STUDIA INFORMATICA 2013
Volume 34 Number 2A (111)

Radosław BOROŃSKI
Politechnika Koszalińska, Wydział Elektroniki i Informatyki

INDEKSOWANIE TABEL DLA GRUPOWYCH ZAPYTAŃ SQL Z UWZGLĘDNIENIEM KRYTERIUM ROZMIARU

Streszczenie. Indeksowanie jest kluczowym elementem optymalizacyjnym systemów relacyjnych baz danych. Komercyjne narzędzia doboru indeksów (np. Toad, SQL Server Database Tuning Advisor) działają na podstawie metod przeznaczonych dla pojedynczych zapytań. W artykule przedstawiono podejście indeksowania tabel w ramach grupowych zapytań SQL uwzględniające kryterium rozmiaru indeksów. Przedstawione przykłady ilustrują, że zastosowanie podejścia grupowego pozwala zmniejszyć czas wykonania zapytań nawet o 30% w stosunku do rozwiązań uzyskanych klasycznymi metodami.

Słowa kluczowe: indeks, indeksowanie, optymalizacja, SQL, SZBD

INDEXING FOR TABLES OF GROUPED SQL QUERIES WITH SIZE CRITERION

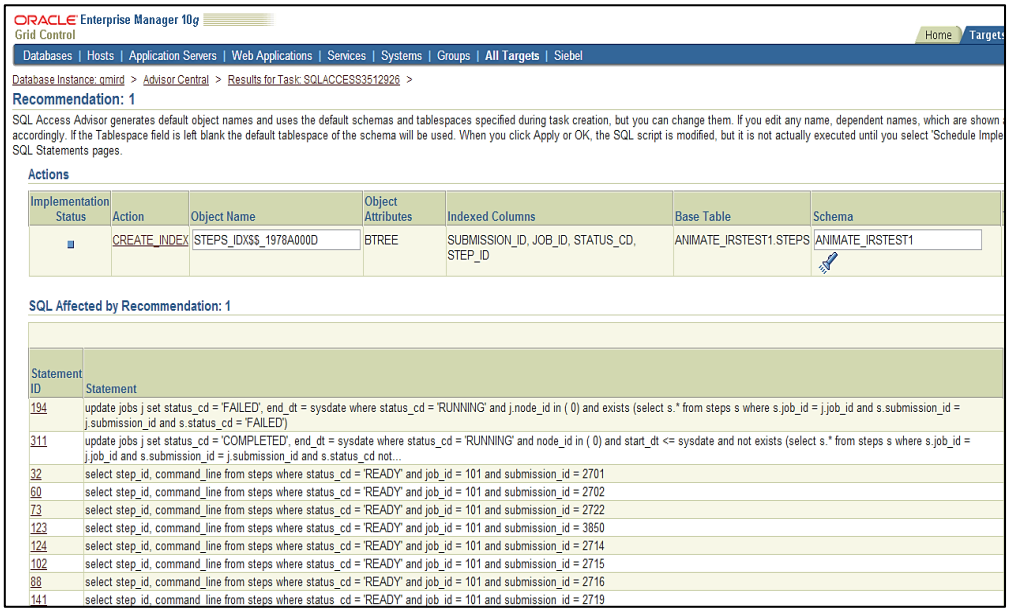
Summary. This paper discusses the problem of minimizing the response time for a given database workload by a proper choice of indexes. The main objective of our contribution is to illustrate the database queries as a group and search for good indexes for the group instead of an individual query, including the size criterion. Examples illustrate that the use of a group approach can reduce queries block execution time of 30% compared to classical methods.

Keywords: index, indexing, optimization, SQL, RDBMS

1. Wstęp

Poszukiwanie dobrych indeksów (tzn. minimalizujących czas wykonania zapytań SQL) dla dużych baz bądź hurtowni danych nigdy nie było zadaniem łatwym. Zazwyczaj użytkownicy i administratorzy baz danych polegają na swoim doświadczeniu, intuicji i dobrych praktykach, niekiedy posługują się narzędziami wspierającymi dobór indeksów. W powszechnym

użyciu są komercyjne aplikacje wspomagające proces doboru indeksów, takie jak Oracle SQL Access Advisor (Rys. 2) [6], Toad firmy Quest Software, SQL Server Database Tuning Advisor [1] czy DB2 Advisor [15].



