УДК 004.651.52

И.Г. Боровской, И.К. Харченко

Модификация иерархических битовых индексов для повышения производительности систем управления базами данных

Описывается модификация bitmap-индекса, именуемая иерархическим битовым индексом (Hierarchical Bitmap Index, HBI), предполагающая группировку ключей индекса и последующее хранение в B^+ -деревьях. Предложено два подхода организации индексной структуры, позволяющих получить двукратный прирост производительности по скорости поиска, а также прирост по количеству операций ввода-вывода.

Ключевые слова: иерархический битовый индекс, HBI, алгоритмы и структуры данных, B^+ -дерево, индексирование в СУБД, C#.

doi: 10.21293/1818-0442-2020-23-2-65-72

Битовые индексы (bitmap-индексы) уже давно стали стандартным способом оптимизации запросов в таких СУБД, как Oracle, Cache и др. [1, 2]. Однако их эффективность сильно зависит от запроса, в том числе в них ограничена поддержка таких операций поиска, как поиск по диапазону. Существует модификация данных индексов, основанная на хранении данных в раздельном виде, где битовые карты используются в качестве разделителей.

Есть ряд факторов, определяющих целесообразность использования битовых индексов. Нужно учитывать тип индексируемого атрибута, сколько уникальных значений хранится в атрибуте, а также запросы какого характера будут использоваться для поиска по данному атрибуту [3, 4].

Идея иерархических битовых индексов (Hierarchical Bitmap Indexes, HBI) подробнее разъяснена в работах [5–8]. Основная идея в том, что вместо хранения всех битовых строк для каждого ключа индекса все значения разделяются на группы, а битовая строка (далее – битовая маска) генерируется для каждой группы так, чтобы каждое значение соответствовало одной и только одной группе. Ключи каждой группы могут храниться в виде обычных битовых карт, в таком случае прирост производительности получается за счёт уменьшения количества конъюнкций, осуществляемых при поиске ключа. Также они могут храниться в виде двусвязного списка (скорость поиска в худшем случае $O(n)$) в случае малого n , либо в виде более производительных структур в случае большого n , таких как B^+ -, B^* - и B^{*+} -деревья.

Прирост производительности достигается в том случае, когда запрашиваемый диапазон значений хранится в одной или нескольких группах: тогда все значения можно прочитать с диска за N операций, где N – количество групп, в которые входит диапазон значений [9].

К примеру, есть множество значений с равномерным распределением от 1 до 256. Их можно разделить на три группы: до 63, от 64 до 127, от 128 до 256. Тогда все элементы будут равномерно распределены по разным деревьям (рис. 1).

Использование B^+ -дерева предпочтительнее B^* - и B^{*+} -деревьев ввиду увеличения сложности процедуры балансировки в последних, а также потому, что

большинство известных на рынке СУБД и файловых систем в операционных системах работают именно с такой структурой данных [10]. Поэтому в данной статье в качестве деревьев в предлагаемой структуре будут использованы B^+ -деревья. Подробнее о генерации таких деревьев рассказано в [11].

B^+ -дерево благодаря своей структуре поддерживает дубликаты ключей, однако при большом количестве дубликатов использовать такой индекс становится менее эффективно, поскольку для поиска конкретного идентификатора кортежа с одинаковыми ключами в индексе для худшего случая необходимо перебрать все элементы, подходящие под заданное условие поиска [9]. В данной работе предполагается альтернативный подход, основывающийся на предположении о том, что количество уникальных элементов невелико относительно общего количества элементов.

Алгоритм поиска

При инициализации индексной структуры передаются следующие параметры: количество бит для хранения типа значения индекса, равного размеру типа, умноженному на 8 бит (обозначим как $SizeOfT$); количество битовых масок (обозначим как $BitmapAmount$); массив числа бит, используемых в каждой битовой маске (обозначим как $BitmapParts_i$).

Для составления битовой маски происходит сдвиг всех байт в числе -1 на $(SizeOfT - BitmapParts_i)$

бит вправо, а затем на $\sum_{j=1}^i BitmapParts_j$ бит влево. По-

сле этого происходит инициализация B^+ -дерева, которому сопоставляется битовая маска.

Алгоритм поиска, какому дереву принадлежит элемент в индексной структуре, следующий:

1. Сначала необходимо выяснить, какому дереву принадлежит элемент. Инициализируется переменная $BTreeIndex$, хранящая индекс B^+ -дерева, в котором будет осуществляться поиск элемента.

