

VI Seminarium PLOUG
Warszawa
Styczeń 2003

Kiedy tworzymy indeksy?

Wojciech Karwowski

Oracle Polska

Na podstawie artykułu „When to create index”, Cary Millsap

Cary Millsap pracuje od 10 lat w firmie Oracle Corporation jako jeden z głównych specjalistów ds. wydajności systemów. Autor wynalazł Optymalnie Elastyczną Architekturę (Optimal Flexible Architecture OFA)

Tłumaczenie i redakcja:

Wojciech Karwowski; Wojciech Karwowski jest pracownikiem firmy Oracle Polska od 6 lat. Jako ISV Channel Consultant zajmuje się wsparciem technologicznym partnerów firmy Oracle. Jest ekspertem od zagadnień wydajnościowych. Wykorzystano materiały Oracle Corporation oraz Hotsos Enterprises Ltd.

Wprowadzenie

Kiedy powinno się tworzyć indeks? Od grubo ponad dekady programiści aplikacji Oracle, kiedy decydowali o tym, czy tworzyć nieunikalny indeks, stosowali prostą, praktyczną zasadę. Jednak pracując u klientów, zdarza nam się trafiać na problemy z wydajnością spowodowane właśnie stosowaniem tej zasady. W niniejszym tekście zostaną przedstawione następujące tezy:

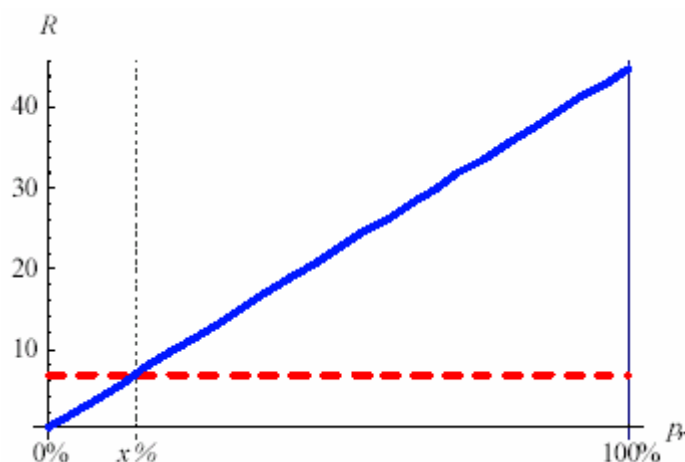
- Praktyczna zasada nie sprawdza się, jeśli decyzję o tym, czy tworzyć indeks, podejmuje się na podstawie punktu zrównania, uzależnionego od selektywności wierszy.
- Indeks może znacznie przyspieszyć działanie bazy nawet w przypadku zapytań na tabeli z jednym wierszem.
- Czynnikiem, który powinien decydować o stosowaniu indeksu, jest selektywność *bloków*, a nie *wierszy*.
- Selektywność bloków przy danym predykcji dyrektywy *where* można określić za pomocą kodu SQL zamieszczonego w niniejszym artykule.
- Wartości kolumn często są naturalnie pogrupowane lub jednolite. Posiadając taką informację, można podjąć lepszą decyzję o tworzeniu indeksu.
- Wiele nowych funkcji bazy Oracle upraszcza zapisywanie danych w porządku fizycznym, a taki zapis pozwala na zbudowanie bardzo szybkiej bazy.

Kiedy tworzyć indeks – sposób tradycyjny

Przynajmniej od czasów Oracle w wersji 5 przy podejmowaniu decyzji o tym, czy tworzyć indeks, stosuje się standardową wskazówkę:

Indeks twórz wtedy, gdy dane zapytanie zwraca mniej niż x % wierszy tabeli.

Na rysunku nr 1 zilustrowano tę zasadę, według której pewna wartość procentowa x jest „punktem zrównania” wydajności bazy Oracle przy skanowaniu indeksu i skanowaniu całej bazy. Na wykresie widać zależność czasu reakcji R (zazwyczaj wyrażanego w sekundach) do proporcji p , wierszy tabeli zwracanych w wyniku danego zapytania.



Rysunek 1. Czas reakcji R w sekundach jako funkcja zwracanego procentu p_r wierszy tabeli. Linią kreskowaną na wysokości $R = 6,75$ (na kolorowym wydruku linia ta jest czerwona) oznaczono czas reakcji w przypadku konieczności przeskanowania całej tabeli. Linią ciągłą (niebieską) oznaczono czas reakcji skanowania przedziału indeksu zwracającej procent p_r wierszy danej tabeli.

Czas reakcji w przypadku planu wykonania zwracającego r wierszy przez skanowanie całej tabeli jest mniej więcej stały, bez względu na to, czy r oznacza jeden wiersz, czy wszystkie wiersze tabeli. Jednak czas reakcji z użyciem skanowania indeksu wzrasta w miarę wzrastania źródłowego zbioru wierszy¹ tabeli. Wartość procentowa $p_r = x$ jest punktem zrównania — wartością p_r , dla której czas reakcji przy skanowaniu całej tabeli i czas reakcji przy skanowaniu przedziału indeksu są identyczne. Dla wartości $p_r < x$ skanowanie przedziału indeksu trwa krócej. Dla wartości $p_r > x$ krócej trwa skanowanie całej tabeli.

Jednak metoda ta ma pewną istotną wadę. Żadna praktyczna zasada dotycząca indeksowania nie sprawdza się, jeśli u jej podstawy leży pewna wartość procentowa — taki „punkt zrównania” jak x .

Dlaczego nie należy polegać na praktycznej zasadzie

Praktyczne zasady, takie jak ta, że „indeks należy tworzyć wtedy, gdy zapytanie zwraca mniej niż x % wierszy tabeli”, oparte są na poniżej podanych obserwacjach:

1. Jeśli zbiór źródłowy będący wynikiem zapytania zawiera tylko jeden wiersz, skanowanie przedziału indeksu trwa krócej, niż skanowanie całej tabeli.

¹ Zbiór źródłowy wierszy (row source) jest po prostu podzbiorem wierszy tabeli (także wszystkich wierszy, jak w przypadku skanowania całej tabeli). Wynik operacji zapytania określa się często mianem *wynikowego zbioru źródłowego wierszy* lub *zbioru wynikowego*. W przypadku złożonego planu wykonywania w bazie Oracle, potomny plan wykonywania przekazuje zbiór wynikowy planowi macierzystemu, który dalej go przetwarza (np. wykonuje operację połączenia lub filtrowania).

2. Jeśli zbiór źródłowy będący wynikiem zapytania zawiera wszystkie wiersze tabeli, skanowanie całej tabeli trwa krócej niż skanowanie przedziału indeksu.
3. A więc musi istnieć pewna wartość procentowa — punkt zrównania — wszystkich wierszy tabeli, przy której szybkość uzyskania zbioru źródłowego przez skanowanie przedziału indeksu jest równa szybkości uzyskania zbioru źródłowego przez skanowanie całej tabeli. W przypadku zapytań zwracających mniej wierszy niż punkt zrównania skanowanie przedziału indeksu jest szybsze. W przypadku zapytań zwracających więcej wierszy niż punkt zrównania szybsze jest skanowanie całej tabeli.

