Уфа: УГАТУ, 2005 Вестник УГАТУ Т. 6, № 1 (12). С. 121–134

УДК 681.3

Ю. С. КАБАЛЬНОВ, А. А. ЛЕВКОВ

МОДЕЛИ ПРЕДСТАВЛЕНИЯ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

Предложены модели хранения данных для использования в информационных системах реального времени. Логическая модель представления данных — битовый гиперкуб, а физическая модель хранения данных основывается на многомерных d-k-d-деревьях. Использование моделей позволяет повысить устойчивость целостности логической модели хранения информации в рамках одной базы данных, производить поиск информации в режиме реального времени при высокой степени компактности данных на физических носителях при потоковом поступлении информации. *Базы данных; системы реального времени; многомерные деревья; быстродействие; надежность; компактность*

ВВЕДЕНИЕ

Функционирование информационных систем, работающих в режиме реального времени (on-line) (далее просто информационных систем реального времени (СРВ)), значительно отличается от работы традиционных информационных систем, работающих в режиме отложенного времени (off-line). Следует отметить, что работа в режиме реального времени подразумевает наличие жестких ограничений по времени принятия решения — управленческое решение должно быть принято до того, как система изменит свое состояние: любая задержка в выдаче информации может привести к сбою. Выполнение данного условия особенно актуально при работе с объектами ответственного назначения, сбой в функционировании которых может привести к серьезным материальным убыткам (к примеру, управление полетом самолета, работой завода и т. д.). В дополнение к этому проблема быстродействия усложняется необходимостью переработки больших массивов информации — современные системы используют сложные алгоритмы, основанные на анализе множества непрерывно поступающих различных по своей природе данных. Эти данные несут информацию о различных физических, химических и других процессах в рамках одной системы. Таким образом, современная СРВ является многоканальной информационной системой.

В процессе работы к информационной СРВ должен быть обеспечен многоуровневый доступ со стороны пользователей и прикладных процессов, с которыми происходит непрерывный обмен информацией (в том числе с помощью локальных и глобальных сетей). Сама же информационная СРВ должна обладать многофункциональностью, выражающейся в реализации функций управления объектом, режимами его работы, контроля, протоколирования, взаимодействия с другими системами и т. д.

Из отмеченных выше особенностей информационных СРВ следует, что они предъявляют повышенные требования к таким характеристикам моделей представления и хранения данных, как:

- надежность хранения и обработки информации, в частности к целостности хранения множества различных данных об одном объекте;
- скорость добавления и поиска данных;
- обеспечение компактного хранения данных на физических носителях;
- гибкость и глубина языкового интерфейса.

От выбора модели представления и хранения данных в информационных СРВ во многом зависит эффективность работы данных систем.

В целом, модели представления и хранения данных принято делить на логические и физические, различающиеся уровнем абстрагирования [1–7].

Логические модели представления данных описывают интерфейсы взаимодействия пользователей с системой, предоставляют различные языковые средства, предназначен-

ные для трансляции логических конструкций на физическую модель хранения данных. От логической модели зависят целостность хранения данных, гибкость и глубина языкового интерфейса [8].

Физические модели хранения данных обеспечивают оптимальное размещение информации на физических носителях (на жестких дисках, в оперативной памяти и т. д.) и описывают взаимодействие различных компонентов аппаратного обеспечения друг с другом. От них зависит скорость добавления и поиска данных, компактность физического хранения информации.

Существующие в настоящее время подходы к построению логических моделей представления данных в СРВ, как правило, базируются на реляционной модели данных. Применение данной модели позволяет осуществлять оперативные поиск и вставку простейших элементов данных. Однако использование реляционной модели для сложных многоканальных информационных систем, какими являются СРВ, приводит к необходимости построения сотен и тысяч взаимосвязанных нормализованных таблиц, что не только значительно замедляет процесс добавления/обработки данных, но и снижает надежность функционирования системы в целом. Кроме того, значительно усложняется построение аналитических запросов к такой системе, так как разработчику необходимо четко и ясно представлять себе всю структуру БД, а при выполнении необходимо одновременное обращение к большинству реляционных таблиц.

Все это привело к появлению гибридных СРВ, так называемых OLAP-систем, где выделяются оперативные, исторические и аналитические данные. Сами СРВ в их случае работают с оперативными и заранее сгенерированными аналитическими данными, не имея возможности напрямую производить анализ исторических данных [9, 10].

Данным системам также присущ ряд отрицательных черт: они неспособны действительно быстро и оперативно реагировать на резко изменяющуюся ситуацию в связи со значительной задержкой между приходом оперативных данных и их агрегацией в аналитические. Кроме того, данные системы не обладают необходимой гибкостью: в OLAP-структурах возможно быстрое выполнение только заранее заданного набора аналитических запросов, выполнение же произвольных запросов происходит крайне медленно [8, 9]. При резкой смене информации во входных потоках данных, например, в условиях критических ситуаций, когда необходимо оперативное выполнение запроса по произвольным критериям, данные системы демонстрируют наиболее низкую производительность.

Эти причины делают OLAP-системы неэффективными для СРВ. Таким образом, современные СРВ по-прежнему вынуждены обрабатывать небольшие наборы данных и не имеют возможности оперативно выполнять сложные аналитические запросы к историческим данным.

Как показал приведенный анализ, реляционный и гибридный OLAP-подходы обладают рядом недостатков. Необходимость построения множества таблиц и связей между ними не только снижает скорость добавления информации, но и замедляет выполнение аналитических запросов, повышает сложность их составления — пользователю необходимо детально представлять полную картину связей всех таблиц друг с другом, снижает надежность всей системы в целом. Использование же OLAP-систем в СРВ затрудняется невозможностью прямого анализа исторических данных.

Таким образом, модель представления данных для СРВ должна отвечать следующим требованиям:

- иметь простую структуру;
- позволять производить анализ любых данных;
- использовать простой язык для написания аналитических запросов.

Далее обратимся к физическим моделям хранения данных. В настоящее время в качестве физических моделей хранения данных, как правило, используются графовые структуры, называемые B+-деревьями. Однако их применение малоэффективно для построения СРВ. Это связано с тем, что функциональные особенности B+-деревьев оптимизированы не для СРВ, а, скорее, для универсальных задач хранения информации.

Как известно, данные для хранения, обработки и анализа в СРВ поступают непрерывно в потоковом режиме. Для таких данных наиболее актуальны операции вставки и поиска. Удаление таких данных — явление редкое, скорее даже наоборот, данные должны быть защищены от удаления и изменения. Эти данные являются «историческими», они отражают состояние системы и окружающей среды на конкретный момент времени. Кроме того, данные, описывающие объект, «много-

мерны», т. е. с одного объекта снимается множество параметров, именно совокупность которых определяет состояние объекта [6].

