STUDIA INFORMATICA 2012
Volume 33 Number 2A (105)

Radosław BOROŃSKI
Politechnika Koszalińska, Wydział Elektroniki i Informatyki

AUTOMATYZACJA I OPTYMALIZACJA PROCESU DOBORU INDEKSÓW DLA DOWOLNEGO WYCINKA CZASOWEGO W RELACYJNEJ BAZIE DANYCH (NA PRZYKŁADZIE ORACLE 11G)

Streszczenie. Artykuł przedstawia metodę automatycznego doboru i optymalizacji indeksów utworzonych na tabelach relacyjnej bazy danych dla dowolnego wycinka czasowego w cyklu przetwarzania powtarzalnego lub zmiennego kodu SQL. Powszechny sposób selekcji indeksów polega na wybraniu najczęściej odpytywanych kolumn dla pojedynczego zapytania z pominięciem innych zapytań SQL w wybranym bloku przetwarzania zapytań. W całościowym cyklu przetwarzania powszechnie stosowana metoda optymalizacji tylko jednego zapytania SQL może okazać się mało wydajna.

Słowa kluczowe: indeks, indeksy, optymalizacja, ISP, SQL, zapytanie

AUTOMATIC INDEX SELECTION AND OPTIMIZATION FOR SQL BLOCK IN RELATIONAL DATABASES

Summary. Article presents new approach to automatic indexes selection problem (ISP) and indexes optimization for any timeframe in process of constant or variable SQL queries block processing for relational database systems. Common index selection methods usually focus on choosing most frequently selected columns for standalone SQL query, omitting other queries present in the same processing block. For a long database processing, containing more than one SQL query, commonly used optimization methods may prove to be inefficient.

Keywords: index, indexes, optimization, ISP, SQL, query

1. Opis problemu i wcześniejsze prace

Szerokie wykorzystanie technologii relacyjnych baz danych przy projektowaniu systemów wspomagających zarządzanie przedsiębiorstwem sprawiło, że konieczne stało się opracowanie metod zwiększających szybkość przetwarzania danych. Wśród licznych dziś sposobów i metod zwiększania wydajności jedną z najważniejszych jest właściwy dobór struktur indeksowych, będących integralną częścią wszystkich aplikacji wykorzystujących technologię relacyjnych baz danych.

Indeks jest to struktura zbudowana na podstawie drzewa binarnego lub mapy bitowej, obejmująca jedną lub więcej kolumn tabeli. Indeks przyśpiesza wykonanie wyrażenia SQL. Budowę indeksu można podzielić na: logiczną (indeksy unikalne i nieunikalne, utworzone na pojedynczych kolumnach lub złożone) lub fizyczną (B-drzewa (B-tree Indexes) o odwróconym kluczu (Reverse Key Indexes), bitmapowe (Bitmap Indexes)) [1]. Najczęściej spotykanymi indeksami są typu B-drzewa, gdyż definiowane są dla atrybutów o dużej selektywności. Do wyszukiwania położenia rekordów stosuje się strukturę podobną do odwróconego drzewa. Na samej górze tego drzewa znajduje się korzeń, poniżej są poziomy z gałęziami, a na samym dole znajdują się liście zawierające informacje o położeniu rekordu bądź rekordów w tabeli.

W literaturze występują różne sposoby i metody doboru indeksów. Każdy z nich zawiera jednak pewne ograniczenia, jak np.: możliwość doboru indeksów tylko jednokolumnowych [2, 3, 4, 5] lub możliwość wyboru wielokolumnowych indeksów kandydackich (ograniczenie do jednego zapytania) [4, 6] bądź generowania dużej liczby indeksów, niewłaściwej ze względów oszczędnościowych [10]. Inne rozwiązania proponują generowanie struktur indeksowych na podstawie wyników kalkulacji wieloparametrowych funkcji, w zamyśle autorów zbliżonych do funkcji kosztowych, zaimplementowanych w systemach zarządzania relacyjnymi bazami danych [7]. Żadne z tych rozwiązań nie proponuje doboru indeksów dla wybranego bloku zapytań SQL, każde skupia się tylko na pojedynczym zapytaniu. Rozwiązania umożliwiające śledzenie sekwencji zapytań SQL i dobór indeksów wielokolumnowych stosują wątpliwy z punktu widzenia optymalizacji, iteracyjny przegląd zupełny lub negują możliwość utworzenia na tej kolumnie indeksu wielokolumnowego, dla którego jednokolumnowy indeks okazał się nieprzydatny [8]. Występuje też metoda pomiaru przydatności indeksu, stosująca szacunkowy koszt oparty na planie wykonania zapytania, nieuwzględniająca faktycznego czasu przetwarzania zapytania [8]. Inne rozwiązanie proponuje przeprowadzenie operacji optymalizacyjnej na samym planie wykonania, a nie na indeksie [9].

