Kiedy indeks szkodzi

> Każdy indeks, który istnieje na tabeli, płaci swoją cenę przy każdym zapisie niezależnie od tego, czy jakiekolwiek zapytanie z niego korzysta — ta strona pokazuje, jak policzyć tę cenę konkretnie i jak znaleźć indeksy, które ją płacą, nie dając nic w zamian.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
MSSQL	Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima

Fiszka

ZMIENIA SYSTEM

Znalezienie zduplikowanych indeksów w PostgreSQL

```
SELECT indrelid::regclass AS tabela, array_agg(indexrelid::regclass) AS indeksy
FROM pg_index
GROUP BY indrelid, indkey
HAVING COUNT(*) > 1;
```

Grupujemy definicje indeksów po tabeli i po dokładnym zestawie kolumn (`indkey`) — grupa z więcej niż jednym indeksem to zduplikowana definicja.

Przykład:

```
     tabela      |              indeksy
-----------------+-------------------------------------
 zamowienia      | {ix_zamowienia_klient,ix_zam_klient_id_v2}
(1 row)
```

TYLKO ODCZYT

Znalezienie nieużywanych indeksów w PostgreSQL

```
SELECT schemaname, relname AS tabela, indexrelname AS indeks, idx_scan
FROM pg_stat_user_indexes
WHERE idx_scan = 0
ORDER BY pg_relation_size(indexrelid) DESC;
```

`idx_scan` liczy, ile razy od ostatniego restartu statystyk silnik faktycznie sięgnął po dany indeks — zero oznacza, że przez cały ten czas nie zrobił tego ani razu.

Przykład:

```
 schemaname |   tabela   |         indeks          | idx_scan
------------+------------+-------------------------+----------
 public     | zamowienia | ix_zamowienia_uwagi      |        0
 public     | zamowienia | ix_zamowienia_status_old |        0
 public     | produkty   | ix_produkty_opis_pelny   |        0
(3 rows)
```

TYLKO ODCZYT

Znalezienie nieużywanych indeksów w MSSQL

```
SELECT OBJECT_NAME(i.object_id) AS tabela, i.name AS indeks,
       ISNULL(s.user_seeks, 0) + ISNULL(s.user_scans, 0) + ISNULL(s.user_lookups, 0) AS
odczyty,
       ISNULL(s.user_updates, 0) AS zapisy
FROM sys.indexes i
LEFT JOIN sys.dm_db_index_usage_stats s
    ON s.object_id = i.object_id AND s.index_id = i.index_id AND s.database_id = DB_ID()
WHERE i.type_desc = 'NONCLUSTERED'
ORDER BY odczyty ASC;
```

Złączenie `sys.indexes` z `sys.dm_db_index_usage_stats` przez `LEFT JOIN` łapie też indeksy, które w ogóle nie mają wpisu w widoku statystyk — czyli takie, po które nikt nigdy nie sięgnął od ostatniego restartu usługi.

Przykład:

```
tabela          indeks                        odczyty     zapisy
--------------- ----------------------------- ----------- -----------
Zamowienia       IX_Zamowienia_UwagiTekst     0           184213
Zamowienia       IX_Zamowienia_StatusStary    0           184213
Klienci          IX_Klienci_Fax               0           2104

(3 rows affected)
```

Indeks z zerem odczytów i dziesiątkami tysięcy zapisów to najgorszy możliwy przypadek — czysty koszt bez jakiejkolwiek korzyści.

MSSQL = Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima · **ASC** = Automatic Self-Calibration: automatyczna samokalibracja czujnika CO2, która zakłada, że co tydzień czujnik zobaczy czyste powietrze o stężeniu około 400 ppm

ZMIENIA SYSTEM

Włączenie monitorowania użycia indeksu w Oracle

```
ALTER INDEX NazwaIndeksu MONITORING USAGE;
```

Oracle nie liczy użycia indeksów domyślnie — trzeba włączyć monitorowanie osobno dla każdego indeksu, poczekać reprezentatywny okres, a potem sprawdzić wynik w `v$object_usage`.

