Indeks złożony, pokrywający, częściowy, funkcyjny

> Zwykły indeks na jednej kolumnie to tylko jeden z czterech wariantów, jakie da się zbudować na tym samym B-drzewie — ta strona przechodzi po kolei przez indeks na kilku kolumnach naraz, przez indeks, który wystarcza sam sobie bez zaglądania do tabeli, przez indeks obejmujący tylko część wierszy i przez indeks liczony z wyrażenia, a nie z gołej kolumny.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

ZMIENIA SYSTEM

Utworzenie indeksu złożonego: CREATE INDEX na kilku kolumnach

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

Budujemy jeden indeks na trzech kolumnach naraz, w podanej kolejności. Silnik sortuje wpisy leksykograficznie: najpierw po pierwszej kolumnie, potem — w obrębie identycznej wartości pierwszej — po drugiej, potem po trzeciej.

Przykład:

```
CREATE INDEX
```

ZMIENIA SYSTEM

Rozszerzenie indeksu o kolumny zwracane: INCLUDE w PostgreSQL

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

Kolumna w `INCLUDE` trafia do liścia indeksu tak samo jak kolumna kluczowa, ale nie bierze udziału w sortowaniu ani w nawigacji po drzewie — służy wyłącznie do tego, żeby zapytanie mogło ją odczytać bez zaglądania do tabeli.

Przykład:

```
CREATE INDEX
```

Dla indeksu `CREATE INDEX ix_zamowienia_klient ON zamowienia (klient_id) INCLUDE (status)` zapytanie `SELECT status FROM zamowienia WHERE klient_id = 4` znajduje `klient_id` przez nawigację po drzewie i od razu odczytuje `status` z tego samego liścia, bez dotykania tabeli.

□□ ZMIENIA SYSTEM

Rozszerzenie indeksu o kolumny zwracane: INCLUDE w MSSQL

```
CREATE NONCLUSTERED INDEX NazwaIndeksu ON NazwaSchematu.NazwaTabeli (NazwaKolumny)
INCLUDE (NazwaKolumny);
```

Ten sam mechanizm w SQL Server nosi identyczną nazwę klauzuli i działa na tej samej zasadzie — kolumny z `INCLUDE` zostają dopisane na poziomie liścia indeksu nieklastrowanego, bez wpływu na kolejność sortowania kluczy.

Przykład:

```
Commands completed successfully.
```

□□ **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

□□ ZMIENIA SYSTEM

Indeks częściowy na wybranym statusie: PostgreSQL

```
CREATE INDEX NazwaIndeksu ON NazwaTabeli (NazwaKolumny) WHERE NazwaKolumny = 'wartosc';
```

Typowe zastosowanie to kolumna statusu, w której zdecydowana większość wierszy ma już wartość końcową, a zapytania operacyjne interesują się właśnie tą mniejszością, która jeszcze nie osiągnęła stanu końcowego.

Przykład:

```
CREATE INDEX
```

Dla tabeli `zamowienia`, w której większość z czterech milionów wierszy ma status `zrealizowane` albo `anulowane`, a operacyjnie liczy się szybkie odnalezienie zamówień `nowe` do obsłużenia, `CREATE INDEX ix_zamowienia_nowe ON zamowienia (klient_id) WHERE status = 'nowe'` daje indeks obejmujący tylko ułamek wierszy tabeli — zamiast pełnego indeksu na `status`, który i tak byłby mało przydatny z powodu niskiej selektywności opisanej na stronie Czym jest indeks: i dlaczego to nie jest „przyspieszacz tabeli”.

◫◫ ZMIENIA SYSTEM

Indeks częściowy na wybranym statusie: filtered index w MSSQL

```
CREATE NONCLUSTERED INDEX NazwaIndeksu ON NazwaSchematu.NazwaTabeli (NazwaKolumny) WHERE NazwaKolumny = N'wartosc';
```

SQL Server nazywa ten sam mechanizm indeksem filtrowanym — warunek `WHERE` w definicji indeksu ogranicza go do podzbioru wierszy dokładnie tak samo jak indeks częściowy w PostgreSQL.

