УДК 004.651.4

ИСПОЛЬЗОВАНИЕ ПРОСТРАНСТВЕННЫХ ИНДЕКСОВ ДЛЯ ОБРАБОТКИ АНАЛИТИЧЕСКИХ ЗАПРОСОВ И АГРЕГИРОВАНИЯ МНОГОМЕРНЫХ ДАННЫХ В ИНФОРМАЦИОННО-АНАЛИТИЧЕСКИХ СИСТЕМАХ

А.М. Бородин, С.В. Поршнев, М.А. Сидоров

Уральский государственный технический университет, г. Екатеринбург
E-mail: Borodin_am@cift.ru

Описан опыт применения технологий многомерного индексирования, использующихся при обработке пространственных данных, для выполнения аналитических запросов к многомерным хранилищам данных. Показано, что использование динамических структур позволяет не только повысить скорость выполнения аналитических запросов, но и более эффективно решать задачу обновления многомерных данных.

Ключевые слова:
Базы данных, многомерные данные, OLAP, R-tree, VAM-Split.

Введение

Для извлечения и анализа информации из многомерных хранилищ данных используют агрегирующие запросы, выполнение которых в информационно-аналитических системах зачастую занимает основное время обработки данных, что свидетельствует о низкой эффективности применяющихся структур индексирования. В этой связи задача разработки механизмов, позволяющих ускорить выполнение агрегирующих запросов, является актуальной.

Одним из возможных подходов к ее решению состоит в использовании методов доступа к данным, доказавшим ранее свою эффективность при обработке других видов информации. К таковым следует отнести механизм пространственного индексирования [1], широко применяемый в геоинформационных системах при решении задач, связанных с хранением и поиском пространственных объектов.

Целью статьи является обсуждение особенностей применения технологий многомерного индексирования, использующихся при обработке пространственных данных, для выполнения аналитических запросов к многомерным хранилищам данных, в частности, подхода, основанного на использовании динамических структур (разновидностей R-дерева), который позволяет, наряду с эффективным выполнением аналитических запросов, решать задачу обновления данных в информационно-аналитических системах.

Изложение сути предлагаемого подхода целесообразно предварить кратким обсуждением основных понятий технологий доступа к многомерным и пространственным данным.

Обобщённые древовидные индексы

Для эффективного вычисления различных аналитических запросов оказывается целесообразным использовать древовидные структуры организации данных, поскольку при этом сложность выполнения запроса (число операций) оказывается логарифмически зависящей от объёма набора исходных данных [2]. Также необходимо учитывать, что древовидная структура должна быть организована на страницах памяти базы данных.

Древовидная структура строится из трёх типов страниц (рис. 1):

1. Листовая страница (страница с данными), содержащая собственно записи о фактах.
2. Внутренняя страница, содержащая ключи, построенные по дочерним страницам (внутренним или страницам данных), а также ссылки на эти страницы.
3. Корневая страница (внутренняя страница верхнего уровня или единственная страница с данными).

Для организации древовидной структуры, в общем случае, необходимо и достаточно выполнить следующие действия [3]:

1) Построить ключа p по набору данных S (например, если набором данных является набор натуральных чисел {1,3,4,7,1000} ключом к нему может быть диапазон [1;1000]).
2) Определить для ключа p и запроса q возможности существования пересечения множества данных, по которым построен ключ p с областью запроса q.
3) Определить для ключа p и элемента данных e «стоимость» вставки e в набор данных с ключом p (например, для элемента {500} и ключей [1,3], [502,600], [1000,1002] мы должны выбрать набор данных с ключом [400,600], т. к. потребуется наименьшее расширение ключа).
4) Разделить переполненную страницу на две (при вставке в заполненную страницу для элемента e и ключа p определить должен ли элемент перейти во вновь создаваемую страницу или остаться в старой).

Необходимо отметить, здесь в качестве элемента данных e может быть использован и ключ (например, при выполнении: действия 1 может конструировать «общий» ключ из набора ключей; действия 4 – разделение внутренней страницы).