Nasze testy i doświadczenie zdobyte w pracy z klientami wykazały, że stwierdzenie pierwsze jest prawdziwe nawet w przypadku bardzo małych tabel. Zapytanie zwracające jeden wiersz jest wydajniejsze, gdy wykonuje się je przez indeks niż przez skanowanie całej tabeli, nawet jeśli ta tabela zawiera tylko jeden wiersz. Wielu osobom, z którymi omawialiśmy ten problem, takie działanie wydaje się zaskakujące. Jest ono również sprzeczne z zasadą samej firmy Oracle, mówiącą, że „w małych tabelach nie są wymagane indeksy” [Oracle 2001a]. Mała tabela może nie *wymagać* indeksu, ale indeks może bardzo przyspieszyć działanie takiej bazy i tym samym uczynić ją bardziej skalowalną.²

Tak więc zgadzamy się z twierdzeniem pierwszym. Jednak drugie jest problematyczne. Czasem o *wiele* mniej zasobów pochłania odczytanie 100% wierszy tabeli przez indeks, niż odczytywanie ich przez skanowanie całej tabeli.

Przykład: Wyobraźmy sobie tabelę o nazwie `interfejs`, której rozmiar maksymalny (*high-water mark*) wynosi 10 000 bloków. Kiedyś w tabeli `interfejs` zawarte były tysiące wierszy, dziś jednak jest ich tylko 100. Wiersze te rozrzucone są po 30 blokach tabeli. Załóżmy, że w tabeli jest kolumna zawierająca klucz podstawowy, `id`, z którą oczywiście skojarzony jest indeks `id_u1`. Na takiej tabeli wykonujemy następujące zapytanie:

```
select id, data, status from interfejs i;
```

Jeśli zapytanie to wykonamy przez skanowanie całej tabeli, będzie to wymagało wykonania przez bazę Oracle 10 000 operacji LIO. Zapytanie możemy jednak nieco zmienić, powodując, że będzie realizowane przez indeks. Jeśli `id` jest kolumną zawierającą tylko nieujemne liczby całkowite, następujące zapytanie spowoduje uzyskanie pożądanego zbioru wierszy przez indeks:

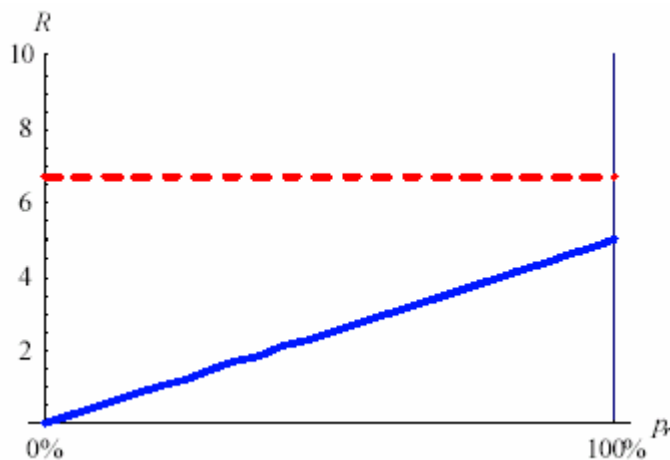
```
select /*+ index(i id_u1) */ id, data, status
from interfejs i
where id > -1;
```

Takie zapytanie będzie wymagało wykonania przez bazę Oracle mniej niż 40 operacji LIO. Czas reakcji będzie rzędu $10\,000/40 = 250$ razy krótszy, jeśli 100% wierszy tabeli wybierzemy przez indeks zamiast przez skanowanie całej tabeli.

² Zaobserwaliśmy, że w Oracle8i zbudowanie i wykorzystywanie indeksu w `system.dual` znacznie zmniejsza liczbę logicznych operacji wejścia-wyjścia (*logical input-output* — LIO), jakie trzeba przetworzyć do uzyskania tego jednego wiersza. Zaobserwaliśmy ok. dziesięciokrotne przyspieszenie reakcji na zapytania `dual`. Jeśli aplikacja wykonuje miliony operacji LIO dziennie na jednowierszowych tabelach referencyjnych, według naszego badania przez indeksację można zaoszczędzić na tych zapytaniach miliony operacji LIO dziennie i ok. 90% mocy obliczeniowej procesora.

W związku z opisywanym przykładem moglibyśmy poruszyć jeszcze inne, szczegółowe zagadnienia. Na przykład, jeśli dyrektywa `select` zawierałaby tylko parametr `id` lub `count(id)` (które to informacje można uzyskać z indeksu w ogóle bez odwoływania się do segmentów danych), skanowanie przez indeks byłoby jeszcze szybsze.

Tak więc, aby wspomniana praktyczna zasada — według której sposób indeksacji zależy od procentu zwracanych wierszy — mogła być stosowana z pożytkiem w przypadkach takich jak ten, musimy brać pod uwagę ewentualność, że indeks może być wydajniejszy niż skanowanie całej tabeli nawet w przypadku zapytań zwracających 100% wierszy tabeli. Zjawisko to zilustrowano na rysunku nr 2.



Rysunek 2. Na wykresie przedstawiono sytuację, w której tabela zawiera dużą liczbę bloków pustych. Skanowanie przedziału indeksu (ciągła, niebieska linia) jest szybsze niż skanowanie całej tabeli (czerwona, przerywana linia), nawet jeśli zapytanie zwraca 100% wierszy tabeli.

Istnieje wiele przypadków, w których praktyczne zasady oparte na procentowym stosunku zwracanych wierszy zawodzą. Jeszcze bardziej problematyczne jest trzecie z wymienionych wcześniej twierdzeń. Problem ten pojawi się ponownie w dalszej części naszej analizy.

Zawirowania wokół wartości x

Jednym z problemów związanych z praktyczną zasadą indeksacji jest trudność w dobraniu właściwej wartości x . Firma Oracle w swojej dokumentacji zalecała różne wartości:³

³ Tabeli tej *nie należy* mylić z wykresami, według których różne wartości x przypisuje się różnym wersjom jądra bazy Oracle. Na przykład, [Niemiec 1999 (38, 318)] wskazuje, że głównym parametrem, od którego zależy wartość x , jest wersja bazy Oracle. W tym artykule zamierzam dowieść, że faktyczny punkt zrównania w dużym stopniu zależy od innych parametrów i jest praktycznie niezależny od wersji Oracle.

Wersja Oracle	Zalecana w dokumentacji Oracle wartość x
5	10-15
6	10-15
7	2-4
8	2-4
8i	15
9i	15

W rzeczywistości jest jeszcze gorzej, niż wynikałoby to z tabeli. O ile mnie pamięć nie myli, w dokumentacji do wczesnej eksploatacyjnej wersji Oracle7 zalecano wartość x „w przedziale 1-15%”. Cóż za rozrzut! Żeby sprawę skomplikować jeszcze bardziej, muszę wspomnieć o znajomych pracujących nad Oracle Applications, którzy twierdzili z pełnym przekonaniem, że w swoich aplikacjach obserwowali wielokrotnie wartości x przekraczające 40.

