Czym jest indeks: i dlaczego to nie jest „przyspieszacz tabeli”

> Bez indeksu znalezienie jednego zamówienia w tabeli liczącej cztery miliony wierszy wymaga przeczytania wszystkich czterech milionów — ta strona tłumaczy, jaką osobną strukturę budujemy, żeby tego uniknąć, i jaką cenę za nią płacimy przy każdym zapisie.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
SSD	Solid State Drive: dysk półprzewodnikowy

Fiszka

ZMIENIA SYSTEM

Utworzenie indeksu na kolumnie: CREATE INDEX

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

Zakładamy indeks na jednej kolumnie tabeli. Silnik czyta całą tabelę raz, buduje z wartości tej kolumny osobną, posortowaną strukturę ze wskaźnikami z powrotem do wierszy i zapisuje ją na dysku obok tabeli — samych danych w tabeli nie rusza.

Przykład:

```
CREATE INDEX
```

◫ OSTROŻNIE · NIEODWRACALNE

Usunięcie indeksu: DROP INDEX

```
DROP INDEX NazwaIndeksu;
```

Kasujemy strukturę indeksu bez dotykania danych w tabeli. Wiersze zostają dokładnie takie same, znika tylko skrót, który pozwalał do nich dotrzeć bez czytania wszystkiego.

Przykład:

```
DROP INDEX
```

Po co to istnieje

Zakładamy, że tabela `zamowienia` ma cztery miliony wierszy i chcemy znaleźć to jedno, o `id = 2841903`. Bez żadnej dodatkowej struktury silnik nie ma innego sposobu niż przejść po tabeli wiersz po wierszu i sprawdzić warunek dla każdego z osobna — taki przebieg nazywamy skanem sekwencyjnym, dokładniej opisanym na stronie Skan sekwencyjny, skan indeksu, lookup: sposoby dotarcia do wiersza.

> **Pojęcie: skan sekwencyjny (pełne przejście po tabeli).** Sposób odczytu, w którym silnik czyta kolejno każdą stronę danych tabeli od początku do końca i dla każdego wiersza sprawdza warunek z osobna, niezależnie od tego, ile wierszy ostatecznie spełni ten warunek. Koszt takiego przejścia rośnie liniowo z liczbą wierszy tabeli — dla czterech milionów wierszy silnik wykonuje pracę proporcjonalną do czterech milionów sprawdzeń, obojętnie czy szukany wiersz jest jeden, czy jest ich sto tysięcy.

Szukanie po `id` w tabeli bez indeksu, przy założeniu, że jedna strona dysku (8 KB w PostgreSQL) mieści około pięćdziesięciu wierszy tej szerokości, oznacza odczyt rzędu osiemdziesięciu tysięcy stron — nawet z dysku SSD to zauważalny czas, a przy obciążonym serwerze i współzawodnictwie o pamięć podręczną znacznie więcej niż „zauważalny”.

> **Pojęcie: indeks.** Osobna struktura danych, przechowywana obok tabeli, która trzyma posortowane wartości jednej lub kilku kolumn razem z odsyłaczem do miejsca, gdzie leży pełny wiersz w tabeli. Indeks nie jest kopią tabeli ani jej przyspieszoną wersją — jest dodatkowym, mniejszym zestawem danych zbudowanym specjalnie po to, żeby dla danej wartości kolumny odnaleźć właściwy wiersz bez przeglądania reszty. Struktura, w jakiej

> najczęściej trzymamy te posortowane wartości, to B-drzewo, opisane osobno na stronie B-drzewo od środka.

Zamiast czytać cztery miliony wierszy tabeli, silnik z indeksem na kolumnie `id` czyta kilka stron samego indeksu — bo indeks jest posortowany, może zejść od korzenia do liścia w kilku krokach, zamiast sprawdzać każdą wartość po kolei — a dopiero znalezione miejsce w indeksie każe mu odczytać dokładnie ten jeden wiersz z tabeli. To jest właściwy sens słowa „indeks”: nie przyspieszenie samej tabeli, tylko skrót obok niej, z którego silnik korzysta zamiast czytać tabelę w całości.

Indeks nie jest bezpłatny. Zajmuje miejsce na dysku, osobne od miejsca zajmowanego przez tabelę — czasem porównywalne z rozmiarem samej tabeli, czasem większe, jeśli indeksowana kolumna jest szeroka albo indeksów na tabeli jest kilka. Ważniejsza cena płaci się przy każdej zmianie danych: `INSERT` dopisujący wiersz do tabeli musi też dopisać nową wartość do każdego indeksu założonego na tej tabeli, `UPDATE` zmieniający wartość indeksowanej kolumny musi usunąć starą pozycję z indeksu i wstawić nową, `DELETE` musi usunąć wpis z indeksu tak samo jak wiersz z tabeli. Tabela z pięcioma indeksami płaci za każdy pojedynczy `INSERT` sześciokrotnie — raz za sam wiersz i pięć razy za aktualizację każdego indeksu z osobna. Strona Kiedy indeks szkodzi rozwija ten koszt na konkretnych liczbach.