Rys. 1. SQL Access Advisor firmy Oracle (wersja 10g2)
Fig. 1. Oracle's 10g2 SQL Access Advisor

Problem doboru indeksów był już niejednokrotnie omawiany w literaturze [3, 5, 8, 12, 14]. Formułowane były różne podejścia do optymalnego doboru indeksów dla pojedynczych zapytań [5, 8, 9]. Badania prowadzone w przeszłości potwierdzały skuteczność narzędzi używanych do doboru indeksów, jednak samemu pomysłowi poświęcono wtedy niewiele uwagi [13, 16]. W ciągu ostatnich lat temat doboru indeksów wrócił na łamy publikacji naukowych, a także zaproponowane zostały bardziej wyrafinowane rozwiązania [10, 11]. Chociaż propozycje te prezentują interesujące próby rozwiązania problemu doboru indeksów, nie są one wystarczające, aby móc zaimplementować je do rzeczywistych systemów produkcyjnych [14]. Powodem tego może być rozpatrywanie problemu doboru indeksów z pominięciem rozmiaru samego indeksu. Kolejną przeszkodą może być możliwość doboru indeksu wyłącznie dla pojedynczego zapytania. Pominięcie innych zapytań odnoszących się do tej samej tabeli może skutkować utworzeniem zbyt dużej liczby podobnych indeksów. Powyższe ograniczenia mogą mieć zasadnicze znaczenie w produkcyjnych systemach baz danych, dla których czas odpowiedzi, a także oszczędność przestrzeni dyskowej są bardzo ważne [10, 12].

Rozdział 2 przedstawia sformułowanie problemu doboru indeksów dla relacyjnych baz danych. Klasyczną metodę doboru indeksów, razem z prostym przykładem przybliżającym zagadnienie, przedstawiono w rozdziale 3. Rozdział 4 omawia nową metodę doboru indek-

sów dla grupy zapytań. Wyniki testów porównane są z wynikami metody klasycznej. Rozdział 5 podsumowuje zagadnienie i przedstawia kierunek dalszych badań.

2. Sformułowanie problemu

Głównym celem artykułu jest ilustracja podejścia poszukiwania indeksów dla grupowych zapytań SQL oraz wskazanie, że w pewnych szczególnych przypadkach podejście to może skutkować lepszymi rozwiązaniami niż podejście klasyczne, polegające na traktowaniu każdego z zapytań w grupie osobno.

Rozważany problem znany jest w literaturze jako Problem Doboru Indeksów (Index Selection Problem (ISP)). W klasycznym ujęciu, problem polega na poszukiwaniu zbiorów indeksów minimalizujących czas odpowiedzi na zadane zapytanie SQL. Jak się okazuje [7], ISP należy do klasy problemów NP-trudnych. Należy podkreślić, że w praktyce dobór wielkości przestrzeni dyskowej w ISP jest problemem typu P (o wielomianowej złożoności obliczeniowej), ponieważ rozmiar bazy danych zazwyczaj rośnie liniowo. Wobec tego pewne zasoby dyskowe w takim rozumowaniu pozostają wolne do wykorzystania na potrzeby ISP [10].

Kryterium czasu nie jest jednak jedynym kryterium doboru indeksów. W praktyce wielokrotnie dochodzi od sytuacji, gdy indeksy podlegają ocenie pod względem rozmiaru zajmowanej przestrzeni dyskowej. Dodatkowo, mechanizmy doboru indeksów przeznaczone są głównie dla pojedynczych zapytań. Rozmiar indeksów oraz możliwość analizy grupowych zapytań są szczególnie istotne w bazach danych systemów wspomagania produkcji. Bazy danych tego typu charakteryzują się koniecznością wykonania tysięcy zapytań SQL (rys. 2), zawierających referencje do setek tabel, na które składają się dziesiątki kolumn. Przestrzeń przeszukiwania indeksów jest bardzo duża i rośnie wykładniczo ze wzrostem danych wejściowych. Jak zauważono na wstępie, rozmiar indeksów ma istotne znaczenie w praktyce.

W tym ujęciu problem rozważany w artykule stanowi rozszerzanie ISP o elementy związane z rozmiarem indeksów oraz możliwość rozpatrywania zapytań grupowych. Problem tego typu jest definiowany następująco:
Dany jest zbiór tabel:

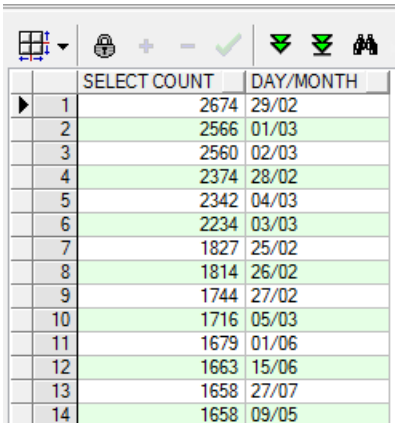
$$T = (T_1, \ldots, T_i, \ldots, T_n), \tag{1}$$

charakteryzowany przez zbiór kolumn wchodzących w skład tych tabel:

$$K = \{k_{1,1}, \ldots, k_{1,l(1)}, \ldots, k_{i,j}, \ldots, k_{n,1}, \ldots, k_{n,l(n)}\}, \tag{2}$$

gdzie $k_{i,j}$ – j-ta kolumna tabeli T_i.