Wiele osób twierdzi, że x zmienia się w wyniku zmian wprowadzanych przez Oracle w optymalizatorze. To jednak nie ma nic do rzeczy. Powodem ciągłych zmian zalecanej wartości x jest to, że autorzy rekomendacji nie dostrzegają prawdziwych czynników wpływających na wysokość punktu zrównania.

A parametrem krytycznym jest liczba bloków bazy Oracle poniżej rozmiaru maksymalnego tabeli, które można pominąć stosując indeks.⁴ Żeby ułatwić sobie ustalenie praktycznej reguły opisującej tworzenie indeksu, należy zadać sobie pytanie: „Który plan wykonywania będzie wymagał dostępu do mniejszej liczby bloków bazy Oracle?”

W przypadku każdego zbioru źródłowego zawierającego ponad jeden wiersz indeks się przydaje — eliminuje wywołania PIO (*programmed input/output*). Liczba operacji PIO na blokach danych, które są pomijane w wyniku użycia indeksu, zależy od następujących czynników:

- Ile bloków poniżej rozmiaru maksymalnego tabeli zawiera przynajmniej *jeden* wiersz zwracany w wyniku zadanego zapytania *where*? Jeśli „interesujące” wiersze są rozproszone jednolicie po tabeli, indeks może się okazać niewydajny nawet w przypadku bardzo „dobrych” wartości, na podstawie których są wybierane wiersze.

Przykład: Chcemy zoptymalizować następujące zapytanie:

```
select id, data from dostawy where znacznik='x'
```

Tabela dostawy zawiera 1 000 000 wierszy zapisanych w 10 000 blokach bazy Oracle. Tylko 10 000 wierszy spełnia kryterium *znacznik='x'*. Selektywność wierszy według kolumny

⁴ O tym, jakie parametry rzeczywiście wpływają na wartość x , Craig Shallahamer, Micah Adler oraz ja pisaliśmy w *Oracle Magazine* w 1993 r. I chyba byliśmy jedynymi, którzy to przeczytali. Kiedy niedawno ponownie spojrziałem na ten tekst, zrozumiałem, dlaczego artykuł ten nie zyskał zbyt dużej popularności. Znalazł się tam między innymi taki wiersz:

$$\text{Równanie 2: } B > H + [pR / \beta] + \alpha(1 - p)^\beta$$

Trzeba przyznać, niełatwy fragment! Nic dziwnego, że nikt nawet nie pytał mnie później, gdzie podział się nawias zamykający (jestem przekonany, że przed oddaniem do druku jeszcze tam był). Poza tym wspomnianego artykułu praktycznie nie można nigdzie znaleźć, co również nie wpływa dobrze na jego popularność. Szczegółowe informacje można znaleźć w punkcie *Materiały referencyjne*.

znacznik i jej wartości x jest więc bardzo „dobra” i wynosi 1%. Jednak fizyczna dystrybucja wierszy w tabeli `dostawy` jest taka, że każdy blok tej tabeli zawiera dokładnie jeden wiersz, w którym `znacznik='x'`. Dlatego zwrócenie wyników zapytania wymaga dostępu do każdego bloku tabeli `dostawy`, bez względu na to, czy dla kolumny `znacznik` użyjemy indeksu. W tym przypadku skanowanie całej tabeli będzie trwało krócej niż skanowanie przedziału indeksu, mimo że zapytanie zwraca tylko 1% wierszy.

- Czy baza Oracle może zrealizować instrukcję `select` korzystając tylko z danych zapisanych w indeksie? Jeśli tak, dzięki indeksowi możemy w ogóle wyeliminować operacje dostępu do tabeli. Kolumny indeksu to zazwyczaj tylko mały podzbiór kolumn indeksowanej tabeli. Dlatego liczba bloków danych zajmowanych przez indeks jest zazwyczaj o wiele mniejsza niż liczba bloków znajdujących się poniżej rozmiaru maksymalnego tabeli. Tak więc nawet skanowanie całego indeksu pochłania mniej zasobów, niż skanowanie przedziału tabeli.

Przypowieść o indeksach

O tym, jak ważna jest koncepcja określana mianem *selektywności bloków*, pokażemy na przykładzie z innej dziedziny życia:

Wyobraźmy sobie książkę *Krótką historia ludzkości*, w której na tysiącu stron streszczono wszystko, co osiągnął rodzaj ludzki, od kiedy tylko potrafimy to przekazać. Wyobraźmy sobie, że w książce tej chcielibyśmy przeczytać o Aleksandrze Wielkim. Jak się do tego zabieramy? Oczywiście, wyszukując odpowiednie hasło w indeksie.

Z indeksu dowiadujemy się, na których dokładnie stronach napisano o Aleksandrze Wielkim. Wkładamy zakładkę do indeksu i przechodzimy bezpośrednio do strony z poszukiwaną informacją. Po przeczytaniu zawartych tam informacji wracamy do strony indeksu z zakładką i sprawdzamy, gdzie znajdują się kolejne informacje. Na koniec zajrzemy jeszcze raz do indeksu, tylko po to, aby dowiedzieć się, że wyczerpały się już strony z poszukiwanymi przez nas informacjami.

Teraz wyobraźmy sobie, że w indeksie znajduje się każde słowo zawarte w książce. Można tam znaleźć nawet strony, na których występuje słowo „na”. Założmy, że chcemy odszukać w książce wszystkie słowa, jakie występują po słowie „na”. Takich informacji nie znajdziemy w indeksie — będziemy musieli przeanalizować sam tekst. Ze względu na dużą częstotliwość występowania słowa „na”, zadanie takie okazałoby się pracochłonne, nawet jeśli istniałby odpowiedni indeks. Słowo „na” wystąpiłoby pewnie już na pierwszej stronie. Wstawilibyśmy więc zakładkę w indeksie, przeszli do strony pierwszej i zlokalizowali słowo „na”. Następnie wrócilibyśmy do indeksu, gdzie okazałoby się, że następne wystąpienie słowa „na” ma miejsce także na stronie pierwszej. W ten sposób każdą stronę książki odwiedzilibyśmy wielokrotnie, nieustannie przerzucając kartki między indeksem i właściwą treścią. Prawdopodobnie książka szybko by się rozleciała.

A teraz wyobraźmy sobie wersję *Krótkiej historii ludzkości* dla niedowidzących. Na potrzeby przykładu założmy, że cały tekst złożono czcionką 72-punktową. Teraz *Krótką historia ludzkości* zawiera ok. 20-30 słów na stronę. Słowo „na” jest na tyle popularne, że występuje praktycznie na każdej stronie książki w zwykłym wydaniu; nie na tyle jednak popularne, żeby występowało na każdej stronie książki złożonej tak dużym drukiem. W takim przypadku indeks okazuje się o wiele bardziej pomocny w realizacji naszego zadania, tj. „znajdź słowa występujące po »na«”, ponieważ dzięki indeksowi pomijamy większą liczbę stron.

„Czcionka”

Tak wygląda czcionka 72-punktowa. Poszczególne strony „Krótkiej historii ludzkości” w wydaniu dla niedowidzących zawierałyby o wiele mniej słów niż w wersji standardowej.