Przykład:

```
Commands completed successfully.
```

Oracle i MySQL nie mają bezpośredniego odpowiednika indeksu częściowego w tej postaci — w Oracle zbliżony efekt osiąga się indeksem funkcyjnym zwracającym `NULL` dla wierszy poza interesującym zakresem, bo Oracle domyślnie nie indeksuje wierszy z samymi wartościami `NULL` w kolumnie indeksowanej.

◫◫ **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

◫◫ ZMIENIA SYSTEM

Indeks na wyrażeniu: ratowanie WHERE LOWER(nazwa) = ... w PostgreSQL

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

Zamiast indeksować `nazwa`, indeksujemy wynik `LOWER(nazwa)` — dokładnie to wyrażenie, które i tak pojawia się po lewej stronie warunku w zapytaniach szukających bez rozróżniania wielkości liter.

Przykład:

```
CREATE INDEX
```

Zapytanie `SELECT nazwa FROM klienci WHERE LOWER(nazwa) = 'kowalski handel'` po założeniu `CREATE INDEX ix_klienci_nazwa_lower ON klienci (LOWER(nazwa))` przestaje wymagać przeliczenia `LOWER(nazwa)` dla każdego wiersza z osobna — silnik znajduje policzoną wartość wprost w indeksie.

◫◫ ZMIENIA SYSTEM

Indeks na wyrażeniu: ten sam mechanizm w Oracle

```
CREATE INDEX NazwaIndeksu ON NazwaSchematu.NazwaTabeli (LOWER(NazwaKolumny));
```

Ten sam mechanizm działa identycznie w Oracle, bo Oracle wspiera indeksy na dowolnym deterministycznym wyrażeniu SQL od dawna.

Przykład:

```
Index created.
```

SQL Server nie ma indeksu bezpośrednio na wyrażeniu w definicji `CREATE INDEX` — ten sam efekt osiąga się kolumną wyliczaną, dodaną do tabeli, a potem zaindeksowaną tak jak każdą inną kolumnę.

◫◫ **SQL** = Structured Query Language: język zapytań do relacyjnych baz danych

◫◫ ZMIENIA SYSTEM

Kolumna wyliczana i zaindeksowana: ratowanie tego samego przypadku w MSSQL

```
ALTER TABLE NazwaSchematu.NazwaTabeli ADD NazwaKolumny AS LOWER(NazwaKolumny);
```

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

Dwa kroki zamiast jednego: najpierw dokładamy do tabeli kolumnę wyliczaną, która przechowuje wynik wyrażenia, potem zakładamy na niej zwykły indeks — z punktu widzenia zapytania efekt końcowy jest taki sam jak indeks funkcyjny w PostgreSQL czy Oracle.

Przykład:

```
Commands completed successfully.
```

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

Indeks złożony

Pojęcie: indeks złożony (wielokolumnowy). Indeks zbudowany na więcej niż jednej kolumnie naraz, w którym wartości są sortowane najpierw według pierwszej kolumny, w obrębie każdej wartości pierwszej kolumny — według drugiej, i tak dalej. Kolejność kolumn w definicji indeksu nie jest kwestią stylu — decyduje o tym, jakie zapytania w ogóle mogą z tego indeksu skorzystać.

Pojęcie: zasada najbardziej lewego prefiksu. Reguła mówiąca, że indeks złożony na kolumnach `(a, b, c)` da się wykorzystać dla warunku dotyczącego samego `a`, dla warunku na `a` i `b` razem, oraz dla warunku na wszystkie trzy — ale nie dla warunku, który pomija `a` i dotyczy tylko `b` albo tylko `c`. Wynika to wprost z tego, jak wpisy są posortowane: bez ustalonej wartości `a` silnik nie wie, w które miejsce drzewa w ogóle zajrzeć, bo cała reszta wpisów jest posortowana dopiero wewnątrz każdej wartości `a`.

