Wykład 2: Agregacja i nulle

Agregacja

Multizbiory

Dlaczego multizbiory?

- Łatwiej przetwarzać zapytania: suma to konkatenacja, nie trzeba usuwać duplikatów; przy rzutowaniu wypisujemy po kolei bez sprawdzania, czy coś już było, czy nie.
- Jak policzyć średnią zarobków? Nie można wziąć zbioru wszystkich zarobków i zastosować operatora AVG; trzeba uwzględnić krotności!

Semantyka podstawowych operacji:

- suma i rzutowanie: jak wyżej;
- różnica wiadomo: różnica liczby wystąpień;
- przecięcie: minimum;
- selekcja: po kolei przeglądamy krotki i wypisujemy pasujące;
- produkt: podwójna pętla.

Multizbiory: pułapki

Wiele praw dotyczących zbiorów nie przenosi się na multizbiory:

$R \cup R = R$

$(R \cap S) - T = R \cap (S - T)$

$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$

$(R \cup S) - T = (R - T) \cup (S - T)$

```
SELECT * FROM R WHERE (C OR D)
   =
(SELECT * FROM R WHERE C)
  UNION ALL
(SELECT * FROM R WHERE D)
```

Agregacja: operatory

- Usuwanie duplikatów:

```
SELECT DISTINCT * FROM R;
```

- Grupowanie i agregacja:

```
SELECT city, SUM(sold) sum
FROM sales
GROUP BY city;
```

month	city	sold
1	Rome	200
2	Paris	500
1	London	100
1	Paris	300
2	Rome	300
2	London	400
3	Rome	400

month	city	sold
1	Paris	300
2	Paris	500
1	Rome	200
2	Rome	300
3	Rome	400
1	London	100
2	London	400

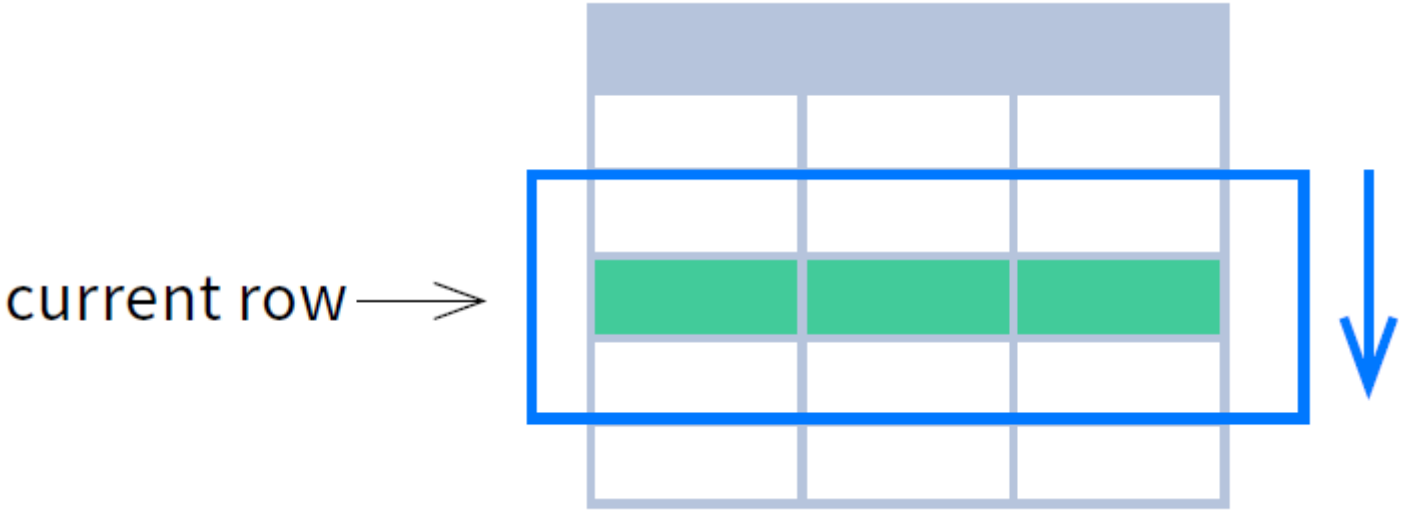
city	sum
Paris	800
Rome	900
London	500

Inne funkcje agregujące: `MIN`, `MAX`, `COUNT`, `AVG`.