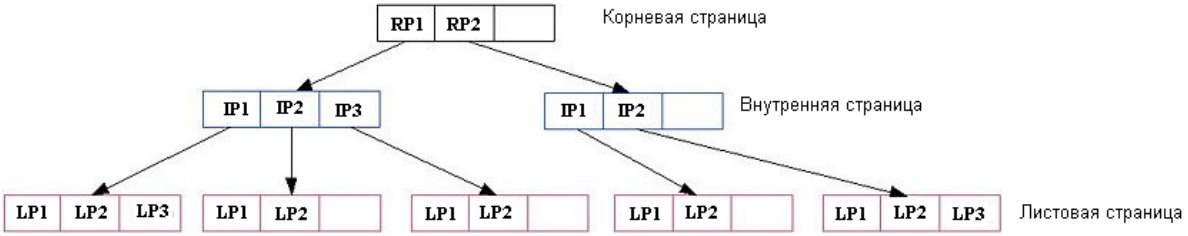

Рис. 1. *Типы страниц дерева*

Описанные действия позволяют создать как алгоритмы построения дерева в процессе вставки элементов в индексирующую структуру и выполнения запросов, так и алгоритмы удаления элементов из структуры, в которых учитывается модификация дерева. Данная особенность позволяет говорить об индексировании изменяющегося набора данных. Большинство современных систем с многомерными хранилищами данных позволяет работать лишь с «мгновенными снимками» базы данных. Также актуальной оказывается построение алгоритмов разовой загрузки (массовой загрузки, *bulk insert*), которые позволяют, используя знания о структуре и наборе данных, более точно строить индексирующую структуру.

В такой структуре выполнение агрегирующего запроса q осуществляется рекурсивным алгоритмом рассмотрения страницы. Рассматриваются ключи p для каждой записи на странице. Если метод 2 даёт положительный результат для p и q, то в случае рассмотрения листовой страницы запись участвует в агрегируемом наборе, а в случае рассмотрения внутренней страницы, следующим шагом рекурсии является рассмотрение страницы, на которую ссылается запись. Рекурсивный алгоритм начинает работу с корневой страницы дерева.

R-деревья

Для выполнения многомерных агрегирующих запросов наиболее представляется целесообразным использовать R-деревья, предложенные в [4]. Здесь ключевым элементом является многомерный параллелотоп, описывающий все элементы данных (minimum bounding rectangle – MBR). В двумерном случае, как очевидно, MBR-параллелотоп вырождается в прямоугольник. Идея использования MBR-параллелотопов изоморфна действиям 1 и 2 для R-деревьев.

Структура данных R-дерева (рис 2) разбивает пространство на множество иерархически вложенных и, возможно, пересекающихся, прямоугольни-

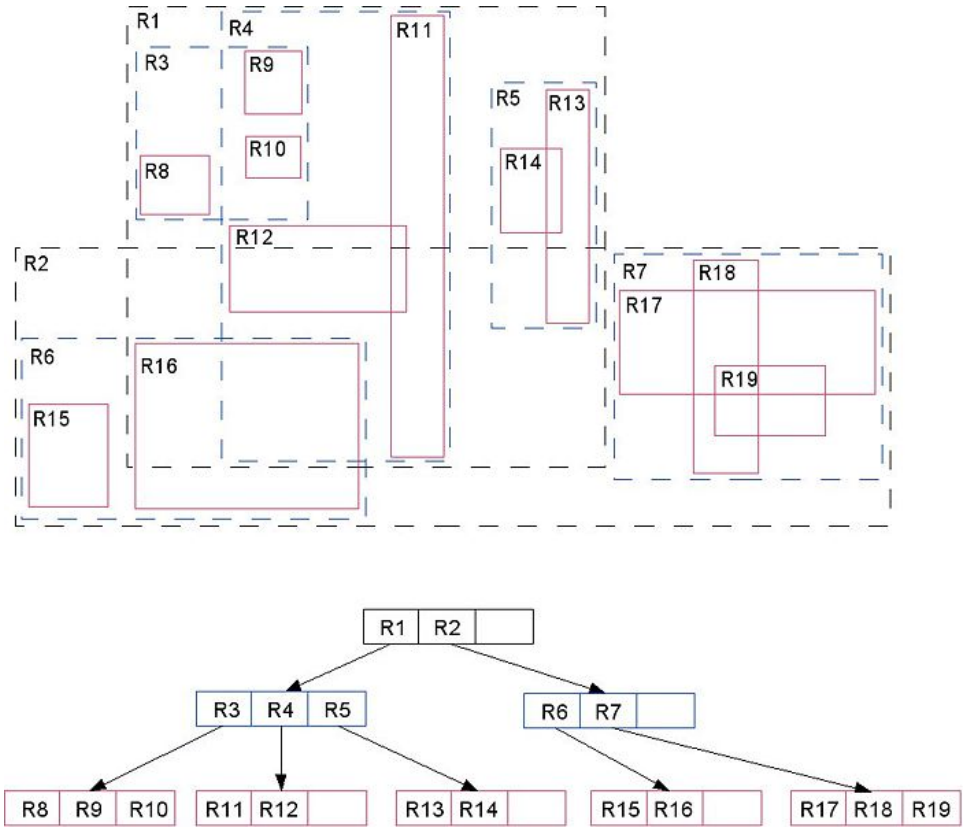

Рис. 2. *Пример структуры R-дерева с ветвлением 3*

ков (для двумерного пространства). В случае трехмерного или многомерного пространства это будут прямоугольные параллелепипеды (кубоиды) или параллелотопы.

