# Plan wykonania: jak go wywołać i jak go czytać

> Zanim silnik wykona zapytanie, decyduje, w jakiej kolejności i jakimi metodami dotrze do danych — ta strona pokazuje, jak podejrzeć tę decyzję w czterech silnikach i jak czytać wynik, który na pierwszy rzut oka wygląda jak nieczytelne drzewo skrótów.

**Skróty użyte na tej stronie (2)**

| Skrót | Co znaczy |
|---|---|
| **SET** | Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP |
| **SQL** | Structured Query Language: język zapytań do relacyjnych baz danych |

# Fiszka

**TYLKO ODCZYT**

## Wywołanie planu szacowanego w PostgreSQL: EXPLAIN

```
EXPLAIN SELECT k.nazwa, z.data, z.wartosc
FROM zamowienia z
JOIN klienci k ON k.id = z.klient_id
WHERE z.status = 'nowe';
```

`EXPLAIN` samo w sobie nie wykonuje zapytania — pyta optymalizator, jaki plan by wybrał, i pokazuje wyłącznie szacunki: koszt i liczbę wierszy, jakich się spodziewa, bez faktycznego uruchomienia.

Przykład:

```
                                  QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=15.50..98.34 rows=412 width=48)
   Hash Cond: (z.klient_id = k.id)
   ->  Seq Scan on zamowienia z  (cost=0.00..78.00 rows=412 width=24)
         Filter: (status = 'nowe'::text)
   ->  Hash  (cost=12.60..12.60 rows=180 width=32)
         ->  Seq Scan on klienci k  (cost=0.00..12.60 rows=180 width=32)
(6 rows)
```

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

**ZMIENIA SYSTEM**

## Wywołanie planu z rzeczywistym wykonaniem w PostgreSQL: EXPLAIN ANALYZE

```
EXPLAIN (ANALYZE, BUFFERS) SELECT k.nazwa, z.data, z.wartosc
FROM zamowienia z
JOIN klienci k ON k.id = z.klient_id
WHERE z.status = 'nowe' AND z.data >= '2024-01-01';
```

`EXPLAIN ANALYZE`, w odróżnieniu od gołego `EXPLAIN`, naprawdę wykonuje zapytanie — dopiero potem pokazuje plan, dokładając do każdego węzła rzeczywisty czas i rzeczywistą liczbę wierszy obok szacunków. To jest kluczowa różnica: wynik zapytania faktycznie powstaje i, przy `SELECT`, faktycznie trafia do klienta.

Przykład:

```
                                                        QUERY PLAN
------------------------------------------------------------------------------------------
---------------------------
 Hash Join  (cost=15234.11..98234.55 rows=41200 width=48) (actual time=142.334..1893.221
rows=38914 loops=1)
   Hash Cond: (z.klient_id = k.id)
   Buffers: shared hit=8102 read=14340
   ->  Bitmap Heap Scan on zamowienia z  (cost=980.22..78450.10 rows=41200 width=24)
(actual time=12.442..1602.887 rows=38914 loops=1)
         Recheck Cond: ((status = 'nowe'::text) AND (data >= '2024-01-01'::date))
```

```
         Heap Blocks: exact=18244
         Buffers: shared hit=8020 read=14320
         ->  Bitmap Index Scan on ix_zamowienia_status_data  (cost=0.00..970.02
rows=41200 width=0) (actual time=9.884..9.884 rows=38914 loops=1)
               Index Cond: ((status = 'nowe'::text) AND (data >= '2024-01-01'::date))
               Buffers: shared hit=210 read=180
   ->  Hash  (cost=12.60..12.60 rows=180 width=32) (actual time=0.128..0.129 rows=180
loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 17kB
         ->  Seq Scan on klienci k  (cost=0.00..12.60 rows=180 width=32) (actual
time=0.006..0.052 rows=180 loops=1)
 Planning Time: 0.842 ms
 Execution Time: 1897.556 ms
(15 rows)
```

Przy `UPDATE` albo `DELETE` `EXPLAIN ANALYZE` naprawdę zmienia dane, bo naprawdę wykonuje polecenie — sprawdzenie planu takiego zapytania na produkcji bez zabezpieczenia dosłownie zmodyfikuje tabelę. Bezpieczny sposób to opakowanie w transakcję i wycofanie jej po obejrzeniu planu:

```
BEGIN;
EXPLAIN ANALYZE UPDATE zamowienia SET status = 'archiwalne' WHERE data < '2020-01-01';
ROLLBACK;
```