Rozwiązanie zagadki

Na przydatność indeksu przy skanowaniu przedziału z użyciem operacji `table access by rowid` wpływają następujące parametry:

- *Wielkość bloku danych bazy Oracle.* Większy blok danych może być interesujący dla większej liczby predykatów `where`, a więc indeks jest wówczas mniej użyteczny. W przypadku większych bloków danych występuje tendencja do niższych wartości x .
- *Wielkość bloku indeksu bazy Oracle.* Im większy jest blok indeksu (który, oczywiście, w wersjach wcześniejszych niż Oracle9i jest zawsze tej samej wielkości co blok danych), tym mniej operacji LIO i PIO będzie na tym indeksie wykonywanych i tym bardziej indeks staje się użyteczny. Z większymi blokami indeksów wiążą się często większe wartości x .
- *Wielkość wiersza.* W naszej przypowieści indeks stawał się bardziej użyteczny, gdy wielkość czcionki wzrastała względem wielkości strony. Jest to analogia do większych wierszy w stosunku do mającego stałą wielkość bloku danych Oracle. Z większymi wierszami zazwyczaj wiążą się wyższe wartości x .
- *Dystrybucja danych.* Gdyby słowa tekstu *Krótkiej historii ludzkości* były posortowane alfabetycznie, nasz indeks okazałby się bardzo wartościowy, nawet w wersji książki składanej małą czcionką i nawet w przypadku słowa „na”. Wszystkie te słowa skumulowałyby się w jednym miejscu, gdzieś w drugiej połowie książki.⁵ Lepsze fizyczne rozmieszczenie danych tabeli wpływa na zmniejszenie wartości x dla danego zapytania.

Zrozumienie parametrów mechanizmu indeksowania pozwala wyjaśnić, dlaczego dobranie właściwej wartości x bywa takie trudne:

- Kiedy autorzy dokumentacji pisali instrukcje dostrajania baz Oracle w wersji 6, prawdopodobnie wykorzystywali tabele w rodzaju `emp` i `dept` ze schematu `scott/tiger`, implementując je na 2-kilobajtowych blokach ówczesnej wersji. Następnie te same zapytania prawdopodobnie wykorzystywali przy pisaniu dokumentacji do Oracle7, tym razem jednak na „nowych”, 4-kilobajtowych blokach. Ponieważ w większych blokach dawało się zapisać więcej wierszy niż wcześniej, zaobserwowano obniżenie wartości x . Stosowanie indeksów wydawało się więc mniej uzasadnione niż w bazie Oracle6. Udokumentowany punkt zrównania spadł z 10-15 do 2-4 procent.

⁵ Oczywiście, alfabetyczne posortowanie tekstu *Krótkiej historii ludzkości* zniszczyłoby znaczenie tekstu, a tym samym nasze wyszukiwanie słów występujących po „na” straciłoby sens.

- Dokumentacja Oracle8i oraz Oracle9i jest o wiele lepsza, jeśli chodzi o mechanizm indeksowania. Teraz jako ogólną wytyczną podaje się wartość $x = 15$, ale równocześnie zaznacza się, że wartość ta „może się znacznie różnić”. Wśród parametrów wpływających na te różnice wymienia się sposób klastrowania oraz szybkość skanowania całej tabeli, natomiast nie wymienia się wielkości bloku lub wiersza jako parametrów klastrowania [Oracle 2001a].
- Wspominałem o znajomych pracujących nad Oracle Applications, którzy twierdzili, że uzyskiwali dobre wyniki przy $x > 40$. Skąd wzięła się tak znaczna różnica w stosunku do wartości wymienianych w oficjalnej dokumentacji Oracle? Nietrudno to zrozumieć, jeśli wziąć pod uwagę środowisko, w jakim pracują ci programiści. Po pierwsze, wynik dotyczy tabel z ogromnymi wierszami. W wielu aplikacjach długość wierszy przekracza 200 kolumn. Ponadto, z różnych przyczyn, dział Oracle Applications „dość wolno” wdraża nowe technologie opracowane w dziale pracującym nad jądrem bazy. W połowie lat 90. wciąż korzystano tam niemal wyłącznie z baz implementowanych na blokach 2-kilobajtowych. Trzeba przyznać, że zmiana wielkości bloku w ogromnych bazach danych Oracle Applications wiązałaby się z ogromnym nakładem pracy, nie wspominając o trudnym do wyobrażenia sprawdzaniu poprawności planów wykonywania SQL. Połączenie dużych wierszy z małymi blokami podwyższyło więc zaobserwowaną wartość punktu zrównania x do niespotykanego gdzie indziej poziomu.

Co więc robić?

Moja rada brzmi tak:

Zapomnieć o **jakichkolwiek** praktycznych zasadach indeksowania opartych na wartościach procentowych.

Nie ma po prostu takich wartości procentowych, które zagwarantują właściwe wyniki. Istnieją zapytania zwracające 1% lub mniej wierszy, które są jednak bardziej wydajne przy skanowaniu całej tabeli niż przedziału indeksu. Są też zapytania zwracające 100% wierszy, które są wydajniejsze przy skanowaniu indeksu. Jeśli więc już upieramy się przy stosowaniu jakiejś wartości x , zalecam znalezienie takiej, która jednocześnie jest mniejsza niż 1% i większa lub równa 100%. Ponieważ wartość taka nie istnieje, zalecam w ogóle poniechanie stosowania zasad indeksowania opartych na wartościach procentowych.

Technologia optymalizatora Oracle bardzo rozwinęła się od czasów optymalizatora ukierunkowanego na koszty; w Oracle8i optymalizator jest już całkiem dobrze dopracowany. Wciąż jednak to od programisty zależy, kiedy ma być tworzony indeks. Jądro bazy Oracle podejmie próbę wykorzystania utworzonego indeksu tylko wtedy, gdy jest to uzasadnione z punktu widzenia wydajności. Ale tworzenie indeksu, który *nigdy* nie może być dobrze wykorzystany, jest stratą miejsca i czasu.⁶ Kiedy więc stosować indeks? Odpowiedzi należy szukać w pojęciu *selektywności bloków*.

⁶ O ile indeks taki nie jest tworzony po to, aby zadośćuczynić zdefiniowanemu ograniczeniu.

Selektywność bloków

Czytelnik prawdopodobnie zna już pojęcie *selektywności wierszy*. Dla danego predykatu `where` możliwe jest zdefiniowanie selektywności wierszy. Jest to liczba wierszy zwracanych przez ten predykat (r) podzielona przez całkowitą liczbę wierszy tabeli (R):

$$p_r = \frac{r}{R}$$

Definicja selektywności wierszy.

Analogicznie, dla danego predykatu `where` można zdefiniować *selektywność bloków*. Jest to liczba bloków danych zawierających przynajmniej jeden wiersz pasujący do warunku predykatu (b) podzielona przez całkowitą liczbę bloków danych znajdujących się poniżej rozmiaru maksymalnego tabeli (B):

$$p_b = \frac{b}{B}$$

Definicja selektywności bloków.

Różnica między selektywnością *wierszy* a selektywnością *bloków* jest bardzo ważna, ponieważ selektywność bloków prawie zawsze wypada gorzej – często o wiele gorzej – niż selektywność wierszy. Jak widzieliśmy wcześniej, przy omawianiu wartości `znacznik='x'` tabeli dostawy, selektywność wierszy dla danego predykatu może wynieść 1%, a selektywność bloków — 100%.