2. Если искомым элемент равен нулю, то в переменную $BTreeIndex$ записывается индекс последнего дерева, алгоритм завершает работу.

3. В цикле по i от одного до $BitmapAmount$ осуществляется конъюнкция над поисковым элементом и битовой маской.

Рис. 1. Структура иерархического битового индекса с тремя B^+ -деревьями

4. Если результат не равен нулю, то элемент принадлежит данному дереву: индекс битовой маски i сохраняется в переменную $BTreeIndex$.

5. Иначе переменная i инкрементируется, происходит возврат к шагу 2.

Если элемент равен нулю, то он принадлежит последнему B^+ -дереву. Это сделано потому, что на шаге 3 результат конъюнкции проверяется на равенство нулю; а элемент, равный нулю, данную проверку не проходит.

Также в данном алгоритме ключ не может принадлежать сразу нескольким B^+ -деревьям: элемент может соответствовать сразу нескольким битовым маскам, но выбирается дерево, битовая маска которого проверена первой, т.е. справедлива формула

$$bitmap_i > bitmap_{i+1}, i = \overline{1, N}, \quad (1)$$

где $bitmap_i$ – i -я битовая маска, N – количество битовых масок.

Для добавления, удаления, изменения или поиска элемента необходимо найти битовую карту, которой соответствует ключ, и осуществить соответствующую операцию в B^+ -дереве, которому соответствует битовая маска.

Время поиска в B^+ -дереве в лучшем случае равно

$$O(\log_{t_i} n_i), \quad (2)$$

где t_i – порядок дерева, n_i – количество ветвей в узлах i -го B^+ -дерева.

В худшем случае, когда все элементы индекса сконцентрированы в единственном B^+ -дереве, скорость равна

$$O(\log_{t_i} n_i + N), \quad (3)$$

где N – количество битовых масок.

Обработка дубликатов ключей

Данный индекс хорошо подходит для случая с высокой селективностью запроса, когда в индексе хранится большое количество уникальных значений [12–15]. Однако в случае, когда большинство значений индекса не уникальны, данный алгоритм не даёт никаких преимуществ.

Для этого, чтобы увеличить скорость поиска элементов в случае низкой селективности, можно реорганизовать структуру индекса так, чтобы вместо битовых масок использовались все уникальные значения, а в B^+ -деревьях в свою очередь хранились бы идентификаторы строк таблицы базы данных.

На рис. 2 представлена организация индексной структуры для 10 уникальных элементов. На первом уровне индекс содержит уникальные ключи, отсортированные по возрастанию (в данном примере от одного до 10). В качестве ключей B^+ -дерева теперь хранятся не ключи атрибута, а значения первичного ключа таблицы либо идентификатора кортежа таблицы ($RowID$) в случае его отсутствия.

Теоретически это даёт выигрыш в скорости поиска ввиду того, что значения первичных ключей или $RowID$ отсортированы в каждом B^+ -дереве, а также того, что деревья проиндексированы по значению атрибута.

Алгоритм поиска одного конкретного элемента изменится на следующий. На вход алгоритма поступают параметры: ключ атрибута (обозначим как Key), значение атрибута (обозначим как $Value$).

1. Необходимо найти индекс B^+ -дерева. Для этого составляется цикл по i от 1 до количества B^+ -деревьев в индексной структуре.

2. На i -м шаге производится конъюнкция переменной Key с битовой маской. Индекс считается найденным, если результат операции ненулевой.

3. Если индекс B^+ -дерева найден, то переход к шагу 5.

4. Если за все шаги цикла индекс B^+ -дерева не найден, то элемент не найден и поиск завершается неудачно.

5. Производится поиск в B^+ -дереве элемента с ключом $Value$. Если элемент найден, то поиск завершается успешно и возвращается значение элемента, иначе поиск завершается неудачно.

Для оценки влияния на производительность данного алгоритма необходимо сравнить скорость поиска в иерархическом индексе со случаем, когда все элементы сосредоточены в одном B^+ -дереве. Также необходимо оценить количество операций ввода-вывода с дискового накопителя для обоих случаев.

С учётом дубликатов скорость поиска элемента в обычном B^+ -дереве равна

$$O(\log_t n + k), \quad (4)$$

где t – порядок дерева, n – количество ветвей в узлах дерева, k – количество элементов, встречающихся в пределах диапазона.

