Bazy danych

6. Podzapytania i grupowanie

P. F. Góra

http://th-www.if.uj.edu.pl/zfs/gora/

2011

Podzapytania

- Podzapytania pozwalają na tworzenie *strukturalnych* podzapytań, co umożliwia izolowanie poszczególnych części instrukcji. Istnieniu podzapytań SQL zawdzięcza słowo “strukturalny” w swojej nazwie.

- Podzapytania zapewniają alternatywny sposób wykonywania zadań, które w inny sposób można realizować tylko poprzez skomplikowane złączenia. Niektórych zadań *nie da* się wykonać bez podzapytań.

- Podzapytania zwiększają czytelność kodu.

Copyright © 2009-11 P. F. Góra

Przykład — typowy problem

```
mysql> SELECT * FROM ludzie;
+----+----------+
| Nr | Nazwisko |
+----+----------+
|  1 | Nowak    |
|  2 | Kowalski |
|  3 | Kowalska |
+----+----------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM zrobione;
+----+-----------+
| Nr | NrZadania |
+----+-----------+
|  1 |         1 |
|  1 |         2 |
|  1 |         3 |
|  1 |         4 |
|  2 |         2 |
|  2 |         3 |
|  2 |         4 |
|  2 |         5 |
|  2 |         6 |
|  3 |         1 |
|  3 |         3 |
|  3 |         6 |
|  3 |         7 |
+----+-----------+
13 rows in set (0.03 sec)
```

Copyright © 2009-11 P. F. Góra

Tabele te mają wspólną kolumnę `Nr` (powstały w wyniku normalizacji). Pytanie: co zrobił użytkownik Nowak? Chcemy się tego dowiedzieć *bez* wypisywania zawartości tabeli `ludzie`, zapamiętywania numeru i wpisywania odpowiedniego zapytania ręcznie.

Po pierwsze, możemy to zrobić przy użyciu zmiennej tymczasowej:

```
mysql> SET @N = (SELECT Nr FROM ludzie WHERE Nazwisko LIKE "Now%");
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM ludzie NATURAL JOIN zrobione
    -> WHERE Nr=@N;
+----+----------+-----------+
| Nr | Nazwisko | NrZadania |
+----+----------+-----------+
|  1 | Nowak    |         1 |
|  1 | Nowak    |         2 |
|  1 | Nowak    |         3 |
|  1 | Nowak    |         4 |
+----+----------+-----------+
4 rows in set (0.64 sec)
```

Copyright © 2009-11 P. F. Góra

Widać, że użycie zmiennej tymczasowej można wyeliminować i napisać wprost, używając podzapytania:

```
mysql> SELECT * FROM ludzie NATURAL JOIN zrobione
    -> WHERE Nr = (SELECT Nr FROM ludzie WHERE Nazwisko LIKE "Now%");
+----+----------+-----------+
| Nr | Nazwisko | NrZadania |
+----+----------+-----------+
|  1 | Nowak    |         1 |
|  1 | Nowak    |         2 |
|  1 | Nowak    |         3 |
|  1 | Nowak    |         4 |
+----+----------+-----------+
4 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Spróbujmy zrobić to samo z użytkownikiem o nazwisku Kowalski:

```
mysql> SELECT * FROM ludzie NATURAL JOIN zrobione
    -> WHERE Nr = (SELECT Nr FROM ludzie WHERE Nazwisko LIKE "Kowal%");
ERROR 1242 (21000): Subquery returns more than 1 row
```

Otrzymaliśmy błąd, gdyż podzapytanie zwraca więcej niż jeden wiersz i operator porównania (=) nie może zadziałać.

Copyright © 2009-11 P. F. Góra

Podzapytania — operatory `ANY`, `IN`, `ALL`

Zacznijmy od założenia dwu tabel

```
mysql> CREATE TABLE Alfa (Litera CHAR(1) NOT NULL PRIMARY KEY,
    -> Liczba SMALLINT);
Query OK, 0 rows affected (0.14 sec)

mysql> CREATE TABLE Beta (Liczba SMALLINT NOT NULL,
    -> InnaLiczba SMALLINT);
Query OK, 0 rows affected (0.75 sec)
```

Copyright © 2009-11 P. F. Góra

Niech ich zawartością będzie:

```
mysql> SELECT * FROM Alfa;
+--------+--------+
| Litera | Liczba |
+--------+--------+
| A      |     10 |
| B      |     20 |
| C      |     25 |
+--------+--------+
3 rows in set (0.01 sec)
```

```
mysql> SELECT * FROM Beta;
+--------+------------+
| Liczba | InnaLiczba |
+--------+------------+
|     20 |         30 |
|     20 |         18 |
|     25 |         30 |
|     27 |         38 |
|     20 |         -5 |
|     10 |       NULL |
+--------+------------+
6 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

coś operator_porównania `ANY` *(podzapytanie)*

oznacza, iż “coś” musi spełniać odpowiednią relację z *jakimiś* wynikami podzapytania

```
mysql> SELECT * FROM Alfa WHERE
    -> Liczba > ANY
    -> (SELECT Liczba FROM Beta);
+--------+--------+
| Litera | Liczba |
+--------+--------+
| B      |     20 |
| C      |     25 |
+--------+--------+
2 rows in set (0.10 sec)
```

```
mysql> SELECT Litera FROM Alfa WHERE
    -> Liczba <= ANY
    -> (SELECT InnaLiczba/3 FROM Beta);
+--------+
| Litera |
+--------+
| A      |
+--------+
1 row in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Słowo kluczowe `ALL` oznacza, że warunek musi być spełniony dla wszystkich wierszy zwracanych przez podzapytanie.

```
mysql> SELECT * FROM Alfa
    -> WHERE Liczba > ALL
    -> (SELECT InnaLiczba FROM Beta);
Empty set (0.09 sec)
```

```
mysql> SELECT * FROM Alfa
    -> WHERE Liczba > ALL
    -> (SELECT 0.5*InnaLiczba FROM Beta);
+--------+--------+
| Litera | Liczba |
+--------+--------+
| B      |     20 |
| C      |     25 |
+--------+--------+
2 rows in set (0.01 sec)
```

Copyright © 2009-11 P. F. Góra

Słowo kluczowe `IN` jest równoważne z warunkiem `= ANY`.

```
mysql> SELECT * FROM Alfa
    -> WHERE Liczba IN
    -> (SELECT InnaLiczba/3 FROM Beta);
+--------+--------+
| Litera | Liczba |
+--------+--------+
| A      |     10 |
+--------+--------+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM Alfa
    -> WHERE Liczba = ANY
    -> (SELECT InnaLiczba/3 FROM Beta);
+--------+--------+
| Litera | Liczba |
+--------+--------+
| A      |     10 |
+--------+--------+
1 row in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Przykład z początku wykładu możemy wobec tego zrealizować następująco:

```
mysql> SELECT * FROM ludzie NATURAL JOIN zrobione
    -> WHERE Nr IN (SELECT Nr FROM ludzie WHERE Nazwisko LIKE "Kowal%");
+----+----------+-----------+
| Nr | Nazwisko | NrZadania |
+----+----------+-----------+
|  2 | Kowalski |         2 |
|  2 | Kowalski |         3 |
|  2 | Kowalski |         4 |
|  2 | Kowalski |         5 |
|  2 | Kowalski |         6 |
|  3 | Kowalska |         1 |
|  3 | Kowalska |         3 |
|  3 | Kowalska |         6 |
|  3 | Kowalska |         7 |
+----+----------+-----------+
9 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Podzapytania — operator `EXISTS`

`... EXISTS (`*podzapytanie*`) ...`

warunek jest prawdziwy jeśli podzapytanie zwraca niepusty zbiór wierszy

```
mysql> SELECT * FROM Alfa
    -> WHERE EXISTS
    -> (SELECT * FROM Beta WHERE
    -> InnaLiczba > 100);
Empty set (0.00 sec)
```

```
mysql> SELECT * FROM Alfa
    -> WHERE EXISTS
    -> (SELECT * FROM Beta WHERE
    -> InnaLiczba < 50);
+--------+--------+
| Litera | Liczba |
+--------+--------+
| A      |     10 |
| B      |     20 |
| C      |     25 |
+--------+--------+
3 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Podzapytania skorelowane

```
mysql> SELECT * FROM Alfa
    -> WHERE EXISTS
    -> (SELECT * FROM Beta WHERE InnaLiczba/3=Alfa.Liczba);
+--------+--------+
| Litera | Liczba |
+--------+--------+
| A      |     10 |
+--------+--------+
1 row in set (0.01 sec)
```

Występujące tutaj podzapytanie zawiera odwołanie do kolumny występującej w zapytaniu zewnętrznym. Podzapytania tego typu nazywamy *skorelowanymi*.