Selektywność wierszy i bloków można obliczyć za pomocą skryptu SQL, takiego jak poniższy skrypt `hds.sql` [Holt 2002].

```

1  rem $Header: /usr/local/hotsos/RCS/hds.sql,v 1.8 2002/01/07 18:12:27 hotsos Exp $
2  rem Copyright (c) 2000-2002 by Hotsos Enterprises, Ltd. All rights reserved.
3  rem Author: jeff.holt@hotsos.com
4  rem Notes: Hotsos data selectivity using a full table scan for the row count.
5
6  define v_substr7 = 'substr(rowid,15,4)||substr(rowid,1,8)'
7  define v_substr8 = 'substr(rowid,7,9)'
8  define v_over = 'substr('&_O_RELEASE'',1,1)'
9
10 col dummy new_value v_substr
11
12 set termout off heading on pause off
13
14 select decode(&v_over, '7', '&v_substr7', '&v_substr8') dummy
15 from dual;
16
17 set termout on verify off feedback off pages 10
18
19 accept p_town prompt 'TableOwner : '
20 accept p_tname prompt 'TableName : '
21 accept p_clst prompt 'ColumnList : '
22 accept p_where prompt 'WhereClause: '
23 accept p_pgs prompt 'PageSize : '
24
25 variable fblks number

```

```

26
27 declare
28 tblks number;
29 tbytes number;
30 ublks number;
31 ubytes number;
32 luefid number;
33 luebid number;
34 lublk number;
35 begin
36 sys.dbms_space.unused_space(
37 upper('&p_town'), upper('&p_tname'), 'TABLE',
38 tblks, tbytes, ublks, ubytes, luefid, luebid, lublk, null
39 );
40 :fblks := tblks - ublks;
41 end;
42 /
43
44 col blks form 9,999,999,999 heading 'Table blocks below hwm|(B)' just c
45 col nrows form 999,999,999,999 heading 'Table rows|(R)' just c new_value v_nrows
46
47 select :fblks blks, count(*) nrows
48 from &p_town..&p_tname;
49
50 col bs form a17 heading 'Block selectivity|(pb = b/B)' just c
51 col nblks form 9,999,999,999 heading 'Block count|(b)' just c
52 col rs form a17 heading 'Row selectivity|(pr = r/R)' just c
53 col nrows form 999,999,999,999 heading 'Row count|(r)' just c
54
55 set pause on pause 'More: ' pages &p_pgs
56
57 select &p_clst,
58 lpad(to_char(count(distinct &v_substr)/:fblks*100,'990.00')||'%',17) as bs,
59 count(distinct &v_substr) nblks,
60 lpad(to_char(count(*)/&v_nrows*100,'990.00')||'%',17) rs,
61 count(*) nrows
62 from &p_town..&p_tname &p_where
63 group by &p_clst
64 order by bs desc;

```

Korzystanie ze skryptu `hds.sql` nie powinno przedstawiać trudności. Jednak wygenerowanie informacji o dystrybucji danych może trwać długo — w zależności od rodzaju danych, takie zapytanie może być realizowane wiele minut lub godzin. To dlatego optymalizator Oracle ukierunkowany na koszty przy obliczaniu lub sprawdzaniu poprawności planu wykonywania wykorzystuje zapisane dane statystyczne, a nie analizuje samych danych tabeli. Poniżej pokazano, jak korzystać z danych uzyskanych ze skryptu `hds.sql`.

Przykład: W systemie istnieje tabela o nazwie `po.cs_ec_po_items`. Naszym celem jest zoptymalizowanie kilku podzapytań, w których jako predykat `where` wykorzystywane jest wyrażenie `ec_po_id=v`. Czy dla kolumny `ec_po_id` powinniśmy utworzyć indeks? Aby uzyskać dobry pogląd na dystrybucję różnych wartości kolumny `ec_po_id`, możemy wykorzystać nasz skrypt `hds.sql`:

```

SQL> @hds
TableOwner : po
TableName  : cs_ec_po_item
ColumnList : ec_po_id
WhereClause:
PageSize   : 20
Table blocks below hwm      Table rows
              (B)              (R)
-----
1,655                299,960
More:
EC_PO_ID      Block selectivity  Block count  Row selectivity  Row count
              (pb = b/B)      (b)          (pr = r/R)      (r)

```

8	63.50%	1,051	0.54%	1,606
0	61.81%	1,023	0.52%	1,572
1	61.27%	1,014	0.49%	1,470
2	60.66%	1,004	0.52%	1,555
4	60.24%	997	0.51%	1,529
6	59.94%	992	0.52%	1,552
7	59.94%	992	0.50%	1,514
5	59.70%	988	0.51%	1,536
9	59.58%	986	0.49%	1,459
3	57.95%	959	0.49%	1,480
45	10.76%	178	0.06%	190
37	10.63%	176	0.06%	183
.1	10.39%	172	0.06%	180
2.5	10.33%	171	0.06%	177
2.3	10.27%	170	0.06%	175
5.8	10.21%	169	0.06%	178

More: ^C

Wyniki zwracane przez skrypt `hds.sql` są posortowane wg malejącej selektywności bloków. Zazwyczaj w wyniku zwracane są tysiące wierszy, ale te najbardziej krytyczne dane — czyli najciekawsza część wydruku — znajdują się na początku. Zazwyczaj wystarczy więc obejrzeć stronę lub dwie takiego wydruku i można zakończyć działanie `hds.sql`.

Zauważmy, że w przypadku tej tabeli selektywność wierszy jest bardzo korzystna dla każdej wartości `ec_po_id`. „Najgorsza” selektywność wierszy wynosi tylko 0,54%.⁷ Oznacza to, że tylko pół procent wierszy tabeli ma w kolumnie `ec_po_id` wartość 8. Jednak selektywność bloków dla tej kolumny przedstawia się inaczej — dla wartości `ec_po_id='8'` wynosi 63,5%. To oznacza, że prawie dwie trzecie bloków tabeli zawiera przynajmniej jeden wiersz, dla którego w kolumnie `ec_po_id` występuje wartość 8.

Czy dla kolumny `ec_po_id` powinniśmy tworzyć indeks? Moglibyśmy spędzić mnóstwo czasu na prognozowaniu kosztów, jakie miałby pochłoniąć plan wykonywania. Jednak może to za nas zrobić optymalizator Oracle. Najbardziej precyzyjną, a jednocześnie najmniej czasochłonną metodą uzyskania odpowiedzi na nasze pytanie jest wykonanie testów na prawdziwym systemie Oracle. Najlepszym sposobem określenia względnego zużycia zasobów w dwóch planach wykonywania jest przetestowanie ich na próbkach danych przy podaniu warunku `sql_trace=true`. Jeśli potrzebne są nam dodatkowe informacje o niezwiązanych z procesorem operacjach wykonywanych przez Oracle, musimy zastosować śledzenie 8 poziomu i rejestrowanie zdarzenia Oracle 10046 [Hotsos 2002]. Jeśli chcemy uzyskać uzasadnienie, dlaczego optymalizator wybrał taki a nie inny plan, musimy zastosować śledzenie pod kątem zdarzenia Oracle 10053 [Lewis 2001].