Рис. 2. Организация индексной структуры для 10 уникальных элементов

«Количество обращений к диску, необходимое для выполнения большинства операций с B^+ -деревом, пропорционально его высоте» [16].

Для B^+ -дерева справедлива следующая теорема: «Если $n \geq 1$, то для B^+ -дерева с n узлами и минимальной степенью $t \geq 2$ справедливо следующее неравенство [16]:»

$$h \leq \log_t \frac{n+1}{2}, \quad (5)$$

где h – высота дерева, n – количество элементов в дереве.

Сравнительная таблица скорости поиска для обычного B^+ -дерева и НВИ представлена в табл. 1, количества операций ввода-вывода – в табл. 2.

Таблица 1
Сравнительная таблица скорости поиска в B^+ -дереве и НВИ

B^+	B^+ с дубликатами	B^+ поиск по диапазону
$O(\log_t n)$	$O(\log_t n + d)$	$O\left(\log_t n + k\right)$
НВИ	НВИ с дубликатами	НВИ поиск по диапазону
$O\left(\log_{t_i} n_i + N\right)$	$O\left(\log_{t_i} n_i + d + N\right)$	$O\left(a \cdot \log_{t_i} n_i + k + N\right)$

Таблица 2
Сравнительная таблица количества операций ввода-вывода при поиске в B^+ -дереве и НВИ

Случай / тип дерева	B^+	B^+ с дубликатами	B^+ поиск по диапазону
Худший случай	h	$h + \frac{d}{t}$	$\frac{n}{t}$
Лучший случай	h	$h + \frac{d}{t}$	$h + \frac{k}{t}$
	НВИ	НВИ с дубликатами	НВИ поиск по диапазону
Худший случай	$h_i + B_m$	$h_i + \frac{d}{t_i} + B_m$	$\left[\sum_{i=1}^a h_i\right] + \frac{k}{t_i} + B_m$
Лучший случай	h_i	$h_i + \frac{d}{t_i}$	$h_i + \frac{k}{t_i}$

Используемые обозначения в таблицах даны ниже.
Глубина i -го B^+ -дерева

$$h_i = \log_{t_i} \frac{n_i + 1}{2}, \quad (6)$$

где t_i – порядок i -го B^+ -дерева, n_i – количество элементов в i -м B^+ -дереве, $n_i \leq n$.

Количество блоков для хранения всех битовых масок

$$B_m = \frac{N * BTreePointerSize}{BlockSize}, \quad (7)$$

где N – количество B^+ -деревьев, $N \geq 1$, $BTreePointerSize$ – размер указателя на B^+ -дерево, $BlockSize$ – размер блока для хранения внутренних и листовых узлов B^+ -дерева.

Количество B^+ -деревьев, в которых располагаются значения, подходящие под диапазон:

$$1 \leq a \leq N, \quad (8)$$

d – количество дубликатов ключей.

Подробнее о вычислении порядка B^+ -дерева описано в работах [16–18].

Предполагается, что битовые маски с указателями на B^+ -дерево хранятся в виде линейного списка в блоках индекса вместе с остальными данными.

Из сравнительных таблиц видно, что при использовании иерархического индекса в худшем случае необходимо обработать всего лишь одно B^+ -дерево при поиске одного элемента, глубина которого будет меньше, чем глубина одного B^+ -дерева, в котором хранятся все ключи, т.е. $h_i \leq h$. Следовательно, теоретически скорость поиска элемента увеличивается.

Однако при поиске диапазона значений придётся так или иначе прибегнуть к поиску элемента во всех сопутствующих деревьях, ключи которых удовлетворяют битовой маске, максимальное значение которых равно a . В лучшем случае $a \leq 2$ (искомые значения вполне могут храниться в одном блоке индекса), в худшем $a = N$.

Тем не менее существует потенциальный выигрыш в количестве операций ввода-вывода. Предположим, что размер блока равен 1 024 байта, на хранение битовой маски выделено четыре байта, всего масок 100. Таким образом, для хранения всех масок необходим один блок. Тогда для случая равномерного распределения ключей в индексе справедливо соотноше-

ние, представленное в табл. 3. Таким образом, количество операций для поиска элемента в конкретном B^+ -дереве НВИ будет таким же, как если бы все значения располагались в одном дереве.

С учётом минимум одной операции ввода-вывода для получения указателя на нужное дерево поиска получим соотношение $h \geq h_i + N$ для $N \leq 2$ и $h < h_i + N$ для $N > 2$.

