Funkcje okna: OVER, PARTITION BY, ROW_NUMBER, LAG i LEAD

> Funkcja okna liczy coś na zbiorze wierszy — sumę, ranking, wartość sąsiedniego wiersza — ale w odróżnieniu od funkcji agregującej z GROUP BY nie zwija tych wierszy w jeden, tylko dokłada wynik do każdego z nich osobno.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
CTE	Common Table Expression: nazwany kawałek zapytania SQL pisany w WITH
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

Agregat kontra funkcja okna: to samo pytanie, inny kształt wyniku

```
SELECT klient_id, SUM(wartosc) AS suma
FROM zamowienia
WHERE klient_id IS NOT NULL
GROUP BY klient_id;
```

Zwykły agregat z `GROUP BY` zwija wszystkie zamówienia jednego klienta w jeden wiersz sumy — traci przy tym poszczególne zamówienia, zostaje tylko wynik zbiorczy.

Przykład:

```
 klient_id |  suma
-----------+---------
         1 | 1680.50
         2 |  150.00
         3 | 2890.00
         4 | 3200.00
         5 |    0.00
(5 rows)
```

```
SELECT k.nazwa, z.data, z.wartosc,
       SUM(z.wartosc) OVER (PARTITION BY z.klient_id) AS suma_klienta
FROM zamowienia z
JOIN klienci k ON k.id = z.klient_id
ORDER BY k.nazwa, z.data;
```

Ta sama suma, policzona funkcją okna zamiast `GROUP BY` — i każde pojedyncze zamówienie zostaje w wyniku widoczne osobno, obok sumy swojego klienta.

Przykład:

```
            nazwa             |    data    | wartosc | suma_klienta
------------------------------+------------+---------+--------------
 Baltic Trade Sp. z o.o.      | 2024-03-25 | 3200.00 |      3200.00
 Kowalski Handel              | 2024-01-20 |    0.00 |       150.00
 Kowalski Handel              | 2024-03-15 |  150.00 |       150.00
 Nowak i Wspólnicy Sp. z o.o. | 2024-01-15 | 1250.00 |      1680.50
 Nowak i Wspólnicy Sp. z o.o. | 2024-02-03 |  430.50 |      1680.50
 Podlaski Handel              | 2024-03-28 |    0.00 |         0.00
 Zielona Dolina S.A.          | 2024-03-10 | 2890.00 |      2890.00
(7 rows)
```

Kowalski Handel ma dwa wiersze, oba z tą samą sumą 150,00 obok siebie — dokładnie to jest różnica względem `GROUP BY`: funkcja okna liczy po zbiorze wierszy określonym przez `PARTITION BY`, ale nie zwija ich w jeden.

join = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

TYLKO ODCZYT

OVER, PARTITION BY i ORDER BY wewnątrz OVER

> **Pojęcie: funkcja okna.** Funkcja, która liczy wartość dla każdego wiersza na podstawie zbioru powiązanych wierszy — jego „okna" — bez zmniejszania liczby wierszy w wyniku. Klauzula `OVER (...)` po nazwie funkcji definiuje to okno: `PARTITION BY` dzieli wiersze na grupy (tak jak `GROUP BY`, ale bez zwijania), a `ORDER BY` wewnątrz `OVER` ustala kolejność, w jakiej funkcja przetwarza wiersze w obrębie każdej grupy — kluczowe dla funkcji zależnych od kolejności, jak `ROW_NUMBER` czy `LAG`.

LAG = Link Aggregation Group: grupa łączy spięta w jedno logiczne; wariant MC-LAG rozciąga ją na dwa przełączniki

TYLKO ODCZYT

ROW_NUMBER, RANK i DENSE_RANK przy remisie

Załóżmy, że do kategorii `elektronika` trafił nowy produkt w tej samej cenie co istniejąca klawiatura:

```
SELECT nazwa, cena,
       ROW_NUMBER() OVER (ORDER BY cena DESC, id) AS row_number,
       RANK()       OVER (ORDER BY cena DESC, id) AS rank,
       DENSE_RANK() OVER (ORDER BY cena DESC, id) AS dense_rank
FROM produkty
WHERE kategoria = 'elektronika';
```

Przykład:

```
         nazwa          |  cena  | row_number | rank | dense_rank
------------------------+--------+------------+------+------------
 Monitor 24 cale        | 650.00 |          1 |    1 |          1
 Klawiatura mechaniczna | 220.00 |          2 |    2 |          2
 Mysz bezprzewodowa     | 220.00 |          3 |    2 |          2
 Kabel HDMI 2m          |  25.00 |          4 |    4 |          3
(4 rows)
```

