Fragmentacja i przebudowa indeksów

> B-drzewo opisane na stronie B-drzewo od środka zakłada porządek — strony ułożone logicznie tak, jak fizycznie leżą na dysku, i wypełnione danymi bez zbędnych dziur — a ciągłe aktualizacje i usunięcia ten porządek z czasem psują. Ta strona pokazuje, jak zmierzyć, ile porządku zostało, i jak go przywrócić.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
DDL	Data Definition Language: polecenia SQL zmieniające strukturę (CREATE, ALTER, DROP)
MSSQL	Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima

Fiszka

TYLKO ODCZYT

Pomiar fragmentacji w MSSQL: sys.dm_db_index_physical_stats

```
SELECT index_id, index_type_desc, avg_fragmentation_in_percent, page_count
FROM sys.dm_db_index_physical_stats(DB_ID(), OBJECT_ID('dbo.Zamowienia'), NULL, NULL,
'DETAILED');
```

Ostatni parametr `'DETAILED'` prosi o pełne, dokładne przejście po wszystkich stronach zamiast szybkiego oszacowania — dokładniejsze, ale wolniejsze na dużych tabelach.

Przykład:

```
index_id  index_type_desc   avg_fragmentation_in_percent  page_count
--------- ---------------- ----------------------------- -----------
```

```
1          CLUSTERED          42.18                          128440
2          NONCLUSTERED       6.94                           28102
3          NONCLUSTERED       67.31                          9840

(3 rows affected)
```

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

TYLKO ODCZYT

Pomiar fragmentacji w PostgreSQL: pgstattuple

```
SELECT * FROM pgstattuple('zamowienia');
```

Rozszerzenie `pgstattuple` (włączane raz przez `CREATE EXTENSION pgstattuple;`) przechodzi po tabeli i liczy dokładny odsetek martwej przestrzeni, zamiast szacować ją pośrednio ze statystyk.

Przykład:

```
 table_len | tuple_count | tuple_len | tuple_percent | dead_tuple_count | dead_tuple_len
| dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+-----------------+--------------------+------------+---------------
 682065920 |     3980102 | 512340000 |          75.11 |            210488 |
26980000 |                3.96 |   72150000 |         10.58
(1 row)
```

TYLKO ODCZYT

Pomiar fragmentacji indeksu w PostgreSQL: pgstatindex

```
SELECT avg_leaf_density, leaf_fragmentation FROM pgstatindex('ix_zamowienia_klient');
```

Ten sam pakiet ma osobną funkcję `pgstatindex` dla samego indeksu.

Przykład:

```
 avg_leaf_density | leaf_fragmentation
------------------+--------------------
             68.42 |              31.06
(1 row)
```

`avg_leaf_density` poniżej stu procent pokazuje fragmentację wewnętrzną — im niższa wartość, tym więcej pustego miejsca w liściach. `leaf_fragmentation` pokazuje fragmentację zewnętrzną — odsetek przypadków, w których fizyczne położenie kolejnej strony liścia nie zgadza się z jej logicznym miejscem w liście.

TYLKO ODCZYT

Zebranie danych o fragmentacji indeksu w Oracle: ANALYZE INDEX VALIDATE STRUCTURE

```
ANALYZE INDEX NazwaIndeksu VALIDATE STRUCTURE;
```

`ANALYZE INDEX ... VALIDATE STRUCTURE` zapisuje szczegółowe dane o indeksie do tymczasowego widoku `INDEX_STATS`, ważnego do następnego takiego wywołania na innym indeksie w tej samej sesji.

TYLKO ODCZYT

Odczytanie zebranych danych o fragmentacji: INDEX_STATS

```
SELECT name, lf_rows, del_lf_rows, pct_used
FROM index_stats
WHERE name = 'NazwaIndeksu';
```

Przykład:

```
NAME                    LF_ROWS  DEL_LF_ROWS  PCT_USED
----------------------- -------- ------------ ----------
IX_ZAMOWIENIA_KLIENT    3980102       412088         71


1 row selected.
```

`DEL_LF_ROWS` to liczba wpisów logicznie usuniętych, ale fizycznie wciąż zajmujących miejsce w liściach indeksu — `412088` usuniętych wpisów na prawie cztery miliony to wyraźny sygnał do

przebudowy.