Niniejszy artykuł ma na celu przedstawienie metody automatycznego doboru struktury indeksów jedna lub wielokolumnowych dla tabeli w relacyjnej bazie danych w taki sposób, aby

brany był pod uwagę wycinek czasowy (blok zapytań SQL), w którym nastąpiło sekwencyjne przetwarzanie zapytań SQL w tabeli.

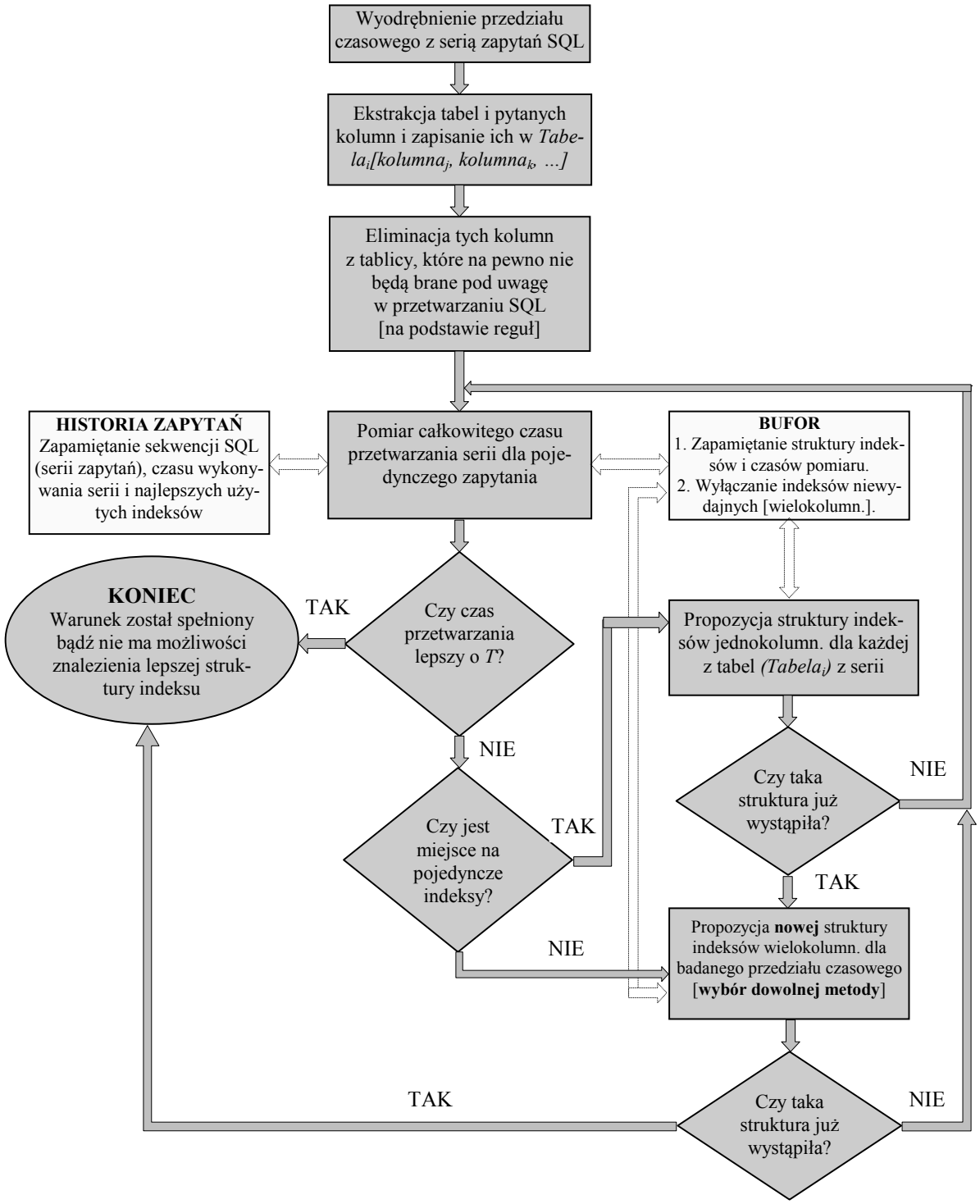
2. Proponowana metoda

2.1. Sposób postępowania

W wybranym bloku zapytań SQL należy wybrać te tabele, w których możliwe jest zastosowanie indeksu, i zapisać je w formacie $Tabela_i[kolumna_1, kolumna_2,...]$, a następnie wyeliminować te kolumny, dla których stworzenie indeksu nie miałoby sensu. Istnieje tu skończona lista reguł, dla których optymalizator kosztowy nie będzie brał danej kolumny pod uwagę (np. dla operatorów warunków: *WHERE: '!='*, *'NOT LIKE'*, *'NOT IN'*, *'LIKE %wyrażenie'* itp.). Należy zmierzyć czas potrzebny do zakończenia przetwarzania serii zapytań SQL dla istniejących już indeksów. Następnie należy zaproponować nowe, niewystępujące wcześniej jednokolumnowe indeksy dla wybranych kolumn w taki sposób, aby czas przetwarzania nie był gorszy od prób poprzednich. Jeżeli jednokolumnowe indeksy nie skróciły całkowitego czasu wykonywania bloku zapytań, autor proponuje zastosowanie doboru struktury indeksów wielokolumnowych. Przy budowie indeksów wielokolumnowych nie należy pomijać kolumn, dla których indeks jednokolumnowy z poprzednich kroków okazał się niewydajny. Istnieje kilka metod doboru kolumn przy procesie tworzenia indeksu wielokolumnowego. Można posłużyć się heurystycznym przeglądem zupełnym [9] na wszystkich możliwych kolumnach, występujących w zapytaniach. Metoda ta jest niewydajna, gdyż wymusza tworzenie dużej liczby indeksów i wyboru tych, dla których czasy pomiarów są najlepsze. Innymi, znanymi w literaturze metodami są