Mając dane ze skryptu `hds.sql`, znamy wartości graniczne, które należy teraz zastosować w teście. Na przykład wiemy teraz, że nasze testy muszą dać odpowiedź na następujące pytania:

- Czy operacja `select cos from cs_ec_po_item where ec_po_id='8'` jest szybsza, jeśli dla kolumny `ec_po_id` utworzono indeks?
- Czy zastosowanie indeksu przyspieszy wykonanie tej operacji dla `ec_po_id='45'`?

⁷ Patrząc tylko na przytoczony tutaj fragment nie możemy określić, że najgorsza selektywność wierszy dla tej tabeli wynosi 0,54%. Wynik zwracany przez `hds.sql` jest posortowany według malejącej selektywności bloków. Nie oznacza to, że wartości selektywności wierszy ułożone są w takim samym porządku (widać to zresztą w pokazanym fragmencie — selektywność wierszy nie zawsze maleje). Aby określić „najgorszą” wartość selektywności wierszy, konieczne byłoby przeanalizowanie całego wyniku działania `hds.sql`.

- Czy zastosowanie indeksu przyspieszy wykonanie tej operacji dla wartości `ec_po_id`, dla których selektywność bloków jest niższa od 1% (dane skryptu zwracane są wg malejącej selektywności bloków, więc nie pokazano tutaj wartości charakteryzujących się najlepszą selektywnością bloków)?

Ostateczna decyzja o tworzeniu indeksu zależy, oczywiście, od porównania korzyści płynących z jego zastosowania z „kosztami” jego tworzenia. Koszty mogą być następujące:

- Przypadkowe obniżenie szybkości planów wykonywania innych zapytań. W aplikacjach, które wciąż wykorzystują optymalizator Oracle ukierunkowany na reguły, ryzyko jest znaczne. Utworzenie indeksu optymalizującego instrukcję *A* może spowodować przypadkowe obniżenie wydajności instrukcji *B*. Na szczęście, w optymalizacji ukierunkowanej na koszty, szczególnie z użyciem histogramów, ryzyko jest mniejsze.
- Wolniejsza reakcja na operacje DML wykonywane na tabeli. Z drugiej jednak strony zauważam, że wiele osób przykłada zbyt dużą wagę do tego czynnika. Nie ma co zgadywać: warto przyjrzeć się danym ze śledzenia operacji DML i określić rzeczywisty „koszt”.
- Większa ilość zajętego miejsca w wyniku zastosowania indeksu. Kiedyś miejsce zajmowane przez indeks było „kosztowne” w finansowym sensie tego słowa. Przy obecnych cenach dysków twardych czynnik ten można prawie pominąć.

W wynikach zwracanych przez narzędzia w rodzaju `hds.sql` można zaobserwować trzy wzorce:

1. Selektywność bloków dla każdej wartości jest tak korzystna, że podejmujemy decyzję o tworzeniu indeksu dla tej kolumny.
2. Selektywność bloków dla każdej wartości jest tak niekorzystna, że podejmujemy decyzję o nietworzeniu indeksu dla tej kolumny.
3. Selektywność bloków jest niekorzystna dla niektórych wartości, a korzystna dla innych. W takim przypadku trzeba zdecydować, czy przydatność indeksu w „korzystnych” przypadkach skompensuje nam jego koszty.

Decyzje w przypadkach 1 i 2 są oczywiste. Naturalnie, najczęściej przyjdzie nam stawić czoła sytuacji 3. Użytkownik bazy Oracle z optymalizatorem ukierunkowanym na koszty w wersji wcześniejszej niż 7.3 miał twardy orzech do zgryzienia. Jeśli nie utworzył indeksu — ryzykował niską wydajność dla niektórych warunków `where`; jeśli utworzył — ryzykował niską wydajność dla innych wartości. W nowszych wersjach optymalizatora Oracle ukierunkowanego na koszty sprawa jest prostsza. Obecnie, gdy tylko zgromadzi się odpowiednie informacje statystyczne⁸, prawdopodobieństwo utworzenia nieprzydatnego indeksu, spowalniającego pracę użytkowników, jest dużo mniejsze.

⁸ To znaczy, jeśli regularnie wykonuje się operację `fnl_stats` (administrator Oracle Applications) lub `dbms_stats` (pozostali użytkownicy).

Przykład: Mamy tabelę `dzial` zawierającą kolumnę `id` i wykazującą następującą dystrybucją danych:

Wartość <code>id</code>	Selektywność wierszy (p_r)	Selektywność bloków (p_b)
01	89,87%	100%
02	2,34%	7%
03	2,16%	6%
04	1,86%	2%
05	1,42%	1%
06	1,17%	1%
07	0,75%	1%
08	0,17%	<1%
09	0,14%	<1%
10	0,11%	<1%
Razem	100%	b/d

Dystrybucja danych jest bardzo nierównomierna. Załóżmy teraz, że na takiej tabeli wykonywane jest następujące zapytanie:

```
select nazwisko from dzial d where id=:a1
```

Jeśli nie ma histogramów, optymalizator ukierunkowany na koszty mógłby założyć, że ponieważ istnieje 10 różnych wartości `id`, każdej z nich odpowiada ok. 1/10 wierszy tabeli. Według tego założenia, dobrym rozwiązaniem byłoby utworzenie indeksu dla kolumny `id`. I tak jest rzeczywiście – o ile `:a1 != '01'`.

Siła optymalizacji opartej na histogramach leży w tym, że przy właściwej implementacji⁹ optymalizator taki „zauważy”, kiedy `:a1 = '01'` i *nie* będzie próbował stosować indeksu `id`. Programista niekorzystający z optymalizacji opartej na histogramach musi albo (1) zoptymalizować zapytanie tak, aby było wydajne przy `:a1 = '01'`, a zupełnie niewydajne w innych przypadkach;¹⁰ albo (2) napisać kod proceduralny, który realizuje jedną instrukcję SQL

⁹ To długa historia, ale w optymalizacji opartej na histogramach firma Oracle niejednokrotnie uciekała się do bolesnych kompromisów. Przed wersją Oracle9i użycie zmiennych dowiązanych uniemożliwiało optymalizację tego typu. To duże ograniczenie, bo — ogólnie — aplikacja niewykorzystująca zmiennych dowiązanych nie da się skalować pod kątem obsługi bardzo wielu użytkowników. W Oracle9i optymalizator działa prawie dokładnie tak, jakbyśmy sobie tego życzyli: „podgląda” treść dowiązaną do takiej zmiennej i podejmuje właściwą decyzję opartą na histogramie. Jednak w kolejnych realizacjach współdzielonej instrukcji SQL wykorzystywany jest plan uzyskany w wyniku pierwszej optymalizacji w danej sesji. Jeśli więc w pierwszym zapytaniu użyto warunku `:a1 = '01'`, drugie (i każde kolejne) wykonanie oparte będzie na optymalnym planie dla `:a1 = '01'`, nawet jeśli wyszukiwanie wykonywane jest wg warunku np. `:a1 = '07'`.

