SELECT: kolejność zapisu a kolejność wykonania

> `SELECT` opisujemy w jednej kolejności, a silnik wykonuje go w zupełnie innej — ta strona pokazuje obie kolejności obok siebie i tłumaczy, dlaczego alias z `SELECT` działa w `ORDER BY`, a w `WHERE` już nie.

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

Proste zapytanie

```
SELECT nazwa, miasto FROM klienci;
```

Zapytanie opisuje, jakie kolumny mają się znaleźć w wyniku i z jakiej tabeli pochodzą. Nic więcej — bez filtrowania, bez sortowania.

Przykład:

```
           nazwa               |  miasto
-------------------------------+-----------
 Nowak i Wspólnicy Sp. z o.o.  | Warszawa
 Kowalski Handel               | Kraków
```

```
 Zielona Dolina S.A.            | Wrocław
 Baltic Trade Sp. z o.o.        | Gdańsk
 Podlaski Handel                | Białystok
(5 rows)
```

TYLKO ODCZYT

Alias kolumny

```
SELECT nazwa AS klient, miasto AS lokalizacja FROM klienci;
```

Słowo kluczowe `AS` nadaje kolumnie wyniku inną nazwę niż nazwa kolumny źródłowej. `AS` jest opcjonalne w większości silników — `nazwa klient` bez niego działa tak samo — ale zostawiamy je, bo odróżnia alias od literówki w liście kolumn.

Przykład:

```
          klient                | lokalizacja
--------------------------------+-------------
 Nowak i Wspólnicy Sp. z o.o.   | Warszawa
 Kowalski Handel                | Kraków
 Zielona Dolina S.A.            | Wrocław
 Baltic Trade Sp. z o.o.        | Gdańsk
 Podlaski Handel                | Białystok
(5 rows)
```

TYLKO ODCZYT

Alias tabeli

```
SELECT k.nazwa, k.miasto FROM klienci AS k WHERE k.miasto = 'Warszawa';
```

Alias tabeli zastępuje jej pełną nazwę skróconym prefiksem przy każdym odwołaniu do kolumny. Przy jednej tabeli to wygoda, przy kilku połączonych złączeniem — jedyny sposób, żeby wskazać, z której tabeli pochodzi kolumna o niejednoznacznej nazwie. Rozdział Czym jest złączenie: iloczyn kartezjański i warunek, który go ogranicza wraca do tego, gdy w grze jest więcej niż jedna tabela.

Przykład:

```
           nazwa              |  miasto
------------------------------+-----------
 Nowak i Wspólnicy Sp. z o.o.  | Warszawa
(1 row)
```

TYLKO ODCZYT

Wszystkie kolumny: SELECT *

```
SELECT * FROM produkty;
```

Gwiazdka rozwija się w pełną listę kolumn tabeli, w kolejności, w jakiej są zdefiniowane w katalogu systemowym. Wygodne przy eksploracji danych z konsoli, ryzykowne w kodzie aplikacji — dodanie kolumny do tabeli zmienia liczbę i kolejność kolumn zwracanych do programu, który mógł się spodziewać stałego kształtu wyniku.

Przykład:

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

TYLKO ODCZYT

Wyrażenie w SELECT

```
SELECT nazwa, cena, cena * 1.23 AS cena_brutto FROM produkty WHERE kategoria = 'meble';
```

Lista w `SELECT` nie musi się ograniczać do nazw kolumn — może zawierać dowolne wyrażenie policzone dla każdego wiersza osobno: działanie arytmetyczne, sklejenie tekstu, wywołanie funkcji. Wynikowa kolumna nie istnieje w tabeli, istnieje wyłącznie w tym jednym wyniku.

Przykład:

```
        nazwa        |  cena   | cena_brutto
---------------------+---------+-------------
 Biurko regulowane   | 1100.00 |     1353.00
 Fotel biurowy       |  890.00 |     1094.70
 Lampka biurkowa     |   90.00 |      110.70
(3 rows)
```

Po co to istnieje

> **Pojęcie: zapytanie deklaratywne.** `SELECT` nie jest przepisem krok po kroku na to, jak policzyć wynik — jest opisem tego, jak wynik ma wyglądać. Piszemy, jakie kolumny, z jakich wierszy i w jakim porządku chcemy zobaczyć, a to, w jakiej kolejności silnik faktycznie przejdzie po danych, którego indeksu użyje i czy posortuje przed czy po filtrowaniu, zostaje jego decyzją — decyzją, którą da się podejrzeć w planie wykonania, opisanym na stronie Plan wykonania: jak go wywołać i jak go czytać. Z punktu widzenia autora zapytania liczy się wynik, nie droga do niego.