К сожалению, B+-деревья не позволяют производить поиск более чем по одному параметру объекта (первичному ключу в таблице). Для обеспечения таких возможностей используются инвертированные файлы, bitmap-индексы и другие дополнительные структуры, которые значительно увеличивают сложность всей системы хранения данных. Кроме того, применение данных структур также ориентировано на поиск по заранее заданным параметрам. Построение же всех возможных индексов по всем параметрам значительно замедляет операции добавления данных и увеличивает объем БД.

Таким образом, использование B+-деревьев в СРВ также является неэффективным и сопряжено с рядом компромиссных решений, которые значительно ограничивают гибкость и мощность системы обработки и анализа информации.

Для эффективной работы в СРВ модель хранения данных должна быть лишена недостатков, присущих B+-деревьям, и отвечать следующим требованиям:

- хранить данные и ключи в рамках единой структуры;
- целостно хранить различные данные об одном объекте;
- позволять производить быстрые добавление/поиск данных по любым параметрам (вторичным ключам).

Данная статья посвящена разработке новой логической и физической модели представления и хранения данных, обеспечивающей:

- высокую целостность данных;
- простой и мощный язык построения аналитических запросов;
- высокую скорость добавления и поиска элементов;
- компактность хранения данных.

Только общая интегральная оптимизация данных моделей на совместное использование позволит решить вышеописанные проблемы построения системы хранения данных для информационных СРВ.

1. ПОСТРОЕНИЕ ЛОГИЧЕСКОЙ МОДЕЛИ ПРЕДСТАВЛЕНИЯ ДАННЫХ[1]

Наиболее полно отвечающей данным требованиям моделью представления данных является битовая многомерная модель данных. В отличие от классической (рис. 1), данная модель имеет ряд существенных отличий, заключающихся в том, что из нее исключено понятие показателей. Вместо них используются так называемые флаги (признаки) присутствия. Суть этого нововведения состоит в том, что все группы показателей, которые имели бы место в традиционной многомерной модели представления данных, сведены каждая в свою отдельную размерность. Второе значительное отличие предлагаемой логической модели представления данных от классической модели состоит в том, что абсолютно все данные хранятся при помощи одного единственного гиперкуба, т. е. база данных имеет гиперкубическую архитектуру [4]. Это стало возможным благодаря тому, что информационное наполнение БД является нетипизированным, а следовательно, позволяющим внести в один и тот же гиперкуб данные любых типов. В классической же модели внесение данных других типов, как правило, влечет за собой появление нового гиперкуба, информационным наполнением которого и являются эти данные, вследствие чего такая база данных становится поликубической.

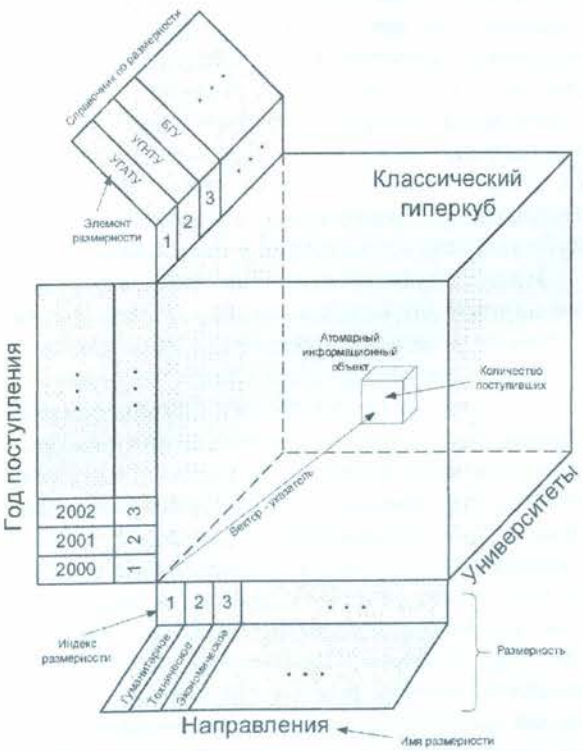

Рис. 1. Общий вид классической многомерной модели представления данных

[1] Раздел 1 основывается на исследованиях, выполненных авторами совместно с канд. техн. наук Д. В. Ивлевым

Основное преимущество гиперкубической модели данных состоит в том, что она позволяет представить данные в виде единой, целостной структуры. Это значительно облегчает решение такого важного вопроса, как централизованность и целостность информации. Кроме того, в классической поликубической модели множество гиперкубов соединяются между собой неким подобием реляционных отношений, что отрицательно сказывается на надежности системы [1, 4, 7]. В гиперкубической модели такой проблемы не возникает ввиду отсутствия подобных связей.

Основной недостаток гиперкубической модели представления данных состоит в сложности работы с так называемыми «независимыми» данными. Для преодоления данной проблемы используется метод синтеза гиперкуба.

Следует сразу заметить, что наборы данных должны быть связаны между собой хотя бы одним определяющим атрибутом, иначе нарушается принцип взаимосвязанности (аналогия ему есть и в реляционной теории баз данных) и в итоге мы получаем несколько отдельных, независимых друг от друга баз данных.

Если же наборы данных не связаны между собой явным образом, то прямой метод формирования гиперкуба представляется невозможным, так как *некоторые срезы этого гиперкуба не имеют смысла*. Из сложившейся ситуации есть два выхода: первый — организация базы данных по поликубической схеме, второй — искусственный синтез гиперкуба. Второй способ является более предпочтительным, так как позволяет сохранить гиперкубическую схему строения базы данных.

При искусственном синтезе гиперкуба атомарный объект последнего формируется путем логического умножения содержимого наборов данных.

Для рассмотрения общего случая синтеза гиперкуба, т. е. для его формирования из произвольного числа различных наборов данных, необходимо ввести понятие транзакции синтеза. В простейшем случае, когда все наборы данных связаны одним определяющим атрибутом, синтез гиперкуба выполняется за один шаг, т. е. является однотранзактным. На практике же наборы независимых данных, как правило, можно разбить на группы, в пределах каждой из которых эти наборы будут связаны по определяющему атрибуту, характерному для этой группы. Число определяющих атрибутов равняется числу таких групп.

Необходимым условием соблюдения принципа взаимосвязанности является наличие для любой произвольно взятой группы соответствующей ей парной группы, такой, чтобы в обеих группах нашлось хотя бы по одному набору данных, которые можно было бы связать по атрибуту, причем этот атрибут не обязательно должен совпадать с определяющими атрибутами любой группы из пары. Поясним данное определение на рис. 2.