⚠ ZMIENIA SYSTEM

Przebudowa indeksu od podstaw w MSSQL: ALTER INDEX REBUILD

```
ALTER INDEX NazwaIndeksu ON NazwaSchematu.NazwaTabeli REBUILD;
```

> **Pojęcie: REBUILD.** Zbudowanie indeksu od nowa, od podstaw — silnik odrzuca starą strukturę i tworzy nową, ciągłą fizycznie i pozbawioną fragmentacji, z aktualnymi statystykami przy okazji.

Przykład:

```
Commands completed successfully.
```

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

⚠ ZMIENIA SYSTEM

Uporządkowanie indeksu bez przebudowy w MSSQL: ALTER INDEX REORGANIZE

```
ALTER INDEX NazwaIndeksu ON NazwaSchematu.NazwaTabeli REORGANIZE;
```

> **Pojęcie: REORGANIZE.** Uporządkowanie istniejących liści indeksu bez budowania go od zera — silnik defragmentuje stopniowo, stronami, w kolejności logicznej listy liści.

Przykład:

```
Commands completed successfully.
```

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

⚠ ZMIENIA SYSTEM

Przebudowa indeksu w PostgreSQL: REINDEX

```
REINDEX INDEX NazwaIndeksu;
```

`REINDEX` buduje indeks od nowa, tak jak `REBUILD` w MSSQL — ale blokuje zapisy do tabeli na cały czas trwania operacji, bo domyślnie działa wewnątrz jednej transakcji.

Przykład:

```
REINDEX
```

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

ZMIENIA SYSTEM

Przebudowa indeksu bez blokowania zapisów: REINDEX CONCURRENTLY

```
REINDEX INDEX CONCURRENTLY NazwaIndeksu;
```

`CONCURRENTLY` buduje nową kopię indeksu obok starej, pozwalając na normalne odczyty i zapisy przez większość czasu trwania operacji, a podmienia starą wersję na nową dopiero na samym końcu, krótką blokadą — kosztem tego, że cała operacja trwa dłużej i wymaga więcej miejsca na dysku w trakcie budowy drugiej kopii.

Przykład:

```
REINDEX
```

ZMIENIA SYSTEM

Przebudowa indeksu w Oracle bez blokowania zapisów: ALTER INDEX REBUILD ONLINE

```
ALTER INDEX NazwaIndeksu REBUILD ONLINE;
```

`ONLINE` pozwala na równoległe operacje `DML` na tabeli podczas przebudowy indeksu — bez tego słowa kluczowego `ALTER INDEX ... REBUILD` blokuje zapisy do tabeli na czas przebudowy, dokładnie jak zwykły `REBUILD` w MSSQL bez opcji online.

Przykład:

```
Index altered.
```

DML = Data Manipulation Language: polecenia SQL zmieniające dane (INSERT, UPDATE, DELETE) · **MSSQL** = Microsoft SQL Server: silnik bazy danych Microsoftu, na nim w Zielonej stoi księgowość Optima

TYLKO ODCZYT

Przebudowa tabeli i jej indeksów w MySQL: OPTIMIZE TABLE

```
OPTIMIZE TABLE NazwaTabeli;
```

Dla tabel InnoDB `OPTIMIZE TABLE` w praktyce przebudowuje tabelę i wszystkie jej indeksy przez odtworzenie ich od zera — MySQL realizuje to mechanizmem online DDL w większości przypadków, ograniczając czas pełnej blokady do samego końca operacji.

Przykład:

```
+--------------------+----------+----------+----------+
| Table              | Op       | Msg_type | Msg_text |
+--------------------+----------+----------+----------+
| sklep.zamowienia   | optimize | status   | OK       |
+--------------------+----------+----------+----------+
1 row in set (4.21 sec)
```

DDL = Data Definition Language: polecenia SQL zmieniające strukturę (CREATE, ALTER, DROP)

ZMIENIA SYSTEM

Obniżenie fillfactora indeksu przy tworzeniu

```
CREATE INDEX NazwaIndeksu ON NazwaTabeli (NazwaKolumny) WITH (fillfactor = 70);
```

Obniżamy fillfactor do siedemdziesięciu procent — trzydzieści procent każdej strony zostaje celowo puste.

