B-drzewo od środka

> Prawie każdy indeks, na który trafiamy w codziennej pracy, jest pod spodem tą samą strukturą — zrównoważonym drzewem o dużym stopniu rozgałęzienia, które sprowadza wyszukanie jednego wiersza spośród milionów do kilku odczytów z dysku, zamiast do przeglądania wszystkiego.

Po co to istnieje

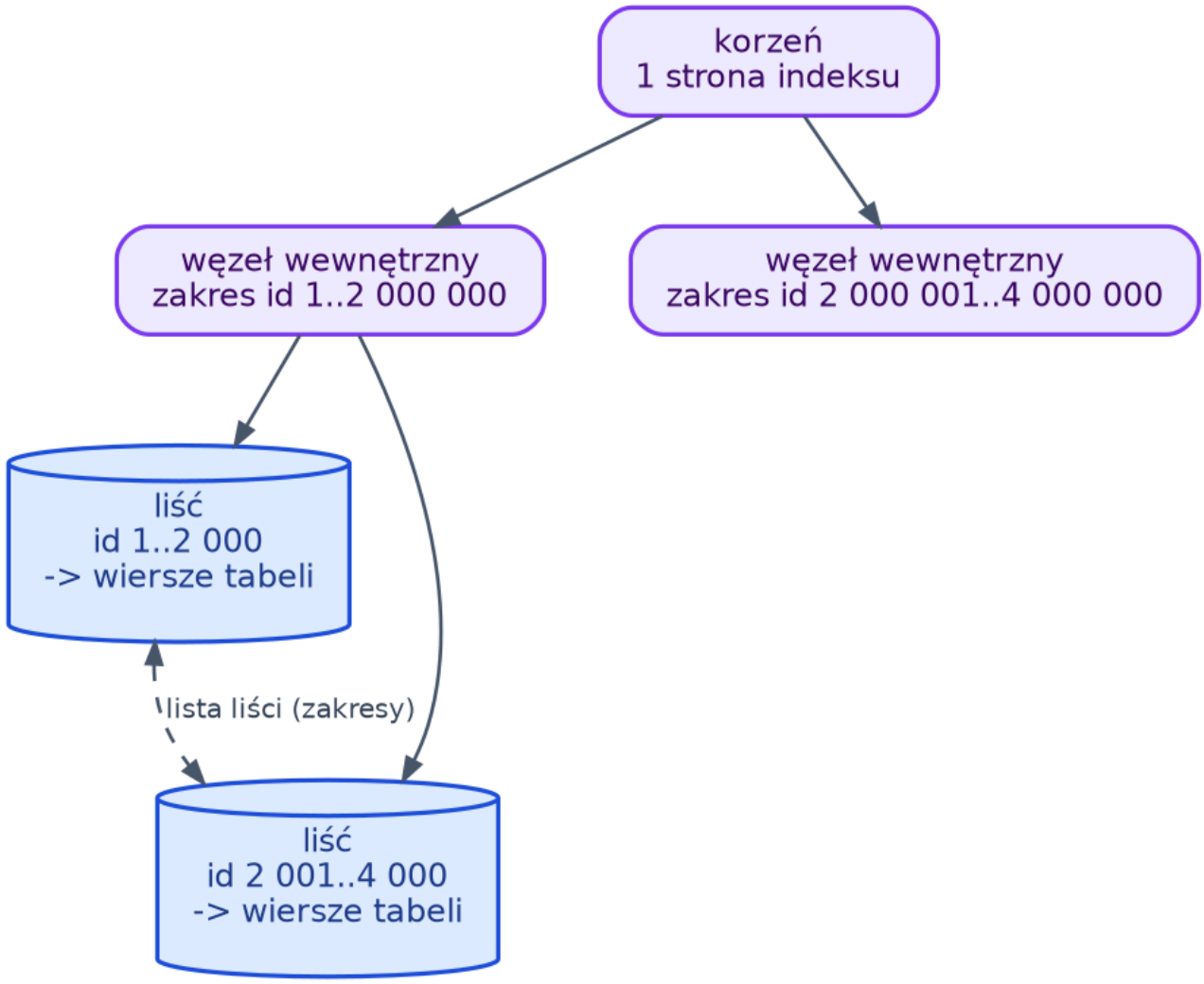
Strona Czym jest indeks: i dlaczego to nie jest „przyspieszacz tabeli” tłumaczy, że indeks to osobna, posortowana struktura ze wskaźnikami do wierszy tabeli. Ta strona odpowiada na pytanie, jak dokładnie ta struktura jest zbudowana wewnątrz — bo od tej budowy zależy, jakie zapytania indeks w ogóle potrafi wesprzeć.