Każdej kolumnie $k_{i,j}$ odpowiada zbiór wartości $V(k_{i,j})$ (zbiór komórek) wchodzących w skład tej kolumny.



	SELECT COUNT	DAY/MONTH
1	2674	29/02
2	2566	01/03
3	2560	02/03
4	2374	28/02
5	2342	04/03
6	2234	03/03
7	1827	25/02
8	1814	26/02
9	1744	27/02
10	1716	05/03
11	1679	01/06
12	1663	15/06
13	1658	27/07
14	1658	09/05

Rys. 2. Przykład liczby zapytań SQL do produkcyjnej bazy danych dla danego dnia
Fig. 2. Example of number of database queries in a given day for a production data warehouse

Dla zbioru tabel T mogą być formułowane różne zapytania Q_i (w SQL są to zapytania typu SELECT). Zapytania te stawiane są względem zadanego zbioru kolumn $K^* \subseteq K$. Wynikiem zapytania Q_i jest zbiór:

$$A_i \subseteq \prod_{k_{i,j} \in K^*} V\left(k_{i,j}\right), \tag{3}$$

gdzie $\prod_{i=1}^{n} Y_i = Y_1 \times Y_2 \times \ldots \times Y_n$ – iloczyn kartezjański zbiorów $Y_1, \ldots, Y_n$.

Dla zadanej bazy danych DB przyjmuje się, że A_i jest wynikiem następującej funkcji:

$$A_i = Q_i\left(K_i^*, Op(DB)\right), \tag{4}$$

gdzie: K_i^* – podzbiór kolumn wykorzystanych w ramach zapytania Q_i, $Op(DB)$ – zbiór operatorów dostępnych w bazie DB, z których zbudowana jest relacja określająca zapytanie Q_i.

Czas związany z wyznaczeniem zbioru A_i jest zależny od wykorzystywanej bazy DB (algorytmów przeszukiwania, struktury indeksów itp.) oraz przyjętego zbioru indeksów $J \subseteq P\left(K^*\right)$ (gdzie $P\left(K^*\right)$ – zbiór potęgowy K^*). W ogólnym przypadku przyjmuje się, że czas wykonywania zapytania Q_i w zadanej bazie DB jest definiowany jako: $t(Q_i, J, DB)$. Ze względu na to, że rozważania dotyczą wyłącznie wpływu indeksów J (warunki pracy bazy określone w ramach DB uznawane są za niezmienne), czas wykonania zapytania Q_i będzie definiowany jako: $t_i(J)$.

Ponadto, zbiór indeksów $J \subseteq P\left(K^*\right)$, a także tabele wykorzystywane w ramach Q_i charakteryzowane są odpowiednio przez rozmiary wykorzystywanej pamięci, $S(J)$ – rozmiar indeksów, $S(Q_i)$ – rozmiar indeksowanych tabel (tabel użytych w ramach Q_i). Stosunek tych wielkości określa współczynnik wykorzystania przestrzeni dyskowej indeksów J względem Q_i: $m\left(J, Q_i\right) = \dfrac{S(J)}{S(Q_i)}$.

W kontekście tak zdefiniowanych parametrów typowy problem ISP wiąże się z odpowiedzią na pytanie:

Jaki zbiór indeksów $J \subseteq P(K^*)$ *minimalizuje czas wykonania zapytania* $Q_i : t_i(J) \to min?$

Rozszerzenie problemu o możliwość uwzględniania wieloelementowych zbiorów zapytań $Q = \{Q_1, \ldots, Q_m\}$ oraz kryterium rozmiaru indeksów prowadzą do pytania o następującej postaci:

Jaki zbiór indeksów $J \subseteq P(K^*)$*, spełniający warunek:* $m(J, Q_i) < m_h$ *(współczynnik wykorzystania przestrzeni dyskowej indeksów jest mniejszy niż zadana wartość* m_h*), minimalizuje czas wykonania zbioru zapytań* $Q : \sum_{Q_i \in Q} t_i(J) \to min?$