Przykład:

```
CREATE INDEX
```

Obniżenie fillfactora ma sens dla indeksów na kolumnach, które są często aktualizowane po tym, jak wiersz już istnieje — tabela `zamowienia`, w której kolumna `status` zmienia się wielokrotnie w cyklu życia zamówienia, korzysta na niższym fillfactorze indeksu obejmującego tę kolumnę, bo kolejne aktualizacje mają gdzie się zmieścić bez wywoływania podziału strony za każdym razem. Dla indeksu na danych, które raz zapisane już się nie zmieniają — na przykład na kolumnie `id` — pełne stopięćdziesięcioprocentowe wypełnienie jest właściwym wyborem, bo nie ma czego rezerwować na przyszłość.

Fragmentacja wewnętrzna i zewnętrzna

Pojęcie: fragmentacja wewnętrzna. Puste miejsce pozostające wewnątrz stron indeksu albo tabeli, powstałe po wierszach usuniętych albo przeniesionych gdzie indziej — strona wciąż zajmuje tyle samo miejsca na dysku i wciąż trzeba ją odczytać w całości, ale przechowuje mniej rzeczywistych danych niż mogłaby. `UPDATE` w silnikach z wielowersyjną kontrolą współbieżności, takich jak PostgreSQL, nie nadpisuje wiersza w miejscu — zapisuje nową wersję wiersza gdzie indziej, a stara wersja zostaje martwym wpisem (dead tuple) do czasu, aż proces porządkujący go usunie. `DELETE` zostawia dokładnie taką samą dziurę.

Pojęcie: fragmentacja zewnętrzna. Rozjazd między logiczną kolejnością stron indeksu (tą, w jakiej łączy je lista liści opisana na stronie B-drzewo od środka) a ich fizycznym położeniem na dysku. Podział strony przy wstawieniu nowej wartości, opisany na tej samej stronie, przydziela nową stronę w dowolnym wolnym miejscu na dysku — logicznie sąsiaduje ona ze starą stroną w liście liści, ale fizycznie może leżeć zupełnie gdzie indziej. Odczyt zakresu wartości, który logicznie jest ciągłym przejściem po kilkunastu sąsiednich liściach, fizycznie staje się skokami po całym pliku danych zamiast jednym uporządkowanym odczytem sekwencyjnym.

Obie postacie fragmentacji rosną z czasem wprost proporcjonalnie do intensywności zapisów na tabeli — tabela z rzadkimi aktualizacjami fragmentuje się powoli, tabela obsługująca tysiące `UPDATE` i `DELETE` na minutę fragmentuje się w ciągu dni czy tygodni na tyle, że warto to zmierzyć.

Przebudowa: REBUILD, REORGANIZE i próg 5/30 procent w MSSQL

Microsoft dokumentuje konkretny próg decyzyjny: przy fragmentacji od pięciu do trzydziestu procent zaleca `REORGANIZE`, powyżej trzydziestu procent — `REBUILD`. Poniżej pięciu procent żadna z dwóch operacji zwykle nie daje wystarczającej korzyści, żeby uzasadnić jej koszt.

Fillfactor

> **Pojęcie: fillfactor.** Ustawienie mówiące silnikowi, jaki procent każdej strony indeksu wypełnić danymi przy jego budowie, zostawiając resztę jako celowo puste miejsce zarezerwowane na przyszłe wstawienia i aktualizacje bez natychmiastowego podziału strony. Domyślna wartość to sto procent — pełne wypełnienie, najlepsze pod kątem rozmiaru indeksu zaraz po zbudowaniu, ale najgorsze pod kątem liczby podziałów stron przy pierwszych kolejnych zapisach.