Каждая вершина R-дерева имеет переменное количество элементов (не более некоторого заранее заданного максимума). Каждый элемент нелистовой вершины хранит два поля данных: способ идентификации дочерней вершины и ограничивающий прямоугольник (кубоид), охватывающий все элементы этой дочерней вершины.

R_a^*-деревья

Одной из наиболее эффективных и широко используемых модификаций R-дерева является R*-дерево [5]. Основными идеями R*-дерева являются:

1. Минимизация объёма, покрываемого каждым MBR-паралелотопом. Эта идея подразумевает минимизацию объёма, входящего в MBR-паралелотоп, но не относящегося к MBR-паралелотопам нижележащего уровня, минимизацию так называемого «мёртвого пространства» (*dead space*).
2. Минимизация перекрытия различных MBR-паралелотопов одного уровня. Перекрытие MBR-паралелотопов одного уровня приводит к тому, что к одному и тому же элементу можно пройти различными возможными путями по структуре дерева, что увеличивает количество элементов, которые необходимо рассмотреть при выполнении запроса.
3. Минимизация границ (периметров) MBR. Другими словами, придание MBR-паралелотопам «более квадратных форм» позволяет уменьшить объём вышестоящих MBR-паралелотопов.
4. Максимизация используемого свободного места на странице, что позволяет сократить общее количество MBR-паралелотопов и понизить высоту дерева.

Описанные идеи оказываются конструктивными при реализации действий 3 и 4 в процессе построения дерева.

Отметим, что в OLAP-системах (On-line Analytical Processing) также успешно применялись R_a^*-деревья [2], представляющие собой модификацию R-деревьев. Их основной идеей является хранение на внутренних страницах предвычисленных значений предполагаемых функций агрегирования (сумм, количества, средних значений) по всем нижележащим данным (рис. 3). Таким образом, если вся внутренняя страница попадает в область запроса, нет необходимости в чтении данных, в то время как имеется возможность использования предвычисленных значений.

Однако данная функция работает эффективно только при выполнении достаточно крупных запросов, либо при использовании разделения дерева таким образом, чтобы границы разделения страниц совпадали с естественными границами разделения данных, по которым наиболее вероятно будут проходить границы запросов. В большинстве случаев этого достаточно сложно достичь.

Массовая загрузка

Зачастую достаточно большая часть данных известна на момент построения индекса. Это даёт дополнительную возможность оптимизации построения индексирующей структуры, за счет использования алгоритмов массовой загрузки, подробное описание которых можно найти, например, в [2]. Все известные алгоритмы массовой загрузки можно разделить по способу разделения деления данных на восходящие и нисходящие алгоритмы. В соответствие с восходящими (*up load*) алгоритмами массовой загрузки осуществляют деление набора данных на группы, которые далее размещают в листовых страницах дерева, и затем из них группируют внутренние страницы более высокого уровня. В этом случае внутренние страницы, зачастую, имеют перекрывающиеся MBR-паралелотопы. В соответствие с нисходящими (*bulk load*) алгоритмами набор данных делят сначала на большие группы, которые затем становятся внутренними страницами верхнего уровня, и далее получившиеся группы делят на группы уровнем ниже. Отметим, что во втором случае внутренние страницы не имеют перекрывающихся параллеотопов.

В случае R-дерева одним из наиболее эффективных нисходящих алгоритмов массовой загрузки является алгоритм VAM-Split (Variance and median split), позволяющий построить сбалансированное дерево. Алгоритм разделения набора данных проходит в два этапа. На первом этапе выбирают ось (измерение) с максимальным среднеквадратичным отклонением координат данных по этой оси, по которой будет проходить разделение (рис. 4). Этим

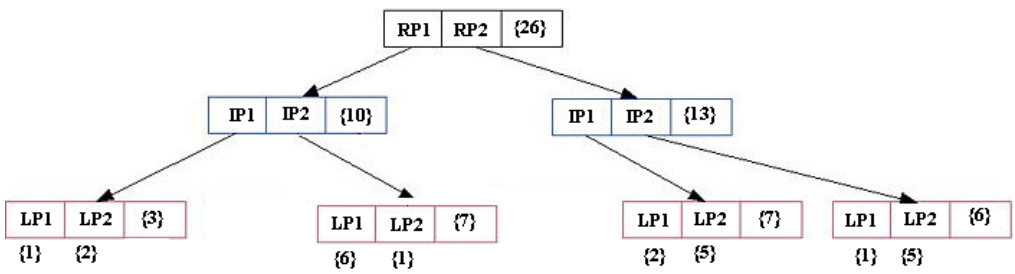

Рис. 3. *Пример структуры R_a-дерева с хранением предвычесленных сумм элементов данных*

