Bazy danych

12. Indeksowanie

P. F. Góra

http://th-www.if.uj.edu.pl/zfs/gora/

2020

Przeszukiwanie posortowanego pliku

Wyszukanie wybranego wiersza w posortowanej tabeli o długości N wymaga dokonania $O(\log_2 N)$ porównań i, typowo, tyluż dostępów do dysku, co samo w sobie może być kosztowne. Jednak tabele SQL nie są posortowane, a co za tym idzie, pliki, w których zapisywane są tabele na dysku, też nie są posortowane — dotyczy to **zwłaszcza** tabel, w których dane są często zmieniane (dodawane, usuwane, dodawane itd). Wymóg zapisywania za każdym razem posortowanej wersji pliku spowodowałby *dramatyczne* spowolnienie działania bazy.

Dodatkowym problemem może być to, że często w tabelach definiujemy wiele "porządków": ta sama tabela mogłaby być sortowana w kolejności numerów operacji (klucz główny), nazwisk klientów (mogą się powtarzać), daty operacji (mogą się powtarzać) itp.

Dlatego też używa się *indeksów*, specjalnych struktur danych, przechowujących tylko atrybuty, względem których sortujemy. Atrybuty te nazywa się *kluczami sortowania* lub krótko *kluczami* (uwaga na mylącą terminologię: klucze sortowania nie muszą być kluczami tabeli!). Wraz z kluczem przechowywany jest fizyczny adres odpowiedniej krotki na dysku — adresu tego nie będziemy pokazywać w następujących definicjach i przykładach, skupiając się na samych kluczach sortowania. Indeksy muszą być tak zaprojektowane, aby wyszukiwanie elementów było szybkie, a dodawanie nowych i usuwanie istniejących niezbyt skomplikowane.

b-drzewo (*b-tree*)

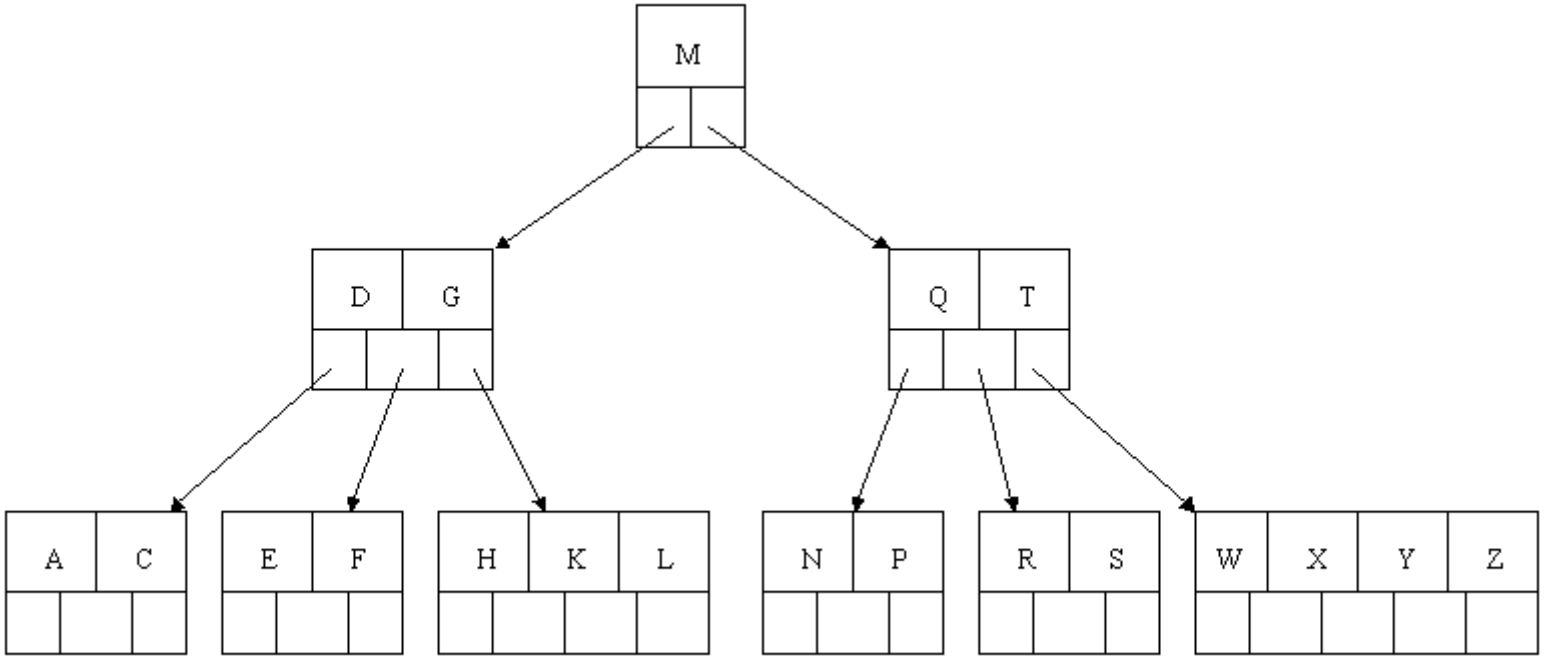
Najczęściej używana struktura danych do indeksowania tabel SQL. Definicja:

1. Drzewo ukorzenione.
2. Każdy węzeł ma nie więcej, niż m potomków.
3. Każdy węzeł, poza korzeniem i liśćmi, ma nie mniej, niż $m/2$ potomków. $m/2$ jest zaokrąglane w dół do liczby całkowitej i nazywane jest *czynnikiem minimalizacji*.
4. Korzeń, jeśli nie jest jednocześnie liściem, ma co najmniej dwóch potomków.
5. Wszystkie liście leżą na tym samym poziomie.
6. Węzeł nie będący liściem, posiadający k potomków, zawiera $k-1$ kluczy.
7. Węzeł posiada nie więcej, niż $m-1$ kluczy. Liść, który nie jest korzeniem, posiada nie mniej, niż $(m/2)-1$ kluczy.

Klucze przechowywane są w porządku niemalejącym. Każdy klucz (w węźle niebędącym liściem) ma potomka, który jest korzeniem poddrzewa zawierającego węzły z kluczami mniejszymi lub równymi od tego klucza, ale większymi od poprzedniego klucza; innymi słowy, klucze przechowywane w węzłach wewnętrznych działają jak separatory. Każdy węzeł ma ponadto dodatkowego potomka, umiejscowionego skrajnie po prawej stronie, będącego korzeniem poddrzewa zawierającego klucze większe od wszystkich kluczy w danym węźle.

B-drzewa wymyślili Rudolf Bayer i Ed McCreight w 1972, pracujący wówczas dla Boeinga. Nie wiedomo, co tak naprawdę oznacza “b” w nazwie b-drzewa. Może to być “Bayer”, “Boeing”, “broad” (szerokie), “bushy” (krzaczaste), “balanced” (zrównoważone) lub jeszcze coś innego.

Przykład b-drzewa



Własności b-drzew

B-drzewo o czynniku minimalizacji $m/2$, przechowujące N węzłów, ma wysokość

$$h < \log_{m/2} \frac{N+1}{2} .$$

W praktycznych implementacjach m bywa duże (rzędu kilkuset). Podstawa logarytmu jest duża, a w konsekwencji **b-drzewo jest szerokie, ale niskie**. Jeśli $m/2 = 100$, drzewo o dwu poziomach może przechować $\sim 10\,000$ kluczy, drzewo o trzech poziomach $\sim 1\,000\,000$ kluczy. Liczba węzłow odwiedzanych w czasie przeszukiwania jest niewielka, rzędu 2-3 — tyle też trzeba wykonać odczytów danych z dysku, plus jeden na odczytanie poszukiwanej krotki, *minus* jeden, gdyż korzenie indeksów często używanych tabel zazwyczaj rezydują w pamięci.

Budując drzewo węzły pozostawia się nie wpełni wypełnione, gdyż ułatwia (przyspiesza) to późniejsze wstawianie nowych kluczy (patrz niżej). Wyjątek stanowią tabele statyczne, niezmieniające się w czasie (lub zmieniające się bardzo rzadko) — w tej sytuacji wypełniamy węzły wewnętrze do ich maksymalnej pojemności.

Wyszukiwanie w b-drzewie

Przeszukiwanie rozpoczynamy od korzenia. Każdy węzeł przeszukujemy przeglądając kolejne klucze. Gdy znajdziemy wartość większą lub równą wartości poszukiwanej, wiemy, że poszukiwana wartość (poszukiwane wartości) znajdują się w poddrzewie, którego korzeniem jest znaleziona wartość (znaleziony separator). Jeśli wszystkie klucze w węźle są mniejsze* od wartości poszukiwanej, znajduje się ona w skrajnie prawym poddrzewie. Przeszukiwanie możemy zakończyć bez schodzenia do poziomu liści, jeśli poszukiwana wartość jest kluczem w jakimś węźle wewnętrznym.

Czas przeszukiwania b-drzewa jest $O(\log_{m/2} N)$.

*Dziekuję studentowi, który zwrócił mi uwagę na błąd, jaki był w poprzedniej wersji tego slajdu!