> **Pojęcie: B-drzewo.** Zrównoważone drzewo wyszukiwań, w którym każdy węzeł może mieć dużo więcej niż dwoje dzieci — nie kilka, jak w klasycznym drzewie binarnym z podręcznika algorytmów, tylko zwykle kilkaset. Nazwa pochodzi od angielskiego „balanced tree”; litera B nie oznacza konkretnego słowa w sposób jednoznacznie ustalony przez autorów oryginalnej pracy z lat siedemdziesiątych, ale najczęściej tłumaczy się ją właśnie jako „zrównoważone”. Zrównoważenie znaczy tu, że droga od korzenia do dowolnego liścia ma zawsze tę samą długość — nie ma gałęzi płytszych ani głębszych od innych, niezależnie od tego, ile razy drzewo rosło czy się kurczyło.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: węzeł wewnętrzny.** Strona B-drzewa, która nie przechowuje samych wierszy, tylko klucze graniczne i wskaźniki do węzłów niższego poziomu — mówi, w którą stronę zejść, żeby trafić do właściwego zakresu wartości.

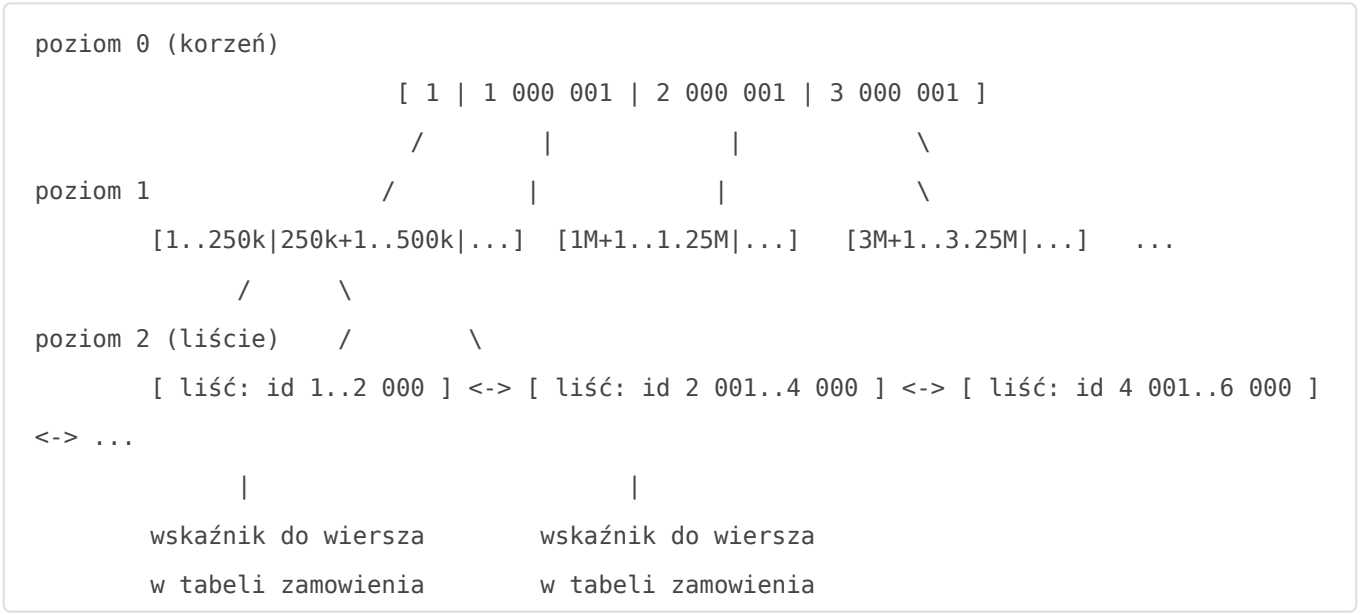


Korzeń wskazuje węzły wewnętrzne, te wskazują liście, a liście są dodatkowo połączone w listę dla zakresów (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: liść.** Najniższy poziom B-drzewa, na którym leżą już właściwe wartości indeksowanej kolumny razem ze wskaźnikiem do wiersza tabeli (albo, w indeksie klastrowym, razem z samym wierszem — o czym mówi tabela porównawcza niżej). Liście B-drzewa są dodatkowo połączone między sobą w listę dwukierunkową, na tym samym poziomie, od lewej do prawej — to jest szczegół, dzięki któremu drzewo obsługuje zakresy, nie tylko pojedyncze wartości.

Struktura ma trzy warstwy funkcjonalne: korzeń na samej górze, jeden lub więcej poziomów węzłów wewnętrznych pod nim, i warstwę liści na samym dole. Szukanie konkretnej wartości

zaczyna się w korzeniu, porównuje szukaną wartość z kluczami granicznymi zapisanymi w węźle i schodzi do dokładnie jednego dziecka — tego, w którego zakres wpada szukana wartość — i tak w dół, poziom po poziomie, aż do liścia.