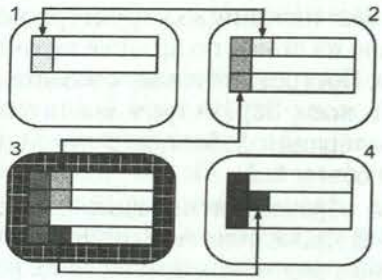

Рис. 2. Иллюстрация принципа взаимосвязанности наборов независимых данных

Здесь цифрами 1–4 помечены группы, в каждой из которых содержатся наборы независимых данных. В каждой группе эти наборы связаны между собой по определяющему атрибуту. Для простейшего случая, рассмотренного выше, когда все наборы данных связаны одним определяющим атрибутом, иллюстрацией является одна первая группа. Общий же случай предполагает наличие нескольких групп со своими определяющими атрибутами. Какую бы группу мы ни взяли, для нее обязательно найдется парная группа. Например, пара 1–2 связана через определяющий атрибут первой группы, пара 2–3 — через определяющий атрибут второй группы, а пара 3–4 — через атрибут, не являющийся определяющим ни в третьей, ни в четвертой группах, но присутствующий в них обеих.

Наборы данных, имеющие атрибуты, через которые производится связь групп между собой, условимся называть «связующими». Синтез гиперкуба в этом случае выполняется за несколько шагов (транзакций), т. е. является многотранзактным. Количество транзакций синтеза равняется числу вышеупомянутых групп. Многотранзактный синтез гиперкуба происходит рекурсивно. Каждая транзакция оперирует двумя аргументами и возвращает в качестве результата два параметра, являющихся аргументами для следующей транзакции. Первым аргументом транзакции является одна из вышеупомянутых групп данных, вторым ее аргументом служит дополнительный набор данных, как прави-

ло, представляющий собой результат выполнения предыдущей транзакции. В процессе выполнения транзакции можно выделить две подзадачи:

• нахождение группы наборов данных, парной первому аргументу текущей транзакции. Поиск такой парной группы производится среди тех групп, которые еще не являлись аргументами транзакции. Найденная таким образом группа передается в следующую транзакцию в качестве ее первого аргумента;

• синтез набора данных на основе обоих аргументов текущей транзакции. Исключение составляет первая транзакция, для которой синтез происходит только по первому аргументу в силу того, что второй ее аргумент еще не определен. Результат такого синтеза передается в следующую транзакцию в качестве ее второго аргумента.

Таким образом, результатом первой транзакции является набор данных, представляющий собой гиперкуб. Следующие же транзакции добавляют в этот гиперкуб новые размерности и соответственно изменяют его информационное наполнение.

Вышеприведенная методика синтеза многомерного гиперкуба имеет один значительный недостаток — она применима лишь для того случая, когда определяющий атрибут, по которому связаны наборы данных, содержит одинаковое число одинаковых элементов для каждого из наборов данных.

Для решения этой проблемы применяется так называемый «расширенный» синтез гиперкуба. Для этого размерности, не являющиеся определяющим атрибутом, дополняются при необходимости «пустым» элементом (nil-элементом), а формирование общего определяющего атрибута происходит посредством выборки из всех наборов данных неповторяющихся значений определяющих атрибутов этих наборов.

Таким образом, применение «расширенного» синтеза гиперкуба позволяет решить проблемы централизованности и целостности хранения информации в рамках одной базы данных. Так как гиперкуб не содержит связей с другими объектами, они не могут быть потеряны или нарушены [4].

Далее обратимся к способу формирования запросов к многомерной БД. За выполнение этой задачи отвечает алгебра операций над объектами многомерного гиперкуба. В основе данной алгебры лежат элементарные операции, производимые над атомарными информационными объектами битовой многомерной модели данных [3]. Элементарными они называются потому, что обусловливают изменения состояний атомарных информационных объектов, т. е. процессы перехода из одного состояния в другое. Так как атомарный объект может принимать только значения из множества {0,1}, то смена его состояния также может быть описана только двумя переходами — из нуля в единицу и обратно. Первый переход назовем «вскидыванием» атомарного объекта, второй — «сбрасыванием».

Такими операциями являются удаление, изменение и добавление. Именно равноправие этих операций позволяет реализовать мощные, гибкие системы, применимые в широком диапазоне проектов, в отличие от известных концепций хранилищ и витрин данных.

Предлагаемая для многомерной битовой модели данных алгебра операций над многомерными объектами состоит из трех групп операций:

• первая группа является набором традиционных операций над множествами, модифицированных с учетом того, что их операндами являются информационные объекты гиперкуба (ИОГ), а не произвольные множества;

• вторую группу составляют модифицированные операции дискретной математики, поскольку любой ИОГ, над которым производятся действия, является битовым пространством;

• в третью группу входят специальные операции, характерные лишь для многомерной алгебры.

Весь спектр этих операций определен для однотипных информационных объектов гиперкуба, т. е. для тех ИОГ, число и типы размерностей которых совпадают. Для разнотипных ИОГ вводится другой базис операций, которые учитывают специфику различия типов операндов.

Кроме того, алгебра операций над многомерными объектами включает в себя операции над множествами размерностей ИОГ, которые тоже будут рассмотрены при обзоре операций над разнотипными информационными объектами.

Рассмотрение основных операций алгебры операций над многомерными объектами начнем с первой группы. В эту группу входят такие операции, как *объединение* и *пересечение*. В реляционной алгебре, кроме них, определены еще операция декартова произведения, возвращающая все возможные сочетания

элементов двух ее операндов, и операция вычитания, возвращающая элементы, имеющиеся лишь в одном из операндов и не имеющиеся в другом. Но в силу того, что сама битовая многомерная модель данных построена таким образом, что изначально содержит в себе все возможные сочетания элементов размерностей, операцию декартова произведения будем считать априорно перекрытой, а операцию вычитания не будем рассматривать вообще, так как в многомерной специфике она имеет смысл лишь при наложении таких значительных ограничений на ее операнды, что оказывается проще декомпозировать ее на элементарные операции.

Следующая группа дискретных операций включает в себя все базовые отношения математической логики (такие как конъюнкция, дизъюнкция, исключающее «или», импликация, эквиваленция, штрих Шеффера и стрелка Пирса). Следует также учесть, что модификация этих операций с учетом многомерной специфики привносит в них определенные аспекты, присущие рассмотренной выше первой группе операций; а именно: на аргументы второй группы операций накладываются те же самые ограничения однотипности. Более того, операции второй группы вообще не могут быть определены вне контекста какой-либо операции из первой группы. Объясняется это тем, что для корректной работы дискретных операций необходимо соблюдение условия равносхемности операндов, а действия с равносхемными частями двух равнотипных ИОГ осуществляются лишь в операциях объединения и пересечения.

