HAVING: dlaczego nie da się tego zrobić w WHERE

> `HAVING` filtruje grupy, `WHERE` filtruje wiersze, a granica między nimi wynika wprost z logicznej kolejności wykonania zapytania — ta strona pokazuje, kiedy warunek należy do jednej klauzuli, kiedy do drugiej, i co się dzieje, gdy `HAVING` pojawia się bez `GROUP BY`.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
ISO	International Organization for Standardization: organizacja, która wydaje normy techniczne używane na całym świecie

Fiszka

TYLKO ODCZYT

Filtrowanie wierszy przed grupowaniem: WHERE

```
SELECT klient_id, COUNT(*) AS liczba_zamowien
FROM zamowienia
WHERE status = 'zrealizowane'
GROUP BY klient_id;
```

Warunek dotyczy pojedynczego wiersza — statusu konkretnego zamówienia — więc trafia do `WHERE`, zanim silnik zacznie cokolwiek grupować. Wiersze niespełniające warunku odpadają najwcześniej, jak to możliwe, i nigdy nie trafiają do grupowania.

Przykład:

```
 klient_id | liczba_zamowien
-----------+------------------
```

```
         1 |                2
         4 |                1
(2 rows)
```

⚠ ZMIENIA SYSTEM

Filtrowanie grup po agregacie: HAVING

```
SELECT klient_id, COUNT(*) AS liczba_zamowien
FROM zamowienia
GROUP BY klient_id
HAVING COUNT(*) > 1;
```

Warunek dotyczy nie pojedynczego wiersza, tylko wyniku policzonego dla całej grupy — liczby zamówień danego klienta. Taki warunek nie ma prawa trafić do `WHERE`, bo `COUNT(*)` dla grupy nie istnieje, dopóki grupowanie się nie wykona, a `WHERE` działa przed nim.

Przykład:

```
 klient_id | liczba_zamowien
-----------+-----------------
         1 |                2
         2 |                2
(2 rows)
```

⚠ ZMIENIA SYSTEM

WHERE i HAVING w jednym zapytaniu

```
SELECT klient_id, SUM(wartosc) AS suma
FROM zamowienia
WHERE status <> 'anulowane'
GROUP BY klient_id
HAVING SUM(wartosc) > 500
ORDER BY klient_id;
```

Nic nie stoi na przeszkodzie, żeby jedno zapytanie odsiewało wiersze przed grupowaniem i grupy po grupowaniu naraz — to normalny, częsty przypadek, nie wyjątek. `WHERE` odrzuca anulowane zamówienia, zanim cokolwiek zostanie zgrupowane, a `HAVING` odrzuca klientów,

których suma pozostałych zamówień nie przekracza 500.

Przykład:

```
 klient_id |  suma
-----------+---------
         1 | 1680.50
         3 | 2890.00
         4 | 3200.00
           |  899.00
(4 rows)
```

Ostatni wiersz to grupa zamówień bez przypisanego klienta (`klient_id IS NULL`) — `GROUP BY` traktuje `NULL` jako osobną, prawidłową grupę, tak jak opisuje to strona Funkcje agregujące i GROUP BY.

◫◫ ZMIENIA SYSTEM

HAVING bez GROUP BY

```
SELECT COUNT(*) AS liczba, SUM(wartosc) AS suma
FROM zamowienia
HAVING SUM(wartosc) > 5000;
```

Bez `GROUP BY` cała tabela liczy się jako jedna, jedyna grupa — `HAVING` filtruje wtedy tę jedną grupę tak samo, jak filtrowałby jedną z wielu. Jeśli warunek jest spełniony, zapytanie zwraca dokładnie jeden wiersz z policzonymi agregatami; jeśli nie, zwraca zero wierszy zamiast jednego.

Przykład:

```
 liczba |  suma
--------+---------
      8 | 8819.50
(1 row)
```

◫◫ ZMIENIA SYSTEM

Kilka warunków na agregatach naraz

```
SELECT klient_id, COUNT(*) AS liczba, SUM(wartosc) AS suma
FROM zamowienia
GROUP BY klient_id
HAVING COUNT(*) > 1 AND SUM(wartosc) > 1000;
```

`HAVING` łączy warunki tak samo jak `WHERE` — przez `AND`, `OR` i `NOT`, z tymi samymi priorytetami. Nic nie stoi na przeszkodzie, żeby jeden warunek odwoływał się do jednego agregatu, a drugi do innego, o ile oba dotyczą tej samej grupy.

Przykład:

```
 klient_id | liczba |  suma
-----------+--------+---------
         1 |      2 | 1680.50
(1 row)
```

Klient 2 odpada mimo dwóch zamówień, bo ich suma (`0.00 + 150.00 = 150.00`) nie przekracza tysiąca — oba warunki muszą być spełnione naraz.