Przykład:

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------
-------------
 Update on zamowienia  (cost=0.43..8412.30 rows=18400 width=42) (actual
time=210.442..3402.118 rows=0 loops=1)
   ->  Index Scan using ix_zamowienia_data on zamowienia  (cost=0.43..8412.30 rows=18400
width=42) (actual time=0.088..940.221 rows=18400 loops=1)
         Index Cond: (data < '2020-01-01'::date)
 Planning Time: 0.311 ms
 Execution Time: 3402.887 ms
(5 rows)
```

```
ROLLBACK
```

`ROLLBACK` cofa modyfikację danych, ale plan i rzeczywiste czasy zostały już pokazane — dokładnie to, o co chodziło.

**join** = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych ·**SET** = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP

**ZMIENIA SYSTEM**

## Wywołanie planu graficznego i statystyk odczytu w MSSQL

```
SET SHOWPLAN_ALL ON;
```

`SET SHOWPLAN_ALL ON` przełącza sesję w tryb, w którym kolejne polecenia nie są wykonywane — SQL Server zwraca sam plan szacowany w postaci tekstowej tabeli zamiast wyniku zapytania. W SQL Server Management Studio ten sam plan szacowany dostępny jest wygodniej jako widok graficzny pod skrótem `Ctrl+L`, a plan rzeczywisty, z wykonaniem zapytania, pod `Ctrl+M`.

Przykład:

```
StmtText                                          StmtId  NodeId  Parent
PhysicalOp        LogicalOp         EstimateRows
------------------------------------------------- ------- ------- -------
----------------- ----------------- ------------
  |--Hash Match(Inner Join, ...)                    1       1       0       Hash
Match        Inner Join        41200.0
       |--Index Seek(...IX_Zamowienia_Status...)    1       2       1       Index
Seek         Index Seek        41200.0
       |--Table Scan(...Klienci...)                 1       3       1       Table
Scan         Table Scan        180.0


(3 rows affected)
```

```
SET STATISTICS IO ON;
```

`SET STATISTICS IO ON` uzupełnia rzeczywiste wykonanie zapytania o liczbę odczytów logicznych i fizycznych dla każdej tabeli — to jest odpowiednik `BUFFERS` z PostgreSQL, tylko doklejony do wykonania zapytania, nie do samego planu.

Przykład:

```
Table 'Zamowienia'. Scan count 1, logical reads 18420, physical reads 210, read-ahead
reads 18100.
Table 'Klienci'. Scan count 1, logical reads 12, physical reads 0, read-ahead reads 0.


(38914 rows affected)
```

**MSSQL** = Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima · **SET** = Switch Embedded Teaming: zespół kart sieciowych zbudowany w przełączniku wirtualnym Hyper-V; łączy dwie karty w jedno łącze bez ustawień po stronie przełącznika fizycznego, więc bez LACP · **SQL** = Structured Query Language: język zapytań do relacyjnych baz danych

**TYLKO ODCZYT**

## Wywołanie planu w Oracle: EXPLAIN PLAN FOR i DBMS_XPLAN.DISPLAY

```
EXPLAIN PLAN FOR
SELECT k.nazwa, z.data, z.wartosc
FROM zamowienia z
JOIN klienci k ON k.id = z.klient_id
WHERE z.status = 'nowe';
```

`EXPLAIN PLAN FOR` nie zwraca planu bezpośrednio — zapisuje go do tabeli planu (`PLAN_TABLE`), skąd trzeba go dopiero odczytać osobnym poleceniem.

Przykład:

```
Explained.
```

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Przykład:

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
--
Plan hash value: 2831940221


-------------------------------------------------------------------------------
| Id  | Operation                    | Name                    | Rows  | Cost  |
```

```
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                          | 41200 |  1842 |
|*  1 |  HASH JOIN                   |                          | 41200 |  1842 |
|   2 |   TABLE ACCESS BY INDEX ROWID | ZAMOWIENIA               | 41200 |  1620 |
|*  3 |    INDEX RANGE SCAN          | IX_ZAMOWIENIA_STATUS     | 41200 |   210 |
|   4 |   TABLE ACCESS FULL          | KLIENCI                  |   180 |    12 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("K"."ID"="Z"."KLIENT_ID")
   3 - access("Z"."STATUS"='nowe')

18 rows selected.
```