Uwaga o blokadach

Operacja	Blokuje tabelę na czas trwania
`ALTER INDEX ... REBUILD` (MSSQL, bez ONLINE)	tak, blokuje odczyty i zapisy
`ALTER INDEX ... REBUILD` z opcją online (MSSQL Enterprise)	nie, dopuszcza równoległe odczyty i zapisy
`ALTER INDEX ... REORGANIZE` (MSSQL)	nie, działa stopniowo z krótkimi blokadami pojedynczych stron
`REINDEX` (PostgreSQL)	tak, blokada `ACCESS EXCLUSIVE` na zapisy przez cały czas trwania
`REINDEX ... CONCURRENTLY` (PostgreSQL)	nie, poza krótką blokadą przy końcowej podmianie
`ALTER INDEX ... REBUILD` bez `ONLINE` (Oracle)	tak, blokuje zapisy DML do tabeli
`ALTER INDEX ... REBUILD ONLINE` (Oracle)	nie, dopuszcza równoległe DML
`OPTIMIZE TABLE` (MySQL, InnoDB)	zwykle nie, poza krótką blokadą na samym końcu

Częste błędy

Przebudowa indeksu bez wcześniejszego pomiaru fragmentacji. Przebudowa indeksu, który ma fragmentację na poziomie dwóch procent, kosztuje czas i zasoby, nie dając praktycznie żadnej korzyści — pomiar zawsze poprzedza decyzję, nie odwrotnie.

Wybór REBUILD bez opcji online na produkcyjnej tabeli w godzinach szczytu. Pełna blokada zapisów na czas przebudowy dużego indeksu może trwać od minut do godzin przy tabelach liczonych w milionach wierszy — operacja tej wagi zasługuje na okno serwisowe albo na wariant online, jeśli licencja i wersja silnika na to pozwalają.

Obniżanie fillfactora na wszystkich indeksach bez rozróżnienia. Niższy fillfactor oznacza większy indeks od samego początku i więcej stron do odczytania przy każdym skanie — sensowny tam, gdzie aktualizacje są częste, szkodliwy tam, gdzie dane są w praktyce tylko do odczytu.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — REBUILD czy REORGANIZE

`sys.dm_db_index_physical_stats` pokazuje `avg_fragmentation_in_percent` równe `18.4` dla jednego indeksu i `56.2` dla drugiego na tej samej tabeli w MSSQL. Którą operację wybierzemy dla każdego z nich?

Odpowiedź

Dla indeksu z fragmentacją `18.4` procent, mieszczącą się w przedziale od pięciu do trzydziestu procent, wybieramy `REORGANIZE`. Dla indeksu z fragmentacją `56.2` procent, powyżej progu trzydziestu procent, wybieramy `REBUILD`.

Pytania kontrolne

1. Dlaczego fragmentacja zewnętrzna szkodzi bardziej odczytom zakresowym niż pojedynczym wyszukaniom po kluczu?

Odpowiedź

Wyszukanie po kluczu i tak schodzi po B-drzewie do jednego konkretnego liścia, niezależnie od tego, gdzie fizycznie leżą inne liście. Odczyt zakresu zakłada przejście po kilku kolejnych liściach w kolejności logicznej listy — jeśli te liście leżą fizycznie w przypadkowych miejscach na dysku, zamiast jednego uporządkowanego odczytu sekwencyjnego silnik wykonuje serię rozproszonych, kosztowniejszych odczytów.

2. Dlaczego REINDEX CONCURRENTLY w PostgreSQL trwa dłużej niż zwykły REINDEX, mimo że efekt końcowy jest ten sam?

Odpowiedź

Zwykły `REINDEX` blokuje zapisy na cały czas trwania i może pracować bez dodatkowych zabezpieczeń przed współbieżnymi zmianami. `REINDEX CONCURRENTLY` musi budować nową kopię indeksu, jednocześnie dopuszczając trwające zapisy do tabeli, więc dodatkowo śledzi i uwzględnia zmiany zachodzące w trakcie budowy — ten dodatkowy nadzór kosztuje czas, ale w zamian pozwala uniknąć blokady tabeli.

Powiązane: B-drzewo od środka · Kiedy indeks szkodzi · Statystyki i optymalizator kosztowy

Źródła: dokumentacja PostgreSQL, rozdział „REINDEX” i moduł „pgstattuple” · Microsoft Docs, „Reorganize and Rebuild Indexes” i „sys.dm_db_index_physical_stats” · Oracle Database Administrator's Guide, rozdział „Managing Indexes”

Revision #46
Created 2026-09-24 22:42:46 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:48 UTC by Tomasz Konieczny