W powyższym przykładzie podzapytanie wykonywane jest dla każdego wiersza tabeli `Alfa`, a więc przy ustalonej wartości wielkości `Alfa.Liczba`. Dla podanych danych, warunek `EXISTS` jest prawdziwy tylko w jednym przypadku.

Copyright © 2009-11 P. F. Góra

Przykład

Zadanie: Wypiszmy te i tylko te wiersze z tabeli `Beta`, w których wartość atrybutu `InnaLiczba` występuje dwukrotnie.

```
mysql> SELECT * FROM Beta AS B1 WHERE
    -> (SELECT COUNT(*) FROM Beta AS B2
    ->   WHERE B2.InnaLiczba=B1.InnaLiczba) = 2;
+--------+------------+
| Liczba | InnaLiczba |
+--------+------------+
|     20 |         30 |
|     25 |         30 |
+--------+------------+
2 rows in set (0.00 sec)
```

Bez podzapytania tego zadania nie dałoby się wykonać — nie da się go zastąpić złączeniem.

Copyright © 2009-11 P. F. Góra

Grupowanie

Grupowanie polega na

- Utworzeniu partycji tabeli, to jest rozbiciu tabeli na *rozłączne* części, których suma mnogościowa równa się całej tabeli; kryterium przynależności do danej części określone jest przez jeden lub więcej atrybutów (lub wielkość obliczaną z atrybutów).
- *Zazwyczaj* odpowiednie zapytanie `SELECT` zwraca pewne wartości zagregowane, charakteryzujące całą grupę, nie zaś pojedyncze krotki.

Copyright © 2009-11 P. F. Góra

Najważniejsze funkcje agregujące

`COUNT(·)`	zlicza wiersze
`COUNT(DISTINCT ·)`	zlicza *różne* wystąpienia w wierszach
`SUM(·)`	podaje sumę wartości liczbowych
`AVG(·)`	podaje średnią arytmetyczną z wartości liczbowych
`STD(·), STDDEV(·)`	podaje odchylenie standardowe wartości liczbowych
`VARIANCE(·)`	podaje wariancję wartości liczbowych
`MIN(·), MAX(·)`	podaje najmniejszą i największą wartość

Copyright © 2009-11 P. F. Góra

```
mysql> SELECT * FROM Beta;
+--------+------------+
| Liczba | InnaLiczba |
+--------+------------+
|     20 |         30 |
|     20 |         18 |
|     25 |         30 |
|     27 |         38 |
|     20 |         -5 |
|     10 |       NULL |
+--------+------------+
6 rows in set (0.00 sec)

mysql> SELECT COUNT(*), COUNT(InnaLiczba), COUNT(DISTINCT InnaLiczba) FROM Beta;
+----------+-------------------+----------------------------+
| COUNT(*) | COUNT(InnaLiczba) | COUNT(DISTINCT InnaLiczba) |
+----------+-------------------+----------------------------+
|        6 |                 5 |                          4 |
+----------+-------------------+----------------------------+
1 row in set (0.00 sec)
```

Proszę zwrócić uwagę jak obsługiwane są wartości `NULL`.

Copyright © 2009-11 P. F. Góra

Typowy przykład grupowania

Utwórzmy tabelę i załadujmy do niej dane zapisane uprzednio w pliku:

```
mysql> CREATE TABLE Sklepy
    -> (Nr SMALLINT UNSIGNED NOT NULL,
    -> Data DATETIME NOT NULL,
    -> Utarg FLOAT,
    -> PRIMARY KEY(Nr,Data));
Query OK, 0 rows affected (0.84 sec)

mysql> LOAD DATA LOCAL INFILE 'sklepy.txt'
    -> INTO TABLE Sklepy
    -> FIELDS TERMINATED BY '\t';
```

Kwalifikator `LOCAL` mówi, że plik jest zlokalizowany na komputerze-kliencie.