Таблица 3
Соотношение количества операций ввода-вывода для обычного B^+ -дерева и НВИ

n	10^5	10^6	10^7	10^8	10^9
n_i	10^3	10^4	10^5	10^6	10^7
$\log_{t_i} n_i$	1,44	1,97	2,51	3,04	3,57
h_i	3	3	4	5	5
$\log_t n$	2,51	3,04	3,57	4,11	4,64
h	3	4	4	5	5

Следует учитывать, что N может быть нулевым, если в оставшейся части блока с битовыми масками хранятся значения B^+ -дерева, либо если принять допущение о том, что индексная структура при инициализации обращается к диску для чтения битовых масок только один раз, а в остальное время выбирает их

из основной памяти. В таком случае справедливо $h > h_i$. Из этого соотношения следует, что в лучшем случае существует небольшой прирост производительности при использовании НВИ, однако в худшем случае преимущество всё ещё будет на стороне обычного B^+ -дерева.

Добавление ссылок на соседние деревья

В B^+ -деревьях для быстрого поиска диапазона элементов листовые узлы хранят внутри себя ссылки на соседние листовые узлы, которые хранят значения [16]. На программном уровне это можно реализовать с помощью связанных списков, данные которых хранятся в виде блоков одинакового размера в одном или нескольких файлах в файловой системе. В процессе поиска нижней границы диапазона находится смещение в файле, указывающее на начальный листовой узел; далее для поиска всего диапазона требуется лишь продвигаться дальше по блокам, проверяя элементы в каждом узле на принадлежность диапазону.

Поскольку для поиска по диапазону в НВИ необходимо в худшем случае обратиться к каждому B^+ -дереву по отдельности, то в крайних листовых узлах B^+ -дерева можно добавить ссылки, указывающие на начало соседнего B^+ -дерева (рис. 3).

NULL – указатель на несуществующее дерево Рис. 3. НВИ с добавлением ссылок на соседние B^+ -деревья

В таком случае для поиска диапазона элементов можно начать поиск в дереве с левой границы диапазона, после чего, двигаясь к следующим элементам и достигнув границы дерева, обратиться по ссылке к началу соседнего дерева и продолжить поиск элементов в нём.

Необходимо заметить, что в данной организации индексной структуры битовые маски рассчитываются по иному алгоритму: сначала идут маски, указывающие на B^+ -деревья с меньшими элементами.

Следовательно, алгоритмическая сложность поиска диапазона элементов в худшем случае равна

$$O(\log_{t_i} n_i + k + N), \quad (9)$$

где t_i – порядок B^+ -дерева, битовая маска которого удовлетворяет нижней границе диапазона; n_i – количество элементов в данном B^+ -дереве, $n_i \leq n$; k – диапазон поиска; N – количество B^+ -деревьев, в которых содержится искомый диапазон значений.

В лучшем случае $N = 1$; следовательно, сложность для лучшего случая равна

$$O(\log_{t_i} n_i + k + 1). \quad (10)$$

Количество операций ввода-вывода для поиска по диапазону в худшем случае будет равно

$$h_i + \frac{d}{t_i} + Bm. \quad (11)$$

В лучшем случае $Bm = 0$, поэтому количество операций ввода-вывода для лучшего случая равно

$$h_i + \frac{d}{t_i}. \quad (12)$$

Реализация индексной структуры на языке программирования C#

Для проверки данной математической модели на практике разработана программа на языке программирования C#. В рамках данной программы реализовано две модификации: добавления ссылок на соседние B^+ -деревья и группировки значений по ключу.

Принципиальное различие двух модификаций заключается в том, что во второй модификации ключи ссылаются на B^+ -дерево со всеми значениями,

которые соответствуют данному ключу, а в самих B^+ -деревьях в качестве ключей выступают искомые значения. Пример: в таблице с девятью миллионами кортежей есть столбец, в котором всего десять уникальных значений (таким образом, значения повторяются 8 999 990 раз). При заполнении индекса создаются 10 отдельных деревьев (по каждому дереву на одно уникальное значение), каждое значение устанавливается в дерево, в котором и ключом, и значением будет являться устанавливаемое значение.

С помощью данной программы реализованы тесты производительности для установления количества операций ввода-вывода для нахождения одного элемента в случае использования одного B^+ -дерева и иерархического индекса в обеих модификациях, а также тесты для установления количества операций ввода-вывода, затраченных на нахождение нижней границы диапазона элементов.