Trzy funkcje, trzy różne odpowiedzi na ten sam remis między klawiaturą a myszą, obie po 220,00. `ROW_NUMBER` ignoruje remis i przydziela kolejne liczby bez wyjątku — dlatego potrzebuje `id` jako dodatkowego kryterium w `ORDER BY`, żeby wynik był powtarzalny, a nie zależny od przypadkowej kolejności fizycznej. `RANK` daje remisującym tę samą pozycję (2), ale przeskakuje pozycję 3 — następny produkt dostaje 4, bo dwóch produktów zajęło już

pozycje 2 i 3 łącznie. `DENSE_RANK` też daje remisującym tę samą pozycję (2), ale nie przeskakuje niczego — następny produkt dostaje 3, kolejną liczbę bez dziury.

□□ TYLKO ODCZYT

LAG — wartość poprzedniego wiersza

```
SELECT k.nazwa, z.data, z.wartosc,
       LAG(z.wartosc) OVER (PARTITION BY z.klient_id ORDER BY z.data) AS
wartosc_poprzedniego
FROM zamowienia z
JOIN klienci k ON k.id = z.klient_id
ORDER BY k.nazwa, z.data;
```

`LAG(kolumna)` sięga o jeden wiersz wstecz w obrębie tej samej partycji, w kolejności ustalonej przez `ORDER BY` wewnątrz `OVER` — bez żadnego samozłączenia ani podzapytania.

Przykład:

```
            nazwa             |    data    | wartosc | wartosc_poprzedniego
------------------------------+------------+---------+----------------------
 Nowak i Wspólnicy Sp. z o.o. | 2024-01-15 | 1250.00 |
 Nowak i Wspólnicy Sp. z o.o. | 2024-02-03 |  430.50 |              1250.00
 Kowalski Handel              | 2024-01-20 |    0.00 |
 Kowalski Handel              | 2024-03-15 |  150.00 |                 0.00
 Zielona Dolina S.A.          | 2024-03-10 | 2890.00 |
 Baltic Trade Sp. z o.o.      | 2024-03-25 | 3200.00 |
 Podlaski Handel              | 2024-03-28 |    0.00 |
(7 rows)
```

Pierwsze zamówienie każdego klienta dostaje `NULL` w kolumnie `wartosc_poprzedniego` — nie ma niczego przed nim w tej partycji, i `LAG` zwraca to wprost, zamiast pomijać wiersz albo zgłaszać błąd. To jest ten sam problem, który strona CROSS JOIN i SELF JOIN rozwiązywała samozłączeniem z podzapytaniem `NOT EXISTS` — tutaj `LAG` daje identyczny efekt jedną funkcją, bez samozłączenia i bez brakującego pierwszego wiersza każdego klienta (który przy samozłączeniu po prostu nie miał pary i znikał z wyniku). `LEAD(kolumna)` działa lustrzanie — sięga o jeden wiersz do przodu.

□□ **LAG** = Link Aggregation Group: grupa łączy spięta w jedno logiczne; wariant MC-LAG rozciąga ją na dwa przełączniki · **join** = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

TYLKO ODCZYT

FIRST_VALUE — punkt odniesienia w obrębie partycji

```
SELECT nazwa, kategoria, cena,
       FIRST_VALUE(nazwa) OVER (PARTITION BY kategoria ORDER BY cena DESC) AS
najdrozszy_w_kategorii
FROM produkty
ORDER BY kategoria, cena DESC;
```

`FIRST_VALUE` zwraca wartość z pierwszego wiersza partycji, według `ORDER BY` w `OVER` — tutaj nazwę najdroższego produktu w każdej kategorii, powtórzoną przy każdym wierszu tej kategorii jako stały punkt odniesienia. `LAST_VALUE` działałaby analogicznie, tyle że wymaga jawnego ustawienia ramki na cały zakres partycji, inaczej zwraca wartość bieżącego wiersza zamiast ostatniego — pułapka opisana w dokumentacji każdego z czterech silników z tej książki.

Przykład:

```
         nazwa          |  kategoria  |  cena   | najdrozszy_w_kategorii
------------------------+-------------+---------+------------------------
 Monitor 24 cale        | elektronika |  650.00 | Monitor 24 cale
 Klawiatura mechaniczna | elektronika |  220.00 | Monitor 24 cale
 Kabel HDMI 2m          | elektronika |   25.00 | Monitor 24 cale
 Biurko regulowane      | meble       | 1100.00 | Biurko regulowane
 Fotel biurowy          | meble       |  890.00 | Biurko regulowane
 Lampka biurkowa        | meble       |   90.00 | Biurko regulowane
(6 rows)
```

TYLKO ODCZYT

Suma narastająca: SUM() OVER z porządkiem

```
SELECT k.nazwa, z.data, z.wartosc,
       SUM(z.wartosc) OVER (PARTITION BY z.klient_id ORDER BY z.data) AS
suma_narastajaca
FROM zamowienia z
JOIN klienci k ON k.id = z.klient_id
ORDER BY k.nazwa, z.data;
```

Dodanie `ORDER BY` do `SUM() OVER (...)` zmienia jej znaczenie z „suma całej partycji" (jak w pierwszej Fiszce tej strony) na „suma od początku partycji do bieżącego wiersza włącznie" — czyli sumę narastającą.