Copyright © 2009-11 P. F. Góra

Początkowe wiersze tabeli zawierają

```
mysql> SELECT * FROM Sklepy ORDER BY Data LIMIT 8;
+----+---------------------+---------+
| Nr | Data                | Utarg   |
+----+---------------------+---------+
|  6 | 2010-09-01 00:00:00 | 1446.55 |
|  4 | 2010-09-01 00:00:00 | 6711.11 |
| 10 | 2010-09-02 00:00:00 | 2407.83 |
|  3 | 2010-09-02 00:00:00 | 3592.91 |
| 10 | 2010-09-03 00:00:00 | 9715.82 |
|  4 | 2010-09-03 00:00:00 |  836.37 |
|  7 | 2010-09-04 00:00:00 | 8560.58 |
|  8 | 2010-09-05 00:00:00 | 9112.47 |
+----+---------------------+---------+
8 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Ile napłynęło utargów ze sklepu Nr 2?

```
mysql> SELECT COUNT(*) FROM Sklepy WHERE Nr=2;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
1 row in set (0.00 sec)
```

A ile ze sklepu Nr 8?

```
mysql> SELECT COUNT(*) FROM Sklepy WHERE Nr=8;
+----------+
| COUNT(*) |
+----------+
|       15 |
+----------+
1 row in set (0.00 sec)
```

Moglibyśmy tak robić dla każdego sklepu, ale to byłoby nudne ☹, przede wszystkim zaś nie wiemy *ile jest sklepów*. A chcielibyśmy to mieć dla wszystkich. . .

Copyright © 2009-11 P. F. Góra

```
mysql> SELECT Nr, COUNT(*) FROM Sklepy
    -> GROUP BY Nr;
+----+----------+
| Nr | COUNT(*) |
+----+----------+
|  1 |       19 |
|  2 |        9 |
|  3 |       18 |
|  4 |       14 |
|  5 |       12 |
|  6 |       12 |
|  7 |       12 |
|  8 |       15 |
|  9 |        6 |
| 10 |        9 |
+----+----------+
10 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Jakie *łączne* utargi zebrano z poszczególnych sklepów w poszczególnych miesiącach?

```
mysql> SELECT Nr, MONTH(Data), COUNT(*), SUM(Utarg) FROM Sklepy
    -> GROUP BY Nr, MONTH(Data);
+----+-------------+----------+------------------+
| Nr | MONTH(Data) | COUNT(*) | SUM(Utarg)       |
+----+-------------+----------+------------------+
|  1 |           9 |        5 | 26535.8900146484 |
|  1 |          10 |        8 | 36799.8699951172 |
|  1 |          11 |        6 | 37257.2102050781 |
|  2 |           9 |        4 |  18497.449798584 |
|  2 |          10 |        2 | 6521.89007568359 |
|  2 |          11 |        3 | 14917.0703353882 |
.....................................................
| 10 |           9 |        4 | 19898.5502929688 |
| 10 |          10 |        3 | 17272.6705932617 |
| 10 |          11 |        2 | 15431.4204101563 |
+----+-------------+----------+------------------+
30 rows in set (0.03 sec)
```

Copyright © 2009-11 P. F. Góra

Przykład

W naszym początkowym przykładzie moglibyśmy policzyć ile kto zrobił zadań:

```
mysql> SELECT Nazwisko, COUNT(NrZadania) AS IleZrobione
    -> FROM ludzie NATURAL JOIN zrobione
    -> GROUP BY Nr;
+----------+-------------+
| Nazwisko | IleZrobione |
+----------+-------------+
| Nowak    |           4 |
| Kowalski |           5 |
| Kowalska |           4 |
+----------+-------------+
3 rows in set (0.09 sec)
```

Copyright © 2009-11 P. F. Góra

Możemy też sprawdzić ile osób zrobiło poszczególne zadania:

```
mysql> SELECT NrZadania, COUNT(Nr) AS IleOsob
    -> FROM ludzie NATURAL JOIN zrobione
    -> GROUP BY NrZadania;
+-----------+---------+
| NrZadania | IleOsob |
+-----------+---------+
|         1 |       2 |
|         2 |       2 |
|         3 |       3 |
|         4 |       2 |
|         5 |       1 |
|         6 |       2 |
|         7 |       1 |
+-----------+---------+
7 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Kto zrobił zadanie o najwyższym numerze?