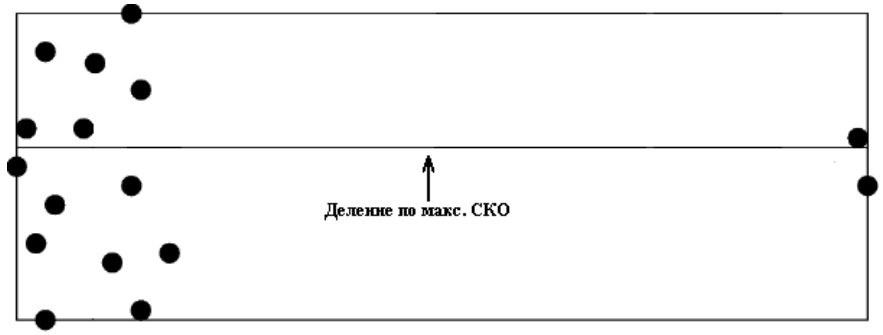

Рис. 4. *Выбор оси разделения с максимальным СКО*

достигается минимизация границ MBR-паралелотопов. Граница разделения выбирается около медианы по выбранной оси так, чтобы получить максимальное использование места на странице памяти, то есть так, чтобы количество элементов в одной из областей, на которые делится набор данных, было кратно максимальному количеству элементов в нижележащей ветви дерева M. Если общее количество данных менее чем $2M$, то используется граница, выбираемая исходя из кратности максимальному количеству элементов в ветви дерева на два уровня ниже. Это необходимо для обеспечения сбалансированности дерева.

Высота дерева может быть вычислена по формуле $h = [\log_f N/M+1]$ (приведено в [2]), где h – высота дерева, f – количество ссылок на дочерние страницы на внутренней, N – количество элементов данных, M – количество элементов данных, помещающихся на листовой странице, прямоугольными скобками обозначена операция взятия наибольшего ближайшего целого числа. В наихудшем случае в результате работы алгоритма VAM-Split из занятых остаются не полностью заполненными $2 \cdot h-1$ страниц.

Таким образом, достигается использование места на страницах памяти, близкое к 100 %.

При построении дерева с использованием алгоритма VAM-Split, для работы с большими объёмами данных на страницах памяти необходимо использовать алгоритмы внешней сортировки, описанные в [6].

Результаты тестирования

Описанные выше элементы технологий обработки многомерных данных были реализованы в виде самостоятельных программных модулей, в которых были использованы следующие алгоритмы индексирования:

1. B-дерева [7], в котором каждый многомерный запрос представляет собой полный перебор данных;
2. R*-дерева;
3. R*-дерева, построенного по алгоритму VAM-Split. (Здесь структура дерева строилась так, чтобы для наиболее часто встречающихся запросов запрос проходил по схеме аналогичной или максимально схожей со схемой поиска одного элемента в древовидной структуре.)

Тестирование проводилось на базе ПК «САПФИР», предназначенного для расчета сводной бюджетной росписи региона России. Здесь общее количество исходных записей в хранилище данных равняется 197 234 элементам. Записи состоят из 12 классификаторов (ключей), суммы и атрибутов данных. Размер записи равен 474 байтам, размер страницы – 8192 байтам. При организации дерева, использовались листовые страницы, на которых умещались 17 записей данных, и внутренние страницы, на которых размещались 36 ссылок на листовые страницы.

В ходе расчета был выполнен 9731 пространственный агрегирующий запрос.

При использовании для индексирования B-дерева [7] было проведено примерно 11000 чтений страниц памяти. Общее время выполнения запросов при использовании индексирующих механизмов Microsoft SQL Server (B-дерево) составило около 9 часов.

В случае использования R*-дерева общее количество обращений к страницам памяти составило 869952, т. е. в среднем 90 обращений для выполнения одного запроса. Высота дерева после вставки всех элементов – 5 уровней. Наполняемость страниц – 78 %. Общее время выполнения запросов с использованием R*-дерева составило 48 с.

В случае использования R*-дерева, построенного по алгоритму VAM-Split, общее количество обращений к страницам памяти составило 476593, т. е. в среднем 49 обращений для выполнения одного запроса. Высота дерева после конструирования индекса – 4 уровня. Наполняемость страниц близка к 100 %. Общее время выполнения запросов с использованием R*-дерева с VAM-Split построением составило 39 с.