Zakładamy indeks złożony `(klient_id, status, data)` na tabeli `zamowienia`:

Warunek WHERE	Korzysta z indeksu?	Dlaczego
`klient_id = 4`	tak, pełne wykorzystanie	równość na pierwszej kolumnie prefiksu
`klient_id = 4 AND status = 'nowe'`	tak, pełne wykorzystanie	równość na obu pierwszych kolumnach prefiksu
`klient_id = 4 AND status = 'nowe' AND data > '2024-01-01'`	tak, pełne wykorzystanie	wszystkie trzy kolumny w kolejności zgodnej z indeksem

Warunek WHERE	Korzysta z indeksu?	Dlaczego
`klient_id = 4 AND data > '2024-01-01'`	częściowo	indeks zawęża do `klient_id = 4`, ale bez warunku na `status` między `klient_id` a `data` nie zawęża dalej po dacie — resztę filtruje przez odczyt wierszy
`status = 'nowe'`	nie	pomija `klient_id`, pierwszą kolumnę prefiksu — silnik nie ma od czego zacząć nawigację po drzewie
`data > '2024-01-01'`	nie	pomija obie wcześniejsze kolumny

Ostatnie dwa wiersze tłumaczą, dlaczego kolejność kolumn w indeksie złożonym warto dobierać pod kątem tego, jak faktycznie wygląda `WHERE` w najczęstszych zapytaniach — kolumnę, po której filtrujemy zawsze albo najczęściej, stawiamy pierwszą.

Indeks pokrywający

> **Pojęcie: indeks pokrywający.** Indeks, który zawiera wszystkie kolumny potrzebne zapytaniu — zarówno te w `WHERE`, jak i te w liście `SELECT` — dzięki czemu silnik odpowiada na zapytanie w całości z samego indeksu, bez ani jednego dodatkowego odczytu wiersza z tabeli.

> **Pojęcie: index-only scan (skan tylko indeksu).** Sposób wykonania zapytania, w którym silnik czyta wyłącznie strony indeksu i buduje z nich wynik, pomijając odczyt tabeli w ogóle — możliwy tylko wtedy, gdy indeks jest pokrywający dla danego zapytania. W PostgreSQL ten plan nazywa się dosłownie `Index Only Scan` i wraca na stronie Skan sekwencyjny, skan indeksu, lookup: sposoby dotarcia do wiersza.

Indeks częściowy

> **Pojęcie: indeks częściowy (filtered index).** Indeks obejmujący tylko podzbiór wierszy tabeli, wybrany warunkiem `WHERE` podanym już przy tworzeniu indeksu, a nie w każdym zapytaniu, które z niego korzysta. Mniejszy zakres oznacza mniejszy indeks — szybszy do przeszukania i tańszy w utrzymaniu, bo silnik aktualizuje go tylko przy zmianach

dotyczących wierszy spełniających ten warunek.

Indeks funkcyjny

Pojęcie: indeks funkcyjny (indeks na wyrażeniu). Indeks zbudowany nie na gołej wartości kolumny, tylko na wyniku wyrażenia policzonego z tej kolumny — silnik przechowuje w indeksie już policzone wartości wyrażenia, więc zapytanie z tym samym wyrażeniem w `WHERE` może z indeksu skorzystać zamiast tracić sargowalność, jak opisuje strona WHERE: filtrowanie wierszy przy okazji `UPPER(nazwa)`.

Częste błędy

Indeks złożony w kolejności niezgodnej z zapytaniami. Indeks `(status, klient_id)` nie pomoże zapytaniu filtrującemu tylko po `klient_id` — zasada najbardziej lewego prefiksu działa w jedną stronę, kolejność kolumn w `CREATE INDEX` musi odpowiadać kolejności, w jakiej warunki faktycznie pojawiają się jako równości w `WHERE`.