Идея последнего теста состоит в том, что в первой модификации общее количество операций ввода-вывода для поиска всех элементов в диапазоне останется почти тем же, однако затраты на поиск уменьшаются именно за счёт уменьшения количества операций при нахождении начала диапазона.

Каждый тест был запущен 10 раз на одном ядре процессора. Для прогрева кэша процессора каждая операция была запущена несколько раз в течение 1 200 мс. Подробнее о проведении замеров производительности на языке программирования C# рассказано в [19–20].

Для тестов производительности промоделирован случай нормального распределения значений; тип данных для хранения ключей и значений – целочисленный (32 бита), значения генерируются в пределах от 1 до 2 147 483 647 (верхняя граница целочисленного четырёхбайтового типа). Для модификации, предполагающей добавление ссылок на соседние деревья, значения распределены по 5 деревьям в НВИ; распределение их битовых масок представлено в табл. 4. Для модификации, предполагающей группировку по уникальным значениям, все элементы

распределены по 10 деревьям, что соответствует 10 уникальным элементам в столбце таблицы.

Таблица 4

Распределение битовых масок в НВИ

Номер дерева	Битовая маска B^+ -дерева
1	0111 1110 0000 0000 0000 0000 0000 0000
2	0000 0001 1111 1000 0000 0000 0000 0000
3	0000 0000 0000 0111 1111 0000 0000 0000
4	0000 0000 0000 0000 0000 0000 1111 1100
5	0000 0000 0000 0000 0000 0000 0011 1111

Для поиска одного элемента в НВИ для модификации добавления ссылок на соседние деревья проверено два условия: в лучшем случае по той части индекса, элементов в которой минимальное количество, а для худшего случая – максимальное. Эти же условия проверены для поиска диапазона элементов: в модификации группировки по ключам при поиске были взяты все элементы из нескольких разных групп, что соответствует запросу «найти все идентификаторы строк таблицы, у которых значение столбца равно одному или множеству значений».

Результаты запуска тестов производительности приведены в табл. 5, графики зависимости скорости поиска от количества элементов в деревьях представлены на рис. 4 и 5.

Из полученных результатов теста производительности видно, что количество операций для нахождения одного элемента уменьшилось на два в лучшем случае, а для худшего случая соразмерно одному B^+ -дереву. Результаты соответствуют полученным в табл. 3 математически.

Разница в скорости поиска в обеих структурах (см. рис. 4) для первой модификации есть только на уровне погрешности: как показал результат, она почти не даёт никакого выигрыша в скорости. Однако для модификации группировки по ключам есть почти двукратный прирост в скорости поиска (см. рис. 5). На этом рисунке не показана скорость для 50 миллионов значений: для B^+ -дерева затраченное время составило 30 756 мс, для НВИ – 15 527 мс.

Таблица 5

Результаты запуска тестов производительности

Модификация	Количество элементов (без дубликатов)	Количество операций для поиска одного элемента			Количество операций для поиска нижней границы диапазона	
		Одно B^+ дерево	НВИ		Одно B^+ дерево	НВИ
			Худший случай	Лучший случай		
С добавлением ссылок на соседние деревья	10 000	4	3	2	4	2
	100 000	4	4	3	4	3
	500 000	5	4	3	5	3
	1 000 000	5	4	3	5	3
	10 000 000	5	4	4	5	4
	50 000 000	6	4	4	6	4
Группировки значений по ключу	10 000	–	–	–	4	3
	100 000	–	–	–	4	4
	500 000	–	–	–	5	4
	1 000 000	–	–	–	5	4
	10 000 000	–	–	–	6	5
	50 000 000	–	–	–	6	5

Рис. 4. График зависимости скорости поиска от количества элементов в индексной структуре для модификации НВИ с добавлением ссылок на соседние деревья

Рис. 5. График зависимости скорости поиска от количества элементов в индексной структуре для модификации НВИ с группировкой значений по ключам

По полученным результатам можно сделать вывод, что модификация, предполагающая группировку значений по ключам индекса, позволяет эффективно работать с большим количеством значений с равными ключами в столбце. Однако это справедливо лишь для небольшого количества уникальных элементов (до 10).

