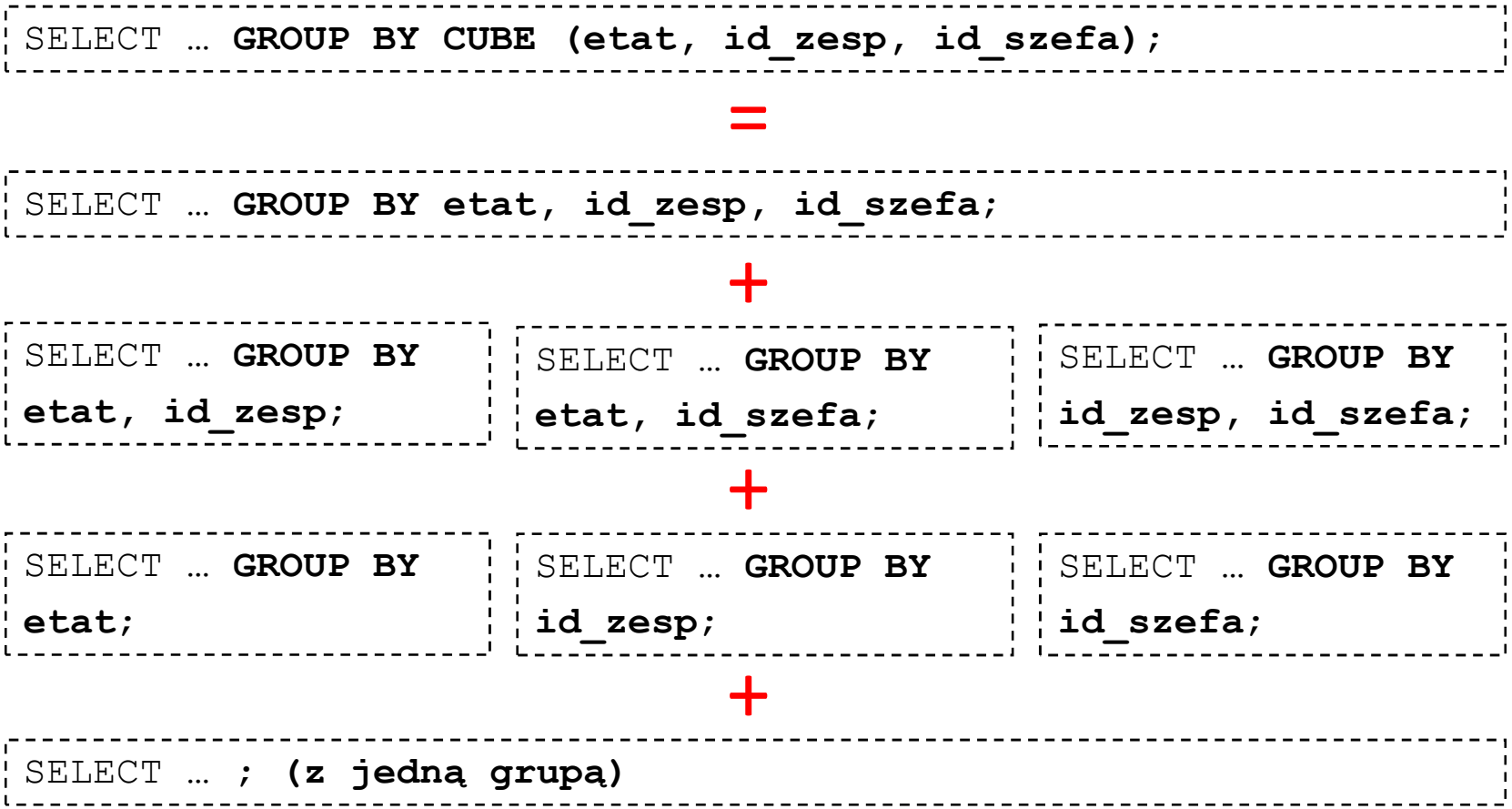
```
SELECT id_zesp, SUM(placa_pod)
FROM pracownicy GROUP BY id_zesp;
```

+

```
SELECT SUM(placa_pod) FROM pracownicy;
```