Podział węzła

Jeśli węzeł staje się przepełniony (po wstawieniu nowego klucza miałby więcej, niż $m-1$ kluczy), trzeba go podzielić. Niech X oznacza dzielony węzeł, Y jego rodzica. Medianę węzła X wstawiamy do jego rodzica, Y. Tworzymy nowy węzeł, Z, i wstawiamy do niego te klucze z X, które są większe od mediany. Z X usuwamy medianę i wartości większe od niej. Jeżeli dzielony węzeł nie jest liściem, wraz z przenoszonymi kluczami trzeba też przenieść wskaźniki do odpowiednich poddrzew. Cały ten proces niesie stały koszt $O(m)$.

W procesie podziału węzła, w tym w obliczaniu mediany, bierze też udział klucz, który chcemy wstawić do węzła (jeśli faktycznie chcemy wstawić).

Uwaga! Wstawianie mediany X do rodzica *może* spowodować przepełnienie rodzica, którego wówczas także dzielimy.

Wstawianie klucza do b-drzewa

Zanim wstawimy klucz, trzeba odszukać (za pomocą podanego wyżej algorytmu) liść, w którym ten klucz powinien się znaleźć. Jeżeli liść nie jest pełny, po prostu wstawiamy nowy klucz. Jeżeli liść jest pełny, należy go podzielić w sposób opisany powyżej. Ponieważ podział oznacza przesunięcie mediany dzielonego węzła na wyższy poziom, rodzic nie może być pełny lub też trzeba podzielić rodzica. Ten proces może powtarzać się w górę drzewa, aż do korzenia. Jeżeli korzeń się przepełnia, dzielimy go, tworząc nowy korzeń, na *wyższym* poziomie. Po takim podziale korzenia nowy korzeń zawiera tylko jeden klucz (medianę kluczy starego korzenia) i ma dwóch potomków.

Algorytm wstawiania kluczy do b-drzewa niesie koszt $O(m \log_{m/2} N)$.

Ten algorytm wymaga dwóch przejść przez drzewo: Pierwszego, w dół, w celu znalezienia liścia, do którego miałby trafić wstawiany element i drugiego, w górę, w celu dokonania podziału przodków. Ponieważ każde odczytanie węzła może oznaczać kosztowną operację dyskową, powyższy algorytm modyfikujemy w ten sposób, aby uniknąć drugiego przejścia przez drzewo. Mianowicie, już na etapie poszukiwania (pierwsze przejście) dzielimy każdy napotkany maksymalnie wypełniony węzeł. Może to oznaczać wykonywanie "niepotrzebnych" podziałów, ale gwarantuje, że po wstawieniu nowego klucza do liścia rodzic nie będzie przepełniony, co eliminuje konieczność ponownego przejścia drzewa w górę.

Usuwanie kluczy z b-drzewa

Usuwanie jest bardziej skomplikowane, niż wstawianie, gdyż trzeba zadbać, aby powstałe drzewo nadal było b-drzewem. Przypadki wymagające specjalnej troski to

- Usunięcie klucza powoduje, że węzeł ma mniej niż minimalną liczbę kluczy i dzieci.
- Usuwany klucz leży w węźle wewnętrznym, a zatem jest separatorem drzew potomnych.

Po usunięciu klucza z drzewa, drzewo często należy zrównoważyć.

Usuwanie z liścia

1. Odszukaj klucz, który chcesz usunąć.
2. Jeśli klucz należy do liścia, usuń go.
3. Jeśli w wyniku usunięcia pojawia się niedopełnienie, sprawdź rodzeństwo (najbliższych sąsiadów) węzła i albo przenieś klucz, albo połącz sąsiednie węzły.
4. Jeśli usunęliśmy wartość ze skrajnie prawego węzła (pod)drzewa, przenieś do niego największą wartość z jego lewego sąsiada, jeśli nie spowoduje to niedopełnienia.
5. Jeśli usunęliśmy wartość ze skrajnie lewego węzła (pod)drzewa, przenieś do niego najmniejszą wartość z jego prawego sąsiada, jeśli nie spowoduje to niedopełnienia.