Для модификации, предполагающей соединение B^+ -деревьев путём добавления ссылок на крайние листовые узлы, также существует прирост, но не по скорости, а по количеству операций. Этому есть объяснение: при поиске в данной модификации НВИ осуществляются одни и те же операции, что и в B^+ -дереве. А поскольку ключи распределены равномерно по всем деревьям, следует вывод о том, что в случае поиска одного элемента и поиска диапазона

элементов количество операций ввода-вывода будет меньше, чем в случае хранения всех ключей в одном B^+ -дереве.

Заключение

Предложенные модификации иерархического битового индекса демонстрируют прирост производительности по сравнению с обычным B^+ -деревом и могут применяться для повышения производительности поиска в СУБД даже для случая малого количества уникальных значений в искомом столбце.

Данные индексные модификации могут быть использованы во многих областях: например, при разработке геоинформационных систем и приложений для улучшения качества поиска в однородных данных [21, 22].

Литература

1. Льюис Д. Разбираемся с индексами на основе битовых карт [Электронный ресурс]. – Режим доступа: http://citforum.ru/database/oracle/bitmap_index/, свободный (дата обращения: 02.02.2020).
2. Каратаев Е. MUMPS СУБД: Практика применения и опыт программирования. – М.: Солон-Пресс, 2013. – 550 с.
3. Труб И.И. Иерархические битовые индексы [Электронный ресурс]. – Режим доступа: <https://www.osp.ru/os/archive/2018/03/>, свободный (дата обращения: 02.02.2020).
4. Труб И.И. Вероятностная модель иерархических индексов базы данных // Программные системы и вычислительные методы. – 2017. – № 4. – С. 15–31.
5. Иерархический битовый индекс: эффективная и расширяемая техника индексирования для атрибутов ключ–значение / М. Морзи, Т. Морзи, А. Нанопулос, И. Манолопулос // Достижения в базах данных и информационных системах: матер. Седьмой Восточно-европейской конф. – Дрезден: ADBIS, 2003. – Т. 2798. – С. 236–252.
6. Иерархический битовый индекс для индексации пространственных данных / Дж. Чмел, Т. Морзи, Р. Врембел // Хранение данных и обнаружение знаний: матер. Девятой Междунар. конф. – Линц: Lecture Notes in Computer Science, 2009. – Т. 5691. – С. 87–98.
7. Otto E. Accelerating Queries on Very Large Datasets // E. Otto, K. Wu. – Chapman & Hall/CRC, 2010. – 558 p.
8. Scientific Data Management: Challenges, Technology, and Deployment [Электронный ресурс]. – Режим доступа: <https://www.crcpress.com/Scientific-Data-Management-Challenges-Technology-and-Deployment/Shoshani-Rotem/p/book/9780367384760>, свободный (дата обращения: 02.02.2020).
9. Сейджвик, Р. Алгоритмы. – 4-е изд. / Р. Сейджвик, К. Уэйн. – Бостон: Addison-Wesley Professional, 2011. – 976 с.
10. Дейт К.Дж. Введение в системы баз данных / К.Дж. Дейт. – 8-е изд. – М.: ИД «Вильямс», 2005. – 1328 с.
11. Кнут Д.Э. Искусство программирования. Генерация всех деревьев. История комбинаторной генерации / Д.Э. Кнут. – М.: Вильямс, 2007. – Т. 4. – 160 с.
12. Bitmap-индекс или B*tree-индекс: какой и когда применять? [Электронный ресурс]. – Режим доступа: http://citforum.ru/database/oracle/bb_indexes/, свободный (дата обращения: 02.02.2020).
13. Чернов А.Ф. Анализ и разработка индекса для поиска последовательностей элементов произвольного типа по их фрагментам в реляционных базах данных: дис. ... канд. техн. наук: 05.13.11 – СПб., 2014. – 156 с.
14. Табличная селективность, индексная селективность, блочная селективность. Базы данных Oracle: Администрирование и программирование СУБД для профи [Электронный ресурс]. – Режим доступа: <https://oracle-patches.com/oracle/prof/341-tablichnaya-selektivnost-indeksnaya-selektivnost-blochnaya-selektivnost>, свободный (дата обращения: 02.02.2020).
15. Tim Henderson Lessons While Implementing a B+Tree [Электронный ресурс]. – Режим доступа: <https://hackthology.com/lessons-learned-while-implementing-a-btree.html>, свободный (дата обращения: 02.02.2020).
16. В-дерево – Викиконспекты [Электронный ресурс]. – Режим доступа: <https://neerc.ifmo.ru/wiki/index.php?title=В-дерево>, свободный (дата обращения: 02.02.2020).
17. Асимптотический анализ алгоритмов [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/78728/>, свободный (дата обращения: 02.02.2020).
18. Bettyf Langtein Example to calculate the order p of a BTree [Электронный ресурс]. – Режим доступа: <https://www.coursehero.com/file/p2gj1mfj/Example-To-calculate-the-order-p-of-a-B-tree-suppose-the-search-field-is-V-8/>, свободный (дата обращения: 02.02.2020).
19. Introduction to benchmarking C# code with benchmark.NET [Электронный ресурс]. – Режим доступа: <https://www.stevejgordon.co.uk/introduction-to-benchmarking-csharp-code-with-benchmark-dot-net>, свободный (дата обращения: 02.02.2020).
20. Learn how to create correct C# benchmarks [Электронный ресурс]. – Режим доступа: <http://blogs.perpetuumsoft.com/dotnet/learn-how-to-create-correct-c-benchmarks/>, свободный (дата обращения: 02.02.2020).
21. Милихин М.М. Комбинированный метод визуализации картографических данных веб-ориентированной геоинформационной системы / М.М. Милихин, Ю.Б. Гриценко, М.М. Рычагов // Доклады ТУСУР. – 2015. – № 1(35). – С. 112–115.
22. Базовые принципы разработки веб-ориентированных информационных систем управления инфраструктурой социально-экономических и технических процессов / Ю.П. Ехлаков, О.И. Жуковский, П.В. Сенченко, Ю.Б. Гриценко, М.М. Милихин // Доклады ТУСУР. – 2017. – Т. 20, № 2. – С. 63–67.

