STUDIA INFORMATICA 2012
Volume 33 Number 2B (106)

Andrzej BARCZAK, Dariusz ZACHARCZUK
Uniwersytet Przyrodniczo-Humanistyczny, Instytut Informatyki

TYPOWE PROBLEMY OPTYMALIZACJI ZAPYTAŃ SQL PRZY TWORZENIU ŚREDNICH I DUŻYCH SERWISÓW/APLIKACJI WWW

Streszczenie. Artykuł przedstawia odpowiedzi na najczęstsze pytania, które zadają sobie webmasterzy, programiści, podczas pisania zapytań SQL. Jakie złączenia tabel stosować? Jaki silnik bazy wybrać? Czy i kiedy lepszym rozwiązaniem jest wykonanie kilku mniejszych zapytań, a kiedy jednego złożonego? Te i inne dylematy zostaną omówione i opatrzone konkretnymi przykładami.

Słowa kluczowe: SQL, MySQL, optymalizacja, wydajność

TYPICAL PROBLEMS OF OPTIMIZING SQL QUERIES FOR CREATING MEDIUM AND LARGE SITES/WEB APPLICATIONS

Summary. Article presents answers to common questions, which have webmasters, programmers when writing SQL queries. How to join tables? Which database engine is better to choose? If and when the better solution is to perform several smaller queries and when complex one? These and other dilemmas will be discussed and provided with concrete examples.

Keywords: SQL, MySQL, optimization, performance

1. Dla kogo i dlaczego powstał ten artykuł?

Problemy optymalizacji są jak rzeka, można je badać długo i wnikliwie, porównywać technologie dostępu, wydajność systemów baz danych itd. itp. Większość z takich testów jest nauką dla samej nauki. Każdy programista intuicyjnie domyśla się, że do przechowywania ogromnych ilości danych lepszy będzie Oracle niż MySQL. Tak samo jest, jeśli posiadamy

dedykowany sterownik obsługujący tylko wybraną bazę danych – będzie on zapewne szybszy i lepszy niż rozwiązanie kompleksowe. Dlaczego więc kolejny artykuł nt. optymalizacji?

Ilość danych, które krążą w internecie, rośnie szybciej niż kiedykolwiek dotąd. Każdy zdaje sobie sprawę z potencjału, jaki daje istnienie w sieci. Każdy (nawet najmniejszy) serwis czy aplikacja WWW poddawana jest wielu zabiegom, które mają na celu zwiększenie jej oglądalności. Skutkiem takich działań są niezliczone ilości zapytań, kierowane do SZBD o dane. W 99% takich przypadków serwisy (wystarczy przejrzeć ofertę hostingów, żadna z wiodących firm nie oferuje nawet PostgreSQL) pracują, oparając się na technologii PHP+MySQL, z ewentualnym wspomaganiem ze strony Ajaxa, wykorzystując gotowe rozwiązania hostingowe (wirtualne lub dedykowane). W takim przypadku optymalizować możemy jedynie strukturę BD oraz zapytania SQL, co w konsekwencji może dać większy efekt niż dobieranie technologii czy SZBD. Skupimy się tylko i wyłącznie na badaniu instrukcji SELECT, ze względu na jej dominujący charakter absolutnie w każdej apliakcji. Komu może pomóc ów artykuł? Każdemu programiście, który tworzy oprogramowanie na podstawie baz danych (BD), bez względu na to, czy jest to prosty serwis WWW czy dedykowany CRM do zarządzania przedsiębiorstwem.

1.1. Typowe dylematy podczas projektowania i programowania

Jednymi z pierwszych pytań, jakie padają podczas tworzenia struktury BD, są:

1. Czy i jak stosować indeksy? Użyć jednego grupowego, a może indywidualnych dla każdego pola?
2. Czy typ danych, jaki zostanie wybrany do przechowywania informacji, ma znaczenie?
3. Klucze obce: czy wykorzystywać wbudowaną referencję (dostępna w InnoDB) czy zapewniać integralność na poziomie kodu?
4. Silnik: czy mechanizm składowania wpływa na prędkość przetwarzania zapytań? Jeśli nie potrzebujemy funkcji, które determinują wybór danego rozwiązania, jakiego silnika użyć?