Теперь рассмотрим последнюю, третью группу операций многомерной алгебры. Аналогами этих операций в базисе Кодда являются *выборка*, *проекция*, *соединение* и *деление*. В случае предлагаемой битовой многомерной модели данных операции выборки и проекции не рассматриваются, так как аналогичные им действия достигаются получением соответствующих срезов. Следовательно, эти операции тоже считаются априорно перекрытыми. Операциям же соединения и деления ставятся в соответствие такие операции многомерной алгебры, как *порождение* и *проецирование*. Кроме того, в эту же группу входят еще две немаловажных операции *надстройки* и *усечения* ИОГ.

Таким образом, существующая алгебра операций над битовым многомерным гиперкубом позволяет осуществлять выборку любых его данных или его изменение. Следует отметить, что все операции, кроме элементарных, позволяют работать с множеством данных, однозначно определяемым координатами на осях гиперкуба. Следовательно, в пределах одной операции возможно обращение к множеству данных любой мощности.

Значительным преимуществом данной алгебры над реляционной является работа с данными в рамках одного объекта (гиперкуба), не имеющего рекурсивных связей с другими объектами и не ограниченного рекурсивными правилами.

Таким образом, применение битовой гиперкубической модели логического представления данных позволяет в значительной мере упростить структуру самой БД, и, следовательно, упростить разработку запросов к ней.

2. ПОСТРОЕНИЕ ФИЗИЧЕСКОЙ МОДЕЛИ ХРАНЕНИЯ ДАННЫХ

Очевидно, что базисом для построения подобных моделей хранения данных должны являться пространственные системы хранения, наиболее типичные для ГИС, так как именно в них возможен поиск более чем по одному параметру (k-d-, R-, bh-деревья). Однако данные модели невозможно напрямую использовать в СРВ в связи с их узкой направленностью на свою предметную область: отдельные из них имеют возможности поиска не более чем по 2–3 параметрам (осей координат для ГИС), в них допускается частичное дублирование информации и т. д. [5].

В результате синтеза и переработки таких моделей хранения данных, как B+-деревья, R-деревья и k-d-деревья, предлагается новая структура организации хранения данных, адаптированная для использования в СРВ — динамическое k-d-дерево (d-k-d-дерево) [1, 2]. Отметим все структурные особенности d-k-d-дерева:

- каждый узел имеет не более m потомков;
- каждый узел, за исключением листьев, имеет не менее n потомков;
- координата ветвления определяется самостоятельно для каждого узла;
- каждый лист заполнен не менее чем на долю s;
- все листья находятся на одном уровне.

На рис. 3 представлен общий вид k-d-B+-дерева для количества измерений (k), равного 2, и при коэффициенте насыщенности (c) 5.

Как правило, все уровни структуры, кроме листового, хранятся в ОЗУ, тогда как листья хранятся на жестком диске.

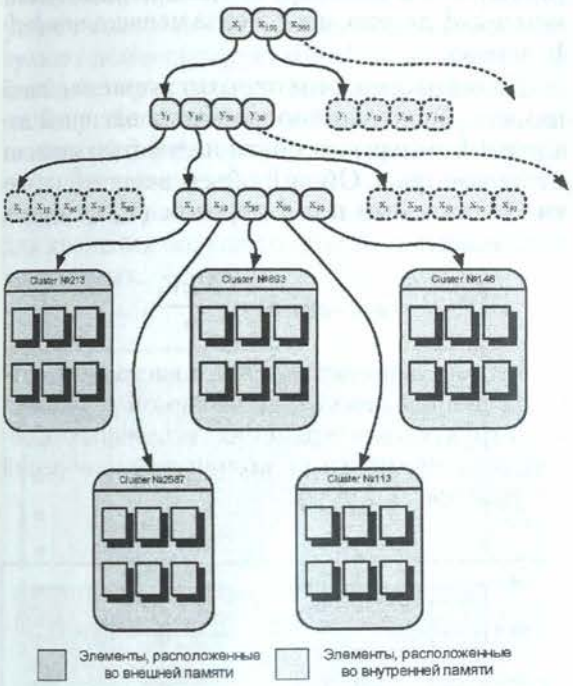

Рис. 3. d-k-d-дерево ($k = 2$)

Именно динамическое определение координаты ветвления для каждой вершины позволяет гарантировать заполненность каждой вершины в пределах от n до m. Это связано с нижеописанными причинами.

Во-первых, при жесткой фиксации определенной координаты ветвления для каждого уровня возможна тупиковая ситуация, при которой ветвление по одной из координат будет невозможным. Такая ситуация возникает, если большинство многомерных данных в какой-то части дерева имеют одинаковое значение данной координаты. В таком случае ветвление по данной координате невозможно, так как она одинакова для всех хранимых элементов. В k-d-B-дереве, предложенном Робинсоном, в данном случае происходило формирование пустого поддерева, не заполненного данными [5]. Количество таких поддеревьев в динамически растущей структуре было достаточно велико, что не позволяло эффективно использовать внутреннюю и внешнюю память.

Во-вторых, даже когда ветвление в узле по данной координате возможно, не существует уверенности в том, что ветвление именно по данной координате будет наиболее эффективным. В данном случае, в результате жестко заданной координаты ветвления, поддеревья узла могут значительно отличаться по мощности (т. е. по количеству хранимых элементов), что приводит к невозможности контроля за количеством памяти, занимаемым k-d-B-деревом [6].

Динамический выбор координаты ветвления в каждом узле позволяет избежать таких проблем, так как для ветвления выбирается координата, наилучшим образом подходящая для образования равномощных поддеревьев.

В случае, когда количество вторичных ключей в структуре велико, необходим выбор из большого количества координат, обеспечивающих примерно одинаково удачные варианты ветвления. Данный выбор может осуществляться по многим критериям. Основной из них: использование для ветвления той координаты, по которой ранее ветвление происходило реже всего. Это условие позволяет обеспечивать примерно одинаковое время поиска элемента по любой из координат и предотвращает построение дерева меньшей мерности (тот случай, когда для построения d-k-d-дерева могла бы использоваться всего одна координата ветвления).

Следует отметить, что при построении d-k-d-дерева не всегда возможно осуществить ветвление по всем существующим координатам. Это происходит тогда, когда в структуре хранится сравнительно небольшое число записей, состоящих из большого количества вторичных ключей. В этом случае количество уровней в дереве может быть меньшим, нежели количество вторичных ключей.