Przykład:

```
            nazwa             |    data    | wartosc | suma_narastajaca
------------------------------+------------+---------+------------------
 Nowak i Wspólnicy Sp. z o.o. | 2024-01-15 | 1250.00 |          1250.00
 Nowak i Wspólnicy Sp. z o.o. | 2024-02-03 |  430.50 |          1680.50
 Kowalski Handel              | 2024-01-20 |    0.00 |             0.00
 Kowalski Handel              | 2024-03-15 |  150.00 |           150.00
 Zielona Dolina S.A.          | 2024-03-10 | 2890.00 |          2890.00
 Baltic Trade Sp. z o.o.      | 2024-03-25 | 3200.00 |          3200.00
 Podlaski Handel              | 2024-03-28 |    0.00 |             0.00
(7 rows)
```

Drugie zamówienie Nowaka i Wspólników pokazuje 1680,50 — sumę obu jego zamówień do tej daty włącznie, nie tylko wartość bieżącego wiersza.

join = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

TYLKO ODCZYT

Najnowszy wiersz per klucz: ROW_NUMBER w podzapytaniu

```
WITH ponumerowane AS (
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY klient_id ORDER BY data DESC) AS rn
    FROM zamowienia
    WHERE klient_id IS NOT NULL
)
SELECT id, klient_id, data, status, wartosc
FROM ponumerowane
WHERE rn = 1;
```

> **Pojęcie: wzorzec „najnowszy wiersz per klucz".** Funkcja okna nie może wystąpić bezpośrednio w `WHERE` (bo logicznie liczy się później niż `WHERE`, zgodnie z kolejnością wykonania ze strony

SELECT: kolejność zapisu a kolejność wykonania), więc numerację trzeba najpierw policzyć w CTE albo podzapytaniu, a dopiero potem przefiltrować po numerze w zapytaniu zewnętrznym. `WHERE rn = 1` po `ROW_NUMBER() OVER (PARTITION BY klient_id ORDER BY data DESC)` daje dokładnie jedno, najnowsze zamówienie każdego klienta.

Przykład:

```
 id  |            nazwa             |    data    |    status    | wartosc
-----+------------------------------+------------+--------------+---------
 102 | Nowak i Wspólnicy Sp. z o.o. | 2024-02-03 | zrealizowane |  430.50
 105 | Kowalski Handel              | 2024-03-15 | nowe         |  150.00
 104 | Zielona Dolina S.A.          | 2024-03-10 | w realizacji | 2890.00
 107 | Baltic Trade Sp. z o.o.      | 2024-03-25 | zrealizowane | 3200.00
 108 | Podlaski Handel              | 2024-03-28 | anulowane    |    0.00
(5 rows)
```

Zapytanie łączy tu kolumnę `nazwa` przez dodatkowe złączenie z `klienci` pominięte w treści zapytania powyżej dla zwięzłości — sam wzorzec numerowania i filtrowania po `rn = 1` jest tym, co warto zapamiętać, bo wraca w bardzo wielu raportach: najnowsza cena produktu, ostatni status zamówienia, najświeższy wpis w logu zdarzeń na użytkownika.

CTE = Common Table Expression: nazwany kawałek zapytania SQL pisany w WITH

Po co to istnieje

Funkcje okna wypełniają lukę między zwykłym `SELECT` (jeden wynik na wiersz, bez wiedzy o sąsiadach) a `GROUP BY` (jeden wynik na grupę, kosztem utraty pojedynczych wierszy). Pytania w rodzaju „jaki jest udział procentowy tego zamówienia w sumie klienta", „które to jest zamówienie klienta z kolei", „ile wynosiła poprzednia transakcja tego samego typu" wymagają jednocześnie zachowania każdego wiersza osobno i dostępu do informacji policzonej na grupie wierszy wokół niego — dokładnie to, czego `GROUP BY` nie potrafi (bo zwija wiersze) i czego zwykły `SELECT` nie potrafi (bo nie widzi niczego poza bieżącym wierszem).