ETAT	ID_ZESP	SUM(PLACA_POD)
		9814,4
	10	2140,2
	20	4316,2
	30	2008
	40	1350
ADIUNKT		1235,5
ADIUNKT	20	1235,5
ASYSTENT		1770,7
ASYSTENT	20	1290,7
ASYSTENT	30	480
DYREKTOR		1730
DYREKTOR	10	1730
PROFESOR		4210
PROFESOR	20	1790
PROFESOR	30	1070
PROFESOR	40	1350
STAZYSTA		458
STAZYSTA	30	458
SEKRETARKA		410,2
SEKRETARKA	10	410,2

Operacja CUBE (3)



Częściowa operacja CUBE (1)

- Wylicza kostkę danych tylko na **podzbiorze** wyrażeń grupujących klauzuli **`GROUP BY CUBE`**.

```
SELECT <lista atrybutów>
FROM  <lista relacji>
WHERE <warunki filtrujące>
GROUP BY <lista wyrażeń grupujących 1>,
         CUBE(<lista wyrażeń grupujących 2>);
```

Częściowa operacja CUBE (2)

```
SELECT etat,id_zesp,id_szefa,SUM(placa_pod)
FROM pracownicy WHERE etat in (…)
GROUP BY etat, CUBE (id_zesp, id_szefa);
```

=

```
SELECT etat, id_zesp, id_szefa, SUM …
GROUP BY etat, id_zesp, id_szefa;
```

+

```
SELECT etat, id_zesp, SUM(placa_pod)…
GROUP BY etat, id_zesp;
```

+

```
SELECT etat, id_szefa, SUM(placa_pod) …
GROUP BY etat, id_szefa;
```

+

```
SELECT etat, SUM(placa_pod)
FROM pracownicy
GROUP BY etat;
```

ETAT	ID_ZESP	ID_SZEFA	SUM
ASYSTENT			1770,7
ASYSTENT		110	480
ASYSTENT		120	480
ASYSTENT		130	439,7
ASYSTENT		140	371
ASYSTENT	20		1290,7
ASYSTENT	20	110	480
ASYSTENT	20	130	439,7
ASYSTENT	20	140	371
ASYSTENT	30		480
ASYSTENT	30	120	480
PROFESOR			4210
PROFESOR		100	3380
PROFESOR		130	830
PROFESOR	20		1790
PROFESOR	20	100	960
PROFESOR	20	130	830
PROFESOR	30		1070
PROFESOR	30	100	1070
PROFESOR	40		1350
PROFESOR	40	100	1350

Kolumny złożone

- Kolumny traktowane **razem** jako jednostka grupowania podczas wyliczania kostki lub półkostki.
- Przykład:

```
GROUP BY ROLLUP(a, b, c, d)
```

=

```
GROUP BY a, b, c, d
GROUP BY a, b, c
GROUP BY a, b
GROUP BY a
GROUP BY ()
```

```
GROUP BY ROLLUP(a, (b, c), d)
```

=

```
GROUP BY a, b, c, d
GROUP BY a, b, c
GROUP BY a
GROUP BY ()
```

Interpretacja wartości pustych (1)

- Jak w zbiorze rekordów wyniku operacji **ROLLUP** i **CUBE** odróżnić wartości puste (NULL), istniejące w analizowanych danych (np. id_szefa = NULL dla pracowników bez szefa) od wartości pustych utworzonych przez operacje do prezentacji podsumowań?

podsumowanie dla grupy etat = ASYSTENT i id_szefa = NULL czy podsumowanie dla grupy etat = ASYSTENT (niezależnie od szefa)?