Боровской Игорь Георгиевич

Д-р физ.-мат. наук, профессор каф. экономики, математики и статистики (ЭМИС) Томского государственного университета систем управления и радиоэлектроники (ТУСУР) Ленина пр-т, д. 40, г. Томск, Россия, 634050
ORCID 0000-0003-2707-7203
Тел.: +7 (382-2) 90-01-85
Эл. почта: igor.g.borovskoi@tusur.ru

Харченко Игорь Константинович

Аспирант каф. автоматизированных систем управления (АСУ) ТУСУР Ленина пр-т, д. 40, г. Томск, Россия, 634050
ORCID 0000-0003-2702-7230
Тел.: +7-923-416-09-12
Эл. почта: igor.k.kharchenko@tusur.ru

Borovskoy I.G., Kharchenko I.K.

Hierarchical bitmap index modification to improve performance of database management systems

This paper describes a modification of the bitmap index called the Hierarchical Bitmap Index (HBI), which involves the grouping of index keys and subsequent storage in B⁺ trees. Two approaches to the organization of the index structure have been proposed, allowing to increase the productivity in the number of input / output operations.

Keywords: hierarchical bitmap index, HBI, algorithms and data structures, B⁺ tree, indexing in DBMS, C#.

doi: 10.21293/1818-0442-2020-23-2-65-72

References

1. Louis D. Understanding bitmap indices. – Available at: http://citforum.ru/database/oracle/bitmap_index/, free (Accessed: February 2, 2020).