Usuwanie z węzła wewnętrznego

Każdy klucz w węźle wewnętrznym (i w korzeniu) jest separatorem dwu poddrzew. Jeśli taki element zostaje usunięty, mogą zdarzyć się dwie sytuacje:

Po pierwsze, każde z dwojga dzieci bezpośrednio na lewo i na prawo od usuwanego elementu ma minimalną dopuszczalną liczbę kluczy, $(m/2)-1$. Węzły te mogą być połączone w jeden, posiadający $2(m/2)-2$ kluczy, co nie przekracza największej dopuszczalnej liczby kluczy. Węzeł ten zostaje przypisany do któregoś z sąsiednich (nieusuniętych) kluczy. Pewna komplikacja może pojawić się, jeżeli klucze nie są unikalne i klucze o wartościach równych usuwanemu występują w łączonych węzłach. Trzeba je także rekursywnie usunąć. *Dodatkowa* trudność powstaje, jeżeli w wyniku takiego rekursywnego usuwania liczba kluczy w połączonym węźle potomnym spadnie poniżej $(m/2)-1$.

Po drugie, co najmniej jedno z dwu dzieci posiada więcej, niż najmniejszą dopuszczalną liczbę kluczy. Wówczas trzeba dla tych węzłów znaleźć nowy separator. Może nim być największy element w lewym poddrzewie lub najmniejszy element w prawym poddrzewie. Element ten usuwamy z węzła, do którego należał i przemieszczamy do węzła, z którego usunęliśmy ten element, od usunięcia którego to wszystko się zaczęło. Ponieważ kreując nowy separator *usunęliśmy* element z węzła niższego rzędu (na ogół z liścia, choć może to być węzeł wewnętrzny niższego poziomu), trzeba teraz zastosować odpowiedni algorytm (usunięcie z liścia, usunięcie z węzła wewnętrznego) do tego węzła.

Równoważenie drzewa po usunięciu

Problemy pojawiają się, jeśli w wyniku usuwania liczba kluczy w którymś węźle spadnie poniżej dopuszczalnego minimum. Uwaga: Ograniczenie na najmniejszą dopuszczalną liczbę kluczy nie dotyczy korzenia, który może zawierać tylko jeden klucz!

1. Jeśli prawy brat węzła z niedoborem ma więcej, niż najmniejszą dopuszczalną liczbę kluczy,
   (a) Przenieś separator z rodzica do węzła z niedoborem, umieszczając go w ostatniej pozycji.
   (b) Najmniejszy element z prawego brata uczyń nowym separatorem i przenieś do rodzica.
   (c) Jeśli przekształcane węzły nie są liśćmi, przenieś pierwszego (skrajnie lewego) potomka prawego brata do węzła, w którym występował niedobór i uczyń go jego ostatnim (skrajnie prawym) dzieckiem.

2. Jeśli nie zachodzi powyższy przypadek, a lewy brat węzła z niedoborem ma więcej, niż najmniejszą dopuszczalną liczbę kluczy,
   (a) Przenieś separator z rodzica do węzła z niedoborem, umieszczając go na pierwszej pozycji.
   (b) Największy element z lewego brata uczyń nowym separatorem i przenieś do rodzica.
   (c) Jeśli przekształcane węzły nie są liśćmi, przenieś ostatniego (skrajnie prawego) potomka lewego brata do węzła, w którym występował niedobór i uczyń go jego pierwszym (skrajnie lewym) dzieckiem.

3. Jeśli lewy i prawy brat mają najmniejszą dopuszczalną liczbę kluczy,
   (a) Stwórz nowy węzeł zawierający klucze z węzła z niedoborem, klucze z jednego z braci oraz separatora (z rodzica), który rozdzielał łączone węzły.
   (b) Usuń separator z rodzica i zastąp dwoje dzieci, które rozdzielał, połączonym węzłem stworzonym w powyższym kroku.
   (c) Jeśli usunięcie separatora z rodzica spowodowało, że liczba kluczy rodzica spadła poniżej minimum i jeśli rodzic nie jest korzeniem, powtórz cały algorytm równoważenia dla rodzica.

Jedynym przypadkiem, jaki pozostaje, jest korzeń, który nie zawiera kluczy i ma jednego potomka. W tym wypadku usuwamy (pusty) korzeń. Nowym korzeniem staje się jego jedyny potomek.

Wizualizacja b-drzew

Na sieci dostępnych jest wiele apletów pozwalających na wizualizację operacji na b-drzewach — na przykład `https://www.cs.usfca.edu/~galles/visualization/BTree.html`.