3. Klasyczna metoda doboru indeksów

Klasyczny dobór indeksów opiera się na pojedynczym zapytaniu SQL i próbie znalezienia zbioru indeksów dla kolumn występujących w tabelach w pojedynczym zapytaniu. Podejście takie pomija występowanie grupy zapytań jako całości. Każde z zapytań występujących w zbiorze zapytań traktowane jest indywidualnie. Wadą tej metody jest możliwość wyznaczenia nadmiernej liczby indeksów. Dodatkowo, mogą zostać utworzone podobne indeksy, bądź indeksy, które zaspokoją potrzeby tylko jednego zapytania z całej rozważanej grupy zapytań SQL. Może to doprowadzić do sytuacji, w której marnowane są zasoby dyskowe i czas potrzebny na utworzenie nadmiarowej liczby indeksów.

Rozważmy przykład, w którym dana jest grupa trzech zapytań SQL $Q = \{Q_1, Q_2, Q_3\}$:

```
Q1: SELECT * FROM T1, T2 WHERE k1,1 > k2,2 AND k1,3=[const],
Q2: SELECT * FROM T2, T3 WHERE k2,2 = k3,2,
Q3: SELECT * FROM T2 WHERE k2,1 < [const].
```

Interpretacja tego typu zapytań (zgodnie z (4)) jest następująca:

Q_1 – zapytanie SELECT, w ramach którego poszukiwany jest zbiór: $A_1 = \{(a,b,c) : a \in V(k_{1,1}), b \in V(k_{2,2}), c \in V(k_{1,3}); a > b, c = [const]\}$, gdzie $K_1^* = \{k_{1,1}, k_{2,2}, k_{1,3}\}$ – zbiór kolumn wykorzystywanych w zapytaniu.

Q_2 – zapytanie SELECT, w ramach którego poszukiwany jest zbiór: $A_2 = \{(a,b) : a \in V(k_{2,2}), b \in V(k_{3,2}); a = b\}$, gdzie $K_2^* = \{k_{2,2}, k_{3,2}\}$ – zbiór kolumn wykorzystywanych w zapytaniu.

Q_3 – zapytanie SELECT, w ramach którego poszukiwany jest zbiór: $A_3 = \{a : a \in V(k_{2,1}); a < [const]\}$, gdzie $K_3^* = \{k_{2,1}\}$ – zbiór kolumn wykorzystywanych w zapytaniu.

Tabele T_1, T_2, T_3 zawierają $1*10^7$ rekordów każda. Całkowity rozmiar tabel wynosi 960 MB. Dla tak określonego zbioru tabel i grupy zapytań Q poszukiwany jest zbiór indek-

sów J, minimalizujący czas odpowiedzi $\sum_{Q_i \in Q} t_i(J) \rightarrow min$. Dodatkowo zakłada się, że współczynnik wykorzystania przestrzeni dyskowej indeksów J względem grupy zapytań Q nie może być większy niż 0,5: $m(J, Q) < 0{,}5$ (oznacza to, że indeksy nie mogą przekraczać 50% sumarycznego rozmiaru tabel T_1, T_2, T_3). W praktyce administratorzy baz danych zazwyczaj nie tworzą indeksów większych niż połowa rozmiaru indeksowanej tabeli. W pierwszej kolejności rozważono sytuację, w której zbiór indeksów jest zbiorem pustym: $J = \emptyset$. W takim przypadku czasy uzyskania odpowiedzi kolejno dla zapytań Q_1, Q_2, Q_3 są następujące: $t_1(J) = 2040$s, $t_2(J) = 3611$s, $t_3(J) = 345$s. Brak indeksów powoduje, że w celu uzyskania odpowiedzi konieczne jest pełne czytanie każdej z tabel (FULL TABLE SCAN) [4]. Zaprezentowane doświadczenia przeprowadzono na bazie danych Oracle 11.2.0.3, zainstalowanej na serwerze z systemem operacyjnym Redhat Enterprise Linux 6, z 64GB pamięci operacyjnej i systemem ASM wykorzystanym do obsługi macierzy dyskowej SAN.