np. heurystyczny losowy dobór selektywny lub algorytm ewolucyjny [10]. Na potrzeby tego artykułu autor posłużył się metodą doboru selektywnego. W przypadku gdy w procesie doboru indeksu pojawił się już wcześniej taki indeks wielokolumnowy i w buforze został zapisany czas wykonywania bloku zapytań SQL, indeksu takiego nie należy tworzyć. Jego wydajność powinna być jednak wzięta pod uwagę przy porównywaniu czasów z innych prób. Seria zapytań zapisywana jest wraz z czasem wykonywania i użytym indeksem w historii zapytań, a następnie aktualizowana, w przypadku gdy zostanie znaleziony lepszy indeks dla tej serii (niezależnie od założonego celu). Proces ten ma na celu zapamiętanie najlepszego indeksu dla serii zapytań w przypadku zmieniającej się lub aktualizowanej sekwencji zapytań SQL w przyszłości.

Próba zakończy się sukcesem, jeżeli przy sprecyzowanych założeniach, w podanym wycinku czasowym, sumaryczny czas wykonywania zapytań SQL będzie zawsze krótszy od najdłuższego badanego o co najmniej *T%*. Na potrzeby tego artykułu autor ustalił *T* na poziomie

20, co wyklucza granicę błędu, a także pokazuje faktyczną redukcję czasu wykonania zapytań w bloku. Na rysunku 1 przedstawiono schemat blokowy proponowanej metody.



Rys. 1. Metoda doboru indeksów dla badanej serii zapytań SQL
Fig. 1. Index selection method for a SQL block

2.2. Środowisko testowe

Środowisko testowe składało się z jednego serwera o konfiguracji: system operacyjny Red Hat Enterprise Linux Server (release 5.5) z jądrem 2.6.18-194.26.1.el5, pamięcią operacyjną 48 GB, 4-rdzeniowym procesorem Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, jednej partycji dyskowej o pojemności 535 GB. Na serwerze zainstalowano bazę danych Oracle w wersji 11.2.0.3, instancję ASM z dostępem do łącznej pojemności dysków 5,6 TB (macierz RAID).