Liczy się to, ile poziomów trzeba pokonać, żeby dojść od korzenia do liścia — a to zależy od tego, ile wskaźników mieści jedna strona węzła. Strona indeksu w PostgreSQL ma domyślnie 8 KB; klucz typu `bigint` razem ze wskaźnikiem do dziecka zajmuje kilkanaście bajtów, więc jedna strona węzła wewnętrznego mieści rząd dwustu, trzystu takich par. Przy stopniu rozgałęzienia dwustu jedna strona korzenia potrafi rozdzielić ruch na dwieście węzłów niższego poziomu, każdy z nich na kolejne dwieście — dwa poziomy dają już czterdzieści tysięcy gałęzi, trzy poziomy osiem milionów.

Dla tabeli `zamowienia` liczącej cztery miliony wierszy to oznacza, że całe drzewo mieści się w trzech poziomach: korzeń, jedna warstwa węzłów wewnętrznych i warstwa liści. Żeby znaleźć jeden konkretny wiersz po kluczu, silnik odczytuje jedną stronę korzenia, jedną stronę węzła wewnętrznego i jedną stronę liścia — trzy odczyty z indeksu — a potem, jeśli zapytanie potrzebuje kolumn spoza indeksu, jeszcze jeden odczyt samej tabeli. Cztery operacje odczytu zamiast, w najgorszym razie, czterech milionów przy skanie sekwencyjnym opisanym na poprzedniej stronie. W praktyce większość tych odczytów trafia zresztą w pamięć podręczną silnika, a nie na fizyczny dysk, bo górne poziomy drzewa są odwiedzane przy każdym wyszukiwaniu i zostają w buforach — realny koszt bywa więc jeszcze niższy niż te cztery operacje sugerują.

Co się dzieje przy wstawieniu

Nowy wiersz w tabeli oznacza nową wartość do wstawienia do indeksu na właściwej pozycji w odpowiednim liściu, żeby zachować porządek sortowania. Dopóki w liściu jest miejsce,

wstawienie jest tanie — dopisanie jednej pozycji na już istniejącej stronie. Kiedy liść się zapełni, silnik dzieli go na dwa: połowa istniejących wartości zostaje na starej stronie, połowa przechodzi na nowo przydzieloną, a klucz graniczny między nimi trzeba dopisać do rodzica na poziomie wyżej. Ten podział strony (page split) może kaskadowo rozejść się w górę, jeśli rodzic też jest pełny — w skrajnym przypadku aż do korzenia, co powiększa drzewo o jeden poziom. To właśnie ten mechanizm utrzymuje drzewo zrównoważone: podział zawsze zachodzi od dołu, na tym samym poziomie co wstawiana wartość, więc wszystkie liście pozostają na tej samej głębokości.