Ta różnica między opisem a przepisem tłumaczy, dlaczego kolejność, w jakiej piszemy klauzule `SELECT`, nie jest kolejnością, w jakiej silnik je liczy.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: logiczna kolejność wykonania.** Ustalona przez standard SQL kolejność, w jakiej silnik logicznie przetwarza klauzule zapytania — niezależnie od tego, w jakiej kolejności fizycznie odczytuje dane z dysku czy z pamięci. „Logicznie” znaczy, że to model do rozumowania o zapytaniu, a nie dosłowny opis pracy silnika: optymalizator wolno mu zmienić kolejność operacji, o ile wynik końcowy wyjdzie taki, jakby trzymał się modelu.

Kolejność zapisu klauzul w zapytaniu:

```
SELECT ...
FROM ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...
LIMIT ...
```

Logiczna kolejność wykonania jest inna:

1. `FROM` — silnik ustala źródłowy zbiór wierszy.
2. `WHERE` — z tego zbioru odsiewa wiersze niespełniające warunku.
3. `GROUP BY` — zwija pozostałe wiersze w grupy.
4. `HAVING` — odsiewa grupy niespełniające warunku na agregacie.
5. `SELECT` — dla tego, co zostało, liczy listę kolumn i wyrażeń wyniku.
6. `DISTINCT` — usuwa zduplikowane wiersze wyniku, jeśli klauzula jest obecna.
7. `ORDER BY` — sortuje to, co zostało.
8. `LIMIT` / `OFFSET` — obcina wynik do żądanej liczby wierszy.

Weźmy zapytanie korzystające z większości tych kroków:

```
SELECT klient_id, SUM(wartosc) AS suma
FROM zamowienia
WHERE status = 'zrealizowane'
GROUP BY klient_id
HAVING SUM(wartosc) > 1000
ORDER BY suma DESC
LIMIT 5;
```

Przykład:

```
 klient_id |  suma
-----------+---------
         4 | 3200.00
         1 | 1680.50
(2 rows)
```

Silnik najpierw bierze całą tabelę `zamowienia` (krok 1), odrzuca wiersze ze statusem innym niż `zrealizowane` (krok 2), zwija pozostałe w grupy po `klient_id` (krok 3), odrzuca grupy, których suma nie przekracza 1000 (krok 4), dopiero teraz liczy kolumny wyniku łącznie z aliasem `suma`

(krok 5), sortuje po tym aliasie (krok 7) i obcina do pięciu wierszy (krok 8) — w tym przypadku i tak zostały tylko dwa.

Z tej kolejności wynika konkretna, praktyczna konsekwencja: alias zdefiniowany w `SELECT` nie istnieje jeszcze na etapie `WHERE`, bo `WHERE` wykonuje się w kroku 2, a `SELECT` dopiero w kroku 5.

```
SELECT klient_id, SUM(wartosc) AS suma
FROM zamowienia
WHERE suma > 1000
GROUP BY klient_id;
```

Przykład:

```
ERROR:  column "suma" does not exist
LINE 3: WHERE suma > 1000
              ^
```

Ten sam alias w `ORDER BY` działa bez zarzutu, bo `ORDER BY` wykonuje się w kroku 7, długo po tym, jak `SELECT` w kroku 5 policzył `suma` i nadał jej nazwę:

```
SELECT klient_id, SUM(wartosc) AS suma
FROM zamowienia
GROUP BY klient_id
ORDER BY suma DESC;
```

Filtr, który logicznie dotyczy agregatu, trafia zamiast tego do `HAVING` — strona HAVING: dlaczego nie da się tego zrobić w WHERE tłumaczy to dokładniej. Warunek dotyczący pojedynczego wiersza, zanim jeszcze trafi do grupowania, zawsze warto trzymać w `WHERE` — silnik odsieje go najwcześniej, jak to możliwe, i będzie miał mniej danych do zgrupowania.