2.3. Warunki proponowanej metody

Należy ustalić warunki, dla których przedsięwzięcie będzie miało sens, będzie powtarzalne i uniwersalne:

- w zastosowanej wersji i w zastosowanym rodzaju bazy danych musi wystąpić możliwość indeksowania obiektów segmentowych (tabel),
- zastosowany silnik bazy danych musi być modelem przetwarzania relacyjnego, z obsługą języka SQL (ANSI SQL),
- parametry konfigurujące serwer i bazę danych nie mogą się zmieniać,
- obciążenie bazy danych i serwera, procesy I/O muszą być na podobnym poziomie dla każdej z prób,
- należy zadbać o wystarczającą ilość miejsca na dysku (lub systemu plików) dla nowo tworzonych indeksów,
- należy umożliwić dostęp do struktury logicznej indeksowanych obiektów, a także samych indeksów,
- należy definiować wyłącznie te struktury indeksowe, które mogą przyczynić się do minimalizacji czasu realizacji zapytań SQL.

3. Przebieg doświadczenia

Wybrano przykładowy wycinek czasowy, w którym nastąpiło przetwarzanie trzech prostych, losowo wygenerowanych zapytań SQL. Miało to na celu pokazanie sensowności metody, a nie próby zmierzenia się ze skomplikowaną strukturą składniową. Zmierzono całkowity czas przetwarzania serii. Następnie spróbowano znaleźć taki indeks (indeksy), który skróciłby czas wykonywania tego bloku zapytań o ustalony wcześniej parametr *T%*.

Z cyklu produkcyjnego wyodrębniono trzy zapytania SQL wyszukujące dane w dwóch kolumnach, odwołujące się do jednej tabeli (TEST). Wyodrębniona tabela była niepartycjonowana i składała się z czterech kolumn losowo wypełnionych danymi *<kol1, kol2, kol3,*

kol4>: *kol1* zawierała wartości liczbowe z przedziału <0,1*10^6>, *kol2* zawierała wartości liczbowe z przedziału <1*10^6, 9*10^6>, *kol3* zawierała wartości znakowe (wielkie litery) o długości 20, natomiast *kol4* zawierała ciągi znakowe (małe litery) o długości 10. Początkowo dla kolumny *kol1* utworzony został (przez doświadczonego administratora baz danych) indeks, który miałby skrócić czas wykonywania całego bloku zapytań SQL. Tabela zawierała 5 milionów niepowtarzających się rekordów. Nie określono stopnia równoległości wykonywanych zapytań w definicji tabeli (przetwarzanie pojedyncze obsługiwane przez jedną sesję).

Zapytanie 1:

```
SELECT kol1, kol2 FROM test WHERE kol1 = <CONST> AND kol2 < [CONST2]
ORDER BY 2 DESC, 1;
```

Zapytanie 2:

```
SELECT kol2, kol3, kol4 FROM test WHERE kol3 LIKE '[ZNAKI]%'
OR kol4 LIKE '%<ZNAKI>%' ORDER BY 1 DESC, 2 DESC;
```

Zapytanie 3

```
SELECT kol1, kol3 FROM test WHERE kol3 LIKE '<ZNAKI>%' OR kol1 = <CONST>;
```

Określono też iloczyn algebraiczny kolumn biorących udział w przetwarzaniu wszystkich zapytań. W opisanym przypadku są to kolumny z zapytania 1, 2, 3: [*kol1, kol2, kol3, kol4*].

W kolejnym kroku dokonano ekstrakcji tabel wraz z kolumnami biorącymi udział w przetwarzaniu zapytań SQL i zapisano je w postaci:

TEST1[kol1, kol2], TEST2[kol3, kol4], TEST3[kol1, kol3].

Na podstawie reguł przetwarzania zapytań przez optymalizator kosztowy z tablicy TEST2 wyeliminowano kolumnę kol4 (warunek *'LIKE %wyrażenie'*). Kolumna ta nie była brana pod uwagę w procesie doboru indeksów.