Na szczęście nie jest konieczne stosowanie zmiennych dowiązanych w tych predykatkach `where`, gdzie domena wartości nie jest duża. Na przykład, stosowanie wartości dosłownych w predykatce `plec = 'm'` jest rozsądne, bo powstają tylko dwie kopie współdzielonej instrukcji SQL w pamięci podręcznej biblioteki. Jednak stosowanie wartości dosłownych w predykatkach typu `nr_zamowienia = '1289942'` miałyby katastrofalne następstwa, ponieważ w pamięci podręcznej powstawałyby potencjalnie tysiące niemal identycznych, a jednak różnych instrukcji SQL.

¹⁰ Staromodnym sposobem takiej optymalizacji byłoby przepisanie instrukcji w następujący sposób:

```
select nazwisko from dzial d where nvl(id,id)=:a1
```

W ten sposób uniemożliwia się bazie Oracle użycie indeksu, w którym jako przedrostek występuje `id`. Bardziej nowoczesnym sposobem byłoby użycie wskazówki:

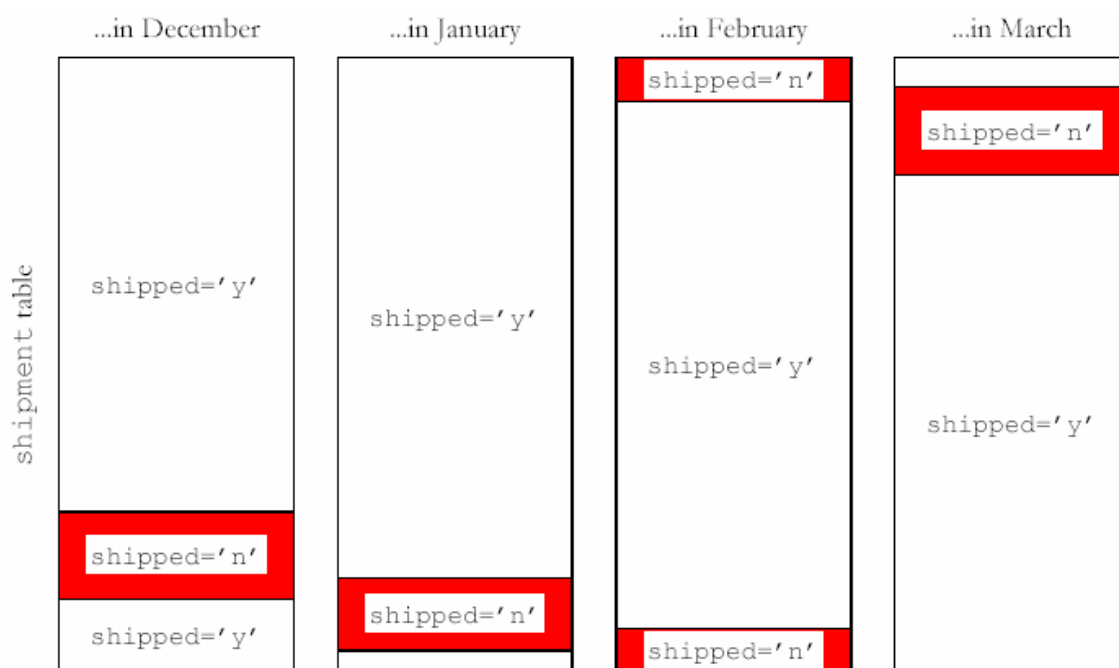
```
select /*+ full(d) */ nazwisko from dzial d where id=:a1
```

dla często wykorzystywanych wartości, a inną dla występujących rzadziej. W module Oracle Księga Główna (*General Ledger*), w funkcjach generatora sprawozdania finansowego (*Financial Statement Generator*) dynamiczny kod SQL jest tworzony właśnie z użyciem metody 2. Rozwiązanie to jest błyskotliwe, ale zagmatwane.

Dystrybucja wartości często nie jest przypadkowa

Ostatnio w dokumentacji Oracle przytacza się opinię, że „wiersze tabeli mają przypadkowy porządek, jeśli chodzi o kolumnę, której dotyczy zapytanie” [Oracle 2001b]. Założenie to upraszcza nieco zadanie autorom dokumentacji, ale powoduje też, że dokumentacja ta jest mniej przydatna.

Doświadczeni użytkownicy skryptu `hds.sql` zauważają, że czasami wartości kolumn są pogrupowane w dość naturalny sposób – i takie pogrupowanie utrzymuje się stale.



Rysunek 3. Wartości kolumny statusu mogą podlegać naturalnemu grupowaniu.

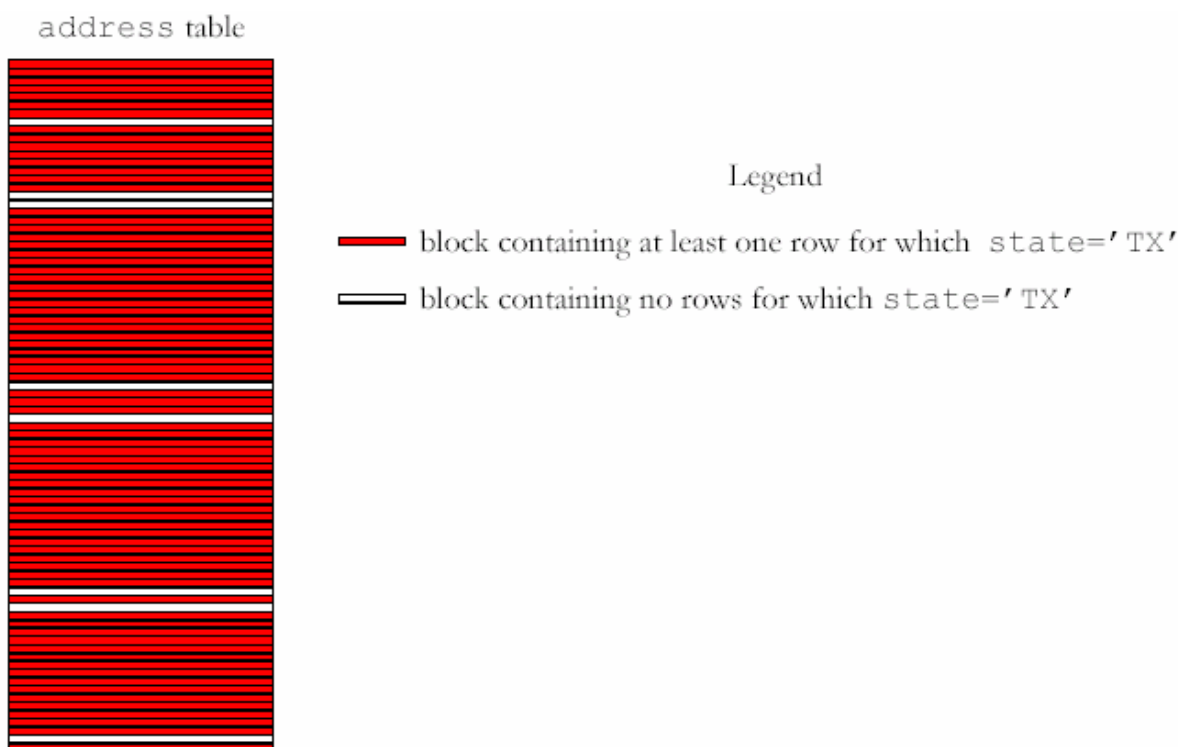
Przykład (powyżej): Tabela `shipment` zawiera kolumnę statusu o nazwie `shipped`. W kolumnie tej umieszczana jest wartość `'t'` wtedy i tylko wtedy, gdy zamówienie na dany produkt zostało zrealizowane. Ponieważ zamówienia są realizowane mniej więcej w takiej samej kolejności, w jakiej wprowadza się je do systemu, wartości `shipped='n'` w tabeli `shipment` w miarę upływu czasu naturalnie grupują się, co zilustrowano na rysunku 3. Takie

Od wersji Oracle8i możemy posunąć się o krok dalej — zoptymalizować zapytanie przez wykorzystanie jego zapisanej struktury. Ta bardzo przydatna funkcja pozwala zoptymalizować instrukcję *bez dostępu do kodu źródłowego tej instrukcji*.

grupowanie zwiększa przydatność indeksu, gdy szukamy właśnie wierszy, w których w kolumnie dostarczone występuje wartość 'n'.