Analogia

Książka bez skorowidza zmusza do przeczytania jej od deski do deski, żeby znaleźć jedno pojęcie — dokładnie tak jak skan sekwencyjny zmusza silnik do przejścia po całej tabeli. Skorowidz na końcu książki to osobne, dodatkowe strony: hasła w porządku alfabetycznym, każde z numerem strony, na której hasło się pojawia. Skorowidz nie zawiera treści rozdziałów — zawiera tylko tyle, ile trzeba, żeby odesłać czytelnika we właściwe miejsce.

Ten skorowidz zajmuje miejsce w książce — kilka dodatkowych stron doliczonych do objętości. I wymaga utrzymania: jeśli redaktor dopisze akapit na stronie 120, który przesunie całą resztę tekstu o jedną stronę, każdy numer strony w skorowidzu, który wskazywał na coś po stronie 120, trzeba poprawić. Książka bez żadnych późniejszych zmian nie płaci tej ceny nigdy więcej niż raz, przy pierwszym złożeniu skorowidza. Tabela w bazie danych zmienia się bez przerwy — i dlatego indeks płaci tę cenę utrzymania przy każdym pojedynczym zapisie, nie raz na zawsze.

Jak to działa

Rozwiń: jak to działa pod spodem

Nie każda kolumna zyskuje na indeksie w tym samym stopniu — decyduje o tym to, jak bardzo dana wartość kolumny wyodrębnia mały podzbiór wierszy z całej tabeli.

> **Pojęcie: selektywność.** Stosunek liczby różnych wartości, jakie przyjmuje kolumna, do liczby wierszy tabeli — im wyższa selektywność, tym mniejszy odsetek wierszy przypada przeciętnie na jedną wartość, i tym więcej silnik zyskuje, mogąc odnaleźć tylko te wiersze zamiast przeglądać wszystkie. Kolumna `id` w tabeli `zamowienia` ma selektywność maksymalną — każda wartość jest unikalna, jeden identyfikator wskazuje dokładnie jeden wiersz spośród czterech milionów. Kolumna `status` w tej samej tabeli, przyjmująca w praktyce cztery czy pięć wartości (`nowe`, `w realizacji`, `zrealizowane`, `anulowane`), ma selektywność dużo niższą: warunek `status = 'zrealizowane'` może zwrócić milion wierszy z czterech milionów, jedną czwartą całej tabeli.

Indeks na kolumnie o dwóch możliwych wartościach — powiedzmy hipotetycznej kolumnie `czy_zaplacone` typu logicznego — zwykle nie ma sensu, i to nie dlatego, że taki indeks nie działałby, tylko dlatego, że nie pomaga. Warunek `czy_zaplacone = true` w tabeli czterech milionów zamówień, przy mniej więcej równym rozkładzie obu wartości, wskaże około dwóch milionów wierszy. Przejście przez indeks, znalezienie w nim dwóch milionów wpisów, a potem doskoczenie po każdym z nich osobno do wiersza w tabeli, jest wolniejsze niż zwykłe przeczytanie tabeli od początku do końca skanem sekwencyjnym — bo skan sekwencyjny czyta dane w kolejności, w jakiej leżą fizycznie na dysku, a doskakiwanie do dwóch milionów rozproszonych miejsc w tabeli to dwa miliony osobnych, kosztowniejszych odczytów.

Optymalizator kosztowy, opisany na stronie Statystyki i optymalizator kosztowy, właśnie to liczy przy wyborze planu — i przy niskiej selektywności świadomie rezygnuje z indeksu, nawet jeśli ten istnieje.

Miejsce, w którym selektywność faktycznie przestaje wystarczać, zależy od tabeli i od rozkładu wartości, nie od jednej sztywnej granicy procentowej — silniki liczą to na podstawie statystyk, nie reguły kciuka. Zasada ogólna brzmi: im mniej wierszy spełnia warunek w stosunku do całej tabeli, tym więcej indeks daje, i im więcej wierszy spełnia warunek, tym mniej ma to sensu, aż do granicy, za którą zwykłe przeczytanie całej tabeli jest po prostu szybsze.