W dalszym etapie użytkowania i pracy na stworzonej bazie pojawiają się kolejne problemy:

1. Który sposób łączenia tabel jest efektywniejszy?
2. Czy w niektórych sytuacjach (np. przy pracy na kilku tabelach) lepiej jest wykonać jedno czy kilka zapytań?
3. Czy i kiedy ograniczać wyniki zapytań?
4. Jaka jest różnica przy przeszukiwaniu danych tekstowych opartych na typach VARCHAR i TEXT (mowa tutaj o zwykłych nieindeksowanych zmiennych tekstowych)?

Na wszystkie powyższe pytania padnie odpowiedź w dalszej części artykułu.

2. Przygotowanie, warunki i cel przeprowadzanych badań

Jak wspomniano wcześniej, przy testach nie są brane pod uwagę ani różne systemy bazodanowe, ani technologia dostępu. Zmierzony zostanie tylko stosunek efektywności zastosowanych rozwiązań. Celem jest więc odpowiedź na pytanie, która metoda jest szybsza, a nie, w jakich warunkach osiągniemy większą wydajność (stosując to samo rozwiązanie)? Nie bierzemy więc pod uwagę bezwzględnych czasów reakcji, a stosunek odpowiedzi systemu. Zapytania SQL także nie będą stałe, a generowane losowo – tożsame pod względem budowy, ale parametryzowane w zależności od sytuacji. Takie podejście wydaje się być bardziej praktyczne.

Do wykonania badań posłuży jednostka o następującej konfiguracji: PC, 8GB RAM, 7200rpm HDD, Intel Core 2 Duo E8400 3.0GHz z oprogramowaniem Win7, Apache 2.2.17, MySQL 5.1.53, PHP 5.3.4. Maszyna nie będzie optymalizowana w żaden sposób, gdyż badamy jedynie konstrukcję bazy oraz instrukcji SQL i w takim przypadku nie liczy się osiągnięty czas, a stosunek wyników, o czym w dalszej części artykułu.

Skrypty będą wykonywały zapytania testujące wyspecyfikowane siedem problemów w sposób cykliczny. W celu uniknięcia zapamiętywania wyników prze SZBD dla stałych poleceń SQL następujących po sobie, skrypty będą realizowały zapytania na przemian dla danych problemów. Wykonanie szybkiego porównania reakcji SZBD non stop na jedno zapytanie testujące problem, a naprzemienne wykonywanie takich instrukcji potwierdziło poprawność założenia – różnice były widoczne mimo braku kasowania buforu. Jego czyszczenie nie jest tutaj bezwzględnie wymagane z uwagi na praktyczny charakter testu. Poza tym, po każdym zapytaniu badającym nastąpi zapisanie rezultatów, co dodatkowo stworzy pewną losowość komend. W konsekwencji powinniśmy dostać odpowiedź na pytanie, w jakim stopniu i jaki rodzaj rozwiązania danego problemu poprawi efektywność projektowanej aplikacji?

2.1. Struktura tabel testujących

Do wykonania badań utworzone zostały dwie grupy tabel. Pierwsza tabela z pierwszej grupy ma następującą konstrukcję:

```
CREATE TABLE `index_on1` (
`id` INT NOT NULL AUTO_INCREMENT,
`int_20` INT(20) UNSIGNED NOT NULL,
`int_4` INT(4) UNSIGNED NOT NULL,
`liczba_v` VARCHAR(20) NOT NULL,
`liczba_i` BIGINT UNSIGNED NOT NULL,
`abstrakt` VARCHAR(300) NOT NULL,
`tekst` TEXT NOT NULL,
PRIMARY KEY (`id`),
INDEX (`int_20`, `int_4`, `liczba_v`, `liczba_i`)
) ENGINE = MyISAM;
```

Tabel w pierwszej grupie będzie trzy (index_on1, index_on2, index_on2_inno), gdzie:

- id: jest kluczem głównym w każdej z dwóch table,
- int_20, int_4: dwie liczby o różnym zakresie {2},
- int_20, liczba_v, liczba_i: ta sama liczba zapisana za pomocą różnych typów {2},
- abstrakt, tekst: dane tekstowe {8},
- INDEX: tutaj grupowy (standardowy w phpMyAdmin), w drugiej tabeli (index_on2) indywidualny, a w trzeciej jak w drugiej, ale na silniku innoDB {1},
- ENGINE: MyISAM (dla tabel 1 i 2) oraz InnoDB (dla tabeli 3) {4}.