□□ ZMIENIA SYSTEM

Filtrowanie po liczbie unikalnych wartości w grupie

```
SELECT klient_id, COUNT(DISTINCT status) AS liczba_statusow
FROM zamowienia
GROUP BY klient_id
HAVING COUNT(DISTINCT status) > 1;
```

`HAVING` sprawdza dowolne wyrażenie agregujące, nie tylko gołe `COUNT(*)` czy `SUM`. Tu szukamy klientów, których zamówienia miały więcej niż jeden różny status — sygnał, że historia zamówień takiego klienta jest bardziej zróżnicowana niż u pozostałych.

Przykład:

```
 klient_id | liczba_statusow
-----------+-----------------
         2 |               2
(1 row)
```

Klient 2 ma jedno zamówienie anulowane i jedno nowe — dwa różne statusy. Pozostali klienci mają albo jedno zamówienie, albo kilka zamówień o tym samym statusie.

> **Uważaj**
>
> - Agregat w `WHERE` (`WHERE COUNT(*) > 1`) kończy się błędem składniowym — agregaty nie istnieją jeszcze na etapie `WHERE`.
> - Warunek bez agregatu w `HAVING` (`HAVING status = 'nowe'`) często działa, ale filtruje później, niż mógłby, marnując pracę włożoną w pogrupowanie wierszy, które i tak zostaną odrzucone.

Po co to istnieje

Strona SELECT: kolejność zapisu a kolejność wykonania ustaliła logiczną kolejność wykonania zapytania: `FROM` → `WHERE` → `GROUP BY` → `HAVING` → `SELECT` → `ORDER BY`. `WHERE` wykonuje się w kroku drugim, długo zanim jakiekolwiek grupowanie miało szansę się zdarzyć — dlatego warunek w `WHERE` może odwoływać się wyłącznie do kolumn pojedynczego wiersza, nigdy do wyniku funkcji agregującej. `HAVING` wykonuje się w kroku czwartym, po `GROUP BY` — dopiero wtedy agregaty policzone dla każdej grupy istnieją i mogą wejść w warunek.

Z tej różnicy w momencie wykonania wynika praktyczna zasada podziału pracy między obiema klauzulami, rozwinięta niżej.

Jak to działa

Kiedy warunek trafia do WHERE. Zawsze wtedy, gdy dotyczy wartości dostępnej w pojedynczym wierszu, zanim cokolwiek zostanie zgrupowane — statusu, daty, identyfikatora klienta, dowolnej kolumny źródłowej albo prostego wyrażenia na niej. Filtrowanie w `WHERE` jest tańsze, bo silnik odrzuca niepotrzebne wiersze najwcześniej, jak to możliwe — mniej wierszy trafia do grupowania, mniej pracy wykonuje sortowanie czy budowanie tabeli haszującej pod agregację.

Kiedy warunek musi trafić do HAVING. Zawsze wtedy, gdy odwołuje się do wyniku funkcji agregującej — sumy, liczby wierszy w grupie, średniej, wartości minimalnej czy maksymalnej policzonej dla całej grupy. Taki warunek fizycznie nie może zostać sprawdzony wcześniej niż grupowanie, bo przed grupowaniem nie ma jeszcze niczego do policzenia — pojedynczy wiersz nie ma własnej sumy zamówień klienta, ma tylko wartość swojego jednego zamówienia.

Reguła praktyczna: jeśli da się przenieść warunek z `HAVING` do `WHERE` bez zmiany znaczenia zapytania, warto to zrobić — silnik odsieje wiersze wcześniej i grupowanie będzie miało mniej

materiału do przetworzenia. Warunek zostaje w `HAVING` tylko wtedy, gdy naprawdę nie da się go inaczej wyrazić — bo dotyczy czegoś, co istnieje wyłącznie na poziomie grupy.

Różnicę widać w planie wykonania, opisanym dokładniej na stronie Plan wykonania: jak go wywołać i jak go czytać. Warunek przeniesiony z `HAVING` do `WHERE` przesuwa się w planie z filtra nałożonego na węzeł grupujący (`GroupAggregate` albo `HashAggregate` z dodatkowym `Filter`) na filtr nałożony na sam odczyt tabeli, często jeszcze przed jakimkolwiek sortowaniem czy haszowaniem — mniej wierszy wchodzi do droższego etapu, więc ten etap kończy się szybciej.

Częste błędy

Agregat wstawiony do WHERE z przyzwyczajenia.

```
SELECT klient_id, COUNT(*) AS liczba
FROM zamowienia
WHERE COUNT(*) > 1
GROUP BY klient_id;
```

Przykład:

```
ERROR:  aggregate functions are not allowed in WHERE
LINE 3: WHERE COUNT(*) > 1
              ^
```

Poprawka to przeniesienie warunku do `HAVING`, jak w drugiej Fiszce tej strony.