Przykład:

```
Index altered.
```

TYLKO ODCZYT

Sprawdzenie nieużywanych indeksów w Oracle: v$object_usage

```
SELECT index_name, table_name, used, start_monitoring
FROM v$object_usage
WHERE used = 'NO';
```

Przykład:

```
INDEX_NAME              TABLE_NAME   USED  START_MONITORING
----------------------- ------------ ----- -------------------
IX_ZAMOWIENIA_UWAGI     ZAMOWIENIA   NO    05/12/2026 09:14:02

1 row selected.
```

◫◫ OSTROŻNIE · NIEODWRACALNE

Usunięcie indeksu przed importem masowym: DROP INDEX

```
DROP INDEX NazwaIndeksu;
```

Usuwamy indeks tuż przed rozpoczęciem masowego ładowania danych do tabeli.

Przykład:

```
DROP INDEX
```

◫◫ ZMIENIA SYSTEM

Odtworzenie indeksu po zakończeniu importu: CREATE INDEX

```
CREATE INDEX NazwaIndeksu ON NazwaTabeli (NazwaKolumny);
```

Po zakończonym imporcie odtwarzamy indeks jednym przejściem po już załadowanych danych — silnik czyta całą tabelę raz i buduje strukturę indeksu od podstaw, co jest tańsze niż aktualizowanie jej milion razy z rzędu przy każdym pojedynczym wierszu importu.

Przykład:

```
CREATE INDEX
```

Ta sama logika stoi za tym, dlaczego narzędzia do masowego ładowania danych — `COPY` w PostgreSQL, `BULK INSERT` w MSSQL, `SQL*Loader` w Oracle — dokumentują wprost zalecenie usunięcia indeksów przed dużym importem i odbudowania ich po, zamiast trzymania ich aktywnych przez cały czas trwania operacji.

◫◫ **MSSQL** = Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima · **SQL** = Structured Query Language: język zapytań do relacyjnych baz danych

Koszt utrzymania przy zapisach

Strona Czym jest indeks: i dlaczego to nie jest „przyspieszacz tabeli” wprowadza zasadę, że `INSERT`, `UPDATE` kolumny indeksowanej i `DELETE` muszą zaktualizować każdy indeks założony na tabeli, nie tylko samą tabelę. Policzmy to na konkretnej tabeli: `zamowienia` z kluczem głównym na `id` i czterema dodatkowymi indeksami — na `klient_id`, na `status`, na `data` i indeksem złożonym `(klient_id, status)`.

Jeden `INSERT` do takiej tabeli oznacza pięć osobnych zapisów: jeden wiersz w samej tabeli plus po jednym wpisie w każdym z czterech indeksów, plus wpis w strukturze klucza głównego, który też jest indeksem. Sześć zapisów fizycznych zamiast jednego. Poniższa tabela pokazuje, jak to przekłada się na przepustowość masowego ładowania danych — liczby są przykładowe, ale rząd wielkości odpowiada temu, co widać w praktyce na tabelach z podobną liczbą indeksów.

Liczba indeksów na tabeli (bez PK)	Przepustowość INSERT (wierszy/s, przykładowy test)
0	52 000
1	34 000
2	24 000
4	13 500
6	8 900

Spadek nie jest liniowy tylko dlatego, że każdy kolejny indeks dokłada tę samą stałą pracę — dokłada też rywalizację o bufory pamięci podręcznej, bo silnik musi trzymać w pamięci strony wielu różnych struktur naraz zamiast jednej. Przy operacjach masowych, opisanych niżej, ten koszt bywa na tyle duży, że warto go świadomie ominąć.