W nawiasach klamrowych podano główne problemy, których konstrukcja dotyczy (odwołanie do wymienionych wcześniej punktów). Druga grupa składa się z tabeli głównej:

```
CREATE TABLE `pk` (
`id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
`osoba` VARCHAR(200) NOT NULL,
`data_ur` DATE NOT NULL,
`wiek` TINYINT NOT NULL,
`pesel` VARCHAR(11) NOT NULL,
INDEX (`data_ur`),
UNIQUE (`pesel`)
) ENGINE = InnoDB;
```

oraz tabeli podrzędnej:

```
CREATE TABLE `fk_on` (
`fv` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
`pk_id` INT NOT NULL ,
`data_zam` DATE NOT NULL,
`suma` DOUBLE NOT NULL,
`lista` VARCHAR(500) NOT NULL,
INDEX (`data_zam`),
INDEX (`suma`),
FOREIGN KEY (pk_id) REFERENCES pk(id) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = InnoDB;
```

która występuje także w wersji bez definicji klucza obcego, a zamiast niego jest zwykły indeks. Na tej grupie tabel zbadamy zagadnienia 3, 5, 6 i 7. Liczba wygenerowanych rekordów dla poszczególnych tabel przedstawia się następująco:

- index_on1, index_on2, index_on2_inno: po 1,5 miliona na tabelę,
- pk: 10 000 wierszy,
- fk_on, fk_off: 600 000 krotek.

Łączna suma rekordów w bazie to około 5,2 miliona. Ważną informacją jest fakt, że wygenerowane dane w obrębie grup tabel są identyczne. Konkretna instrukcja SQL będzie więc zwracała identyczny wynik, a różnicą będzie tylko czas odpowiedzi.

3. Testy i osiągnięte rezultaty

Zapytania, które posłużą do przeprowadzenia prób czasowych, mają prostą konstrukcję, jednak ich losowy charakter (rodzaj porównania: <, >, =, wartości liczbowe i daty, spójniki

warunków: OR, AND oraz ograniczenia: offset i limit są generowane losowo) sprawia, iż czasy ich wykonania dla pojedynczego sprawdzenia mogą się różnic nawet kilkaset razy. Tabela 1 zawiera listę schematów komend SQL oraz wskazanie na problem, którego dotyczą. Dla każdego z zapytań stworzony został skrypt odpytujący (łącznie osiem). W danym pliku losowane są zmienne potrzebne do zbudowania zapytania. Następnie polecenie w takiej formie zostaje wykonane po kolei na tabelach, na których badamy dane zagadnienie. Obliczany jest czas wykonania pierwszego zapytania oraz stosunek następnych czasów do pierwszego uzyskanego, a wynik zapisywany do jest BD – na tym działanie skryptu się kończy. Każdy z tych ośmiu plików będzie uruchamiany kolejno.

Jak odczytać wyniki badań? Ze względu na różne dane, o które baza jest odpytywana, bezwzględny czas wykonania nie jest wartością miarodajną i na tej podstawie nie da się wyciągnąć żadnych wniosków. Obliczany jest więc każdorazowo stosunek czasu wykonania. Oczywiście, aby to zrobić, należy mieć do czego się ustosunkować. Przyjmiemy, że pierwsze wykonane zapytanie będzie wyznacznikiem i w stosunku do niego będą określane pozostałe rezultaty. Przykład: uzyskany czas z dwóch kolejnych instrukcji to 0,75 s i 1,25 s W takim wypadku wynik badań zaprezentowany zostanie w postaci 1,67, co oznacza, iż rozwiązanie drugie jest mniej optymalne i wolniejsze. Dodatkowo na pierwszą komendę zostanie wybrana ta, która zdaniem autora ma szanse osiągnąć najlepsze rezultaty.