Indeksy w SQL

Klucz główny zawsze jest indeksem:

```
mysql> CREATE TABLE foo
    -> (KolA INT NOT NULL PRIMARY KEY,
    -> ...);
Query OK, 0 rows affected (0.02 sec)
```

Torzenie klucza głównego na więcej, niż jednej kolumnie — wszystkie *muszą* być `NOT NULL`:

```
mysql> CREATE TABLE bar
    -> (Kol1 CHAR(5) NOT NULL,
    ->  Kol2 INT NOT NULL,
    ->  KOL3 CHAR(12),
    ->  PRIMARY KEY(Kol1,Kol2));
Query OK, 0 rows affected (0.09 sec)
```

Dodatkowe indeksy

Dodatkowe indeksy możemy tworzyć od razu w definicji tabeli...

```
mysql> CREATE TABLE popo
    -> (X INT NOT NULL PRIMARY KEY,
    ->  Y VARCHAR(16),
    ->  INDEX NazwaIndeksu (Y));
Query OK, 0 rows affected (0.08 sec)
```

...albo osobno, poleceniem `CREATE INDEX nazwaindeksu ON nazwatabeli (kolumna)`:

```
mysql> CREATE TABLE lupo
    -> (nr INT UNSIGNED NOT NULL PRIMARY KEY,
    ->  A VARCHAR(16),
    ->  B DATE);
Query OK, 0 rows affected (0.19 sec)

mysql> CREATE INDEX indeksnadwukolumnach ON lupo (A,B);
Query OK, 0 rows affected (0.27 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Nazwa indeksu jest opcjonalna. Jest ona potrzebna, jeżeli chcemy *explicite* zażądać użycia (lub zignorowania) jakiegoś indeksu.

Indeks może mieć kwalifikator `UNIQUE` (`CREATE UNIQUE INDEX...`). Wartości atrybutu wchodzącego w skład indeksu unikalnego nie mogą się powtarzać. W MySQL kolumna wchodząca w skład unikalnego indeksu *może* przyjmować wartości `NULL`, które *mogą* się powtarzać.

Przykład

`bigdata` jest tabelą zawierającą 2 000 000 krotek. Zawiera trzy kolumny, z których pierwsza, będąca kluczem głównym, jest typu `INTEGER`, druga `TIMESTAMP`, trzecia `BIGINT`. Zakładamy dodatkowe indeksy na tych dwu kolumnach.

```
mysql> CREATE INDEX indeksczasu ON bigdata (czas);
Query OK, 2000000 rows affected (9 min 45.19 sec)
Records: 2000000  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX indeksliczb ON bigdata (wielkosc);
Query OK, 2000000 rows affected (35 min 21.59 sec)
Records: 2000000  Duplicates: 0  Warnings: 0
```

Proszę zwócić uwagę na czasy wykonania zapytań zakładających te indeksy!

Zróbmy kopię tej tabeli, nie zawierającą nowych indeksów:

```
mysql> CREATE TABLE bezindeksow
    -> (nr INT UNSIGNED NOT NULL PRIMARY KEY,
    -> czas TIMESTAMP NOT NULL,
    -> wielkosc BIGINT UNSIGNED NOT NULL)
    -> SELECT * FROM bigdata;
Query OK, 2000000 rows affected (36.55 sec)
Records: 2000000  Duplicates: 0  Warnings: 0
```

Traz porównajmy czasy wykonania dwu zapytań:

```
mysql> SELECT COUNT(DISTINCT wielkosc)
    -> FROM bigdata USE INDEX (indeksliczb)
    -> WHERE wielkosc < 16000000;
+--------------------------+
| COUNT(DISTINCT wielkosc) |
+--------------------------+
|                       96 |
+--------------------------+
1 row in set (0.24 sec)


mysql> SELECT COUNT(DISTINCT wielkosc)
    -> FROM bezindeksow
    -> WHERE wielkosc < 16000000;
+--------------------------+
| COUNT(DISTINCT wielkosc) |
+--------------------------+
|                       96 |
+--------------------------+
1 row in set (3.34 sec)
```

Uwagi o wykorzystaniu indeksów

Efektywność wykorzystania indeksów zależy od selektywności zapytania, to jest od tego, jak duży procent wierszy przeszukiwanej tabeli zwraca zapytanie. I tak:

- Jeżeli zapytanie zwraca nie więcej, niż 1% wierszy, warto używać indeksów.
- Jeżeli zapytanie zwraca więcej, niż 20% wierszy, *nie należy* używać indeksów. Strategia pełnego odczytu danych z tabeli będzie bardziej efektywna.
- To, co się dzieje dla wartości pośrednich, może zależeć od wielu czynników, w tym historii usuwania i dodawania nowych wierszy do tabeli, a nawet historii odczytywań danych z tabeli przez bieżący wątek kliencki.

Jak widzieliśmy, samo zakładanie indeksów na wielkich tabelach może być kosztowne. Dodatkowo każdy indeks powoduje, że proces dodawania/usuwania krotek z tabeli może być znacznie wolniejszy. Indeksów nie należy więc zakładać “bo tak”, ale tylko wówczas, gdy ich przewidywane wykorzystanie przyspieszy działanie bazy, to znaczy działanie często wykonywanych/kluczowych zapytań.

Wymuszanie indeksów

W większości przypadków wbudowany optymalizator zapytań sam zdecyduje, jaki plan wykonania wybrać, jakich indeksów użyć, czy nie lepsza będzie strategia pełnego przeglądania tabeli itp. W sytuacjach, w których podejrzewamy, że optymalizator nie znajduje właściwego rozwiązania, możemy narzucić to, z jakiego indeksu skorzystać