Ilustracja: https://learnsql.com

Funkcje okienkowe

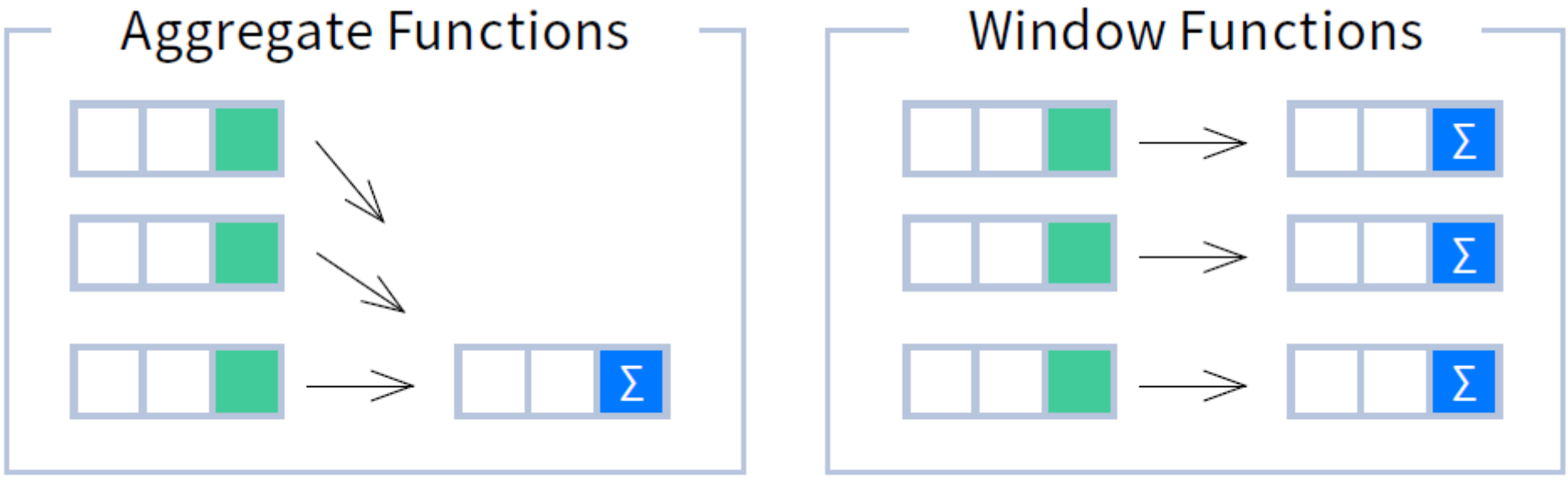
Funkcje okienkowe wyliczają rezultat na podstawie okienka (zbioru wierszy), które jest wyznaczane przez przesuwającą się ramkę dookoła bieżącego wiersza.



Źródło: https://learnsql.com/blog/sql-window-functions-cheat-sheet/

Funkcje okienkowe a agregacja

Inaczej niż zwykła agregacja, funkcje okienkowe nie kolapsują wierszy.



Źródło: https://learnsql.com/blog/sql-window-functions-cheat-sheet/

Składnia

```
SELECT city, month,
  sum(sold) OVER (
    PARTITION BY city
    ORDER BY month
    RANGE BETWEEN UNBOUNDED PRECEDING
      AND CURRENT ROW
  ) sum
FROM sales;
```

```
SELECT <column_1>, <column_2>, ...
  <window_function> OVER (
    PARTITION BY <...>
    ORDER BY <...>
    <window_frame>

  ) <column_alias>
FROM <table_name>;
```

`PARTITION BY`, `ORDER BY`, `<window_frame>` są opcjonalne.