Объем памяти, занимаемый деревьями, главным образом зависит от количества хранимых данных и их размера. Так как структура d-k-d-дерева во многом зависит от распределения поступающих данных, то для их анализа будет использоваться «усредненная» модель, находящаяся между «оптимальным» и «вырожденным» деревом. Для k-d-B-деревьев наиболее актуален порядок поступления данных, поэтому будем считать, что данные поступают в случайном порядке [5].

Объем памяти, занимаемый несжатым многомерным массивом, зависит только от количества измерений в нем и от количества элементов в измерении. Однако в настоящее время наиболее широко распространены сжатые многомерные массивы, в которых вместо хранения длинной последовательности нулей хранится их количество. Кроме того, данное сжатие применяется к блокам многомерного массива, размер и структура которых определяются конкретным программным продуктом и недоступны в открытых источниках. Таким образом, сравнения будут производиться с несжатым многомерным массивом с эмпири-

чески полученными усредненными значениями для программного продукта MS-MOLAP.

Кроме того, значительным отличием деревьев от массивов является то, что деревья изначально ориентированы на хранение информации как во внутренней, так и во внешней памяти, в то время как массивы ориентированы на работу только с внутренней. В то же время объем памяти, занимаемый многомерными массивами, сравним с объемом памяти, занимаемым деревьями во внешней памяти, поэтому сравнение многомерных массивов с деревьями будет производиться только по использованию общей памяти всей структурой.

Количество уровней (U) в d-k-d-дереве:

$$U = \frac{\ln N_{cl}}{\ln c},$$

где c — коэффициент насыщенности, N_{cl} — количество кластеров на жестком диске, содержащих данные.

Отсюда, если принять размер всех ключей одинаковым: $V_{s\,i} = V_{s\,j}$, где $i = \{1..k\}$, $j = \{1..k\}$, объем оперативной памяти, занимаемой d-k-d-деревом, составит

$$V_{\text{RAM}} = \frac{c^U - 1}{c - 1}(16 + cV_s).$$

На рис. 4 представлена зависимость объема оперативной памяти, занимаемой d-k-d-, a-k-d- и k-d-B-деревьями, от их коэффициента насыщенности (объем оперативной памяти, занимаемый последними, рассчитан согласно методам, описанным в [1, 2, 5, 6]).

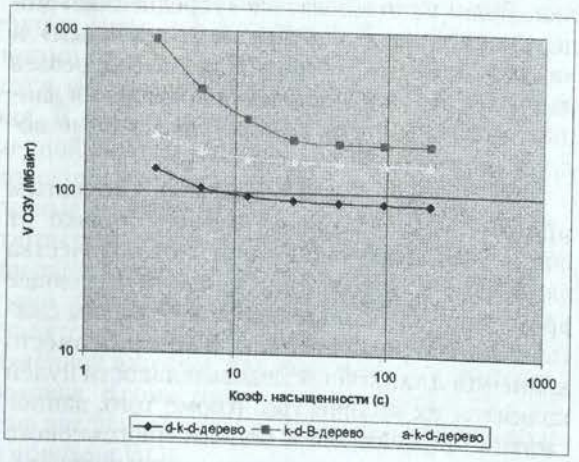

Рис. 4. Зависимость объема оперативной памяти, занимаемой k-d-деревьями, от их коэффициента насыщенности

Как можно видеть из данных зависимостей, предложенное d-k-d-дерево занимает в оперативной памяти в 1,8 раза меньше места, чем a-k-d-дерево, и в 2,5 раза меньше, чем k-d-B-дерево.

Далее рассмотрим использование внешней памяти. Степень использования внешней памяти d-k-d-деревом почти полностью зависит от параметра s. Общий объем внешней памяти, занимаемый d-k-d-деревом ($V_{d-k-d\,\text{HDD}}$), составляет:

$$V_{d-k-d\,\text{HDD}} = \frac{NkV_s}{s}.$$

На рис. 5 представлены зависимости объема внешней памяти, занимаемой различными структурами хранения вторичных ключей, в зависимости от количества измерений в структуре [1, 2, 5, 6].

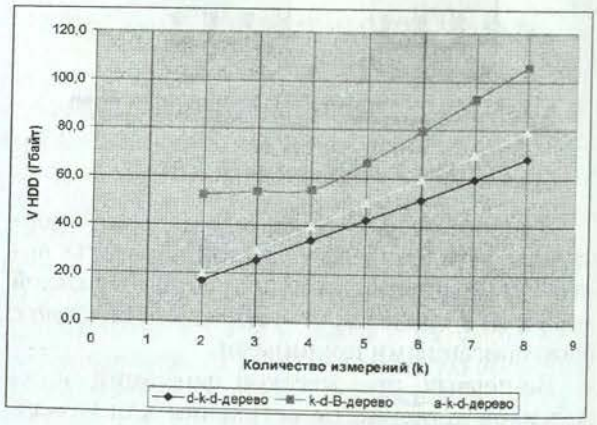

Рис. 5. Зависимости объема внешней памяти от количества измерений в структуре

Как видно из данных зависимостей, d-k-d-дерево занимает на жестком диске чуть меньший объем памяти, нежели a-k-d-деревья (меньше в 1,17 раза), и значительно меньший объем памяти, нежели k-d-B-дерево (меньше в 1,5–3 раза).

Далее рассмотрим общий объем памяти, занимаемый данными структурами и многомерными массивами (рис. 6).

На данном графике изображены зависимости для d-k-d-дерева, «идеального» хранилища, т. е. такого, которое не занимает дополнительной памяти, а состоит исключительно из хранимых данных для несжатого многомерного массива и для многомерного массива, сжатого по технологии фирмы Microsoft. Как можно видеть из данных зависимостей, несжатый многомерный массив занимает наибольший объем памяти, на порядки больший, чем «идеальное» хранилище. Сжатый многомерный массив показывает неплохие показатели при небольшом количестве

измерений, при их возрастании его размер увеличивается по экспоненте. Наилучший результат демонстрирует *d-k-d*-дерево, которое наиболее приближено к «идеальному» хранилищу. Дополнительный объем памяти, занимаемый *d-k-d*-деревом, находится в линейной зависимости от количества измерений в структуре, что делает их наилучшим выбором для хранения большого объема многофакторных данных.