ETAT	ID_ZESP	ID_SZEFA	SUM
ASYSTENT			1770,7
ASYSTENT		110	480
ASYSTENT		120	480
ASYSTENT		130	439,7
ASYSTENT	20		1290,7
ASYSTENT	20	110	480
ASYSTENT	20	130	439,7
PROFESOR			4210
…			

Interpretacja wartości pustych (2)

- Funkcja **GROUPING**, zwraca wartość:
  - 1 – jeśli pusta wartość wyrażenia grupującego (NULL), które jest jej parametrem, powstała w wyniku działania operacji **ROLLUP** lub **CUBE** (rekord jest podsumowaniem),
  - 0 – w przeciwnym wypadku.

```
SELECT <lista atrybutów>, GROUPING(wyrażenie grupujące)
FROM <lista relacji>
...
GROUP BY ROLLUP | CUBE(<lista wyrażeń grupujących>)
...;
```

Interpretacja wartości pustych (3)

```
SELECT etat, id_zesp, id_szefa,
       SUM(placa_pod)
FROM pracownicy
WHERE etat in (…)
GROUP BY CUBE (etat, id_zesp, id_szefa);
```

?

ETAT	ID_ZESP	ID_SZEFA	SUM
			1730
			6350,2
		100	3790,2
		130	830
	10		1730
	10		2140,2
	10	100	410,2
	20		1790
	20	100	960
	20	130	830
	30		1070
	30	100	1070
	40		1350
	40	100	1350
DYREKTOR			1730
DYREKTOR			1730
DYREKTOR	10		1730
DYREKTOR	10		1730
PROFESOR			4210
PROFESOR		130	830
PROFESOR	20		1790
…			

Interpretacja wartości pustych (4)

```
SELECT etat, id_zesp, id_szefa,
       SUM(placa_pod), GROUPING(id_szefa)
FROM pracownicy
WHERE etat in (…)
GROUP BY CUBE (etat, id_zesp, id_szefa);
```

ETAT	ID_ZESP	ID_SZEFA	SUM	GR
			1730	0
			6350,2	1
		100	3790,2	0
		130	830	0
	10		1730	0
	10		2140,2	1
	10	100	410,2	0
	20		1790	1
	20	100	960	0
	20	130	830	0
	30		1070	1
	30	100	1070	0
	40		1350	1
	40	100	1350	0
DYREKTOR			1730	0
DYREKTOR			1730	1
DYREKTOR	10		1730	0
DYREKTOR	10		1730	1
PROFESOR			4210	1
PROFESOR		130	830	0
PROFESOR	20		1790	1
…				

podsumowanie dla grupy etat = DYREKTOR i id_szefa = NULL

podsumowanie dla grupy etat = DYREKTOR

Interpretacja wartości pustych (5)

```
SELECT case when GROUPING(etat)=1 then 'wszystkie etaty' else etat end as etat,
       case when GROUPING(id_zesp)=1 then 'wszystkie zespoły'
         else to_char(id_zesp) end as id_zesp,
       SUM(placa_pod)
FROM pracownicy
GROUP BY CUBE (etat, id_zesp);
```

ETAT	ID_ZESP	SUM(PLACA_POD)
wszystkie etaty	wszystkie zespoły	9814,4
wszystkie etaty	10	2140,2
wszystkie etaty	20	4316,2
wszystkie etaty	30	2008
wszystkie etaty	40	1350
ADIUNKT	wszystkie zespoły	1235,5
ADIUNKT	20	1235,5
ASYSTENT	wszystkie zespoły	1770,7
ASYSTENT	20	1290,7
ASYSTENT	30	480
DYREKTOR	wszystkie zespoły	1730
DYREKTOR	10	1730
PROFESOR	wszystkie zespoły	4210

…

Funkcja GROUPING_ID (1)

- Wylicza poziom grupowania – liczbę odpowiadającą bitowemu wektorowi wyników funkcji **`GROUPING`**, skojarzonemu z bieżącym wierszem wyniku.
- Stosowana w celu uniknięcia wielokrotnego wywołania funkcji **`GROUPING`** np. przy filtrowaniu rekordów.
- Dla **`CUBE(x, y)`**:

Poziom grupowania	Wektor bitowy	`GROUPING_ID(x,y)`
`x y`	0 0	0
`x`	0 1	1
`y`	1 0	2
całkowite	1 1	3

Funkcja GROUPING_ID (2)

```
SELECT etat, id_zesp,
       SUM(placa_pod) AS suma,
       GROUPING_ID(etat) AS g1,
       GROUPING_ID(id_zesp) AS g2,
       GROUPING_ID(etat, id_zesp) AS g3,
       GROUPING_ID(id_zesp, etat) AS g4
FROM pracownicy
GROUP BY CUBE(etat, id_zesp)
ORDER BY etat, id_zesp;
```

```
G1=0: GROUPING(etat) = 0
G2=1: GROUPING(id_zesp) = 1
G3=1: GROUPING(etat)|GROUPING(id_zesp) = 0|1 = 1
G4=2: GROUPING(id_zesp)|GROUPING(etat) = 1|0 = 2
```

```
G1=1: GROUPING(etat) = 1
G2=1: GROUPING(id_zesp) = 1
G3=3: GROUPING(etat)|GROUPING(id_zesp) = 1|1 = 3
G4=3: GROUPING(id_zesp)|GROUPING(etat) = 1|1 = 3
```

ETAT	ID_ZESP	SUMA	G1	G2	G3	G4
ADIUNKT	20	1235.5	0	0	0	0
ADIUNKT		1235.5	0	1	1	2
ASYSTENT	20	1290.7	0	0	0	0
ASYSTENT	30	480	0	0	0	0
ASYSTENT		1770.7	0	1	1	2
DYREKTOR	10	1730	0	0	0	0
DYREKTOR		1730	0	1	1	2
PROFESOR	20	1790	0	0	0	0
PROFESOR	30	1070	0	0	0	0
PROFESOR	40	1350	0	0	0	0
PROFESOR		4210	0	1	1	2
SEKRETARKA	10	410.2	0	0	0	0
SEKRETARKA		410.2	0	1	1	2
STAZYSTA	30	458	0	0	0	0
STAZYSTA		458	0	1	1	2
	10	2140.2	1	0	2	1
	20	4316.2	1	0	2	1
	30	2008	1	0	2	1
	40	1350	1	0	2	1
		9814.4	1	1	3	3

Funkcja GROUPING_ID (3)

OPIS	SUMA
Etat: ADIUNKT Wszystkie zespoły	1235.5
Etat: ADIUNKT Zespół: 20	1235.5
Etat: ASYSTENT Wszystkie zespoły	1770.7
Etat: ASYSTENT Zespół: 20	1290.7
Etat: ASYSTENT Zespół: 30	480
Etat: DYREKTOR Wszystkie zespoły	1730
Etat: DYREKTOR Zespół: 10	1730
Etat: PROFESOR Wszystkie zespoły	4210
Etat: PROFESOR Zespół: 20	1790
Etat: PROFESOR Zespół: 30	1070
Etat: PROFESOR Zespół: 40	1350
Etat: SEKRETARKA Wszystkie zespoły	410.2
Etat: SEKRETARKA Zespół: 10	410.2
Etat: STAZYSTA Wszystkie zespoły	458
Etat: STAZYSTA Zespół: 30	458
Wszystkie etaty Wszystkie zespoły	9814.4
Wszystkie etaty Zespół: 10	2140.2
Wszystkie etaty Zespół: 20	4316.2
Wszystkie etaty Zespół: 30	2008
Wszystkie etaty Zespół: 40	1350

