Funkcje agregujące i GROUP BY

> Agregacja zwija wiele wierszy w jeden, a `GROUP BY` decyduje, według jakiego klucza to zwijanie ma przebiegać — ta strona pokazuje obie rzeczy razem, bo funkcja agregująca bez `GROUP BY` liczy jedną wartość dla całej tabeli, a z `GROUP BY` — jedną wartość dla każdej grupy osobno.

◫◫ Skróty użyte na tej stronie (2)

Skrót	Co znaczy
ISO	International Organization for Standardization: organizacja, która wydaje normy techniczne używane na całym świecie
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

◫◫ TYLKO ODCZYT

Zliczenie wszystkich wierszy: COUNT(*)

```
SELECT COUNT(*) AS liczba_zamowien FROM zamowienia;
```

`COUNT(*)` liczy wiersze, niezależnie od tego, co mają w kolumnach — nawet jeśli wszystkie kolumny w danym wierszu są `NULL`em, wiersz i tak się liczy.

Przykład:

```
 liczba_zamowien
-----------------
               8
```

```
(1 row)
```

TYLKO ODCZYT

Zliczenie wypełnionych wartości: COUNT(kolumna)

```
SELECT COUNT(klient_id) AS zamowien_z_klientem FROM zamowienia;
```

`COUNT` z nazwą kolumny liczy tylko wiersze, w których ta kolumna nie jest `NULL`em. Strona NULL: trzecia wartość logiczna i pułapki, które z niej wynikają opisuje to zachowanie wraz z tabelami prawdy, które je tłumaczą.

Przykład:

```
 zamowien_z_klientem
----------------------
                    7
(1 row)
```

TYLKO ODCZYT

Suma: SUM

```
SELECT SUM(wartosc) AS suma_zamowien FROM zamowienia WHERE status = 'zrealizowane';
```

`SUM` dodaje wartości z kolumny liczbowej po wszystkich wierszach, które przeszły przez `WHERE`. Wiersze z `NULL`em w sumowanej kolumnie są pomijane, tak jakby ich nie było — nie liczą się jako zero.

Przykład:

```
 suma_zamowien
---------------
       4880.50
(1 row)
```

TYLKO ODCZYT

Średnia: AVG

```
SELECT AVG(cena) AS srednia_cena FROM produkty WHERE kategoria = 'elektronika';
```

`AVG` liczy średnią arytmetyczną z wartości niepustych. Mianownikiem jest liczba wierszy z wypełnioną kolumną, nie liczba wszystkich wierszy — `NULL`e nie wchodzą ani do sumy, ani do liczby dzielników.

Przykład:

```
    srednia_cena
---------------------
 298.3333333333333333
(1 row)
```

□□ TYLKO ODCZYT

Minimum i maksimum: MIN, MAX

```
SELECT MIN(cena) AS najtansza, MAX(cena) AS najdrozsza FROM produkty;
```

`MIN` i `MAX` działają na dowolnym typie porównywalnym — liczbach, datach, tekście — nie tylko na liczbach. Podobnie jak pozostałe agregaty, ignorują `NULL`e.

Przykład:

```
 najtansza | najdrozsza
-----------+------------
     25.00 |    1100.00
(1 row)
```

□□ TYLKO ODCZYT

Sklejenie tekstu z grupy: STRING_AGG

```
SELECT kategoria, STRING_AGG(nazwa, ', ' ORDER BY nazwa) AS produkty
FROM produkty
GROUP BY kategoria;
```

`STRING_AGG` skleja wartości tekstowe z wielu wierszy jednej grupy w jeden ciąg znaków, rozdzielony podanym separatorem. Nazwa i dokładna składnia różnią się między silnikami mocniej niż w przypadku pozostałych agregatów.

Przykład:

```
  kategoria  |                    produkty
-------------+-------------------------------------------------
 elektronika | Kabel HDMI 2m, Klawiatura mechaniczna, Monitor 24 cale
 meble       | Biurko regulowane, Fotel biurowy, Lampka biurkowa
(2 rows)
```

Silnik	Funkcja sklejająca tekst z grupy
PostgreSQL	`STRING_AGG(kolumna, separator)`
SQL Server (2017+)	`STRING_AGG(kolumna, separator)`
MySQL / MariaDB	`GROUP_CONCAT(kolumna SEPARATOR separator)`
Oracle	`LISTAGG(kolumna, separator)`

SQL = Structured Query Language: język zapytań do relacyjnych baz danych

TYLKO ODCZYT

Grupowanie po jednej kolumnie

```
SELECT kategoria, COUNT(*) AS liczba, AVG(cena) AS srednia_cena
FROM produkty
GROUP BY kategoria;
```

`GROUP BY` zwija wiersze o tej samej wartości wskazanej kolumny w jeden wiersz wyniku, a funkcje agregujące w `SELECT` liczą się osobno dla każdej takiej grupy.