Źródło: https://learnsql.com/blog/sql-window-functions-cheat-sheet/

PARTITION BY

`PARTITION BY` dzieli wiersze na **partycje**, do których oddzielnie stosuje się funkcję okienkową.

```
SELECT city, month, sum(sold) OVER (PARTITION BY city) sum FROM sales;
```

month	city	sold
1	Rome	200
2	Paris	500
1	London	100
1	Paris	300
2	Rome	300
2	London	400
3	Rome	400

PARTITION BY city

month	city	sold	sum
1	Paris	300	800
2	Paris	500	800
1	Rome	200	900
2	Rome	300	900
3	Rome	400	900
1	London	100	500
2	London	400	500

Domyślnie: Jeśli nie ma `PARTITION BY`, to cały zbiór krotek jest jedną partycją.

Źródło: https://learnsql.com/blog/sql-window-functions-cheat-sheet/

ORDER BY

`ORDER BY` specyfikuje porządek na wierszach w obrębie partycji.

```
SELECT city, month, sum(sold) OVER (PARTITION BY city ORDER BY month) sum FROM sales;
```

sold	city	month
200	Rome	1
500	Paris	2
100	London	1
300	Paris	1
300	Rome	2
400	London	2
400	Rome	3

PARTITION BY city ORDER BY month

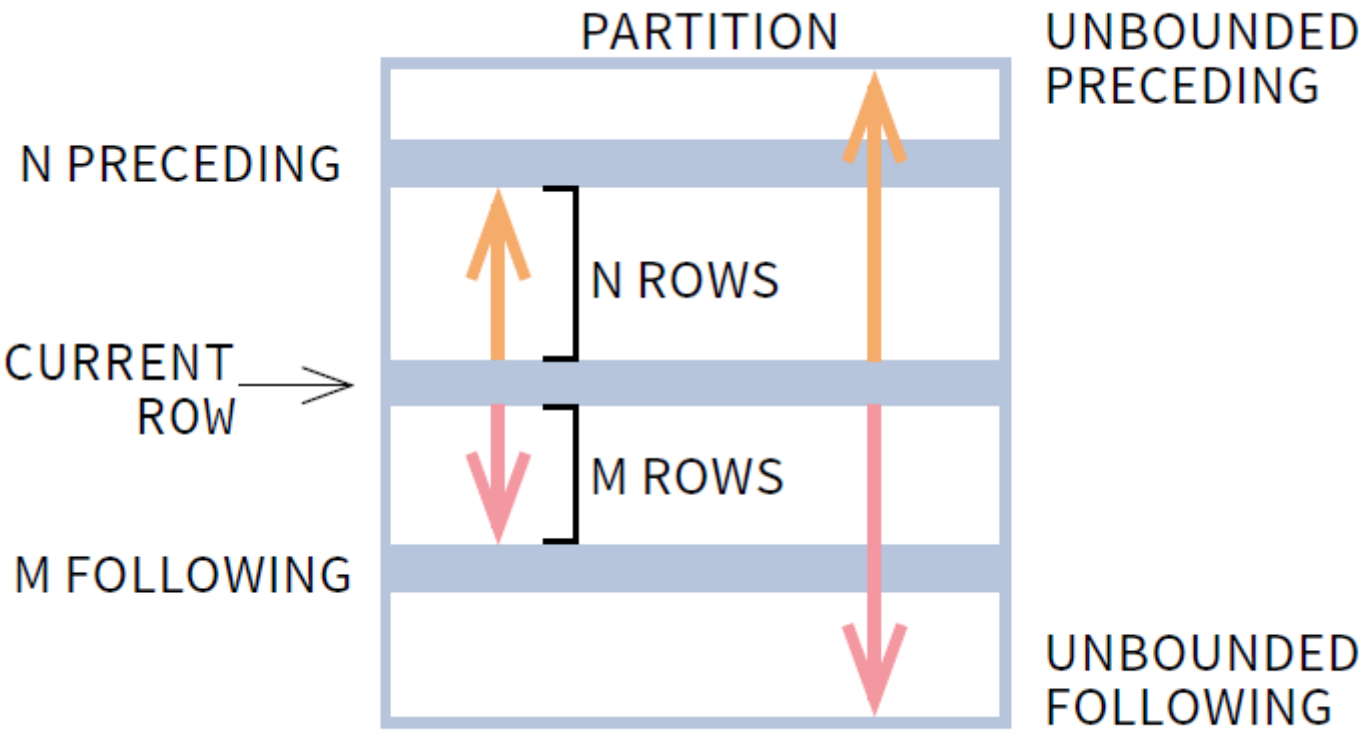
sold	city	month
300	Paris	1
500	Paris	2
200	Rome	1
300	Rome	2
400	Rome	3
100	London	1
400	London	2