Indeksów, jakie da się założyć na tabeli, jest kilka rodzajów — różnią się strukturą wewnętrzną i tym, jakie zapytania potrafią wesprzeć. Strona B-drzewo od środka opisuje najpowszechniejszy z nich, a strony Indeks złożony, pokrywający, częściowy, funkcyjny rozszerzają temat o warianty budowane na tym samym B-drzewie oraz o alternatywne struktury dopasowane do innych typów zapytań.

Częste błędy

Zakładanie indeksu na każdej kolumnie „na wszelki wypadek”. Każdy dodatkowy indeks to dodatkowy koszt przy każdym `INSERT`, `UPDATE` i `DELETE`, opisany wyżej — indeks, którego żadne zapytanie faktycznie nie używa, płaci tę cenę bez żadnej korzyści w zamian. Strona Kiedy indeks szkodzi pokazuje, jak sprawdzić, które indeksy w ogóle są używane.

Zakładanie indeksu na kolumnie o niskiej selektywności. Indeks na kolumnie `status` z czterema wartościami w tabeli `zamowienia` sam w sobie nie pomoże warunkowi `status = 'zrealizowane'`, jeśli ta wartość obejmuje jedną czwartą tabeli — optymalizator go po prostu pominie. Taki indeks może się za to przydać w innej roli, na przykład jako część indeksu złożonego albo indeksu częściowego, o czym mówi kolejna strona rozdziału.

Mylenie indeksu z kopią tabeli. Indeks nie przechowuje wszystkich kolumn wiersza — domyślnie tylko wartość kolumny indeksowanej i odsyłacz do wiersza. Zapytanie, które potrzebuje innych kolumn niż ta indeksowana, i tak musi dodatkowo odwiedzić samą tabelę, chyba że wchodzi w grę indeks pokrywający, opisany na kolejnej stronie.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — selektywność na oko

Tabela `zamowienia` ma cztery miliony wierszy. Kolumna `klient_id` przyjmuje około stu dwudziestu tysięcy różnych wartości, kolumna `status` — cztery wartości. Na której z tych dwóch kolumn indeks da większą korzyść typowemu zapytaniu filtrującemu po jednej konkretnej wartości, i dlaczego?

Odpowiedź

Na kolumnie `klient_id`. Sto dwadzieścia tysięcy różnych wartości rozłożonych na cztery miliony wierszy daje przeciętnie około trzydziestu trzech wierszy na jednego klienta — warunek `klient_id = wartość` wyodrębnia niewielki ułamek tabeli, więc indeks pozwala silnikowi ominąć zdecydowaną większość wierszy. Warunek na `status` z czterema wartościami wyodrębnia przeciętnie milion wierszy na wartość — jedną czwartą całej tabeli — więc indeks na tej kolumnie samodzielnie niewiele pomaga typowemu zapytaniu filtrującemu po jednym statusie.

Pytania kontrolne

1. Dlaczego zdanie „indeks przyspiesza tabelę” jest nieprecyzyjne?

Odpowiedź

Indeks nie zmienia samej tabeli ani sposobu, w jaki leżą w niej dane — jest osobną, dodatkową strukturą obok tabeli. Przyspiesza konkretne zapytania, które potrafią z niego skorzystać, filtrując po kolumnie indeksowanej w sposób sargowalny, a jednocześnie spowalnia każdy zapis do tabeli, bo ten zapis musi zaktualizować także indeks. „Przyspiesza tabelę” sugeruje jednostronną, bezkosztową korzyść, której w rzeczywistości nie ma.

2. Czym różni się koszt indeksu przy odczycie od kosztu przy zapisie?

Odpowiedź

Przy odczycie indeks jest czystym zyskiem, jeśli zapytanie potrafi z niego skorzystać — nie ma sytuacji, w której obecność pasującego indeksu spowalnia `SELECT`. Przy zapisie indeks jest czystym kosztem: `INSERT`, `UPDATE` kolumny indeksowanej i `DELETE` muszą zaktualizować strukturę indeksu dodatkowo do samego wiersza tabeli, niezależnie od tego, czy jakiekolwiek zapytanie kiedykolwiek z tego indeksu skorzysta.

Powiązane: B-drzewo od środka · Indeks złożony, pokrywający, częściowy, funkcyjny · Kiedy indeks szkodzi · WHERE: filtrowanie wierszy

Źródła: dokumentacja PostgreSQL, rozdział „Indexes” · Microsoft Docs, „Indexes” · Oracle Database Concepts, rozdział „Indexes and Index-Organized Tables”

Revision #47
Created 2026-09-24 22:42:35 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:38 UTC by Tomasz Konieczny