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

□□ **TYLKO ODCZYT**

## Wywołanie planu w MySQL 8: EXPLAIN FORMAT=JSON i EXPLAIN ANALYZE

```
EXPLAIN FORMAT=JSON
SELECT k.nazwa, z.data, z.wartosc
FROM zamowienia z
JOIN klienci k ON k.id = z.klient_id
WHERE z.status = 'nowe';
```

Przykład:

**Pokaż pełny przykład (30 linii)**

```
EXPLAIN
{
  "query_block": {
    "select_id": 1,
    "cost_info": { "query_cost": "9821.40" },
```

```
    "nested_loop": [
      {
        "table": {
          "table_name": "z",
          "access_type": "ref",
          "possible_keys": ["ix_zamowienia_status"],
          "key": "ix_zamowienia_status",
          "rows_examined_per_scan": 41200,
          "filtered": "100.00"
        }
      },
      {
        "table": {
          "table_name": "k",
          "access_type": "eq_ref",
          "possible_keys": ["PRIMARY"],
          "key": "PRIMARY",
          "rows_examined_per_scan": 1,
          "filtered": "100.00"
        }
      }
    ]
  }
}
1 row in set (0.00 sec)
```

```
EXPLAIN ANALYZE
SELECT k.nazwa, z.data, z.wartosc
FROM zamowienia z
JOIN klienci k ON k.id = z.klient_id
WHERE z.status = 'nowe';
```

`EXPLAIN ANALYZE` w MySQL 8, tak jak w PostgreSQL, naprawdę wykonuje zapytanie i dokłada rzeczywisty czas do każdego węzła — dostępne od wersji 8.0.18, wcześniej MySQL miał wyłącznie plan szacowany.

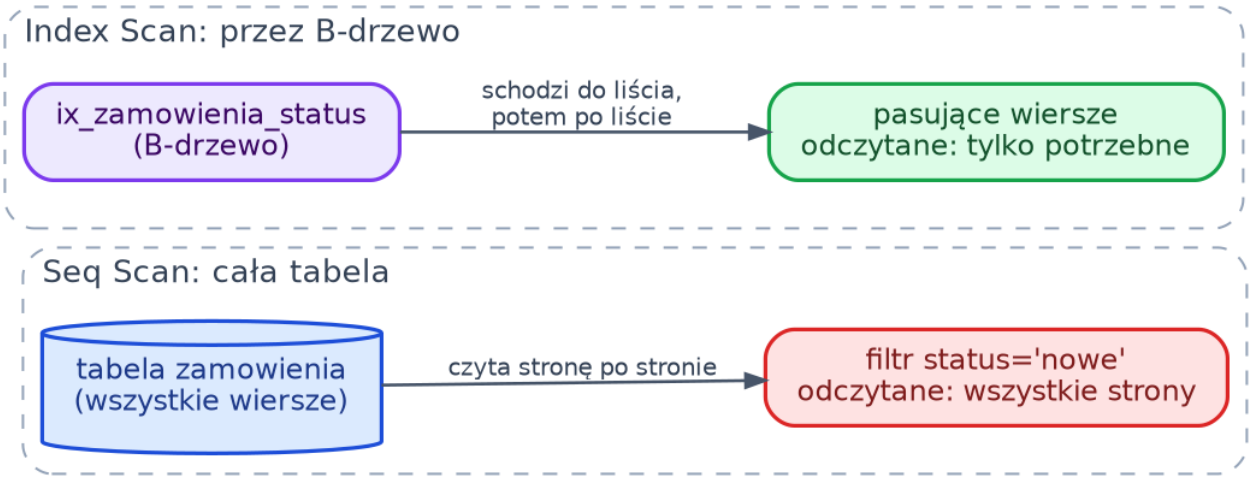
Przykład:

```
-> Nested loop inner join  (cost=9821.40 rows=41200) (actual time=0.244..1842.331
rows=38914 loops=1)
    -> Index lookup on z using ix_zamowienia_status (status='nowe')  (cost=8420.10
rows=41200) (actual time=0.201..1402.884 rows=38914 loops=1)
    -> Single-row index lookup on k using PRIMARY (id=z.klient_id)  (cost=0.25 rows=1)
(actual time=0.008..0.008 rows=1 loops=38914)
1 row in set (1.85 sec)
```