Таким образом, для выбранного хранилища данных наиболее предпочтительным оказывается индексирование по методу R*-дерева, построенного по алгоритму VAM-Split.

В заключение отметим, что для обобщения полученного результат на другие хранилища данных требуются дополнительные исследования, т. к.,

очевидно, что оценки эффективности работы любого способа индексирования зависит от конкретного набора данных. Тестирование методов доступа к многомерным данным на случайных наборах данных приведенных, например, в [8] представляется нам непоказательным. Поскольку реальные наборы данных, особенно в случаях большого количества измерений, обнаруживают ярко выраженную кластеризацию. Как следствие оценки эффективности индексирования на случайных наборах, по нашему мнению, не имеет прямой связи с эффективностью индексирования реальных данных.

СПИСОК ЛИТЕРАТУРЫ

1. Gaede V., Gunter O. Multidimensional Access Methods // ACM Comput. Surv. – 1998. – V. 30. – № 2. – p. 170–231.
2. Manolopoulos Y., Nanopoulos A., Papadopoulos A., Theodoridis Y. R-trees: Theory and Applications. – Springer, 2006. – 194 p.
3. Hellerstein J., Kornacker M., Mohan C. Concurrency and Recovery in Generalized Search Trees // SIGMOD Record. – 1997. – V. 26. – № 2. – P. 62–72.
4. Guttman A. R-trees: A dynamic index structure for spatial searching // Proc. of the ACM SIGMOD Intern. Conf. on Management of Data, 1984. – P. 47–54.
5. Beckmann N., Kriegel H.-P., Schneider R., Seeger B. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles // Proc. of the ACM SIGMOD Intern. Conf. on Management of Data, 1990. – P. 47–54.
6. Graefe G., Implementing Sorting in Database Systems // ACM Comput. Surv. – 2006. – V. 38. – № 2. – P. 87–94.
7. Гарсиа-Молина Г., Ульман Дж., Уидом Дж. Системы баз данных. Полный курс – М.-СПб.: Вильямс, 2004. – 1088 с.
8. Шестаков Н.А. Индексирование пространственных данных в MS SQL Server 2000 // Известия Томского политехнического университета. – 2006. – Т. 309. – № 4. – С. 157–162.

Поступила 14.07.2008 г.

УДК 004.89

ОБ ИСПОЛЬЗОВАНИИ В ПРОГРАММИРОВАНИИ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ

С.С. Васильев, В.Б.Новосельцев

Томский политехнический университет
E-mail: vitaliin@gmail.com

Рассматривается популярный подход к повышению эффективности процесса разработки надежного программного обеспечения, который предполагает систематизированное применение механизма абстракции за счет использования так называемых проблемно-ориентированных языков. С учетом опыта работы в рамках представляемой парадигмы анализируются и обсуждаются достоинства и недостатки подхода, а также перспективы его развития.

Ключевые слова:
Проблемно-ориентированные языки, механизм абстракции, ИТ-проект, программный инструментарий.

Введение

В настоящее время вслед за теоретиками в области информационных технологий (ИТ) многие разработчики программного обеспечения сходятся во мнении, что механизм абстракции является одним из ключевых факторов, определяющих качество и эффективность реализации сложного программного продукта. Своего рода «венцом» целого ряда исследований по тематике *абстрактных типов данных* в 70-х годах прошлого века явился язык *CLU*, предложенный Барбарой Лисков [1]. Несомненна концептуальная значимость языка *CLU* – он определил одну из основ *объектно-ориентированного подхода*. В то же время продукт Б. Лисков не получил сколько-нибудь серьезного распространения в среде профессиональных разработчиков, прежде всего, вследствие излишней академичности и перегруженности специфическими конструкциями.

Достаточно очевидным является то, что разумно примененный механизм абстракции позволяет весьма эффективно осуществлять командную разработку, поддерживая при этом высокое качество программного продукта и эффективное управление процессом. Отложенные вычисления [2–6], модули [2–6], объекты [6, 7], манданты [5, 8] – все это лишь некоторые из современных инструментов введения и поддержки абстракции. Разные языковые среды поддерживают эти механизмы в той или иной степени, одни хорошо, другие не очень. Общеизвестно, что любые алгоритмические решения могут быть реализованы в рамках различных существующих парадигм программирования (выбор, в значительной степени, определяется личными предпочтениями). Закономерно возникает вопрос, что же может претендовать на роль «идеального» механизма абстракции при написании приложений, лежит ли он в плоскости технических средств