Tabela 1

Czasy wykonywania zapytań dla początkowego indeksu *KOL1_IDX*

Nr pomiaru	Zapytanie 1	Zapytanie 2	Zapytanie 3	Suma (Zapytania 1+2+3)
1	112 s	22 s	115 s	249 s
2	114 s	25 s	128 s	267 s
3	118 s	28 s	135 s	281 s

Kolejnym krokiem był pomiar całkowitego czasu wykonywania wyodrębnionych zapytań SQL. Serię zapytań uruchomiono trzy razy i zmierzono czas wykonywania dla każdego z za-

pytań osobno, a następnie zmierzono czas dla całej serii (przy istniejącym indeksie na kolumnie *kol1*). Tabela 1 przedstawia czasy wykonywania zapytań przy zastosowaniu indeksu *KOL1_IDX*.

Kolejnym krokiem było znalezienie indeksu, który skróciłby sumaryczny czas wykonywania obydwu zapytań o co najmniej 20%, czyli dla najdłuższego z badanych prób (281 s) czas wykonywania nie mógłby być większy niż 224,28 s dla każdej próby, po zastosowaniu proponowanego indeksu (indeksów).

4. Proponowane rozwiązanie

Proponowane rozwiązanie wyłącza obecnie używany indeks (nie usuwa) i proponuje zastosowanie nowego indeksu (indeksów), ale tylko dla kolumn biorących udział w przetwarzaniu zapytań SQL. Inne kolumny tabeli, niebiorące udziału w zapytaniu, były pomijane. Wykonano trzy próby z trzema nowymi indeksami jednokolumnowymi.

Próba 1: Utworzenie trzech indeksów jednokolumnowych:

```
CREATE INDEX KOL2_IDX ON test (kol2);
CREATE INDEX KOL3_IDX ON test (kol3);
CREATE INDEX KOL4_IDX ON test (kol4);
```

W zaproponowanym rozwiązaniu optymalizator kosztowy nie skorzystał z żadnego z nowo utworzonych indeksów i we wszystkich trzech przypadkach wystąpiło pełne czytanie tabeli (*TABLE FULL SCAN*). Proponowane indeksy były nieprzydatne w tej serii zapytań i czasy potrzebne na uzyskanie wyników były podobne do czasów uzyskanych przy próbie z indeksem KOL1_IDX. W tabeli 2 przedstawiono czasy wykonywania zapytań dla próby 1.

Tabela 2

Czasy wykonywania zapytań po utworzeniu trzech niezależnych indeksów

Nr pomiaru	Zapytanie 1	Zapytanie 2	Zapytanie 3	Suma (Zapyt.1+2+3)
4	116 s	24 s	114 s	254 s
5	115 s	30 s	118 s	263 s
6	110 s	31 s	109 s	250 s

W żadnym z pomiarów w próbie 1 nie udało się uzyskać czasu mniejszego niż 224,28 s. Zastosowanie trzech niezależnych indeksów nie było rozwiązaniem lepszym o 20% od zastosowanego początkowego indeksu *KOL1_IDX*.

Próba 2: Utworzenie indeksu kompozytowego (wielokolumnowego) na kolumnach dopuszczonych do indeksowania, występujących we wszystkich zapytaniach (na kolumnach *kol1, kol2 i kol3*):

```
CREATE INDEX KOL1_KOL2_KOL3_IDX ON TEST(KOL1,KOL2, KOL3);
```

Stworzenie takiego indeksu nie skróciło czasu wykonywania zapytań 1 i 2, jednak był on wykorzystany w zapytaniu 1 (tu nastąpiło zwiększenie czasu wykonania o ponad 80% dla operacji czytania indeksu *INDEX RANGE SCAN DESCENDNG*). W przypadku zapytania 3 indeks również był wykorzystany. Nie wpłynęło to jednak na zwiększenie bądź zmniejszenie czasu wykonywania dla żadnej z prób. Wystąpiło tu pełne czytanie tabeli. W tabeli 3 przedstawiono czasy wykonywania zapytań dla próby 2.