2. Karataev E. *MUMPS SUBD: Praktika primeneniya I opit programmirovaniya* [MUMPS DBMS: Application practice and programming experience]. Moscow, Solon-Press Publ., 2013. 550 p. (in Russ.).
3. Trub I.I. Hierarchical bitmap indices. – Available at: <https://www.osp.ru/os/archive/2018/03/>, free (Accessed: February 2, 2020).
4. Trub I.I. [A probabilistic model of hierarchical database indices]. *Software systems and computational methods*, 2017, no. 4, pp. 15–31 (in Russ.).
5. Morzy M., Morzy T., Nanopoulos A., Manolopoulos Y. Hierarchical Bitmap Index: an Efficient and Scalable Indexing Technique for Set-Valued Attributes. Proc. of the Advances in Databases and Information Systems, Seventh East European Conference ADBIS 2003, Dresden, Springer Berlin Heidelberg Publ., 2003, vol. 2798, pp. 236–252.
6. Chmiel J., Morzy T., Wrembel R. HOBI: Hierarchically Organized Bitmap Index for Indexing Dimensional Data. Proc. of the Eleventh International Conference on Data Warehousing and Knowledge Discovery (DaWaK). Springer Berlin Heidelberg Publ., 2009, vol. 5691, pp. 87–98.
7. Otto E., Wu K. Accelerating Queries on Very Large Datasets. London, Chapman & Hall/CRC Publ., 2010. 558 p.
8. Scientific Data Management: Challenges, Technology, and Deployment. – Available at: <https://www.crcpress.com/Scientific-Data-Management-Challenges-Technology-and-Deployment/Shoshani-Rotem/p/book/9780367384760>, free (Accessed: February 2, 2020).
9. Sedgewick R., Wayne K. Algorithms. 4th Edition. Boston: Addison-Wesley Professional Publ., 2011. 976 p.
10. Deit, K., Dzh. *Vvedeniye v sistemy baz dannykh* [An introduction to database management systems]. Moscow, Williams Publ., 2005. 1328 p. (in Russ.).
11. Knuth, D.E. *Iskusstvo programmirovaniya. Generatsiya vseh dereviev. Istoriya kombinatornykh generatsiy* [The art of programming. Generation of all trees. The history of combinatorial generation]. Moscow, Williams Publ., 2007, 160 p. (in Russ.).
12. Bitmap-index or B*tree-index: which and when to use? Available at: http://citforum.ru/database/oracle/bb_indexes/, free (Accessed: February 2, 2020).
13. Chernov A.F. *Analiz i razrabotka indeksa dlya poiska posledovatel'nostey elementov proizvol'nogo tipa po ikh fragmentam v relyatsionnykh bazakh dannykh. Diss. kand. nauk* [Analysis and development of an index for searching sequences of elements of an arbitrary type by their fragments in relational databases. Cand. Diss.]. Saint-Petersburg, 2014. 156 p. (in Russ.).
14. Table selectivity, index selectivity, block selectivity. Oracle Databases. Administration and programming of DBMS for professionals. Available at: <https://oracle-patches.com/oracle/prof/341-tablichnaya-selektivnost-indeksnaya-selektivnost-blochnaya-selektivnost>, free (Accessed: February 2, 2020).
15. Tim Henderson Lessons While Implementing a B+Tree. Available at: <https://hackthology.com/lessons-learned-while-implementing-a-btree.html>, free (Accessed: February 2, 2020).
16. B-tree – Wikiconspects. Available at: <https://neerc.ifmo.ru/wiki/index.php?title=В-дерево>, free (Accessed: February 2, 2020).
17. Asymptotic Analysis of Algorithms. – Available at: <https://habr.com/ru/post/78728/>, free (Accessed: February 2, 2020).
18. Bettyf Langtein Example to calculate the order p of a BTree. Available at: <https://www.coursehero.com/file/p2gj1mfj/Example-To-calculate-the-order-p-of-a-B-tree-suppse-the-search-field-is-V-8/>, free (Accessed: February 2, 2020).
19. Introduction to benchmarking C# code with benchmark .NET. Available at: <https://www.stevejgordon.co.uk/introduction-to-benchmarking-csharp-code-with-benchmark-dot-net>, free (Accessed: February 2, 2020).
20. Learn how to create correct C# benchmarks. Available at: <http://blogs.perpetuumsoft.com/dotnet/learn-how-to-create-correct-c-benchmarks/>, free (Accessed: February 2, 2020).
21. Milikhin M.M., Gritsenko Y.B., Rychagov M.M. [Combined geospatial data visualization method for web-based geographic information system]. *TUSUR reports*, 2015, no. 1, pp. 112–115 (in Russ.).
22. Ekhlakov Y.P., Zhukovskiy O.I., Senchenko P.V., Gritsenko Y.B., Milikhin M.M. [Basic principles of developing web-oriented information systems for managing the infrastructure of socio-economic and technical processes]. *TUSUR reports*, 2017, vol. 20, no. 2, pp. 63–67 (in Russ.).

Borovskoy Igor Georgievich

Doctor of Physics and Mathematics, Professor,
Department of Economics, Mathematics and Statistics,
Tomsk State University of Control Systems
and Radioelectronics (TUSUR)
40, Lenin pr., Tomsk, Russia, 634050
ORCID 0000-0003-2707-7203
Phone: +7 (382-2) 90-01-85
Email: igor.g.borovskoi@tusur.ru

Kharchenko Igor Konstantinovich

Postgraduate student, Department of Automated Control
Systems TUSUR
40, Lenin pr., Tomsk, Russia, 634050
ORCID 0000-0003-2702-7230
Phone: +7-923-416-09-12
Email: igor.k.kharchenko@tusur.ru