Częste błędy

Filtrowanie po aliasie w WHERE. Opisane wyżej `column "suma" does not exist` — pierwszy odruch to poprawka na `HAVING SUM(wartosc) > 1000`, drugi to powtórzenie całego wyrażenia w WHERE`, jeśli warunek faktycznie dotyczy pojedynczego wiersza, a nie grupy.

Założenie, że kolejność zapisu klauzul jest dowolna. `WHERE` musi stać przed `GROUP BY`, a `GROUP BY` przed `HAVING` — to nie jest kwestia stylu, tylko wymóg składni, zgodny zresztą z logiczną kolejnością wykonania. Zapisanie ich w innej kolejności kończy się błędem składniowym, nie ostrzeżeniem.

Poleganie na kolejności zwracanej przez SELECT * bez ORDER BY. Bez jawnego `ORDER BY` kolejność wierszy w wyniku jest nieokreślona, nawet jeśli w praktyce wygląda stabilnie przez miesiące — strona ORDER BY, LIMIT, TOP, ROWNUM, FETCH: porcjowanie wyniku tłumaczy, dlaczego.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — kolejność wykonania

Dla zapytania `SELECT status, COUNT(*) AS ile FROM zamowienia WHERE klient_id IS NOT NULL GROUP BY status HAVING COUNT(*) > 1 ORDER BY ile DESC;` wypisujemy numerami kolejność, w jakiej silnik logicznie wykona poszczególne klauzule.

Odpowiedź

1. `FROM zamowienia`
2. `WHERE klient_id IS NOT NULL`
3. `GROUP BY status`
4. `HAVING COUNT(*) > 1`
5. `SELECT status, COUNT(*) AS ile`
6. `ORDER BY ile DESC`

`LIMIT` i `DISTINCT` nie występują w tym zapytaniu, więc kroki 6 i 8 z pełnej listy są pomijane.

Pytania kontrolne

1. Dlaczego `ORDER BY suma` działa, a `WHERE suma > 1000` w tym samym zapytaniu zgłasza błąd, skoro `suma` to alias zdefiniowany w tym samym SELECT?

Odpowiedź

Bo `WHERE` wykonuje się logicznie przed `SELECT`, więc alias jeszcze nie istnieje, a `ORDER BY` wykonuje się po `SELECT`, więc alias jest już policzony i dostępny.

2. Czy `SELECT *` gwarantuje tę samą kolejność kolumn w każdym wywołaniu tego samego zapytania na tej samej tabeli?

Odpowiedź

Tak, dopóki nikt nie zmieni struktury tabeli. Kolejność odpowiada kolejności kolumn w katalogu systemowym, ale `ALTER TABLE ... ADD COLUMN` dopisuje nową kolumnę na końcu, więc kod polegający na stałej liczbie i kolejności kolumn z `SELECT *` jest kruchy na zmiany struktury.

3. Zapytanie `SELECT nazwa AS n FROM klienci WHERE n = 'Kowalski Handel';` kończy się błędem. Dlaczego, skoro to samo zapytanie z `ORDER BY n` zamiast `WHERE n` działa bez zarzutu?

Odpowiedź

Alias `n` powstaje dopiero w kroku piątym logicznej kolejności wykonania, razem z resztą listy `SELECT`. `WHERE` wykonuje się w kroku drugim, więc `n` jeszcze nie istnieje i silnik zgłasza błąd „column n does not exist”. `ORDER BY` wykonuje się w kroku siódmym, długo po tym, jak `SELECT` policzył i nazwał `n`, więc w tym miejscu alias jest już dostępny.

Powiązane: WHERE: filtrowanie wierszy · HAVING: dlaczego nie da się tego zrobić w WHERE · Funkcje agregujące i GROUP BY · Plan wykonania: jak go wywołać i jak go czytać

Źródła: ISO/IEC 9075-2:2016 § 7.16 · dokumentacja PostgreSQL, rozdział „The Statistics Collector” i „Select Lists”

Revision #47
Created 2026-09-24 22:41:44 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:46 UTC by Tomasz Konieczny