Tabela 3

Czasy wykonywania zapytań po utworzeniu indeksu kompozytowego dla trzech kolumn występujących we wszystkich zapytaniach testowych

Nr pomiaru	Zapytanie 1	Zapytanie 2	Zapytanie 3	Suma (Zapytania 1+2+3)
7	210 s	22 s	111 s	343 s
8	211 s	28 s	120 s	359 s
9	204 s	24 s	118 s	346 s

W żadnym z pomiarów w próbie 2 nie udało się uzyskać czasu mniejszego niż 224,28 s. Zastosowanie indeksu kompozytowego na wszystkich kolumnach kandydackich nie jest rozwiązaniem lepszym o 20% od zastosowanego początkowego indeksu *KOL1_IDX*.

Próba 3: Utworzenie indeksu kompozytowego (wielokolumnowego) na kolumnach występujących w największej liczbie zapytań (na kolumnach *kol1* i *kol3*):

```
CREATE INDEX KOL1_KOL3_IDX ON TEST(KOL1,KOL3) NOLOGGING NOPARALLEL;
```

Stworzenie takiego indeksu nie skróciło czasu wykonywania dla zapytań 1 i 2. W obydwu przypadkach wystąpiło pełne czytanie tabeli (*TABLE FULL SCAN*). Natomiast w przypadku zapytania 3 nastąpiła istotna redukcja czasu wykonania od największego uzyskanego w poprzednich próbach aż o 40% W tabeli 4 przedstawiono czasy wykonywania zapytań dla próby 3.

Tabela 4

Czasy wykonywania zapytań po utworzeniu indeksu kompozytowego dla dwóch kolumn najczęściej występujących w zapytaniach testowych

Nr pomiaru	Zapytanie 1	Zapytanie 2	Zapytanie 3	Suma (Zapytania 1+2+3)
10	116 s	23 s	82 s	221 s
11	114 s	31 s	95 s	240 s
12	112 s	28 s	86 s	226 s

Po zastosowaniu takiego indeksu udało się uzyskać sumaryczny czas wykonania mniejszy od początkowego o 20% (zakładane 224,28 s, uzyskane 221 s), co dowodzi skuteczności me-

tody dla wybranego bloku zapytań SQL i nie wyczerpuje wszystkich możliwych struktur indeksowych. Nie jest to wymagane, ale może okazać się pomocne jako ustawienie bazowe dla kolejnej próby zmniejszenia całkowitego czasu wykonywania bloku zapytań SQL. Dzięki historii serii i czasu ich wykonania każda kolejna próba zmniejsza się o indeksy już raz odrzucone, co z kolei przyczynia się do redukcji kroków przy następnych próbach.

5. Zastosowanie metody

Proponowana metoda jest tylko wstępem do automatyzacji procesu opierającego się na większej ilości danych (tabel, kolumn i zastosowanych indeksów), przy precyzyjnym określeniu ograniczeń sprzętowo-programowych. Rozwiązanie jest niezależne od zastosowanej wersji i rodzaju relacyjnej bazy danych i cechuje się niezależnością platformową. Metoda ta może zostać użyta przy próbie ustawienia lepszych indeksów dla krytycznego bloku zapytań SQL w środowisku testowym i późniejszego wdrożenia jej w środowisku produkcyjnym. Zapamiętywanie serii zapytań z zastosowaniem najlepszego dla tej serii indeksu razem z całkowitym czasem wykonania może być punktem wyjściowym w przypadku zmieniającej się sekwencji zapytań SQL w badanym bloku. Proponowana metoda jest skalowalna: istnieje możliwość łączenia mniejszych bloków w większe serie zapytań SQL i opracowanie lepszego rozwiązania dla większego wycinka czasowego na podstawie historii serii i jednoczesnej eliminacji nieużytecznych indeksów dla nowego wycinka czasowego. Istnieje także możliwość podwyższenia wskaźnika celu, co może doprowadzić do próby znalezienia najlepszego indeksu, a nie tylko lepszego o ustalone wcześniej *T%*.