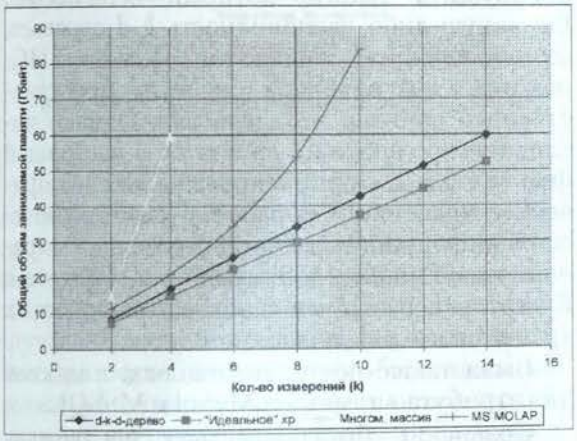

Рис. 6. Общий объем памяти, занимаемый различными структурами хранения данных

Далее рассмотрим быстродействие предложенной структуры хранения данных. В данном разделе будут сравниваться *d-k-d-*, *a-k-d-* и *k-d*-B-деревья. Сравнение с многомерными массивами в данном разделе не производится в связи с тем, что последние не предлагают никаких средств по работе с внешней памятью, размещая все данные в оперативной памяти. Разумеется, при объеме хранимых данных в десятки и сотни гигабайтов многомерные массивы будут вынуждены постоянно транслировать данные из внешней памяти во внутреннюю, что вызовет значительное падение их производительности. При объеме хранимых данных в 10 Гбайт ответ даже на простейший запрос поиска может занять 10–15 минут — именно столько времени необходимо на чтение 10 Гбайт данных с жесткого диска.

Скорость поиска по древовидной структуре зависит от двух параметров: скорости поиска в узле дерева и количества уровней в структуре, т. е. количества выполнения поисков по узлам. Используя алгоритм граничного интерполяционного поиска в *d-k-d*-дереве, мы можем уменьшить время выполнения поиска в узле дерева по сравнению с a-k-d- и *k-d*-B-деревьями, в которых используется бинарный поиск. Рассчитанные данные для алгоритмов успешного поиска во внутренней памяти отражены на рис. 7.

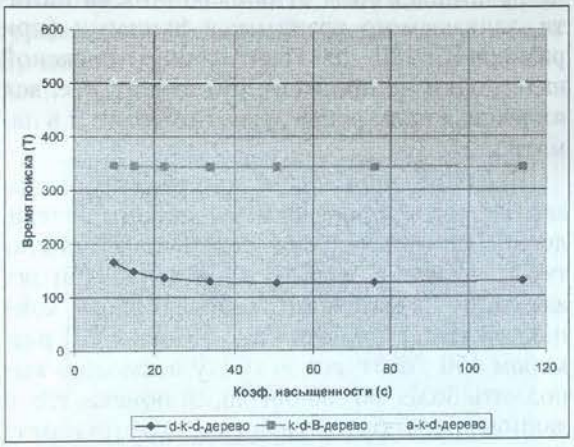

Рис. 7. Зависимость времени выполнения поиска от коэффициента насыщенности

Как можно видеть из графика, *d-k-d*-дерево демонстрирует самое высокое быстродействие. Это связано не только с использованием алгоритма граничного интерполяционного поиска, но и с хорошей сбалансированностью дерева.

На рис. 8 изображены зависимости времени поиска элемента в структуре от количества измерений в ней.

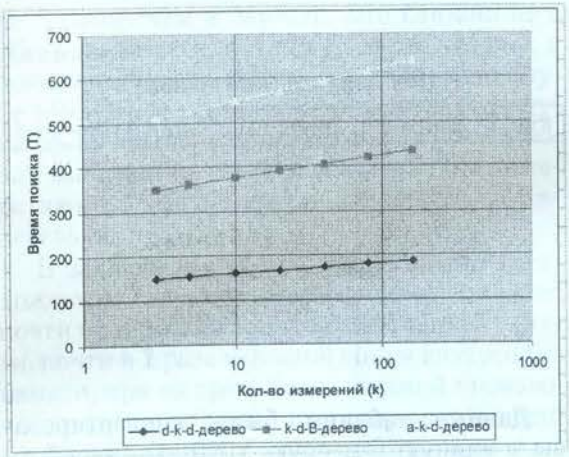

Рис. 8. Зависимость времени выполнения поиска от количества измерений в структуре

Как видно из данных зависимостей, предложенное *d-k-d*-дерево сохраняет свою высокую эффективность при росте количества измерений в структуре.

В заключение можно сказать, что предлагаемая структура обеспечивает экономное

хранение данных в памяти: хранятся только реально присутствующие данные, а не весь разреженный массив. Доля дополнительной памяти, занимаемой индексной структурой, не превышает 0,2% от общего объема памяти, занимаемого хранимыми данными (при размере БД 150–200 Гбайт размер индексной надстройки не превысит 180 Мбайт, т. е. вся индексная надстройка легко поместится в памяти).

Предложенная индексная структура позволяет также производить операции поиска, добавления и удаления элементов из структуры в режиме реального времени. При реализации приведенного выше кода на компьютере Intel Pentium IV – 1 GHz в БД размером 150 Гбайт за 1 секунду возможно выполнить более 30 000 операций поиска, т. е. в данном случае быстродействие ограничивается исключительно быстродействием жесткого диска. Опыты показали, что реально удается провести около 200 операций поиска (применение массива SCSI-винчестеров позволяет существенно поднять данный параметр).

3. ПРИМЕР

В качестве тестовых данных выступали данные учета пользовательского трафика, хранимые в СУБД MySQL. Размер хранимых данных в БД составлял 154,3 Гбайт при общем количестве хранимых записей около 2,3 млрд, хранимых в табл. 1, 2.

Таблица 1

Пользовательские данные

#id	ip_sourse	ip_target	time_start	time_end

Таблица 2

Данные трафика

#id	v_data	n_spec

Данные таблицы были конвертированы в единую 6-мерную МБД. По атрибуту «n_spec», содержащему описание передаваемой информации, был сформирован справочник на основе Bp-дерева. Количество хранимых ключей в Bp-дереве составило 1,1 млрд. В d-k-d-дерево хранения информации по вторичным ключам были помещены следующие атрибуты: «ip_sourse», «ip_target», «time_start», «time_end», «v_data», индекс по «n_spec».

Для проведения комплексного сравнения данная структура была экспортирована в MSSQL в виде аналогичных реляционных таблиц. Далее, было построено ROLAP-хранилище данных. Построение MOLAP-хранилища оказалось невозможным, так как все попытки создать МБД на основе реляционной БД объемом более 2 Гбайт заканчивались безуспешно.

В существующих на данный момент на рынке коммерческих продуктах не удалось обнаружить такого, который использовал бы какую-либо разновидность k-d-деревьев, за исключением специализированных ГИС-пакетов, позволяющих хранить пространственные данные в R-деревьях. Однако последние продукты не являются в настоящей форме СУБД, а ориентированы на специфическое хранение и обработку картографической информации, хотя эффективность хранения двухмерных и трехмерных объектов в R-деревьях сравнима с эффективностью их хранения в d-k-d-деревьях.