Domyślnie: Jeśli nie ma `ORDER BY`, to porządek w obrębie partycji jest dowolny.

Źródło: https://learnsql.com/blog/sql-window-functions-cheat-sheet/

Ramka okienka `<window_frame>`

Ramka okienka to zbiór wierszy danej partycji używany do ewaluacji funkcji okienkowej, określony względem bieżącego wiersza. (W każdej partycji jest wyliczany niezależnie.)



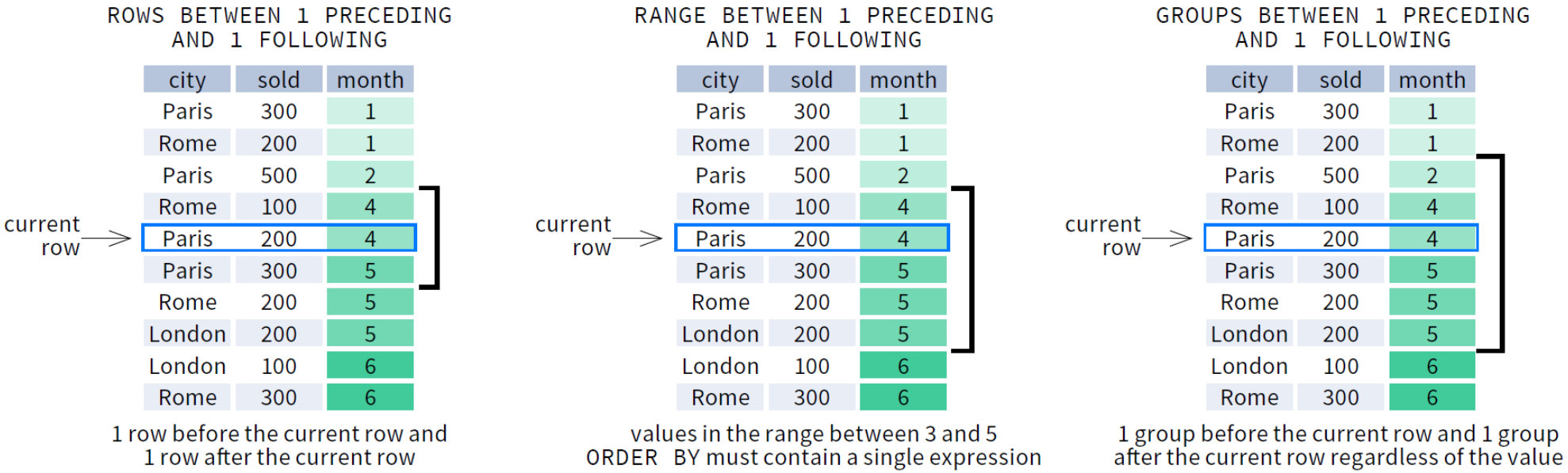
Źródło: https://learnsql.com/blog/sql-window-functions-cheat-sheet/

Ramka okienka `<window_frame>` (2)

```
ROWS | RANGE | GROUPS BETWEEN lower_bound AND upper_bound
```

(`GROUPS` tylko PostgreSQL), gdzie `lower_bound` i `upper_bound` można wybrać spośród

```
UNBOUNDED PRECEDING | n PRECEDING | CURRENT ROW | n FOLLOWING | UNBOUNDED FOLLOWING
```



Domyślnie: `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW` z `ORDER BY`;

`ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING` bez `ORDER BY`.