**JSON** = JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami ·**join** = połącz dwa pliki tekstowe po wspólnej kolumnie, jak złączenie tabel w bazie danych

# Czym jest plan wykonania

**Pojęcie: plan wykonania.** Drzewo operacji, które optymalizator wybrał jako sposób obliczenia wyniku konkretnego zapytania — każdy węzeł tego drzewa to jedna operacja (skan tabeli, skan indeksu, złączenie, sortowanie, agregacja), z liśćmi na dole reprezentującymi dostęp do surowych danych i korzeniem na górze reprezentującym ostateczny wynik zwrócony klientowi. Ten sam SQL da inny plan zależnie od tego, jakie indeksy istnieją, ile wierszy mają tabele i jakie statystyki silnik ma na ich temat — plan nie jest właściwością zapytania, tylko decyzją podjętą dla tego zapytania w tym konkretnym stanie bazy, tematem który rozwija strona Statystyki i optymalizator kosztowy.



*Seq Scan czyta całą tabelę stronę po stronie, Index Scan schodzi przez B-drzewo tylko do pasujących wierszy (kliknięcie otwiera pełny rozmiar)*

# Jak czytać plan

Niezależnie od silnika, plan czyta się od najgłębszego wcięcia w stronę najmniejszego — węzeł najbardziej wcięty w tekście wykonuje się fizycznie jako pierwszy, dostarcza dane węzłowi nad sobą, ten kolejnemu wyżej, aż do korzenia na samej górze, który reprezentuje ostateczny wynik. W przykładzie `EXPLAIN ANALYZE` z PostgreSQL wyżej `Bitmap Index Scan` na dole wykonuje się pierwszy — znajduje pozycje w indeksie — `Bitmap Heap Scan` nad nim odczytuje z tych pozycji rzeczywiste wiersze tabeli, a `Hash Join` na samej górze zestawia je z wynikiem skanu `klienci`.

**Koszt szacowany kontra czas rzeczywisty.** Liczba w nawiasie `cost=X..Y` to umowna jednostka kosztu, jaką optymalizator przypisał operacji, zanim cokolwiek wykonał — `X` to koszt do zwrócenia pierwszego wiersza, `Y` to koszt całkowity. `actual time=X..Y` to już rzeczywisty czas w milisekundach, zmierzony podczas wykonania — te dwie liczby nie są bezpośrednio porównywalne jednostkami, koszt to abstrakcyjna miara użyta do wyboru planu, czas to fizyczny pomiar tego, co się faktycznie stało. W przykładzie wyżej `Hash Join` ma koszt całkowity `98234.55` i rzeczywisty czas `1893.221` ms — same liczby nie mówią nic o milisekundach na jednostkę kosztu, liczy się raczej to, który węzeł w drzewie zjada najwięcej czasu rzeczywistego, bo to on jest kandydatem do optymalizacji.

**Rows szacowane kontra rows rzeczywiste.** `rows=41200` w części kosztowej to liczba wierszy, jaką optymalizator się spodziewał na podstawie statystyk, `rows=38914` w części `actual` to liczba wierszy, jaka faktycznie wypadła z tego węzła. W przykładzie wyżej te dwie liczby są bliskie sobie — różnica rzędu kilku procent to normalne zaokrąglenie wynikające z tego, że statystyki są próbką, nie pełnym przeliczeniem. Duża rozbieżność, na przykład szacunek dziesięciu wierszy przy rzeczywistych czterdziestu tysiącach, oznacza, że optymalizator podjął decyzję o planie na podstawie błędnego założenia o rozmiarze wyniku — a plan dobry dla dziesięciu wierszy (na przykład zagnieżdżona pętla) bywa katastrofalny dla czterdziestu tysięcy. Strona Statystyki i optymalizator kosztowy tłumaczy, skąd biorą się takie rozbieżności i jak je naprawić.