Tabela 1

Lista schematów instrukcji SQL wykorzystanych do przeprowadzenia analizy efektywności

NR	Zapytanie SQL	Referencja do problemu							
		1	2	3	4	5	6	7	8
1	`SELECT id FROM index_... WHERE ([int_20|int_4|liczba_v|liczba_i] [=|<|>] N [AND|OR])+ LIMIT offset,limit`	x			x				
2	`SELECT id FROM index_... WHERE [int_20|liczba_v|liczba_i] [=|<|>] N LIMIT offset,limit`		x		x				
3	`SELECT id FROM index_... WHERE [abstrakt|tekst] LIKE '%N%' LIMIT offset,limit`				x				x
4	`SELECT id, fv FROM pk,[fk_on|fk_off] WHERE id=pk_id AND pesel = 'N'`			x		x			
5	`SELECT id, fv FROM pk LEFT JOIN [fk_on|fk_off] ON id=pk_id WHERE pesel = 'N'`			x		x			
6	`SELECT id, fv FROM pk, [fk_on|fk_off] WHERE id=pk_id AND (data_zam > 'N' OR suma>M) LIMIT offset,limit`						x		
7	`SELECT osoba FROM pk WHERE pesel [=|IN] ('P') [LIMIT 1]`							x	

Wyniki badań zostały przedstawione w tabelach 2 i 3. Analizie zostały poddane wszystkie uzyskane czasy i odrzucono skrajne wyniki, które ewidentnie odbiegały od pozostałych i mogły być spowodowane czynnikami zewnętrznymi (np. kopiowanie plików z partycji, na której

rezyduje SZBD). Ponieważ niektóre rezultaty były dość zaskakujące, wyniki mogą być opisowe, a dokładne spostrzeżenia zostaną wyszczególnione w podsumowaniu.

Tabela 2

Wyniki czasów osiąganych przez poszczególne zapytania dla pierwszej grupy tabel

Lp	Opis i przykładowe zapytanie(a) badające dany przypadek	Wyniki	
		MyISAM	InnoDB

1	Praca na różnych indeksach. Stosunek indeksu prostego do złożonego dla MyISAM oraz czasu MyISAM do InnoDB dla indeksu prostego.
	`SELECT id FROM index_on2 WHERE int_20>7036132 OR int_4 =1654 AND cyfra_v ='5059204' AND cyfra_i <7121582 LIMIT 377,8`
	`SELECT id FROM index_on1 WHERE int_20>7036132 OR int_4 =1654 AND cyfra_v ='5059204' AND cyfra_i <7121582 LIMIT 377,8`
	`SELECT id FROM index_on2_inno WHERE int_20>7036132 OR int_4 =1654 AND cyfra_v ='5059204' AND cyfra_i <7121582 LIMIT 377,8`
		Zróżnicowany
		dla czasu wyjściowego <1 s średni stosunek wynosi 0,65;
		dla czasów >>1 s śr. stosunek wynosi 0,01
			Zróżnicowany
			dla czasu wyjściowego <1 s średni stosunek wynosi 1,68; dla czasów >>1 s śr. stosunek wynosi 0,04

2	Ta sama informacja – różny typ danych. Stosunek INT do VARCHAR.
	`SELECT id FROM index_on2 WHERE int_20<2765502 LIMIT 677,94`
	`SELECT id FROM index_on2 WHERE liczba_v<2765502 LIMIT 677,94`
	`SELECT id FROM index_on2_inno WHERE int_20<2765502 LIMIT 677,94`
	`SELECT id FROM index_on2_inno WHERE liczba_v<2765502 LIMIT 677,94`
		Bardzo zróżnicowany
		dla czasów <1 s śr, stosunek 4059!
		dla >1 s – 0,03
			Bardzo zróżnicowany
			dla czasów <1 s śr, stosunek 293!
			dla >1 s – wyniki niespójne

3	Ta sama informacja – różny typ danych. Stosunek VARCHAR do TEXT.
	`SELECT id FROM index_on2 WHERE abstrakt LIKE '%a eu %' LIMIT 974,23`
	`SELECT id FROM index_on2 WHERE tekst LIKE '%a eu %' LIMIT 974,23`
	`SELECT id FROM index_on2_inno WHERE abstrakt LIKE '%a eu %' LIMIT 974,23`
	`SELECT id FROM index_on2_inno WHERE tekst LIKE '%a eu %' LIMIT 974,23`
		Śr stosunek: 0,43	śr. stosunek: 0,85

4. Podsumowanie

Interpretacja otrzymanych wyników badań dostarczyła następujących wniosków (numery odpowiadają liczbom porządkowym z tabel 2 i 3):

1. Indeksy – oczywiście należy stosować je zawsze dla tych kolumn, na których nasze zapytania opierają swoje warunki. Dodatkowo:
   - Jeśli w warunku zapytania występuje kilka zmiennych indeksowanych, szybciej działa indeks grupowy.