Funkcja GROUPING_ID (4)

```
SELECT CASE WHEN GROUPING(etat) = 1 THEN 'Wszystkie etaty'
            ELSE'Etat: '||etat
       END ||
       CASE WHEN GROUPING(id_zesp) = 1 THEN' Wszystkie zespoły'
            ELSE ' Zespół: '||to_char(id_zesp)
       END AS opis,
       SUM(placa_pod) AS suma
FROM pracownicy
GROUP BY cube (etat, id_zesp)
ORDER BY opis;
```

```
SELECT CASE GROUPING_ID(etat,id_zesp)
          WHEN 3 THEN 'Wszystkie etaty Wszystkie zespoły'
          WHEN 2 THEN 'Wszystkie etaty Zespół: '||to_char(id_zesp)
          WHEN 1 THEN 'Etat: '||etat||' Wszystkie zespoły'
          ELSE 'Etat: '|| etat || ' Zespół: '||to_char(id_zesp) END AS opis,
       SUM(placa_pod) AS suma
FROM pracownicy
GROUP BY CUBE (etat, id_zesp)
ORDER BY opis;
```

Operacja GROUPING SETS (1)

- Rozszerza funkcjonalność klauzuli **GROUP BY** o możliwość **wyboru zbiorów grup**, które chcemy uzyskać w wyniku zapytania.

```
SELECT <lista atrybutów>
FROM <lista relacji>
...
GROUP BY GROUPING SETS((<lista wyrażeń grupujących 1>),
                       (<lista wyrażeń grupujących 2>),
                       (<lista wyrażeń grupujących 3>)
                        …)
...;
```

Operacja GROUPING SETS (2)

```
SELECT etat, id_zesp, id_szefa,
SUM(placa_pod) AS suma
FROM pracownicy WHERE …
GROUP BY GROUPING SETS ((etat, id_zesp),
(etat, id_szefa),(etat),());
```

=

```
SELECT etat, id_zesp, SUM(placa_pod)
… GROUP BY etat, id_zesp;
```

+

```
SELECT etat, id_szefa, SUM(placa_pod) …
GROUP BY etat, id_szefa;
```

+

```
SELECT etat, SUM(placa_pod)…
GROUP BY etat;
```

+

```
SELECT SUM(placa_pod)… FROM pracownicy;
```

ETAT	ID_ZESP	ID_SZEFA	SUMA
PROFESOR	30		1070
ASYSTENT	30		480
DYREKTOR	10		1730
ADIUNKT	20		1235,5
PROFESOR	20		1790
ASYSTENT	20		1290,7
PROFESOR	40		1350
STAZYSTA	30		458
ADIUNKT		130	1235,5
ADIUNKT			1235,5
ASYSTENT		110	480
ASYSTENT		120	480
ASYSTENT		130	439,7
ASYSTENT		140	371
ASYSTENT			1770,7
DYREKTOR			1730
DYREKTOR			1730
PROFESOR		100	3380
PROFESOR		130	830
PROFESOR			4210
STAZYSTA		130	250
STAZYSTA		140	208
STAZYSTA			458
			9404,2

Funkcja GROUP_ID

- Umożliwia identyfikację grup-duplikatów powstałych w wyniku działania operacji **GROUP BY**.
- Jeśli przy grupowaniu powstało n duplikatów, funkcja zwraca wartości z przedziału <0, n-1>.

```
SELECT etat, id_zesp,
       SUM(placa_pod) AS suma,
       GROUP_ID() AS gr
FROM pracownicy
GROUP BY etat, ROLLUP(etat, id_zesp)
ORDER BY etat, id_zesp, GROUP_ID();
```

ETAT	ID_ZESP	SUMA	GR
ADIUNKT	20	1235.5	0
ADIUNKT		1235.5	0
ADIUNKT		1235.5	1
ASYSTENT	20	1290.7	0
ASYSTENT	30	480	0
ASYSTENT		1770.7	0
ASYSTENT		1770.7	1
DYREKTOR	10	1730	0
DYREKTOR		1730	0
…			

Klauzula PIVOT (1)

- Realizuje agregację danych wierszowych i konwersję do danych w układzie kolumnowym.
- Składnia:

```
SELECT ... FROM ...
PIVOT (funkcja_agr(<wyrażenie) [AS alias]
       [,funkcja_agr(<wyrażenie>)[AS alias]…]
       FOR (<lista kolumn przestawnych>) IN (
              lista_wartości_1 [AS alias],
              lista_wartości_2 [AS alias], …))
```

  - Lista kolumn w klauzuli FOR jest statyczna (brak możliwości definicji przez zapytanie).