```
SELECT ... FROM nazwatabeli USE INDEX(nazwaindeksu) ...;
```
spowoduje, że wyszukiwanie będzie odbywać się z wykorzystaniem wskazanego indeksu.

```
SELECT ... FROM nazwatabeli FORCE INDEX(nazwaindeksu) ...;
```
spowoduje, że wyszukiwanie będzie odbywać się z wykorzystaniem wskazanego indeksu, przy czym dodatkowo optymalizator uzna, że pełne przeglądanie tabeli jest bardzo kosztowne, a więc nie wolno go zastować.

```
SELECT ... FROM nazwatabeli IGNORE INDEX(nazwaindeksu) ...;
```
spowoduje, że wskazany indeks nie zostanie użyty.

Analogicznej składni można używać w złączeniach.

Indeksy i zapytanie `UPDATE`

Przypuśćmy, że w zapytaniu `UPDATE` zmieniamy wartości indeksowanego atrybutu, przy czym zmianom podlega niewiele elementów, tak, aby wyszukiwanie odbywało się z użyciem indeksu, nie w trybie pełnego przeglądania tabeli. Na przykład chcemy podnieść o 10% pensje pracownikom zarabiającym najmniej:

```
UPDATE pracownicy
SET pensja = 1.1*pensja
WHERE pensja < 3000;
```

Jeśli kolumna `pensja` jest indeksowana, narzucający się sposób wykonania jest taki:

1. Wyszukaj element spełniający warunek `WHERE`.
2. Zmień jego wartość.
3. Wyszukaj kolejny element etc

Rodzi się tu jednak pewne niebezpieczeństwo: Zmieniona wartość może nadal spełniać kryterium (wyobraźmy sobie kogoś zarabiającego 2500 PLN), zaś po dokonaniu zmiany zostanie przebudowany indeks i *nowa* wartość może zostać znaleziona przy przeszukiwaniu indeksu jako kandydat do zmiany. To byłoby wbrew naszym intencjom! Zapytanie SQL nie może widzieć wprowadzanych przez siebie zmian.

`UPDATE` z wykorzystaniem indeksu odbywa się zatem następująco:

I. W pierwszej fazie, korzystając z indeksu, tylko wyszukujemy krotki, które należy zmienić, i zapamiętujemy ich klucze w tabeli pomocniczej. Ponieważ zmienianych krotek nie będzie zbyt wiele, zmaterializowana tabela pomocnicza najprawdopodobniej zmieści się w pamięci.

II. W drugiej fazie, przeglądamy tabelę z kluczami krotek, które trzeba zmienić, i korzystając z indeksu związanego z kluczem głównym, znajdujemy odpowiednie krotki i dokonujemy zmiany wartości w krotce. Po każdej zmianie *następuje* przebudowanie indeksu, ale nie jest on już przeszukiwany w tym samym zapytaniu.

Wyszukiwanie pełnotekstowe

Wyszukiwanie pełnotekstowe służy do wyszukiwania podanych napisów (ciągów znaków) w kolumnach typu `CHAR`, `VARCHAR`, `TEXT`. W tym celu na tabeli, którą chcemy w ten sposób przeszukiwać, należy założyć specjalny indeks. W MySQL wyszukiwanie pełnotekstowe działa tylko na tabelach typu MyISAM.

```
mysql> CREATE TABLE Teksty
    -> (Id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Tytul VARCHAR(32),
    -> Tresc VARCHAR(64),
    -> FULLTEXT(Tytul,Tresc)
    -> ) ENGINE=MyISAM, CHARSET cp1250;
Query OK, 0 rows affected (0.28 sec)
```

Niech przykładowa tabela zawiera następujące dane:

```
mysql> SELECT * FROM Teksty;
+----+----------------------+-----------------------------------------------------+
| Id | Tytul                | Tresc                                               |
+----+----------------------+-----------------------------------------------------+
|  1 | Prof. Andrzej bim    | Rzepliński, bom, kryminolog: - Nie znam akt          |
|  2 | operacyjnych bim bom | gdzie mogą być dowody, ale nie możemy               |
|  3 | wykluczyć,           | że XXX bim był milicyjnym agentem.                  |
|  4 | Powtórka             | Operacyjnych akt nie znam ja, bim, mówi Rzepliński. |
|  5 | Bim                  | Operacyjnych, operacyjnych, operacyjnych, milicyjych. |
|  6 | Bim bim              | akt operacyjnych                                    |
|  7 | Bom                  | bim                                                 |
|  8 | wypełniacz           | wypełniacz                                          |
|  9 | bim                  | wypełniacz                                          |
| 10 | wypełniacz           | bim                                                 |
+----+----------------------+-----------------------------------------------------+
10 rows in set (0.10 sec)
```

Wyszukiwanie odbywa się poprzez wywołanie zapytania `SELECT` z klauzulą `WHERE MATCH(`*nazwy kolumn*`) AGAINST(`*wzorzec*`)`.

```
mysql> SELECT * FROM TEKSTY
    -> WHERE MATCH (Tytul,Tresc) AGAINST('operacyjnych');
+----+----------------------+-----------------------------------------------------+
| Id | Tytul                | Tresc                                               |
+----+----------------------+-----------------------------------------------------+
|  5 | Bim                  | Operacyjnych, operacyjnych, operacyjnych, milicyjnych. |
|  6 | Bim bim              | akt operacyjnych                                    |
|  4 | Powtórka             | Operacyjnych akt nie znam ja, bim, mówi Rzepliński. |
|  2 | operacyjnych bim bom | gdzie mogą być dowody, ale nie możemy               |
+----+----------------------+-----------------------------------------------------+
4 rows in set (0.00 sec)
```

Wartość semantyczna

Wyrażenie `MATCH...AGAINST(`*wzorzec*`)` określa wartość semantyczną podanego wzorca w poszczególnych wierszach tabeli:

```
mysql> SELECT Id, CONCAT(Tytul,' ',Tresc) AS Tekst,
    -> MATCH (Tytul,Tresc) AGAINST('operacyjnych') AS Wartosc
    -> FROM Teksty;
+----+-----------------------------------------------------------+------------------+
| Id | Tekst                                                     | Wartosc          |
+----+-----------------------------------------------------------+------------------+
|  1 | Prof. Andrzej bim Rzepliński, bom, kryminolog: - Nie znam akt |      0           |
|  2 | operacyjnych bim bom gdzie mogą być dowody, ale nie możemy | 0.38341854994499 |
|  3 | wykluczyć, że XXX bim był milicyjnym agentem.             |      0           |
|  4 | Powtórka Operacyjnych akt nie znam ja, bim, mówi Rzepliński. | 0.38763393589171 |
|  5 | Bim Operacyjnych, operacyjnych, operacyjnych, milicyjnych. | 0.53687456835829 |
|  6 | Bim bim akt operacyjnych                                  | 0.40085528270084 |
|  7 | Bom bim                                                   |      0           |
|  8 | wypełniacz wypełniacz                                     |      0           |
|  9 | bim wypełniacz                                            |      0           |
| 10 | wypełniacz bim                                            |      0           |
+----+-----------------------------------------------------------+------------------+
10 rows in set (0.00 sec)
```

Wartość semantyczna wzorca w wierszu

- jest tym większa, im więcej razy wzorzec występuje w wierszu,
- jest tym większa, im wiersz jest krótszy,
- przyjmuje wartość zero, jeśli wzorzec *nie* występuje w wierszu.

Uwaga: Wartość semantyczna wyrazów krótkich lub występujących w co najmniej połowie wierszy wynosi *zero*.

```
mysql> SELECT * FROM TEKSTY
    -> WHERE MATCH (Tytul,Tresc) AGAINST('bim bom');
Empty set (0.00 sec)
```

Zapytanie `SELECT... WHERE MATCH... AGAINST...` wyświetla tylko wiersze o niezerowej wartości semantycznej.

Boolowskie wyszukiwanie pełnotekstowe

`AGAINST(`*wzorzec* `IN BOOLEAN MODE)` pozwala na stosowanie operatorów lepiej określających poszukiwany wzorzec. Przy wyszukiwaniu boolowskim tabela *nie musi* mieć zdefiniowanego indeksu `FULLTEXT`.

- $+$ fragment musi być obecny, $-$ fragment nie może być obecny.