Была также построена таблица индексов для атрибута «n_spec» в MySql и MSSQL.

Сравнение быстродействия предложенных методик хранения производилось на компьютере Intel Pentium IV 2,1 ГГц, 2048 Мбайт ОЗУ, RAID 2 SCSI 4×120 Гбайт, операционная система Windows 2003 Server.

Оценивались:

- время поиска индекса «n_spec» по заданному атрибуту в Bp-дереве;
- объем памяти, занимаемый Bp-деревом;
- общее время поиска записи в системе;
- объем памяти, занимаемый d-k-d-деревом.

Все измеренные данные сравнивались с аналогичными показателями в СУБД MySQL, MSSQL и MS-ROLAP.

На рис. 9 показана зависимость времени поиска индекса «n_spec» по заданному атрибуту от коэффициента насыщенности Bp-дерева. Данные были получены путем измерения времени поиска всех элементов в БД и делением полученного результата на количество элементов в структуре. Проведение испытания подобным образом обеспечивало одинаковое кэширование кластеров жесткого диска операционной системой (общий объем памяти, выделенный под дисковый кэш, составил 512 Мбайт). Определить коэффициент насыщенности B+-деревьев в данном испытании не представлялось возможным. По косвенным данным, эти коэффициенты могли находиться в пределах 30 … 100.

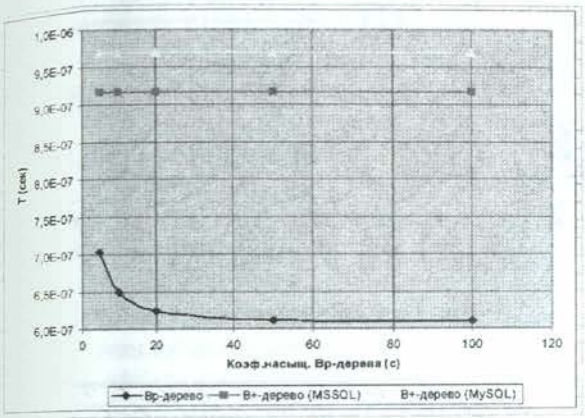

Рис. 9. Зависимость времени поиска от коэффициента насыщенности

Как видно из данных зависимостей, Вр-дерево демонстрирует на 30–40% более высокую производительность, нежели B+-деревья в СУБД MSSQL и MySQL. Более высокая производительность MSSQL в данном случае объясняется более высокой степенью заполненности кластеров и собственным механизмом хеширования, работающим независимо от операционной системы. Разница в производительности между B+-деревьями от MSSQL и MySQL не превышает 6% и объясняется незначительными отличиями алгоритмов и структур.

В целом, данное практическое испытание подтверждает теоретические расчеты. Меньшее влияние коэффициента насыщенности на скорость работы Вр-дерева объясняется тем, что значительная доля времени тратится на обращения к жесткому диску, количество которых не зависит от коэффициента насыщенности.

В табл. 3 отображен объем памяти, занимаемый Вр- и B+-деревьями на жестком диске. Коэффициент заполненности кластера (s) был задан в 3/4.

Таблица 3

Объем памяти, занимаемый структурами хранения по первичным ключам

	B+-дерево (MSSQL)	B+-дерево (MySQL)	Вр-дерево
Объем на жестком диске, Гбайт	22,05	23,79	17,56

Как можно видеть из данной таблицы, Вр-дерево занимает на 27% меньший объем на жестком диске, нежели B+-дерево в MySQL, и на 20% меньший объем, нежели B+-дерево в MSSQL. При размере структуры в 10–20 Гбайт это является серьезным улучшением структуры.

На рис. 10 представлены зависимости объема оперативной памяти, занимаемой структурами, от коэффициента насыщенности (для Вр-дерева).

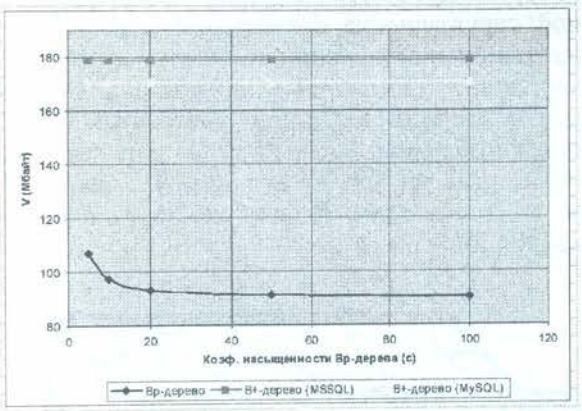

Рис. 10. Зависимость объема ОЗУ от коэффициента насыщенности

Следует отметить, что данная информация получена косвенным путем: путем вычисления разности объема оперативной памяти, занимаемой загруженными модулями программ MSSQL и MySQL, без подключенных баз данных и с ними. Как видно из данного графика, объем оперативной памяти, занимаемой B+-деревом в MSSQL, несколько больше, чем в MySQL. Это связано не с объемом памяти, занимаемым структурой, а дополнительным кэшем, который использует MSSQL. В целом, предложенная структура организации данных по первичным ключам занимает на 40–50% меньший объем оперативной памяти, нежели структуры на B+-деревьях.

В заключение можно отметить, что предложенная структура справочников занимает почти на треть меньший объем внешней памяти, почти в 2 раза меньший объем внутренней памяти, при на треть более высокой производительности, нежели у известных реализаций справочников на B+-деревьях. Это позволяет добиться решения более сложных задач поиска данных без усовершенствования аппаратного обеспечения.

Так как стандартные средства OLAP-систем Microsoft и Oracle не предлагают встроенных средств по решению проблемы высокой избыточности многомерного хранения данных, авторами предлагается использовать предложенные d-k-d-деревья в каче-

стве структуры размещения данных на физических носителях.

Многомерное хранилище данных строится на основе справочников размерностей, описанных выше. Задачей многомерного хранилища является связь значений справочников (отложенных по осям МБД) и показателей, являющихся элементами хранения многомерных кубов. При переложении многомерной структуры на *d-k-d*-деревья это выражается в сопоставлении конкретному набору координат конкретной хранимой записи. На рис. 11 представлена такая трехмерная МБД, сформированная по атрибутам «ip_source», «ip_target», «n_spec» и «v_data». Показателями в данном случае являются объемы переданной информации.