W kolejnym etapie wyznaczone zostały czasy odpowiedzi $t_1(J)$, $t_2(J)$, $t_3(J)$ dla zbioru indeksów J uzyskanych w klasycznym podejściu. Podejście to charakteryzuje się tym, że każde z zapytań SQL traktowane jest indywidualnie. Uzyskane indeksy mają następującą postać: $k_{1,1}$ i $k_{1,3}$ na tabeli T_1; $k_{2,1}$, $k_{2,2}$ na tabeli T_2; $k_{3,2}$ na tabeli T_3. Indeksy te reprezentowane są przez zbiór: $J = \{\{k_{1,1}, k_{1,3}\}, \{k_{2,2}\}, \{k_{3,2}\}, \{k_{2,1}\}\}$, zawierający cztery elementy. Każdy element (podzbiór) zbioru J zawiera te kolumny, z których (na podstawie mechanizmu Oracle SQL Advisor, wersja 11.2.0.3) zbudowane zostaną indeksy bazy danych. Przykładowo, zbiór $\{k_{1,1}, k_{1,3}\}$ oznacza, że zbudowany zostanie jeden indeks na kolumnach $k_{1,1}, k_{1,3}$.

Utworzony zbiór indeksów J zajmuje 510 MB dodatkowej przestrzeni dyskowej $m(J, Q) = 0{,}53$. Uzyskane czasy odpowiedzi są następujące: $t_1(J) = 2612$s, $t_2(J) = 2580$s, $t_3(J) = 5$s.

O ile czas odpowiedzi jest lepszy o około 10%, o tyle rozwiązanie nie spełnia kryterium współczynnika wykorzystania przestrzeni dyskowej indeksów $m(J, Q) > 0{,}5$.

Użycie przez silnik bazy danych zbioru indeksów J powoduje, że czas odpowiedzi dla zapytania Q_1 zwiększył się. Wynikało to z konieczności przeczytania całego indeksu dla kolumny $k_{1,1}$, a następnie, z powodu braku występowania w tym indeksie danych dla kolumny $k_{1,3}$, czytania całej tabeli T_1. Silniki baz danych nie są w stanie posłużyć się dwoma różnymi indeksami dla jednej tabeli w zakresie jednego zapytania niezawierającego podzapytań bądź złączeń z tą samą tabelą.

Przedstawiony przykład pokazuje, że można spotkać się z zapytaniami, dla których klasyczny dobór indeksów (implementowany w tradycyjnych środowiskach produkcyjnych) może skutkować opóźnieniem odpowiedzi bazy danych spowodowanym zbyt dużym rozmiarem indeksów. W skrajnych przypadkach indeksy mogą wymusić czas odpowiedzi większy niż w przypadku ich braku ($J = \emptyset$) ze względu na dwustopniowe czytanie: indeksu i tabeli.

4. Indeksowanie grupowych zapytań

Grupowa metoda doboru indeksów opiera się na wzajemnej relacji grupy zapytań SQL. Przyjmuje się, że zapytania są ze sobą w relacji, jeśli odnoszą się do kolumn z tych samych tabel. Indeksowanie grupy Q, w której zapytania powiązane są ze sobą przez odwołania do wspólnych tabel, może prowadzić do indeksów, które jednocześnie obejmują kilka zapytań. W odróżnieniu od klasycznych metod proponowane podejście uwzględnia poszukiwanie indeksów dla wszystkich zapytań jednocześnie. Umiejętność wyznaczania indeksów dla grup zapytań pozwala uniknąć nadmiarowości indeksów i tym samym ograniczyć czas uzyskania odpowiedzi (znaczenie umiejętnego indeksowania było rozważane m.in. w [3]). Korzyści związane z wyznaczeniem indeksów grupowych zostały przedstawione na poniższych przykładach.

Analizowana w poprzednim przykładzie grupa Q zawiera zapytania powiązane ze sobą przez wspólny zbiór tabel T_1, T_2, T_3. Niech KW będzie sekwencją par $(k_{i,j}, c_{i,j})$ określających liczbę wystąpień $c_{i,j}$ każdej kolumny $k_{i,j}$ w zapytaniach Q. W rozważanym przypadku sekwencja KW ma postać:

$$KW = \left((k_{1,1},1), (k_{1,3},1), (k_{2,1},1), \boxed{(k_{2,2},2)}, (k_{3,2},1) \right). \tag{5}$$