Przykład:

```
  kategoria  | liczba |    srednia_cena
-------------+--------+--------------------
 elektronika |      3 | 298.3333333333333333
 meble       |      3 | 693.3333333333333333
(2 rows)
```

TYLKO ODCZYT

Grupowanie po wielu kolumnach

```
SELECT klient_id, status, COUNT(*) AS liczba, SUM(wartosc) AS suma
FROM zamowienia
GROUP BY klient_id, status
ORDER BY klient_id, status;
```

Kombinacja wartości ze wszystkich kolumn `GROUP BY` razem definiuje jedną grupę — dwa wiersze trafiają do tej samej grupy tylko wtedy, gdy zgadzają się na wszystkich wymienionych kolumnach naraz.

Przykład:

```
 klient_id |    status     | liczba |  suma
-----------+---------------+--------+---------
         1 | zrealizowane  |      2 | 1680.50
         2 | anulowane     |      1 |    0.00
         2 | nowe          |      1 |  150.00
         3 | w realizacji  |      1 | 2890.00
         4 | zrealizowane  |      1 | 3200.00
         5 | anulowane     |      1 |    0.00
           | nowe          |      1 |  899.00
(7 rows)
```

Zamówienie 106, z `klient_id` równym `NULL`, dostało własną grupę zamiast zniknąć z wyniku — `GROUP BY` traktuje `NULL` jako jedną, konkretną wartość do celów grupowania, w odróżnieniu od `WHERE`, gdzie porównanie z `NULL`em nigdy nie daje prawdy.

TYLKO ODCZYT

Grupowanie po wyrażeniu

```
SELECT EXTRACT(YEAR FROM data) AS rok, EXTRACT(MONTH FROM data) AS miesiac, COUNT(*) AS
liczba
FROM zamowienia
GROUP BY EXTRACT(YEAR FROM data), EXTRACT(MONTH FROM data)
ORDER BY rok, miesiac;
```

`GROUP BY` nie ogranicza się do nazw kolumn — może grupować po dowolnym wyrażeniu, o ile to samo wyrażenie da się policzyć dla każdego wiersza. Tu grupujemy zamówienia po miesiącu, mimo że w tabeli nie ma osobnej kolumny na miesiąc.

Przykład:

```
 rok | miesiac | liczba
-----+---------+--------
2024 |       1 |      2
2024 |       2 |      1
2024 |       3 |      3
2024 |       4 |      2
(4 rows)
```

Uważaj

- Kolumna w `SELECT`, której nie ma ani w `GROUP BY`, ani wewnątrz funkcji agregującej, jest błędem w PostgreSQL i Oracle — i niebezpiecznym wyjątkiem w starszym MySQL, opisanym niżej.
- `COUNT(kolumna)` i `COUNT(*)` w zapytaniu bez `GROUP BY` dają różne liczby tylko wtedy, gdy kolumna ma choć jeden `NULL` — przy kompletnych danych są identyczne, co maskuje pomyłkę do dnia, w którym dane przestają być kompletne.

Po co to istnieje

Pojęcie: agregacja. Zwinięcie wielu wierszy w jeden przez policzenie z nich pojedynczej wartości — sumy, średniej, licznika, wartości skrajnej albo sklejonego tekstu. Bez `GROUP BY` cała tabela (albo to, co z niej zostało po `WHERE`) jest jedną wielką grupą i funkcja agregująca zwraca dokładnie jeden wiersz wyniku. Z `GROUP BY` tabela dzieli się na tyle grup, ile różnych kombinacji wartości występuje w kolumnach grupujących, a agregacja liczy się osobno w każdej z nich.

Jak to działa

Rozwiń: jak to działa pod spodem

Co wolno w SELECT przy grupowaniu. Jeśli zapytanie ma `GROUP BY`, każda kolumna wymieniona w `SELECT` musi być albo kolumną grupującą, albo argumentem funkcji agregującej. Powód jest logiczny, nie arbitralny: grupa złożona z kilku wierszy nie ma jednej wspólnej wartości kolumny, która nie była częścią grupowania — `status` w grupie o `klient_id = 1` mógłby dla różnych wierszy tej grupy być inny, więc pytanie „który status pokazać w tym jednym wierszu wyniku” nie ma jednoznacznej odpowiedzi bez dodatkowej reguły.

```
SELECT klient_id, status, wartosc FROM zamowienia GROUP BY klient_id;
```

Przykład:

```
ERROR:  column "zamowienia.status" must appear in the GROUP BY clause
        or be used in an aggregate function
LINE 1: SELECT klient_id, status, wartosc FROM zamowienia GROUP BY...
                          ^
```

MySQL i ONLY_FULL_GROUP_BY. MySQL przed wersją 5.7 domyślnie pozwalał na dokładnie to, co wyżej kończy się błędem w PostgreSQL — zwracał wtedy wartość `status` i `wartosc` z jednego, dowolnie wybranego wiersza danej grupy, bez żadnej gwarancji, z którego. Zapytanie nie zgłaszało błędu i wyglądało na działające, dopóki ktoś nie porównał wyniku z oczekiwaniem. Od wersji 5.7.5 tryb `ONLY_FULL_GROUP_BY` jest domyślnie włączony i MySQL zaczyna zgłaszać ten sam błąd co PostgreSQL. Zapytania napisane pod starsze MySQL i migrowane na nowszą wersję albo na inny silnik są jednym z częstszych źródeł awarii przy takiej migracji — kod latami działał, bo nikt nie sprawdził, że wynik był przypadkowy.