Źródło: https://learnsql.com/blog/sql-window-functions-cheat-sheet/

Logiczna kolejność operacji w SQLu

1. `FROM`, `JOIN`
2. `WHERE`
3. `GROUP BY`
4. funkcje agregujące
5. `HAVING`
6. funkcje okienkowe
7. `SELECT`
8. `DISTINCT`
9. `UNION`/`INTERSECT`/`EXCEPT`
10. `ORDER BY`
11. `OFFSET`
12. `LIMIT`/`FETCH`/`TOP`

Funkcji okienkowych można używać w klauzulach `SELECT` i `ORDER BY`. Nie można ich używać w klauzulach `FROM`, `WHERE`, `GROUP BY`, ani `HAVING`.

Źródło: https://learnsql.com/blog/sql-window-functions-cheat-sheet/

Dostępne funkcje okienkowe

- Funkcje rankingowe:
  - `row_number()` - unikatowy numer dla każdego wiersza partycji, różne numery przy remisach;
  - `rank()` - ranking w obrębie partycji, przy remisach ten sam, z lukami;
  - `dense_rank()` - ranking w obrębie partycji, przy remisach ten sam, bez luk.
- Funkcje dystrybucji (rozkładu):
  - `percent_rank()` - (rank-1) / (liczba wierszy - 1),
  - `cume_dist()` - dystrybuanta empiryczna, tj. liczba wierszy nie większych / łączna liczba wierszy.
- Funkcje analityczne:
  - `lead(expr,offset,default)` - wartość dla wiersza *offset* za bieżącym;
  - `lag(expr,offset,default)` - wartość dla wiersza *offset* przed bieżącym;
  - `ntile(n)` - podziel wiersze partycji na *n* równych grup, przypisz każdemu wierszowi numer grupy;
  - `first_value(expr)` - wartość `expr` dla pierwszego wiersza w ramce;
  - `last_value(expr)` - wartość `expr` dla ostatniego wiersza w ramce;
  - `nth_value(expr, n)` - wartość `expr` dla `n`-tego wiersza w ramce.
- Funkcje agregujące:
  - `avg(expr)` - średnia wartość dla wierszy w ramce
  - `count(expr)` - liczba wartości dla wierszy w ramce
  - `max(expr)` - maksymalna wartość dla wierszy w ramce
  - `min(expr)`- minimalna wartość dla wierszy w ramce
  - `sum(expr)` - suma wartości dla wierszy w ramce

Źródło: https://learnsql.com/blog/sql-window-functions-cheat-sheet/

Nulle

Co reprezentuje null?

1. Nieznana/brakująca wartość
    - np. brakująca data urodzenia
2. Nieistniejąca wartość
    - np. mąż dla kobiety niezamężnej
3. Ukryta wartość
    - np. zastrzeżony numer telefonu
    - Podobnie do 1, ale można żądać, żeby była podobna do 2:
        - nie chcemy, żeby ktoś poznał nasz numer telefonu,
        - ale również aby coś wnioskował z tego, że posiadamy numer telefonu albo że go nie posiadamy.

Kiedy SQL produkuje nulle?

- `AVG(A)` na pustym zbiorze krotek.
- Tak samo `MIN`, `MAX`, `SUM` (czemu nie 0?).
- Tylko `COUNT` daje 0.
- Produkt zewnętrzny
  - `SELECT * FROM R, S WHERE R.A = S.A;`

    niesparowane wiersze się gubią,
  - `SELECT * FROM R FULL JOIN S ON R.A = S.A;`

    parujemy z wierszem nulli (po lewej lub prawej)
  - `SELECT * FROM R LEFT JOIN S ON R.A = S.A;`

    tylko lewe, a prawe się gubią

Reguły w standardzie SQL

1. Każda operacja arytmetyczna na nullu daje null.
2. Każde porównanie na nullu daje UNKNOWN (wartość logiczna ½),
   - koniunkcja = min,
   - alternatywa = max,
   - negacja = 1 - x,

   bierze się krotki o wartości TRUE (wartość 1).

```
SELECT * FROM Movie WHERE len < 120;
```

```
SELECT * FROM Movie WHERE len >= 120;
```

```
SELECT * FROM Movie WHERE len < 120 OR len >= 120;
```

Dziwności

- `SUM(A)` sumuje tylko nie-nulle; **niezgodność z (1)**

  `SELECT SUM(A) + SUM(B) FROM T ≠ SELECT SUM(A + B) FROM T`

- `COUNT(A)` zlicza tylko nie-nulle, count(*) zlicza wszystkie krotki; **niezgodność wewnętrzna:** co jak jest tylko jedna kolumna?