Zakładanie, że indeks pokrywający pomaga każdemu zapytaniu na tej tabeli. Indeks pokrywający jest pokrywający tylko względem konkretnego zestawu kolumn w konkretnym zapytaniu — dopisanie do `SELECT` jednej kolumny spoza `INCLUDE` natychmiast zmusza silnik z powrotem do odwiedzenia tabeli.

Indeks częściowy z warunkiem, który zapytania nie powtarzają dosłownie. Indeks częściowy `WHERE status = 'nowe'` wesprze zapytanie z warunkiem `status = 'nowe'`, ale nie wesprze zapytania z `status IN ('nowe', 'w realizacji')` — optymalizator musi rozpoznać, że warunek zapytania logicznie mieści się w warunku indeksu, a to rozpoznawanie ma swoje granice zależne od silnika.

□□ Ćwiczenia

Zadania sprawdzające.

Ćwiczenie 1 — najbardziej lewy prefiks

Mając indeks złożony `(a, b, c)`, które z poniższych warunków skorzystają z niego w pełni, które częściowo, a które wcale: `WHERE a = 1 AND c = 3`, `WHERE b = 2`, `WHERE a = 1 AND b = 2`?

Odpowiedź

`WHERE a = 1 AND c = 3` skorzysta częściowo — indeks zawęzi po `a`, ale bez warunku na `b` nie zawęzi dalej po `c`, mimo że warunek na `c` istnieje. `WHERE b = 2` nie skorzysta wcale, bo pomija `a`, pierwszą kolumnę prefiksu. `WHERE a = 1 AND b = 2` skorzysta w pełni, bo obie kolumny są kolejnym prefiksem indeksu, licząc od lewej.

Pytania kontrolne

1. Dlaczego kolumna w klauzuli INCLUDE nie pomaga warunkom w WHERE?

Odpowiedź

Kolumna z `INCLUDE` nie bierze udziału w sortowaniu ani w nawigacji po drzewie indeksu — jest dopisana na poziomie liścia wyłącznie po to, żeby dało się ją odczytać razem z kluczem. Warunek `WHERE` na takiej kolumnie nie ma po czym nawigować, więc silnik musiałby i tak przejrzeć wszystkie liście, tracąc sens indeksu.

2. Dlaczego WHERE UPPER(nazwa) = 'KOWALSKI HANDEL' przestaje być sargowalne bez indeksu funkcyjnego, mimo że zwykły indeks na nazwa już istnieje?

Odpowiedź

Zwykły indeks na `nazwa` przechowuje wartości `nazwa` takie, jakie są zapisane w tabeli, nie wynik `UPPER(nazwa)`. Żeby sprawdzić warunek, silnik musiałby policzyć `UPPER(nazwa)` dla każdego wiersza z osobna, co wymaga pełnego przejścia po tabeli. Indeks funkcyjny na `UPPER(nazwa)` rozwiązuje to, bo przechowuje już policzoną wartość wyrażenia, dokładnie tę, której szuka warunek.

Powiązane: Czym jest indeks: i dlaczego to nie jest „przyspieszacz tabeli” · B-drzewo od środka · WHERE: filtrowanie wierszy · Skan sekwencyjny, skan indeksu, lookup: sposoby dotarcia do wiersza

Źródła: dokumentacja PostgreSQL, rozdział „Indexes on Expressions”, „Partial Indexes” i „Index-Only Scans and Covering Indexes” · Microsoft Docs, „Create Indexes with Included Columns” i „Create Filtered Indexes” · Oracle Database SQL Language Reference, „Function-Based Indexes”

Revision #46
Created 2026-09-24 22:42:38 UTC by Tomasz Konieczny

Updated 2026-10-26 14:11:41 UTC by Tomasz Konieczny