Klauzula PIVOT (2)

```
SELECT id_zesp, placa_pod FROM pracownicy
ORDER BY id_zesp;
```

ID_ZESP	PLACA_POD
10	1730
10	410.2
20	590
20	480
20	371
20	439.7
20	645.5
20	960
20	830
30	480
30	250
30	208
30	1070
40	1350

```
SELECT * FROM
 (SELECT id_zesp, placa_pod FROM pracownicy)
 PIVOT(SUM(placa_pod) FOR id_zesp IN
  (10 AS Zesp_10,
   20 AS Zesp_20,
   30 AS Zesp_30,
   40 AS Zesp_40));
```

ZESP_10	ZESP_20	ZESP_30	ZESP_40
2140.2	4316.2	2008	1350

Klauzula PIVOT (3)

```
SELECT id_zesp, etat, placa_pod
FROM pracownicy
ORDER BY id_zesp, etat;
```

ID_ZESP	ETAT	PLACA_POD
10	DYREKTOR	1730
10	SEKRETARKA	410.2
20	ADIUNKT	590
20	ADIUNKT	645.5
20	ASYSTENT	371
20	ASYSTENT	439.7
20	ASYSTENT	480
20	PROFESOR	960
20	PROFESOR	830
30	ASYSTENT	480
30	PROFESOR	1070
30	STAZYSTA	208
30	STAZYSTA	250
40	PROFESOR	1350

```
SELECT * FROM
  (SELECT etat, id_zesp, placa_pod
   FROM pracownicy)
 PIVOT(SUM(placa_pod) FOR id_zesp IN
  (10 AS Zesp_10,
   20 AS Zesp_20,
   30 AS Zesp_30,
   40 AS Zesp_40))
ORDER BY etat;
```

ETAT	ZESP_10	ZESP_20	ZESP_30	ZESP_40
ADIUNKT		1235.5		
ASYSTENT		1290.7	480	
DYREKTOR	1730			
PROFESOR		1790	1070	1350
SEKRETARKA	410.2			
STAZYSTA			458	

Klauzula PIVOT (4)

```
SELECT * FROM
 (SELECT etat, id_zesp, placa_pod
  FROM pracownicy)
 PIVOT(AVG(placa_pod) FOR id_zesp
  (10 AS Zesp_10, 20 AS Zesp_20,
   30 AS Zesp_30, 40 AS Zesp_40))
ORDER BY etat;
```

ETAT	ZESP_10	ZESP_20	ZESP_30	ZESP_40
ADIUNKT		617.75		
ASYSTENT		430.2333333	480	
DYREKTOR	1730			
PROFESOR		895	1070	1350
SEKRETARKA	410.2			
STAZYSTA			229	

```
SELECT etat,
 round(Zesp_10, 2) AS Zesp_10,
 round(Zesp_20, 2) AS Zesp_20,
 round(Zesp_30, 2) AS Zesp_30,
 round(Zesp_40, 2) AS Zesp_40 FROM
 (SELECT etat, id_zesp, placa_pod FROM pracownicy)
 PIVOT(AVG(placa_pod) FOR id_zesp IN
  (10 AS Zesp_10, 20 AS Zesp_20, 30 AS Zesp_30, 40 AS Zesp_40))
ORDER BY etat;
```

ETAT	ZESP_10	ZESP_20	ZESP_30	ZESP_40
ADIUNKT		617.75		
ASYSTENT		430.23	480	
DYREKTOR	1730			
PROFESOR		895	1070	1350
SEKRETARKA	410.2			
STAZYSTA			229	

Klauzula PIVOT XML (1)