ROLLUP, CUBE, GROUPING SETS. Trzy rozszerzenia `GROUP BY`, które oprócz zwykłych grup dokładają do wyniku wiersze podsumowań.

> **Pojęcie: ROLLUP.** Rozszerzenie `GROUP BY`, które oprócz każdej grupy dokłada wiersze sum częściowych, cofając się po kolei wzdłuż listy kolumn grupujących, aż do jednego wiersza sumy całkowitej.

```
SELECT status, SUM(wartosc) AS suma
FROM zamowienia
GROUP BY ROLLUP (status);
```

Przykład:

```
    status     |  suma
---------------+---------
 anulowane     |    0.00
 nowe          | 1049.00
 w realizacji  | 2890.00
 zrealizowane  | 4880.50
               | 8819.50
(5 rows)
```

Ostatni wiersz, z pustym `status`, to suma po wszystkich statusach naraz — `NULL` w tym miejscu nie znaczy „brak danych o statusie”, tylko „ten wiersz to podsumowanie, nie pojedyncza grupa”. Rozróżnienie tych dwóch znaczeń tego samego `NULL`a jest źródłem pomyłek opisanych niżej.

`CUBE` rozszerza ten pomysł na wszystkie możliwe kombinacje kolumn grupujących, nie tylko kolejne cofnięcia od pełnej listy do pustej. `GROUPING SETS` pozwala wprost wymienić, które kombinacje grupowania mają się znaleźć w wyniku, zamiast polegać na automatycznym schemacie `ROLLUP` czy `CUBE`.

Częste błędy

Mylenie NULL z ROLLUP z prawdziwym brakiem danych. Wiersz podsumowania z `ROLLUP` ma `NULL` w kolumnie grupującej z zupełnie innego powodu niż wiersz zamówienia 106, w którym `klient_id` naprawdę jest nieznany. Kod, który filtruje `WHERE status IS NOT NULL` po zapytaniu z `ROLLUP`, usunie wiersz podsumowania razem z prawdziwymi brakami danych, jeśli oba źródła `NULL` wystąpią w tej samej kolumnie.

Filtrowanie po agregacie w WHERE zamiast w HAVING. `WHERE SUM(wartosc) > 1000` nie skompiluje się — `WHERE` działa przed grupowaniem, więc agregat jeszcze nie istnieje. Strona HAVING: dlaczego nie da się tego zrobić w WHERE rozwija ten temat.

Zakładanie, że COUNT(kolumna) i COUNT(*) zawsze dają to samo. Różnią się tylko wtedy, gdy kolumna zawiera `NULL`e — dopóki dane są kompletne, różnica jest niewidoczna, co usypia czujność do dnia, w którym w danych pojawi się pierwszy brak.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — grupowanie po wielu kolumnach

Na podstawie tabeli `pozycje(id, zamowienie_id, produkt_id, ilosc, cena)` piszemy zapytanie liczące, ile sztuk każdego produktu sprzedano łącznie (`SUM(ilosc)`), grupując po `produkt_id`.

Odpowiedź

```
SELECT produkt_id, SUM(ilosc) AS sztuk_lacznie
FROM pozycje
GROUP BY produkt_id
ORDER BY produkt_id;
```

`SELECT` zawiera tylko `produkt_id` (kolumna grupująca) i `SUM(ilosc)` (agregat) — dokładnie to, co reguła „co wolno w SELECT przy grupowaniu” dopuszcza.

Pytania kontrolne

1. Dlaczego `SELECT klient_id, wartosc FROM zamowienia GROUP BY klient_id;` jest błędem, mimo że `wartosc` istnieje w tabeli?

Odpowiedź

Bo `wartosc` nie jest kolumną grupującą ani argumentem funkcji agregującej. Grupa dla danego `klient_id` może zawierać kilka wierszy o różnej `wartosc`, więc silnik nie ma jednoznacznej odpowiedzi, którą z nich pokazać w jednym wierszu wyniku.

2. Co się stanie z wierszem, w którym kolumna grupująca ma wartość NULL?

Odpowiedź

Dostanie własną grupę, tak jakby `NULL` był jedną, konkretną wartością do porównania z innymi `NULL`ami w tej samej kolumnie — inaczej niż w `WHERE`, gdzie porównanie z `NULL` nigdy nie daje prawdy.

Powiązane: NULL: trzecia wartość logiczna i pułapki, które z niej wynikają · HAVING: dlaczego nie da się tego zrobić w WHERE · SELECT: kolejność zapisu a kolejność wykonania

Źródła: dokumentacja PostgreSQL, rozdział „Aggregate Functions” i „GROUPING SETS, CUBE, and ROLLUP” · dokumentacja MySQL, sekcja „ONLY_FULL_GROUP_BY SQL Mode” · ISO/IEC 9075-2:2016 § 10.9

Revision #46
Created 2026-09-24 22:41:50 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:52 UTC by Tomasz Konieczny