Jak to działa

> **Pojęcie: ramka okna (window frame).** Podzbiór wierszy w obrębie partycji, na którym funkcja okna faktycznie liczy swój wynik dla danego wiersza — domyślnie, przy obecności `ORDER BY` w `OVER`, ramka sięga od początku partycji do bieżącego wiersza włącznie, co tłumaczy zachowanie sumy narastającej z Fiszki wyżej. Ramkę da się ustawić jawnie klauzulą `ROWS BETWEEN`, na przykład `ROWS BETWEEN 1 PRECEDING AND CURRENT ROW` liczy tylko bieżący wiersz i jeden poprzedni — przydatne do prostej średniej kroczącej z dwóch ostatnich wartości, w odróżnieniu od sumy czy średniej narastającej liczonej od samego początku partycji.

Częste błędy

Funkcja okna bezpośrednio w WHERE. `SELECT * FROM zamowienia WHERE ROW_NUMBER() OVER (...) = 1` kończy się błędem składni w każdym z czterech silników — funkcje okna liczą się logicznie po `WHERE`, więc `WHERE` nie ma jeszcze dostępu do ich wyniku. Rozwiązaniem jest zawsze podzapytanie albo CTE, jak w ostatniej Fiszce tej strony.

```
ERROR:  window functions are not allowed in WHERE
```

ROW_NUMBER bez wystarczającego ORDER BY przy remisach. `ROW_NUMBER() OVER (ORDER BY cena DESC)` bez dodatkowego kryterium rozstrzygającego remis (tu `id`) da powtarzalny w ramach jednego wykonania, ale niegwarantowany między wykonaniami wynik dla wierszy o identycznej cenie — silnik może przydzielić numer 2 i 3 klawiaturze i myszy w innej kolejności przy każdym uruchomieniu, jeśli nic więcej nie odróżnia ich w `ORDER BY`.

LAST_VALUE bez jawnej ramki. `LAST_VALUE(nazwa) OVER (PARTITION BY kategoria ORDER BY cena DESC)` bez dopisania `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING` zwraca — wbrew nazwie — wartość bieżącego wiersza, nie ostatniego wiersza całej partycji, bo domyślna ramka z `ORDER BY` sięga tylko do bieżącego wiersza, tak jak przy sumie narastającej.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — RANK kontra DENSE_RANK

Cztery produkty w kategorii mają ceny 500, 500, 300, 100. Jakie wartości przydzieli im `RANK() OVER (ORDER BY cena DESC)`, a jakie `DENSE_RANK()` z tym samym `ORDER BY`?

Odpowiedź

`RANK`: 1, 1, 3, 4 — dwa produkty remisują na pozycji 1, a kolejny produkt (300) dostaje pozycję 3, bo dwie pozycje (1 i 2) zostały już zajęte przez remisujących. `DENSE_RANK`: 1, 1, 2, 3 — remis na pozycji 1, a kolejny produkt dostaje pozycję 2, bez żadnej przerwy w numeracji.

Pytania kontrolne

1. Czym różni się SUM(kolumna) OVER (PARTITION BY x) od SUM(kolumna) OVER (PARTITION BY x ORDER BY y)?

Odpowiedź

Pierwsza, bez `ORDER BY`, liczy sumę całej partycji i przypisuje tę samą wartość każdemu wierszowi partycji. Druga, z `ORDER BY`, domyślnie liczy sumę narastającą — od początku partycji do bieżącego wiersza włącznie — więc każdy kolejny wiersz w partycji dostaje inną, rosnącą wartość.

2. Dlaczego pierwsze zamówienie każdego klienta ma NULL w kolumnie LAG(wartosc), a nie na przykład zero?

Odpowiedź

Bo `LAG` sięga o wiersz wstecz w obrębie partycji, a dla pierwszego wiersza partycji żaden poprzedni wiersz nie istnieje. `NULL` oznacza tu brak wartości, dokładnie w tym samym sensie co gdziekolwiek indziej w SQL — zero sugerowałoby fałszywie, że poprzednie zamówienie istniało i miało wartość zero, co nie jest prawdą.

Powiązane: CROSS JOIN i SELF JOIN · CTE: WITH i zapytania rekurencyjne · SELECT: kolejność zapisu a kolejność wykonania · Podzapytania: skalarne, wierszowe, skorelowane, EXISTS

Źródła: ISO/IEC 9075-2:2016 § 7.11 (window clause) · dokumentacja PostgreSQL, rozdział „Window Functions" · dokumentacja Microsoft SQL Server, „OVER Clause (Transact-SQL)"

Revision #47
Created 2026-09-24 22:42:09 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:12 UTC by Tomasz Konieczny