```
mysql> SELECT Nazwisko, NrZadania
    -> FROM Ludzie NATURAL JOIN zrobione
    -> WHERE NrZadania IN (SELECT MAX(NrZadania) FROM zrobione);
+----------+-----------+
| Nazwisko | NrZadania |
+----------+-----------+
| Kowalska |         7 |
+----------+-----------+
1 row in set (0.00 sec)
```

Mamy tu funkcję agregującą `MAX( )` oraz podzapytanie. Użyliśmy predykatu `IN`, gdyż zadanie o najwyższym numerze mogła zrobić więcej niż jedna osoba.

Copyright © 2009-11 P. F. Góra

Klauzula `HAVING` — filtrowanie po grupowaniu

Zastosowanie klauzuli `HAVING` zwraca tylko wartości dla grup spełniających podane kryterium.

```
mysql> SELECT Nr, MONTH(Data), COUNT(*), SUM(Utarg) As Total FROM Sklepy
    -> GROUP BY Nr, MONTH(Data)
    -> HAVING Total > 30000;
+----+-------------+----------+------------------+
| Nr | MONTH(Data) | COUNT(*) | Total            |
+----+-------------+----------+------------------+
|  1 |          10 |        8 | 36799.8699951172 |
|  1 |          11 |        6 | 37257.2102050781 |
|  3 |          10 |        8 | 33674.0201416016 |
|  5 |          11 |        7 | 36028.9307861328 |
+----+-------------+----------+------------------+
4 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

HAVING i podzapytania

Wypisz tylko te sklepy, z których spłynęło *więcej niż średnia* liczba utargów:

```
mysql> SELECT NR, COUNT(*) AS IleUtargow FROM Sklepy GROUP BY Nr
    -> HAVING IleUtargow >
    -> (SELECT AVG(u) FROM
    -> (SELECT COUNT(*) AS u FROM Sklepy GROUP BY Nr) AS TuMusiBycAlias);
+----+------------+
| NR | IleUtargow |
+----+------------+
|  1 |         19 |
|  3 |         18 |
|  4 |         14 |
|  8 |         15 |
+----+------------+
4 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Klauzula `HAVING` przydaje się takżed w wypadku aliasów, gdyż alias jest nie-rozpoznawany przez klauzulę `WHERE`, ale jest rozpoznawany przez klauzulę `HAVING`. Formalnie rzecz biorąc klauzula `HAVING` działa po grupowaniu, tyle że w poniższym przykładzie istnieje tylko jedna grupa, obejmująca wszystkie krotki.

```
mysql> SELECT Utarg AS Uuu FROM Sklepy WHERE Uuu>9500;
ERROR 1054 (42S22): Unknown column 'Uuu' in 'where clause'

mysql> SELECT UTARG AS Uuu FROM Sklepy HAVING Uuu>9500;
+---------+
| Uuu     |
+---------+
| 9827.57 |
| 9539.41 |
| 9813.53 |
| 9841.34 |
| 9715.82 |
| 9732.15 |
+---------+
6 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Średnia ruchoma

W tabeli mamy zgromadzone dane numeryczne, indeksowane “czasem” (szereg czasowy) — na przykład kolejne kursy akcji, kolejne odczyty temperatury, dane o sprzedaży z kolejnych miesięcy itp. Należy obliczyć średnią ruchomą

$$y_i = \frac{1}{p+1} \sum_{j=i-p/2}^{i+p/2} x_j$$

Obliczenie tego po stronie aplikacji jest proste. *Jak to zrobić w SQL?*

Copyright © 2009-11 P. F. Góra

```
mysql> CREATE TABLE W
    -> (Nr SMALLINT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
    -> Wartosc FLOAT NOT NULL);
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO W (Wartosc) VALUES
    -> (1),(5),(2.2),(0.8),(4),(3.1),(3.9),(2.7),(3.1),(0.9);
Query OK, 10 rows affected (0.07 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

Copyright © 2009-11 P. F. Góra