Przeciwnieństwem dystrybucji *klastrowej* (pogrupowanej) jest dystrybucja *jednolita*. Jeśli dystrybucja pewnej wartości kolumny jest naprawdę równomierna na przestrzeni tabeli, egzemplarze tej wartości są fizycznie oddalone od siebie na takie same odległości.

Przykład: Tabela *address* zawiera kolumnę typu *state*, ta zaś zawiera dwuliterowy kod stanu lub województwa klienta. W aplikacji wykorzystującej tę tabelę nie zachodzi wyraźna relacja między czasem wstawienia wiersza opisującego klienta a wartością kolumny *state* tego wiersza. Stąd fizyczna dystrybucja wartości *state* jest praktycznie jednolita. Choć wartość *state='TX'* występuje, powiedzmy, tylko raz na 30 wierszy, istnieje bardzo niewiele bloków tabeli, w których wartość taka nie występuje w ogóle. Zilustrowano to na rysunku nr 4.



Rysunek 4. Indeks zbudowany dla kolumny *state* byłby mało przydatny w kontekście wartości *state='TX'*.

Legenda:

- *block containing at least one row for which `state='TX'`* – blok zawierający przynajmniej jeden wiersz, w którym *state='TX'*

- *block containing no rows for which state='TX' – blok niezawierający żadnego wiersza, w którym stan='TX'*

Tutaj zastosowanie indeksu dla kolumny `state` byłoby prawdopodobnie rozwiązaniem niewydajnym przy poszukiwaniu „popularnych” wartości tej kolumny. Gdyby jednak, na przykład, istniał jeden lub kilka stanów, które występują w o wiele mniejszej liczbie wierszy niż liczba bloków w tabeli `address` oraz gdyby często poszukiwano tych wartości oraz stosowano histogramy, wtedy utworzenie indeksu dla kolumny `state` prawdopodobnie przyspieszyłoby działanie aplikacji.

Czasem kolumny `statusu` podlegają naturalnemu grupowaniu. Jednak jeśli nie istnieje żaden „czynnik zewnętrzny”, dystrybucja wartości kolumn `typu` bywa bardziej jednolita. Na porządek fizycznego zapisu danych w tabeli możemy wpływać w różny sposób:

- przez *partycjonowanie* tabeli i indeksu bazy Oracle;
- przez stosowanie tabel Oracle zorganizowanych wg indeksu;
- przez okresowe procedury pielęgnacyjne (usuwanie wierszy i ponowne ich wstawianie w preferowanym porządku fizycznym);
- przez stosowanie segmentów *klastrów* zamiast segmentów *tabel*.

Nie należy bezkrytycznie przyjmować, że rozmieszczenie danych w tabeli jest przypadkowe. Wystarczy użyć skryptu `hds.sql`, aby przekonać się, jak często jest to nieprawdziwe założenie. Każdy sposób fizycznego grupowania danych wiąże się zarówno z korzyściami, jak i z kosztami. Jeśli zmiana fizycznej dystrybucji danych jednocześnie wpływa na zwiększenie zysku netto, przepływu gotówki oraz zwrotu z inwestycji — warto ją przeprowadzać. [Goldratt 1992]

Podsumowanie

Wiele źródeł zaleca, aby decyzję o stosowaniu indeksu podejmować na podstawie *selektywności wierszy* dla danego predykatu `where`. Co gorsza, czasem zaleca się podejmowanie tej decyzji na podstawie selektywności wierszy w całej *kolumnie*, zupełnie ignorując możliwość nierównomiernego rozłożenia danych. Jednak selektywność wierszy jest zawodna, jeśli chodzi o decyzje o tworzeniu indeksu. Najlepszym sposobem zminimalizowania ryzyka jest przetestowanie wydajności właściwego kodu SQL na poprawnej próbce danych. Narzędzia w rodzaju `hds.sql`, zwracające informacje o *selektywności bloków*, zwiększają niezawodność i wydajność takich testów — pokazują najbardziej krytyczne wartości kolumn, dla których należałoby przeprowadzić testy.

Optymalizator Oracle ukierunkowany na koszty ułatwia podjęcie decyzji o tworzeniu indeksu — sam korzysta z indeksu w sposób bardziej inteligentny niż optymalizator ukierunkowany na reguły. Jednak w implementacjach, które wciąż oparte są na optymalizatorze ukierunkowanym na reguły, zrozumienie znaczenia selektywności bloków może odgrywać ważną rolę przy zwiększaniu wydajności aplikacji Oracle. Takie zrozumienie pozwala podejść w sposób bardziej aktywny do fizycznego rozmieszczenia danych w bazie. Od wersji Oracle 7.3 wprowadzono wiele funkcji ułatwiających zapisywanie danych w takim porządku fizycznym, który najbardziej przyspiesza działanie bazy.

Materiały referencyjne

- GOLDRATT, E.M.; COX, J. 1992. *The Goal: A Process of Ongoing Improvement*. 2d ed. Great Barrington MA: North River Press.
- GORMAN, T. 2001. *The Search for Intelligent Life in the Cost-Based Optimizer*. Evergreen Database Technologies.
- LEWIS, J. 2001. *Practical Oracle8i: Building Efficient Databases*. Upper Saddle River NJ: Addison-Wesley.
- MILLSAP, C; SHALLAHAMER, C.; ADLER, M. 1993. „Predicting the Utility of the Nonunique Index”, *Oracle Magazine*, wol. VII, nr 2 (wiosna 1993 r.): 48-53. Artykuł ten został również wydrukowany i był rozprowadzany w firmie Oracle Corporation w latach 1992-1995. Niestety, nie istnieje dostępna publicznie elektroniczna kopia artykułu. Autorowi znany jest tylko jeden sposób na uzyskanie dostępu do artykułu: znalezienie fizycznego egzemplarza numeru *Oracle Magazine* z wiosny 1993 r.
- NIEMIEC, R. 1999. *Oracle Performance Tuning Tips & Techniques*. Berkeley CA: Osborne/McGraw Hill.
- ORACLE. 2001. *Oracle9i Database Administrator's Guide Release 1 (9.0.1)*.
- ORACLE. 2001. *Oracle9i Database Performance Guide and Reference Release 1 (9.0.1)*.