- Działa identycznie jak PIVOT, ale:
  - w wyniku daje kolumnę typu XMLType, zawierającą ciąg XML z danymi,
  - lista kolumn w klauzuli FOR może być zdefiniowana podzapytaniem.

```
SELECT ... FROM ...
PIVOT XML (funkcja_agr(<wyrażenie) [AS alias]
          [,funkcja_agr(<wyrażenie>)[AS alias]…]
       FOR (<lista kolumn przestawnych>) IN (
              lista_wartości_1 [AS alias],
              lista_wartości_2 [AS alias], …) |
              (podzapytanie))
```

Klauzula PIVOT XML (2)

```
SELECT * FROM
  (SELECT id_zesp, placa_pod FROM pracownicy)
  PIVOT XML (SUM(placa_pod) AS suma FOR id_zesp IN
    (SELECT id_zesp FROM zespoly));
```

```
ID_ZESP_XML
--------------------------------------------------------------------------------
<PivotSet>
    <item><column name="ID_ZESP">10</column><column name="SUMA">2140,2</column></item>
    <item><column name="ID_ZESP">20</column><column name="SUMA">4316,2</column></item>
    <item><column name="ID_ZESP">30</column><column name="SUMA">2008</column></item>
    <item><column name="ID_ZESP">40</column><column name="SUMA">1350</column></item>
    <item><column name="ID_ZESP">50</column><column name="SUMA"></column></item>
</PivotSet>
```

Klauzula UNPIVOT (1)

- Realizuje konwersję danych z układu kolumnowego do układu wierszowego.
- Składnia:

```
SELECT ... FROM ...
UNPIVOT [INCLUDE NULLS | EXCLUDE NULLS](
          (lista kolumn)
          FOR (<lista kolumn przestawnych>) IN (
                 lista_kolumn_1 [AS alias],
                 lista_kolumn_2 [AS alias], …))
```

- **INCLUDE NULLS** – uwzględnia w zbiorze wynikowym puste wiersze,
- **EXCLUDE_NULLS** – odrzuca puste wiersze ze zbioru wynikowego (domyślnie).

Klauzula UNPIVOT (2)

```
CREATE VIEW uklad_kolumnowy AS SELECT * FROM
 (SELECT id_zesp, placa_pod FROM pracownicy)
 PIVOT(SUM(placa_pod) FOR id_zesp IN
  (10 AS zesp_10, 20 AS zesp_20,
   30 AS zesp_30, 40 AS zesp_40,
   50 AS zesp_50));
```

```
ZESP_10 ZESP_20 ZESP_30 ZESP_40 ZESP_50
------- ------- ------- ------- -------
 2140.2  4316.2    2008    1350
```

```
SELECT * FROM (
 (SELECT zesp_10, zesp_20, zesp_30, zesp_40, zesp_50
  FROM uklad_kolumnowy)
 UNPIVOT((suma_plac_w_zespole) FOR zespol IN
  (zesp_10 AS 'Zesp_10',
   zesp_20 AS 'Zesp_20',
   zesp_30 AS 'Zesp_30',
   zesp_40 AS 'Zesp_40',
   zesp_50 AS 'Zesp_50')));
```

```
ZESPOL  SUMA_PLAC_W_ZESPOLE
------- -------------------
Zesp_10                2140.2
Zesp_20                4316.2
Zesp_30                  2008
Zesp_40                  1350
```

Klauzula UNPIVOT (3)

```
SELECT * FROM (
 (SELECT zesp_10, zesp_20, zesp_30, zesp_40, zesp_50
  FROM uklad_kolumnowy)
 UNPIVOT INCLUDE NULLS((suma_plac_w_zespole) FOR zespol IN
  (zesp_10 AS 'Zesp_10',
   zesp_20 AS 'Zesp_20',
   zesp_30 AS 'Zesp_30',
   zesp_40 AS 'Zesp_40',
   zesp_50 AS 'Zesp_50'));
```

ZESPOL	SUMA_PLAC_W_ZESPOLE
Zesp_10	2140.2
Zesp_20	4316.2
Zesp_30	2008
Zesp_40	1350
Zesp_50	

Bibliografia

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