```
mysql> SELECT * FROM W;
+----+---------+
| Nr | Wartosc |
+----+---------+
| 1  | 1       |
| 2  | 5       |
| 3  | 2.2     |
| 4  | 0.8     |
| 5  | 4       |
| 6  | 3.1     |
| 7  | 3.9     |
| 8  | 2.7     |
| 9  | 3.1     |
| 10 | 0.9     |
+----+---------+
10 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra

Problem rozwiążemy za pomocą samozłączenia (self–join).

```
mysql> SET @p = 3;
mysql> SELECT x.wartosc, AVG(y.wartosc) AS Srednia
    -> FROM W AS x, W AS y
    -> WHERE y.Nr >= x.Nr - @p/2 AND y.Nr <= x.Nr + @p/2
    -> GROUP BY x.Nr;
```

Dwukrotnie odwolujemy się do tej samej tabeli, a więc musi ona występować pod różnymi aliasami. Wystąpienie `x` służy do identyfikowania wierszy, więc występuje także w klauzuli `GROUP BY`. Z odpowiednich wystąpień `y` obliczana jest średnia.

Copyright © 2009-11 P. F. Góra

```
+---------+-----------------+
| wartosc | Srednia         |
+---------+-----------------+
| 1       | 3               |
| 5       | 2.7333333492279 |
| 2.2     | 2.6666666865349 |
| 0.8     | 2.3333333532015 |
| 4       | 2.6333333055178 |
| 3.1     | 3.6666666666667 |
| 3.9     | 3.2333333492279 |
| 2.7     | 3.2333333492279 |
| 3.1     | 2.2333333094915 |
| 0.9     | 1.9999999403954 |
+---------+-----------------+
10 rows in set (0.94 sec)
```

Skrajne wartości są źle obsługiwane.

Copyright © 2009-11 P. F. Góra

```
mysql> SET @p = 3;
mysql> SET @gorny = (SELECT MAX(Nr) FROM W) + 1;
mysql> SET @dolny = (SELECT MIN(Nr) FROM W) - 1;
mysql> SELECT x.wartosc, AVG(y.wartosc) AS Srednia
    -> FROM W as x, W as y
    -> WHERE y.Nr >= x.Nr - @p/2 AND y.Nr <= x.Nr + @p/2
    ->       AND x.Nr - @p/2 >= @dolny AND x.Nr + @p/2 <= @gorny
    -> GROUP BY x.Nr;
+---------+-----------------+
| wartosc | Srednia         |
+---------+-----------------+
| 5       | 2.7333333492279 |
| 2.2     | 2.6666666865349 |
| 0.8     | 2.3333333532015 |
| 4       | 2.6333333055178 |
| 3.1     | 3.6666666666667 |
| 3.9     | 3.2333333492279 |
| 2.7     | 3.2333333492279 |
| 3.1     | 2.2333333094915 |
+---------+-----------------+
8 rows in set (0.01 sec)
```

Copyright © 2009-11 P. F. Góra

Suma narastająca

```
mysql> SELECT x.Nr, x.wartosc, FORMAT(SUM(y.wartosc),1) AS RunningTotal
    -> FROM W AS x, W AS y
    -> WHERE y.Nr <= x.Nr
    -> GROUP BY x.Nr;
+----+---------+--------------+
| Nr | wartosc | RunningTotal |
+----+---------+--------------+
| 1  | 1       | 1.0          |
| 2  | 5       | 6.0          |
| 3  | 2.2     | 8.2          |
| 4  | 0.8     | 9.0          |
| 5  | 4       | 13.0         |
| 6  | 3.1     | 16.1         |
| 7  | 3.9     | 20.0         |
| 8  | 2.7     | 22.7         |
| 9  | 3.1     | 25.8         |
| 10 | 0.9     | 26.7         |
+----+---------+--------------+
10 rows in set (0.00 sec)
```

Copyright © 2009-11 P. F. Góra