**Loops.** `loops=N` mówi, ile razy dany węzeł wykonał się w sumie — węzeł wewnętrzny zagnieżdżonej pętli złączenia (`Nested Loop`) wykonuje się raz na każdy wiersz węzła zewnętrznego, więc jego `actual time` trzeba w głowie pomnożyć przez `loops`, żeby zobaczyć rzeczywisty łączny wkład w czas całego zapytania — pojedyncze wykonanie może wyglądać tanio, a zsumowane po tysiącach powtórzeń okazać się główną przyczyną wolnego zapytania.

# Częste błędy

**Uruchamianie EXPLAIN ANALYZE na modyfikującym zapytaniu bez transakcji.** `EXPLAIN ANALYZE DELETE FROM zamowienia WHERE ...` naprawdę usuwa wiersze — nawyk opakowywania takich sprawdzeń w `BEGIN` / `ROLLBACK`, pokazany w fiszce wyżej, jest tu obowiązkowy, nie opcjonalny.

**Porównywanie liczb kosztu między dwoma różnymi zapytaniami.** Koszt w `EXPLAIN` ma sens tylko jako porównanie planów alternatywnych dla tego samego zapytania w tym samym momencie — nie jest jednostką czasu i nie da się nim porównać dwóch zupełnie różnych zapytań między sobą.

**Czytanie planu od góry do dołu zamiast od dołu do góry.** Kolejność wykonania jest odwrotna do kolejności, w jakiej większość osób czyta tekst — najgłębiej wcięty węzeł wykonuje się pierwszy, korzeń na samej górze ostatni.

# □□ Ćwiczenia

> Zadania sprawdzające.

# Ćwiczenie 1 — czytanie rozbieżności

Plan pokazuje `(cost=0.43..120.00 rows=15 width=48) (actual time=0.088..4820.221 rows=612000 loops=1)`. Co ta rozbieżność mówi o statystykach silnika, i jakie mogą być tego konsekwencje dla wybranego planu?

**Odpowiedź**

Optymalizator spodziewał się piętnastu wierszy, a otrzymał sześćset dwanaście tysięcy — statystyki, na których podstawie oszacował selektywność warunku, są mocno nieaktualne albo błędne. Konsekwencja to plan dobrany pod założenie małego wyniku — na przykład zagnieżdżona pętla albo skan indeksu zamiast skanu sekwencyjnego — który przy takiej liczbie wierszy jest znacznie wolniejszy niż plan, jaki optymalizator wybrałby, znając prawdziwą liczbę od początku.

# Pytania kontrolne

**1. Dlaczego EXPLAIN bez ANALYZE nie pokazuje rzeczywistego czasu wykonania?**

**Odpowiedź**

Bo `EXPLAIN` bez `ANALYZE` nie wykonuje zapytania w ogóle — pyta wyłącznie optymalizator, jaki plan wybrałby i jakie są jego szacunki, bez faktycznego odczytu danych. Rzeczywisty czas wymaga rzeczywistego wykonania, a to robi dopiero `EXPLAIN ANALYZE`.

**2. Który węzeł planu wykonuje się jako pierwszy: ten na samej górze wydruku, czy ten najbardziej wcięty?**

**Odpowiedź**

Najbardziej wcięty. Plan czyta się od dołu i od najgłębszego wcięcia w stronę korzenia na samej górze — węzły niżej dostarczają dane węzłom wyżej, więc muszą wykonać się wcześniej.

**Powiązane:** Statystyki i optymalizator kosztowy · Skan sekwencyjny, skan indeksu, lookup: sposoby dotarcia do wiersza · Czym jest złączenie: iloczyn kartezjański i warunek, który go ogranicza

**Źródła:** dokumentacja PostgreSQL, rozdział „Using EXPLAIN” · Microsoft Docs, „Showplan Text (SHOWPLAN_ALL)” i „SET STATISTICS IO” · Oracle Database SQL Tuning Guide, rozdział „Using EXPLAIN PLAN” · dokumentacja MySQL, rozdział „EXPLAIN Output Format” i „EXPLAIN ANALYZE Statement”

---

Revision #47
Created 2026-09-24 22:42:42 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:44 UTC by Tomasz Konieczny