Co się dzieje przy usunięciu

Usunięcie wartości z liścia zwalnia w nim miejsce; sam wpis znika, a otaczające go wartości nie muszą się przesuwać, bo liść nie musi być fizycznie ciągły w pamięci strony. Przy dużej liczbie usunięć strona może zostać częściowo pusta — niektóre silniki wtedy łączą sąsiednie, prawie puste strony w jedną (merge stron), inne, w tym PostgreSQL, tego automatycznie nie robią i zostawiają odzyskanie miejsca procesowi porządkującemu w tle, czemu poświęcona jest strona Fragmentacja i przebudowa indeksów.

Dlaczego B-drzewo obsługuje równość, zakresy i ORDER BY

Wartości w liściach B-drzewa leżą w pełnym porządku sortowania, a same liście są dodatkowo połączone w listę od najmniejszej wartości do największej. To daje trzy rzeczy naraz z tej samej struktury.

Równość (`WHERE id = 2841903`) to zejście od korzenia do jednego konkretnego liścia — dokładnie ten przypadek, który opisaliśmy wyżej.

Zakres (`WHERE data BETWEEN '2024-01-01' AND '2024-01-31'`) to zejście do liścia zawierającego dolną granicę zakresu, a potem przejście po połączonych liściach w prawo, aż wartość przekroczy górną granicę — bez ponownego schodzenia od korzenia przy każdym kolejnym wierszu. To jest dokładnie mechanizm, na którym opiera się sargowalność opisana na stronie WHERE: filtrowanie wierszy.

`ORDER BY` na kolumnie indeksowanej może w ogóle pominąć osobne sortowanie wyniku — jeśli silnik i tak czyta liście w kolejności listy połączonej, wynik wychodzi posortowany bez dodatkowego kroku, tej samej konstrukcji, która obsługuje zakresy.

Indeks haszujący nie ma żadnej z tych dwóch własności. Buduje z wartości kolumny skrót (hash) i przechowuje wpisy w kubełkach według wartości tego skrótu — dwie bliskie sobie wartości, na przykład `1000` i `1001`, mogą trafić do zupełnie odległych kubełków, bo funkcja haszująca nie zachowuje porządku. Taki indeks świetnie odpowiada na pytanie „czy istnieje wiersz o dokładnie tej wartości”, ale nie ma żadnego sensownego sposobu odpowiedzieć na „wszystkie wartości większe niż X” ani na „wypisz posortowane po tej kolumnie” — nie ma tam żadnego porządku do przejścia.

Mapa: B-drzewo i inne struktury indeksów

Struktura	Co obsługuje dobrze	Czego nie obsługuje
B-drzewo	równość, zakresy, `ORDER BY`, `LIKE 'prefiks%'`	dopasowania niesargowalne, pełnotekstowe wyszukiwanie
Hash	tylko równość (`=`)	zakresy, sortowanie, `<` / `>`
GiST (PostgreSQL)	dane geometryczne, zakresy wielowymiarowe, wyszukiwanie „najbliższego sąsiada”	zwykłą równość obsługuje gorzej niż B-drzewo
GIN (PostgreSQL)	wartości złożone z wielu elementów — tablice, `jsonb`, pełnotekstowe wyszukiwanie	pojedyncze proste wartości skalarne, gdzie B-drzewo jest tańsze
Bitmapowy	kolumny o niskiej selektywności, łączenie kilku warunków bitową operacją AND/OR	częste zapisy — aktualizacja bitmapy jest kosztowna
Klastrowy	odczyt zakresu bez dodatkowego doskoku do tabeli, bo liść to sam wiersz	tylko jeden na tabelę, bo fizycznie determinuje kolejność danych

Ten podział jest tu tylko mapą orientacyjną — hash, bitmapowy i klastrowy wracają szerzej na stronach Indeks złożony, pokrywający, częściowy, funkcyjny i Skan sekwencyjny, skan indeksu, lookup: sposoby dotarcia do wiersza, bo każdy z nich odpowiada innemu pytaniu postawionemu przez zapytanie, a nie innemu stopniowi „szybkości”.