Indeksy zduplikowane i nadmiarowe

> **Pojęcie: indeks nadmiarowy.** Indeks, którego zakres zastosowań w całości pokrywa inny, już istniejący indeks na tej samej tabeli — istnieje, zajmuje miejsce i płaci koszt zapisu, ale żadne zapytanie nie potrzebuje go osobno, bo szerszy indeks robi dokładnie to samo i więcej. Wynika to wprost z zasady najbardziej lewego prefiksu opisanej na stronie Indeks złożony, pokrywający, częściowy, funkcyjny: indeks `(klient_id)` jest nadmiarowy, jeśli na tej samej tabeli istnieje już indeks `(klient_id, status)`, bo każdy warunek, który skorzystałby z samego `(klient_id)`, równie dobrze skorzysta z prefiksu `(klient_id, status)`.

> **Pojęcie: indeks zduplikowany.** Dwa indeksy zbudowane na dokładnie tych samych kolumnach, w tej samej kolejności — najczęściej powstaje przez pomyłkę, kiedy ktoś zakłada nowy indeks, nie sprawdziwszy, że taki sam już istnieje, tylko pod inną nazwą.

Indeksy nigdy nieużywane

> **Pojęcie: indeks nieużywany.** Indeks, który silnik nigdy albo prawie nigdy nie wybiera do wykonania żadnego zapytania — jego istnienie kosztuje tyle samo, ile kosztowałby przydatny indeks, ale nie przynosi żadnej korzyści przy odczycie, więc bilans wychodzi wyłącznie na minus.

Indeks na małej tabeli

Tabela `pracownicy` licząca kilkadziesiąt wierszy zmieści się w jednej albo dwóch stronach pamięci — skan sekwencyjny całej tabeli to jeden odczyt, porównywalny albo tańszy niż zejście po B-drzewie indeksu, które i tak musiałoby najpierw odczytać swoje własne strony, a potem doskoczyć do tabeli. Optymalizator kosztowy, opisany na stronie Statystyki i optymalizator kosztowy, rutynowo ignoruje istniejący indeks na takiej tabeli i wybiera skan sekwencyjny — nie dlatego, że coś jest nie tak, tylko dlatego, że przy garstce wierszy dodatkowa struktura nie ma się czym opłacić.

Indeks na kolumnie o niskiej selektywności

Strona Czym jest indeks: i dlaczego to nie jest „przyspieszacz tabeli” wprowadza selektywność jako miarę tego, ile wierszy przeciętnie odpowiada jednej wartości kolumny. Indeks na kolumnie `status` w tabeli `zamowienia`, gdzie cztery wartości status dzielą między siebie cztery miliony wierszy niemal po równo, nie tylko nie pomaga typowemu zapytaniu filtrującemu po jednym statusie — dodatkowo płaci pełny koszt zapisu opisany na początku tej strony za każdy `INSERT` i za każdą zmianę statusu, bez żadnej korzyści w zamian, bo optymalizator i tak go pomija.

Rozdęcie indeksu przy ciągłych aktualizacjach

Częste `UPDATE` i `DELETE` na tabeli z indeksami zostawiają w strukturze indeksu miejsca po usuniętych albo przeniesionych wpisach, których silnik nie zawsze odzyskuje na bieżąco — indeks rośnie fizycznie szybciej niż rośnie liczba aktywnych wpisów, które faktycznie reprezentuje. To zjawisko, nazywane rozdęciem indeksu, ma osobną, pełną stronę pod tytułem Fragmentacja i przebudowa indeksów — tutaj wystarczy zapamiętać, że jest to kolejny koszt strony zapisu, doliczający się do tych policzonych wyżej z biegiem czasu, nie tylko w chwili wstawienia wiersza.

Świadome usunięcie indeksu przed ładowaniem masowym

Import dużej liczby wierszy jednym zadaniem — na przykład ładowanie archiwum zamówień z poprzedniego roku — to sytuacja, w której koszt utrzymania indeksów przy każdym pojedynczym wierszu przestaje się opłacać. Taniej wychodzi usunąć indeksy, wykonać import bez nich, a potem odbudować je raz, jednym przejściem po już załadowanych danych, niż aktualizować je wiersz po wierszu przez cały czas trwania importu.