Tabela 3

Wyniki czasów osiąganych przez poszczególne zapytania dla drugiej grupy tabel

Lp	Opis i przykładowe zapytanie(a) badające dany przypadek	śr. stosunek czasów

4	Stosunek czasu przy pracy na referencji klucza obcego do indeksu.
	`SELECT id, fv FROM pk, fk_on`
	`WHERE id=pk_id AND pesel='4489' LIMIT 593500,87`
	
	`SELECT id, fv FROM pk, fk_off`
	`WHERE id=pk_id AND pesel='4489' LIMIT 593500,87`
		0,69

5	jw., ale przy złączeniu typu LEFT JOIN
	`SELECT id, fv FROM pk LEFT JOIN fk_on ON id=pk_id`
	`WHERE pesel='9289' LIMIT 430518,22`
	
	`SELECT id, fv FROM pk LEFT JOIN fk_off ON id=pk_id`
	`WHERE pesel='9289' LIMIT 430518,22`
		0,67

6	Stosunek pobierania danych złączonych przez WHERE do złączenia JOIN
	`SELECT id, fv FROM pk, fk_on`
	`WHERE id=pk_id AND (wiek <59 AND suma >2118) LIMIT 609,100`
	
	`SELECT id, fv FROM pk LEFT JOIN fk_on ON id=pk_id`
	`WHERE (wiek <59 AND suma >2118) LIMIT 609,100`
		0,70

7	Stosunek jednego zapytania na dwóch tabelach do dwóch zapytań na poszczególnych tabelach (czas przetwarzania danych, które trzeba pobrać z wyniku jednego zapytania, aby uformować drugie, **jest brany pod uwagę**).
	`SELECT id, fv FROM pk, fk_on WHERE id=pk_id AND (data_zam>'1934-06-25' OR suma >4032) ORDER BY id, fv LIMIT 174, 81`
	
	`SELECT fv, pk_id FROM fk_on WHERE data_zam >'1934-06-25' OR suma>4032 ORDER BY fv LIMIT 174, 81`
	
	`76 » SELECT wiek FROM pk WHERE id IN (2058,2205,...) ORDER BY id LIMIT 81`
		Dla czasów >20 s st: 4,99 dla <20 s – 0,03

8	Stosunek zapytania bez ograniczenia do zapytania z ograniczeniem.
	`SELECT wiek FROM pk WHERE pesel IN (2363) OR id IN (5239)`
	
	`SELECT wiek FROM pk WHERE pesel IN (2363) OR id IN (5239) LIMIT 2`
		0,08

2. INT kontra VARCHAR – praktycznie każdą zmienną liczbową da się zapisać w postaci tekstu. Wyciągnięte wnioski akurat z tego dość kontrowersyjnego porównania mogą zaskoczyć, ale na pewno otwierają ścieżkę do dalszych badań i pogłębienia tej konkretnej kwestii. Wracając do konkluzji, warto przedstawiać liczbę za pomocą tekstu (wg wyników badań):
   - Dla operacji równości dane oparte na typie liczbowym w MyISAM mają ogromną przewagę. Dla operacji <, > jest dokładnie na odwrót. Różnice są bardzo wyraźne.