Warunek na pojedynczym wierszu trzymany w HAVING zamiast w WHERE.

```
SELECT klient_id, COUNT(*) AS liczba
FROM zamowienia
GROUP BY klient_id
HAVING klient_id IS NOT NULL;
```

Zapytanie zadziała i da poprawny wynik, ale silnik najpierw pogrupuje wszystkie wiersze, łącznie z tymi o `klient_id IS NULL`, i dopiero po fakcie odrzuci niepotrzebną grupę. Ten sam warunek w `WHERE` odsiałby wiersz 106 przed grupowaniem, więc grupowanie miałoby mniej pracy — przy ośmiu wierszach różnica jest niemierzalna, przy ośmiu milionach już nie.

HAVING bez GROUP BY mylone z pustym wynikiem WHERE. Zapytanie `SELECT SUM(wartosc) FROM zamowienia HAVING SUM(wartosc) > 100000;` przy niespełnionym warunku zwraca zero wierszy, nie jeden wiersz z wartością `NULL` czy zerem — łatwo to pomylić z sytuacją, w której zapytanie się

nie wykonało, podczas gdy wykonało się poprawnie i po prostu odrzuciło jedyną istniejącą grupę.

Odwołanie do aliasu z SELECT wewnątrz HAVING.

```
SELECT klient_id, COUNT(*) AS liczba
FROM zamowienia
GROUP BY klient_id
HAVING liczba > 1;
```

Przykład:

```
ERROR:  column "liczba" does not exist
LINE 4: HAVING liczba > 1;
               ^
```

Ten sam mechanizm, który w SELECT: kolejność zapisu a kolejność wykonania tłumaczy, dlaczego alias z `SELECT` nie działa w `WHERE`, obowiązuje też tutaj — `HAVING` wykonuje się w kroku czwartym logicznej kolejności, `SELECT` dopiero w piątym, więc alias `liczba` jeszcze nie istnieje w momencie sprawdzania `HAVING`. Poprawka to powtórzenie pełnego wyrażenia agregującego: `HAVING COUNT(*) > 1`.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — dobór klauzuli

Piszemy zapytanie zwracające kategorie produktów, w których średnia cena przekracza 300, pomijając przy tym z obliczeń produkty z kategorii `elektronika` tańsze niż 50 (traktujemy je jako akcesoria, nie główny asortyment).

Odpowiedź

```
SELECT kategoria, AVG(cena) AS srednia_cena
FROM produkty
WHERE cena >= 50
GROUP BY kategoria
```

```
HAVING AVG(cena) > 300;
```

Warunek na pojedynczym produkcie (`cena >= 50`) trafia do `WHERE`, bo dotyczy pojedynczego wiersza przed grupowaniem. Warunek na średniej dla całej kategorii trafia do `HAVING`, bo `AVG(cena)` nie istnieje przed zgrupowaniem produktów.

Pytania kontrolne

1. Dlaczego `WHERE SUM(wartosc) > 1000` jest błędem składniowym, a `HAVING SUM(wartosc) > 1000` nie?

Odpowiedź

`WHERE` wykonuje się przed `GROUP BY` w logicznej kolejności zapytania, więc `SUM(wartosc)` dla grupy jeszcze nie istnieje w tym momencie. `HAVING` wykonuje się po `GROUP BY`, kiedy agregaty są już policzone.

2. Co zwróci zapytanie z HAVING, ale bez GROUP BY, jeśli warunek w HAVING nie jest spełniony?

Odpowiedź

Zero wierszy. Cała tabela liczy się wtedy jako jedna grupa, a `HAVING` odrzuca tę jedną grupę w całości, zamiast zwrócić wiersz z wartością `NULL` czy zerem.

3. Dlaczego `HAVING liczba > 1` zgłasza błąd, skoro `liczba` to nazwa kolumny widocznej w wyniku tego samego zapytania?

Odpowiedź

`liczba` to alias nadany w klauzuli `SELECT`, a `SELECT` wykonuje się logicznie po `HAVING`. W momencie sprawdzania warunku `HAVING` ten alias jeszcze nie istnieje — trzeba w to miejsce wstawić pełne wyrażenie, na przykład `COUNT(*)`.

Powiązane: SELECT: kolejność zapisu a kolejność wykonania · Funkcje agregujące i GROUP BY · WHERE: filtrowanie wierszy

Źródła: ISO/IEC 9075-2:2016 § 7.9 · dokumentacja PostgreSQL, rozdział „The HAVING Clause”

Revision #47
Created 2026-09-24 22:41:52 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:54 UTC by Tomasz Konieczny