Przykładowo, dla kolumny $k_{2,2}$ (oznaczona w (5) ramką) wartość $c_{2,2} = 2$, co oznacza, że kolumna $k_{2,2}$ występuje dwukrotnie w zapytaniach grupy Q (zapytania Q_1 i Q_2). Częstość występowania kolumny w bloku zapytań może zostać wykorzystana przy doborze indeksów J. Prz założeniu, że indeksy konstruowane są przy uwzględnieniu częstości występowania kolumn $k_{i,j}$, zbiór indeksów J dla zadanej grupy Q ma postać: $J = \{\{k_{2,1}, k_{2,2}\}\}$. Zbiór J składa się z jednego indeksu kompozytowego, który zbudowany jest dla kolumn $k_{2,1}$ i $k_{2,2}$ tabeli T_2 ($k_{2,2}$ wykorzystano ze względu na maksymalną liczbę wystąpień $c_{2,2}$, $k_{2,1}$ wykorzystano ze względu na przynależność do tej samej tabeli co $k_{2,2}$). Utworzony indeks zajmuje 184 MB dodatkowej przestrzeni dyskowej ($m(J, Q) = 0{,}19$). Takie rozwiązanie nie tylko przyspiesza wykonywanie bloku zapytań Q, lecz także znacznie redukuje ilość miejsca na dysku, potrzebnego do utworzenia indeksu. Przy trzeciej próbie pomiarowej baza danych zwraca następujące czasy odpowiedzi: $t_1(J) = 1235$s, $t_2(J) = 2430$s, $t_3(J) = 5$s, zmniejszając nie tylko sumaryczny czas wykonania bloku zapytań o 35%, lecz także oszczędzając miejsce na dysku. Rozwiązanie spełnia kryterium współczynnika wykorzystania przestrzeni dyskowej indeksów $m(J, Q) > 0{,}5$ (19%) i jest o 34% lepsze od indeksu przedstawionego w poprzednim przypadku. Wynik taki jest następstwem czytania tylko indeksu dla indeksowanej tabeli Q_2 i pełnego czytania nieindeksowanej tabeli Q_1. Oznacza to, że optymalizator kosztowy nie wykonuje dodatkowej operacji czytania osobno dla indeksu i osobno dla tabeli. Wobec powyższego należy stwierdzić, że indeksy powinny być dobierane z rozwagą.

Przedstawiony przykład pokazuje, że czas odpowiedzi bazy danych dla grupy zapytań Q może być pomniejszony w przypadku, gdy przy doborze indeksów uwzględnia się całą grupę zapytań, a nie pojedyncze zapytanie (jak to ma miejsce w metodzie klasycznej).

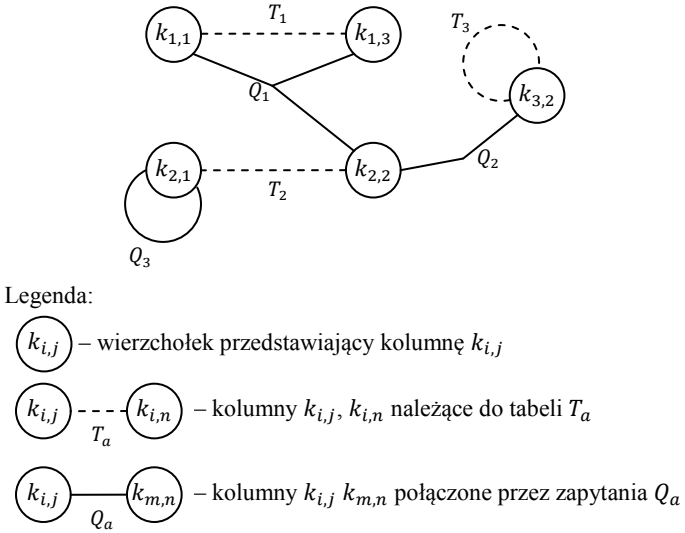
Należy zwrócić uwagę na to, że poszukiwanie indeksów dla grupowych zapytań ma sens wtedy, kiedy zapytania w grupie spełniają warunek wzajemnej zależności. Zapytania Q_1, Q_2, Q_3 z poprzedniego przykładu są zależne.

Wzajemna zależność zapytań grup Q może być graficznie zilustrowana w postaci hipergrafu $G(Q)$, którego zbiór krawędzi determinowany jest przez postać Q.

Przykład hipergrafu rozważanej grupy zapytań Q przedstawiony został na rys. 3. Dla grafu tego typu wierzchołki przedstawiają kolumny użyte w zapytaniach Q, krawędzie natomiast łączą kolumny należące do wspólnej tabeli T_a (linia przerywana hiperkrawędzi) oraz należące do wspólnych zapytań Q_i (linia ciągła hiperkrawędzi). Na przykład hiperkrawędź łącząca wierzchołki $k_{1,1}$, $k_{2,2}$, $k_{1,3}$ przedstawia relację w zapytaniu Q_1.

Przyjmuje się, że zbiór zapytań Q jest w relacji, jeżeli odpowiadający mu hipergraf G(Q) jest spójny.

Omawiane przykłady pozwalają sądzić, że indeksowanie grupy zapytań Q będzie miało przewagę nad indeksowaniem klasycznym tylko dla zapytań relacyjnych.