Częste błędy

Oczekiwanie, że B-drzewo pomoże przy `LIKE '%tekst'`. Wiodący znak wieloznaczny psuje porządek sortowania, na którym opiera się cała nawigacja po drzewie — silnik nie potrafi wskazać jednego miejsca w posortowanych wartościach, od którego zacząć, bo pasujące wartości mogą zaczynać się od czegokolwiek. Strona WHERE: filtrowanie wierszy opisuje to dokładniej.

Traktowanie wysokości drzewa jako stałej niezależnie od rozmiaru tabeli. Trzy poziomy dla czterech milionów wierszy to konkretna liczba dla konkretnego rozmiaru i konkretnego stopnia rozgałęzienia — tabela stukrotnie większa nie potrzebuje stukrotnie więcej poziomów, bo rozgałęzienie rośnie wykładniczo z każdym kolejnym poziomem, ale w pewnym momencie i tak dokłada kolejny poziom, co widać przy bardzo dużych tabelach jako niewielki, skokowy wzrost kosztu wyszukiwania.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — liczenie poziomów

Przy stopniu rozgałęzienia dwustu na poziom, ile poziomów potrzebuje B-drzewo, żeby pomieścić tabelę o ośmiu milionach wierszy, i ile odczytów z indeksu wymaga wtedy jedno wyszukanie po kluczu?

Odpowiedź

Dwieście do potęgi drugiej to czterdzieści tysięcy, dwieście do potęgi trzeciej to osiem milionów — dokładnie tyle, ile ma tabela. Trzy poziomy wystarczają: korzeń, jeden poziom węzłów wewnętrznych, liście. Jedno wyszukanie po kluczu wymaga trzech odczytów stron indeksu — po jednym na każdy poziom — plus jednego dodatkowego odczytu samej tabeli, jeśli zapytanie potrzebuje kolumn spoza indeksu.

Pytania kontrolne

1. Dlaczego liście B-drzewa są połączone w listę, skoro do samego zejścia od korzenia do liścia to połączenie nie jest potrzebne?

Odpowiedź

Połączenie liści w listę pozwala obsłużyć zakresy bez ponownego schodzenia od korzenia dla każdej kolejnej wartości. Silnik schodzi raz do liścia zawierającego dolną granicę zakresu, a potem przesuwa się po połączonych liściach w prawo, dopóki wartości nie przekroczą górnej granicy — dużo tańsze niż powtarzanie pełnego wyszukiwania od korzenia dla każdego kolejnego wiersza w zakresie.

2. Dlaczego indeks haszujący nie potrafi wesprzeć `ORDER BY`, a B-drzewo tak?

Odpowiedź

Funkcja haszująca celowo rozprasza wartości po kubełkach bez zachowania ich wzajemnego porządku — dwie bliskie sobie wartości mogą trafić do zupełnie odległych kubełków. B-drzewo przechowuje wartości w pełnym porządku sortowania na poziomie liści, więc przejście po liściach od lewej do prawej daje wynik posortowany bez dodatkowego kroku.

Powiązane: Czym jest indeks: i dlaczego to nie jest „przyspieszacz tabeli” · Indeks złożony, pokrywający, częściowy, funkcyjny · Skan sekwencyjny, skan indeksu, lookup: sposoby dotarcia do wiersza · Fragmentacja i przebudowa indeksów

Źródła: dokumentacja PostgreSQL, rozdział „Index Types” i „B-Tree Indexes” · Microsoft Docs, „Clustered and Nonclustered Indexes Described” · Oracle Database Concepts, rozdział „B-Tree Indexes”

Revision #46
Created 2026-09-24 22:42:36 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:39 UTC by Tomasz Konieczny