- `SELECT a, AVG(b) FROM R GROUP BY a;`

  daje jeden wiersz z nullem w kolumnie a; **niezgodność z (2)**, bo przyjmujemy, że nulle są sobie równe.

- w `FULL JOIN` wiersze z nullami nie spełniają warunku `ON`; **niezgodność z (2)**.

Można powiedzieć, że to kwestia wyboru i że tak jest wygodniej, ale taka semantyka jest bardziej skomplikowana, trudniej się jej nauczyć.

Paradoksy

- `int x;` jeśli `x` jest null, to `0*x` też jest null. A nie zero, mimo że dowolna wartość pomnożona przez 0 daje 0. Bez sensu?
- `SELECT * FROM Movie WHERE len < 120 OR len >= 120;` nulle się nie wybiorą, bo dają unknown.

  (Bez sensu? Ale jak pytamy o wzrost męża, a dana kobieta jest niezamężna, to pewnie lepiej, żeby jej nie wybierało…)
- `SELECT R.a FROM R WHERE R.a NOT IN (SELECT a FROM S);`

  dla `R = {1,2,3,4}`, `S = {1}`, daje `{2,3,4}`;

  ale gdy dodamy null do S, `S = {1, null}`, to dostajemy `{}`.

  Jeden element wyrzuca trzy elementy? Ludzie nie myślą w logice 3-wartościowej…

Można powiedzieć, że taka jest semantyka i że to nie jest paradoks. Ale na pewno jest to dysonans poznawczy. Zachowanie sprzeczne z intuicją zwiększa liczbę błędów.

Semantyka pewnych odpowiedzi (*)

Prawdą jest to, co zachodzi po każdym ustawieniu nulli na wartości:

- zwracamy te krotki, które są zwrócone wg. zwykłej semantyki *dla każdego* podstawienia nulli wartościami.

Np. zapytanie

```
SELECT * FROM Movie WHERE len >= 120 OR len < 120;
```

zwraca wszystko.

Semantyka pewnych odpowiedzi: problemy (*)

Paradoksy dalej sa:

```
SELECT R.a FROM R WHERE R.a NOT IN (SELECT a FROM S)
```

daje takie odpowiedzi jak wcześniej.

Dla $R = \{1, 2, 3, 4\}$, $S = \{1, null\}$ zapytanie

```
SELECT count(R.a) FROM R
WHERE R.a NOT IN (SELECT a FROM S)
```

daje albo 3 albo 2, więc nie wiadomo co zwrócić; chyba null :-)

Ta semantyka pasuje tylko do pierwszej roli nulla: nieznana wartość.

Ale jak sobie radzić z wartościami nieistniejącymi? (Może oddzielne tabele? Albo wprowadzić specjalnego nulla “atrybut nie istnieje”?)

Semantyka pewnych odpowiedzi: złożoność (*)

- Trudno obliczyć odpowiedź. Wygląda na to, że trzeba rozważać dowolne podstawienia wartości pod nulle - wykładniczo dużo…
- Problem jest NP-trudny. Gdyby istniał algorytm rozwiązujący go w czasie wielomianowym, to byłyby też algorytmy wielomianowe dla wielu słynnych problemów, dla których nie znamy takich algorytmów: np. problem komiwojażera, problem plecakowy, problem SAT.
- Da się łatwo obliczyć wynik dla zapytań pozytywnych, czyli SPCU:
  - każdego nulla zastępujemy specjalną wartością, inną niż wszystkie inne, odróżnialną od normalnych stałych;
  - obliczamy zapytanie traktując te nowe wartości jak zwykłe stałe;
  - wyrzucamy z odpowiedzi wszystkie krotki zawierające te specjalne wartości.