Rys. 3. Hipergraf dla grupy zapytań Q
Fig. 3. Hypergraph for considered set of queries Q

Jako kontrprzykład można przedstawić następującą grupę zapytań SQL: $Q^* = \{Q_1^*, Q_2^*, Q_3^*\}$:

Q_1^*: SELECT * FROM T_1, T_2 WHERE $k_{1,1}$ > $k_{1,2}$,
Q_2^*: SELECT * FROM T_2, T_3 WHERE $k_{2,1}$ = $k_{3,2}$,
Q_3^*: SELECT * FROM T_4 WHERE $k_{4,1}$ > [const].

Przykład hipergrafu dla rozważanej grupy zapytań Q został przedstawiony na rys. 4.

Przedstawiony hipergraf jest niespójny. W tym kontekście zapytania Q^* uznawane są za niepowiązane (bez relacji).

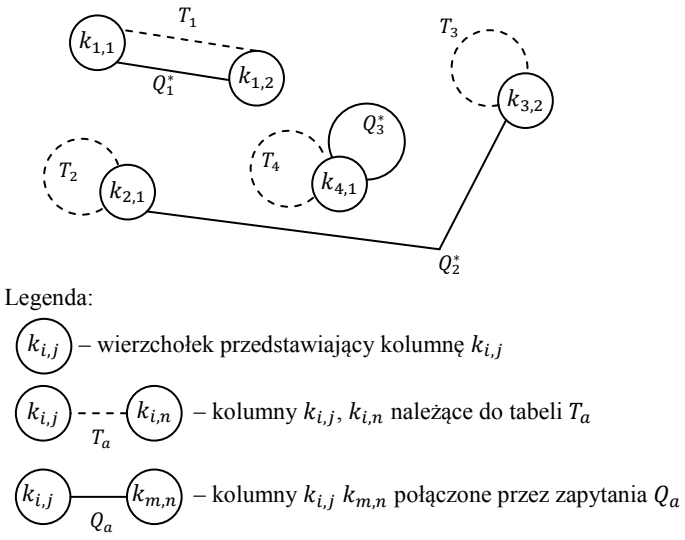
Niepowiązane zapytania w procesie doboru indeksów nie mogą być traktowane jako grupa. W takim przypadku najlepsze indeksy określane są dla każdego z zapytań z osobna:

$$J^* = \{\{k_{1,1}, k_{1,2}\}, \{k_{2,1}\}, \{k_{3,2}\}, \{k_{4,1}\}\}. \quad 6)$$

Sekwencja KW^* grupy zapytań Q^* ma postać:

$$KW^* = \big((k_{1,1},1),(k_{1,2},1),(k_{2,1},1),(k_{3,2},1),(k_{4,1},1)\big). \quad (7)$$

Należy zauważyć, że dla elementów grupy (kolumn kandydackich) KW^* nie występują powiązania pomiędzy zapytaniami (liczba wystąpień dla każdej z kolumn jest równa 1). Wobec powyższego kolumny te nie mogą być brane pod uwagę jako przydatne do indeksowania w relacyjnej grupie zapytań. W takim przypadku każda z tabel T_i będzie indeksowana indywidualnie dla każdego z zapytań Q^*.



Rys. 4. Hipergraf dla grupy zapytań Q*
Fig. 4. Hypergraph for considered set of queries Q*

5. Podsumowanie

Poszukiwanie dobrego indeksu dla tabeli jest odpowiedzią na potrzebę minimalizacji czasu wykonania zapytania i kosztu związanego z utworzeniem indeksu. Indeksy mogą przyspieszyć wykonanie zapytania w relacyjnej bazie danych, lecz znalezienie takich, które jednocześnie zajmą mało miejsca na dysku, często jest zadaniem trudnym dla rozbudowanych systemów bazodanowych.

Przedstawione przykłady pokazują, że istnieje potrzeba opracowania mechanizmu doboru indeksów zorientowanego na grupę zapytań, a nie na pojedyncze zapytanie. Jest to szczegól-

nie istotne dla cyklicznie powtarzających się grup zapytań w produkcyjnych bazach danych. Praktyka pokazuje, że dla powtarzającej się grupy dużej liczby zapytań, na które składa się duża liczba tabel, podejście grupowe może wypracować lepsze rezultaty niż metoda klasycznego doboru indeksów. Może to skutkować nie tylko lepszym czasem odpowiedzi z bazy danych, lecz także i zaoszczędzonym miejscem w często ograniczonych zasobach dyskowych. W przykładzie pokazano istotę problemu, w którym grupowe podejście ma sens dla zapytań, między którymi istnieje warunek wzajemnej zależności (zapytania Q_1 i Q_2). Dodatkowo, wprowadzenie współczynnika wykorzystania przestrzeni dyskowej pozwala zaoszczędzić miejsce na dysku.