```
mysql> SELECT * FROM TEKSTY
    -> WHERE MATCH (Tytul,Tresc) AGAINST('+operacyjnych -znam' IN BOOLEAN MODE);
+----+----------------------+-----------------------------------------------------+
| Id | Tytul                | Tresc                                               |
+----+----------------------+-----------------------------------------------------+
|  2 | operacyjnych bim bom | gdzie mogą być dowody, ale nie możemy               |
|  5 | Bim                  | Operacyjnych, operacyjnych, operacyjnych, milicyjnych. |
|  6 | Bim bim              | akt operacyjnych                                    |
+----+----------------------+-----------------------------------------------------+
3 rows in set (0.04 sec)
```

- Brak operatora — podany fragment jest opcjonalny. Wiersze zawierające fragment opcjonalny są oceniane wyżej.

- () — nawiasy służą do grupowania słów w podwyrażenia. Takie grupy można zagnieżdżać.
- `"` — zwrot ujęty w podwójne cudzysłowy powoduje dopasowanie tylko wierszy, które zawierają podany zwrot *dokładnie w tej formie, w jakiej został napisany*.

Są także inne operatory.

```
mysql> SELECT * FROM Teksty
    -> WHERE MATCH(Tytul,Tresc) AGAINST('+(akt operacyjnych)' IN BOOLEAN MODE);
+----+----------------------+-----------------------------------------------------+
| Id | Tytul                | Tresc                                               |
+----+----------------------+-----------------------------------------------------+
|  2 | operacyjnych bim bom | gdzie mogą być dowody, ale nie możemy               |
|  4 | Powtórka             | Operacyjnych akt nie znam ja, bim, mówi Rzepliński. |
|  5 | Bim                  | Operacyjnych, operacyjnych, operacyjnych, milicyjnych. |
|  6 | Bim bim              | akt operacyjnych                                    |
+----+----------------------+-----------------------------------------------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM Teksty
    -> WHERE MATCH(Tytul,Tresc) AGAINST('+"akt operacyjnych"' IN BOOLEAN MODE);
+----+---------+------------------+
| Id | Tytul   | Tresc            |
+----+---------+------------------+
|  6 | Bim bim | akt operacyjnych |
+----+---------+------------------+
1 row in set (0.00 sec)
```

Wyszukiwanie z rozwijaniem zapytania

Użytkownik bardzo często polega na “wiedzy niejawnej” — człowiek wie, że jakieś terminy są równoznaczne lub bliskoznaczne, ale jak to zalgorytmizować? Powiedzmy, szukając wzorca “daza danych”, chcemy też uzyskać wiersze zawierające napisy “Oracle” i “MySQL”, nawet jeśli napis “baza danych” w nich nie występuje. Albo też szukając informacji o profesorze, chcemy także znaleźć wiersze, w których profesor wymieniony jest z nazwiska, z pominięciem tytułu.

Realizujemy to za pomocą konstrukcji `AGAINST(`*wzorzec* `WITH QUERY EXPANSION)`. W takim wypadku tabela przeszukiwana jest dwa razy. Przy drugim wyszukiwaniu wzorzec zostaje połączony z kilkoma innymi wzorcami, znajdującymi w wierszach będących na górze (mających największą wartość semantyczną *pierwotnego wzorca*) po pierwszym wyszukiwaniu.

Przykład

```
mysql> SELECT * FROM TEKSTY
    -> WHERE MATCH (Tytul,Tresc) AGAINST('Prof');
+----+------------------+-----------------------------------------+
| Id | Tytul            | Tresc                                   |
+----+------------------+-----------------------------------------+
|  1 | Prof. Andrzej bim | Rzepliński, bom, kryminolog: - Nie znam akt |
+----+------------------+-----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM TEKSTY
    -> WHERE MATCH (Tytul,Tresc) AGAINST('Prof' WITH QUERY EXPANSION);
+----+------------------+-----------------------------------------------+
| Id | Tytul            | Tresc                                         |
+----+------------------+-----------------------------------------------+
|  1 | Prof. Andrzej bim | Rzepliński, bom, kryminolog: - Nie znam akt   |
|  4 | Powtórka         | Operacyjnych akt nie znam ja, bim, mówi Rzepliński. |
+----+------------------+-----------------------------------------------+
2 rows in set (0.01 sec)
```

Uwagi końcowe

- Wyszukiwanie pełnotekstowe jest wolne i kosztowne.
- Jeżeli planujemy przeszukiwać naprawdę dużą tabelę, zaleca się
  - Utworzyć tabelę bez definiowania indeksu `FULLTEX`.
  - Wprowadzić dane do tabeli.
  - Po wprowadzeniu danych dodać indeks za pomocą polecenia `ALTER TABLE`.

  W przeciwnym wypadku wprowadzanie danych do tabeli może być bardzo powolne.
- Istnieje sporo *komercyjnych* nakładek na RDBMSy, znacznie usprawniających wyszukiwanie pełnotekstowe.