Częste błędy

Zakładanie indeksu identycznego z już istniejącym pod inną nazwą. Bez systematycznego przeglądu, jak w fiszce wyżej, taki duplikat potrafi istnieć latami, płacąc podwójny koszt zapisu za identyczną, zduplikowaną korzyść przy odczycie.

Traktowanie zerowego idx_scan jako natychmiastowego wyroku. Indeks wspierający rzadką, ale krytyczną operację — na przykład raport miesięczny uruchamiany raz w miesiącu — może mieć zero użyć w oknie obserwacji krótszym niż jego rzeczywisty cykl użycia. Zanim usuniemy indeks na podstawie statystyk, sprawdzamy, od kiedy te statystyki są liczone, i czy okres obserwacji obejmuje pełen cykl pracy systemu.

Usuwanie indeksu przed importem bez odtworzenia go po. Import zakończony, a indeksy nieodtworzone, to tabela, która nagle przestaje wspierać zapytania, które wcześniej działały szybko — częsty błąd operacyjny, kiedy krok odtworzenia zostaje przypadkiem pominięty w skrypcie albo w procedurze ręcznej.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — czy ten indeks jest nadmiarowy

Tabela `zamowienia` ma indeks `(klient_id, status, data)` oraz osobny indeks `(klient_id)`. Czy ten drugi jest nadmiarowy, i jakim zapytaniem dałoby się to sprawdzić w PostgreSQL?

Odpowiedź

Tak, indeks `(klient_id)` jest nadmiarowy — każde zapytanie filtrujące po samym `klient_id` skorzysta równie dobrze z lewego prefiksu indeksu `(klient_id, status, data)`, więc osobny, węższy indeks nie daje żadnej dodatkowej korzyści, a płaci pełny koszt zapisu. Sprawdzenie tego w PostgreSQL to zapytanie z fiszki „Znalezienie zduplikowanych indeksów", rozszerzone o porównanie, czy kolumny jednego indeksu są prefiksem kolumn drugiego, nie tylko o identyczność pełnego zestawu kolumn.

Pytania kontrolne

1. Dlaczego indeks na tabeli liczącej kilkadziesiąt wierszy zwykle nie pomaga, mimo że sam indeks jest zbudowany poprawnie?

Odpowiedź

Skan sekwencyjny malej tabeli to jeden albo dwa odczyty stron, bo cała tabela mieści się w garstce stron pamięci. Zejście po B-drzewie indeksu wymaga własnych odczytów, a potem dodatkowego doskoku do tabeli — przy tak małej liczbie stron te dwa podejścia kosztują podobnie albo indeks kosztuje więcej, więc optymalizator wybiera prostszy skan sekwencyjny.

2. Dlaczego usuwamy indeksy przed masowym importem, skoro w końcu i tak trzeba je odtworzyć?

Odpowiedź

Bo koszt jednorazowego zbudowania indeksu od zera po zakończeniu importu jest niższy niż koszt aktualizowania tego samego indeksu osobno przy każdym z milionów pojedynczych wierszy w trakcie importu. Odbudowa indeksu to jedno przejście po już gotowych danych, podczas gdy utrzymywanie go aktywnym przez cały import to miliony osobnych, drobnych operacji zapisu na strukturze drzewa.

Powiązane: Czym jest indeks: i dlaczego to nie jest „przyspieszacz tabeli” · Indeks złożony, pokrywający, częściowy, funkcyjny · Fragmentacja i przebudowa indeksów · Statystyki i optymalizator kosztowy

Źródła: dokumentacja PostgreSQL, rozdział „The Statistics Collector” i „Populating a Database” · Microsoft Docs, „sys.dm_db_index_usage_stats” · Oracle Database Reference, „V$OBJECT_USAGE”

Revision #46
Created 2026-09-24 22:42:40 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:42 UTC by Tomasz Konieczny