6. Podsumowanie i dalsze prace

Dobór indeksów dla tabel i wskazówek do zapytań w relacyjnej bazie danych odbywa się na zasadzie prób i błędów. Utworzony indeks, który ma zastosowanie dla jednego zapytania, często okazuje się całkowicie nieprzydatny dla innego zapytania lub wręcz jest szkodliwy (próba 2), co spowalnia proces przetwarzania zapytania do bazy danych. Istniejące i stosowane rozwiązania automatyzujące dobór indeksu opierają się na analizie częstości pobierania danych z kolumn, czyli tworzone są dla najczęściej odpytywanych kolumn, dla pojedynczego zapytania. Pomijany jest cały proces produkcyjny w wycinku czasowym. Jeżeli pojawi się zapytanie zawierające inny zestaw kolumn, indeks taki staje się zbędny. Następuje wtedy pełne czytanie całej tabeli, a nie tylko wymaganych danych, z wykorzystaniem indeksu. Jest to

sztywne i nieelastyczne rozwiązanie, angażujące programistę i zmuszające go do tworzenia nowych lub zmiany istniejących indeksów, które również mogą okazać się nieprzydatne.

Autor pracuje nad następującym rozwiązaniem automatycznego doboru indeksów: wykorzystującym algorytm genetyczny, analizującym czas wykonania bloku zapytań, proponującym zmiany i śledzącym wpływ propozycji zmian indeksów na sumaryczny czas wykonania zapytań w badanym wycinku czasowym.

BIBLIOGRAFIA

1. Gryglewicz-Kacerka W., Szymczak B.: Administracja bazą danych. Wydawnictwo Politechniki Łódzkiej, Łódź 2003.
2. Schkolnick M.: The Optimal Selection of Indices for Files. Information Systems, Vol. 1, 1975.
3. Hammer M., Chan A.: Index Selection in a Self-Adaptive DataBase Management Systems. SIGMOD'76 Proceedings of the 1976 ACM SIGMOD International Conference on Management of Data, New York 1976.
4. Finkelstein S., Schkolnick M.: Physical Database Design for Relational Databases. ACM-TOOLS, 1988.
5. Frank M., Omiecinski M.: Adaptive and Automated Index Selection in RDBMS. Proceedings of EDBT, 1992.
6. Chaudhuri S., Narasayya V.: An efficient Cost-Driven Index Selection Tool for MS SQL Server. Very Large Data Bases Endowment Inc, 1997.
7. Harrison G.: Algorytmy automatycznego doboru indeksów z możliwością zastosowania na wielu platformach. Oracle SQL High Performance Tuning, 1997.
8. Jurkowski W.: Metoda automatycznego doboru indeksów w systemach relacyjnych baz danych. Rozprawa doktorska, Politechnika Szczecińska, Wydział Informatyki, Szczecin 2003.
9. Kołaczkowski P., Rybiński H.: Automatic Index Selection in RDBMS by Exploring Query Execution Plan Space. Studies in Computational Intelligence, Vol. 223, Springer, 2009, s. 3÷24.
10. Kratica J., Ljubic I., Tosic D.: A Genetic Algorithm for the Index Selection Problem. EvoWorkshops'03 Proceedings of the 2003 International Conference on Applications of Evolutionary Computing, 2003.

Wpłynęło do Redakcji 8 stycznia 2012 r.

Abstract

Common index selection methods usually focus on choosing most frequently selected columns for standalone SQL query, omitting other queries present in the same processing block. Suggested solution not only overcomes this limitation but also is platform and database engine independent. Author's solution could potentially be used in an attempt to find better indexes for a critical part of long-running SQL queries in testing environment and for further production database implementation. Recording SQL series together with good indexes applied for these series and total execution time could be a starting point for an environment of changing SQL code within problematic block. Simple test presented in this article proves reasonableness of proposed method (results in Table 4). The suggested solution is scalable: there is a potentiality of combining smaller SQL blocks into larger series and finding better solution based on smaller blocks' history with exclusion of bad indexes. Author is currently working on index selection method with genetic algorithm use that will analyze SQL series, suggest indexes' improvements and track their influence on the series execution time.

Adres

Radosław BOROŃSKI: Politechnika Koszalińska, Wydział Elektroniki i Informatyki,
ul. Śniadeckich 2, 75-453 Koszalin, Polska, radoslaw.boronski@tu.koszalin.pl.