Przedstawioną metodę grupowych zapytań należy zweryfikować pod względem efektywności i wiarygodności przez serię badań numerycznych dla różnych danych wejściowych.

Należy zauważyć, że przytoczony przykład i zaproponowana metoda doboru indeksów biorą pod uwagę czas przetwarzania bloku zapytań, a także rozmiar indeksu. W ogólnym przypadku należy również rozważyć dodatkowe aspekty mające wpływ na te dwa czynniki, takie jak: czas potrzebny na utworzenie indeksu, koszt jego utrzymania (aktualizacja indeksu), a także czas jego aktualizacji. Przyszłe badania autora będą brały te aspekty pod uwagę.

Przedstawione badania będą kontynuowane w kierunku pracy nad systemem automatycznego doboru indeksów dla zapytań grupowych z wykorzystaniem algorytmu genetycznego [2]. System taki będzie analizował grupę zapytań SQL, proponował indeksy dla tabel w grupie, a także śledził czas wykonywania zapytań.

BIBLIOGRAFIA

1. Agrawal S., Chaudhuri S., Kollar L., Marathe A., Narasayya V., Syamala M.: Database Tuning Advisor for Microsoft SQL Server 2005. Proceedings of the 30th International Conference on Very Large Databases, 2004.
2. Back T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press Oxford, UK 1996.
3. Boroński R.: Automatyzacja i optymalizacja procesu doboru indeksów dla dowolnego wycinka czasowego w relacyjnej bazie danych (na przykładzie Oracle 11g). Studia Informatica, Vol. 33, No. 2A (105), Gliwice 2012, s. 229.
4. Boroński R.: Wpływ ustawień parametru wieloblokowego sekwencyjnego czytania danych na czas wykonania zapytania SQL w bazie danych Oracle. Materiały VI Krajowej Konferencji Naukowej Infobazy 2011, Gdańsk 2011, s. 135.
5. Chaudhuri S., Narasayya V.: An efficient Cost-Driven Index Selection Tool for MS SQL Server. Very Large Data Bases Endowment Inc, 1997.

6. Dageville B., Das D., Dias K., Yagoub K., Zait M. Ziauddin M.: Automatic SQL Tuning in Oracle 10g. Proceedings of the 30th International Conference on Very Large Databases, 2004.
7. Finkelstein S., Schkolnick M., Tiberio P.: Physical database design for relational databases. ACM Trans. Database Syst, 13(1), 1988, s. 91÷128.
8. Frank M, Omiecinski M.: Adaptive and Automated Index Selection in RDBMS. Proceedings of EDBT, 1992.
9. Gupta H., Harinarayan Y., Rajaraman A., Ullman J.D.: Index Selection for OLAP. Proceedings of the Internatoinal Conference on Data Engineering, Birmingham 1997, s. 208÷219.
10. 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
11. 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.
12. Maggie Y., Ip L., Saxton L. V., Raghavan V.: On the Selection of an Optimal Set of Indexes. IEEE Transactions on Software Engineering, 9(2), 1983, s. 135÷143.
13. Schkolnick M.: The Optimal Selection of Indices for Files. Information Systems, Vol. 1, 1975.
14. Schnaitter K.: On-line Index Selection for Physical Database Tuning. ProQuest, UMI Dissertation Publishing, 2011.
15. Valentin G., Zuliani M., Zilio D., Lohman G.: DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. Proceeding ICDE '00, Proceedings of the 16th International Conference on Data Engineering, s. 101.
16. Wedekind H.: On the selection of access paths in a data base system, [in:] Klimbie J. W., Koffeman K. L. (eds.): Data Base Management. North-Holland, Amsterdam 1974, s. 385÷397.

Wpłynęło do Redakcji 9 stycznia 2013 r.

Abstract

Finding a good index for a database table is crucial for every relational database system, not only from the productivity point but also because of the cost aspect. Index may be important for a relational database to process queries with reasonable efficiency, but the selec-

tion of a good index may be difficult. Presented examples show that there is a need for finding an automatic index selection mechanism for grouped queries. Practice shows that in some circumstances an index focused on grouped queries may give better results and also enables user to save time needed for index creation. It also saves system hardware resources (disk space). In the examples it is shown that grouped queries indexes are more effective than those obtained by classical methods because queries Q_1 and Q_2 satisfy the relation condition.

Adres

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