   - Dla InnoDB powyższy wniosek sprawdza się tylko w przypadku równości. Nierówności nie trzymały się żadnego wzorca.
3. VARCHAR kontra TEXT:
   - w przypadku MyISAM 2x szybsze jest przeszukiwanie typu TEXT,
   - dla InnoDB różnice nie są aż tak duże, ale nadal typ TEXT ma przewagę.
4. Konkatenacja tabel w połączeniu z kluczem obcym daje gorsze wyniki. W większości przypadków bardziej optymalnie działa zwykły indeks w miejscu klucza obcego.
5. Przy lewostronnym złączeniu sytuacja z pkt. 4 powtórzyła się z zadziwiająco podobnym wynikiem. Jest to wyraźne potwierdzenie, że baza danych wykonuje szybciej operacje na zwykłym indeksie niż na kluczu obcym.
6. Warunek złączenia klauzuli WHERE czy JOIN? Użycie lewostronnego złączenia zaowocowało przyspieszeniem rzędu 20% – 30% nad tą samą operacją wykonywaną za pomocą konkatenacji.
7. Przy badaniu opcji wyższości jednego zapytania nad kilkoma czasy, osiągane przez pierwszą instrukcję SQL były wyraźnie podzielone na stosunkowo krótkie (od 0,07 s do 20 s) oraz bardzo długie (~3350 s). W tym drugim przypadku losowość zapytań spowodowała, że SZBD musiał przeglądać dokładnie wszystkie istniejące rekordy. Czas takiego zapytania oscylował w granicy 56 min, a ewentualny raport w aplikacji, korzystający z dwóch zapytań zamiast z jednego, wykonywałby się prawie 5 razy dłużej. Zupełnie odwrotnie przedstawia się sytuacja, kiedy wynik dostajemy bez przeglądania całości (czyli sytuacja z czasami krótkimi) – wtedy rozbicie na dwa zapytania ma sens i czas ich wykonania łącznie z czasem potrzebnym na przetworzenie danych niezbędnych do drugiej instrukcji jest kilkadziesiąt razy mniejszy.
8. Badanie ósme jest chyba jednym z ciekawszych, a wnioski są następujące: ograniczenie wyników za pomocą instrukcji LIMIT może kilkanaście razy skrócić czas oczekiwania na odpowiedź z systemu. Należy zwrócić uwagę, że zapytanie bez limitu pracowało na zmiennych, które w całej tabeli są unikalne (klucz główny i zmienna z indeksem UNIQUE). Można by wymagać od SZBD, aby wiedział, że skoro atrybut jest niepowtarzalny, to po znalezieniu dopasowania, innego wiersza o tej wartości już nie będzie. Mimo wszystko wygląda na to, iż takiego założenia system nie posiada.

Zapewne istnieje niezliczona liczba innych przypadków, warunków, opcji, których ten artykuł nie uwzględnił. Uważamy jednak, że w omówionych problemach każdy z czytelników znajdzie przynajmniej jeden, z którym się spotkał, nad którym nie raz się zastanawiał, ale nie miał sposobności lub czasu, aby go zbadać, a po przeczytaniu opracowania problem ten zostanie rozwiązany.

BIBLIOGRAFIA

1. Pachev S.: MySQL. Mechanizmy wewnętrzne bazy danych. Helion, Gliwice 2008.
2. Henderson C.: Skalowalne witryny internetowe. Helion, Gliwice 2007.
3. Schwartz B., Zaitsev P., Tkochenko V., Zawodny J. D., Lentz A., Balling D. J.: Wysoko wydajne MYSQL. Optymalizacja, archiwizacja, replikacja. Wydanie II, Helion, Gliwice 2009.
4. Harrington J. L.: SQL dla każdego. Wydanie II, Mikom, Warszawa 2000.

Wpłynęło do Redakcji 16 stycznia 2012 r.

Abstract

Article addresses common optimize and performance problems of SQL queries, however, with a completely different point of view as a typical development in this topic. Completed tests are designed to reveal more optimal solutions to common issues. We do not compare the absolute times achieved here by the SQL command. Randomness used in the construction of queries brings us closer to the reality in which these queries are used.

Conclusions and observations put forward on the basis of the results can be used both in simple web applications, more complex web sites as well as dedicated systems supporting companies management. Any database programmer will certainly find here a theme that interested him.

Adresy

Andrzej BARCZAK: Uniwersytet Przyrodniczo-Humanistyczny, Instytut Informatyki, ul. 3 Maja 54, 08-110 Siedlce, Polska, andrzej.barczak@neostrada.pl.
Dariusz ZACHARCZUK: Uniwersytet Przyrodniczo-Humanistyczny, Instytut Informatyki, ul. 3 Maja 54, 08-110 Siedlce, Polska, dzariusz@dzariusz.pl.