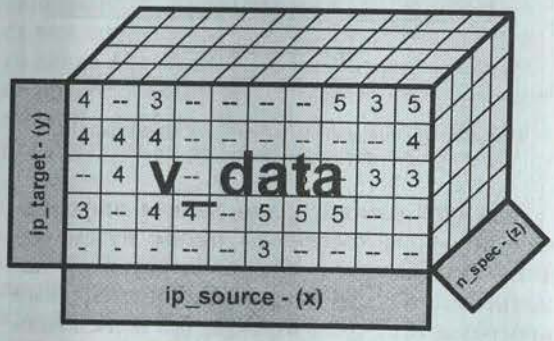

Рис. 11. Трехмерная БД успеваемости

Данная МБД в наглядном виде может быть представлена срезами (табл. 4).

Таблица 4

Срез таблицы по атрибуту n_spec

		Ip_source			
		(1)	(2)	(3)	(4)
ip_target	(1)	4	4	–	3
	(2)	3	–	–	–
	(3)	5	4	5	–

В виде *d-k-d*-дерева данная структура будет выглядеть так, как показано на рис. 12 (коэффициент насыщенности = 3).

В случае использования битовой МБД, в которой отсутствуют показатели как таковые, а наполнение лишь идентифицирует существование объекта, конкретный набор координат и будет описывать собой хранимый объект, без сопоставления ему каких бы то ни было записей. На рис. 13 и в табл. 5 представлена такая трехмерная битовая МБД, сформированная по атрибутам «ip_source», «ip_target» и «v_data».

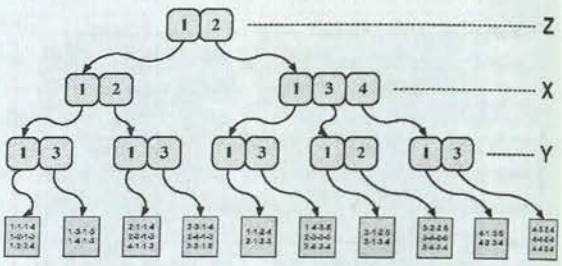

Рис. 12. *d-k-d*-дерево успеваемости

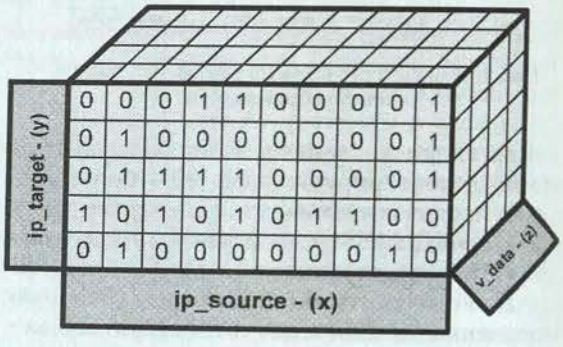

Рис. 13. Битовая многомерная БД успеваемости

Таблица 5

Срез таблицы по атрибуту v_data

		Ip_source			
		(1)	(2)	(3)	(4)
ip_target	(1)	1	0	0	0
	(2)	0	1	1	0
	(3)	0	0	0	0
	(4)	0	0	0	0
	(5)	1	0	0	0
	(6)	0	0	0	0

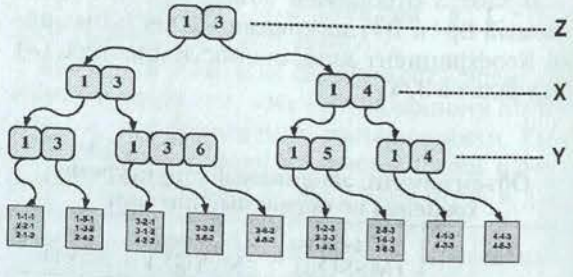

Рис. 14. *d-k-d*-дерево успеваемости

В виде *d-k-d*-дерева данная структура будет выглядеть так, как показано на рис. 14 (коэффициент насыщенности = 3).

Как можно видеть из примеров, *d-k-d*-деревья способны хранить данные как для поликубических, так и гиперкубических моделей, без потери эффективности.

Далее сравним время поиска данных в предложенной структуре d-k-d-дерева в реляционных хранилищах MSSQL и MySQL, а также в ROLAP-системе, построенной на основе MSSQL. Для проведения корректного сравнения в реляционных системах измерялось быстродействие индексированных реляционных таблиц. Дополнительный объем памяти, занимаемый индексами, в дальнейшем учитывался при измерении общего объема памяти, занимаемого структурами.

В первом испытании был осуществлен поиск «точечных» данных, т. е. тех данных, для которых были известны все атрибуты, кроме одного. Был проведен 1 млн операций такого поиска со случайными неизвестными атрибутами. Полученные времена поиска были усреднены. На рис. 15 изображены зависимости скорости поиска элементов в различных структурах от коэффициента насыщенности (для d-k-d-дерева).

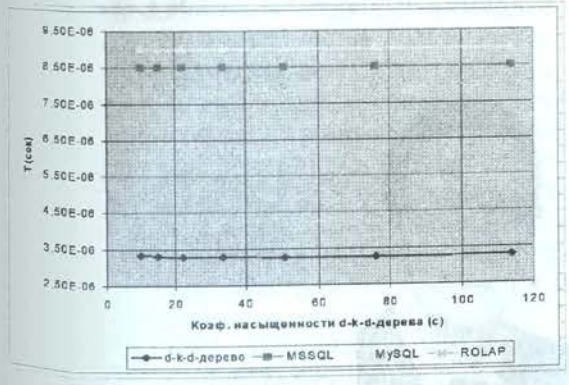

Рис. 15. Зависимости скорости поиска элемента в различных структурах от коэффициента насыщенности (для d-k-d-дерева)

Как можно видеть из данных зависимостей, время поиска элемента в d-k-d-дереве мало зависит от коэффициента насыщенности. Это связано с тем, что основное время в данном случае тратится на обращение к жесткому диску, в то время как поиск данных в оперативной памяти происходит много быстрее. В данном случае эта разница не так заметна благодаря большому объему дискового кэша. Скорость поиска данных в d-k-d-дереве втрое более высока, чем в реляционном хранилище. Это связано с тем, что для поиска в реляционных таблицах по группе вторичных ключей приходится проводить несколько поисков в B+-деревьях индексов. ROLAP-хранилище показывает чуть более высокое быстродействие в связи с тем, что ее структура изначально оптимизирована для проведения подобных запросов.

Следующее испытание проводилось для множеств данных, т. е. при трех неизвестных атрибутах, что позволяет оценить способность структур к выполнению аналитических запросов. В связи с тем, что объем данных, возвращаемых в результате каждого запроса, в данном случае значительно больше, чем при «точечных» запросах, количество запросов в испытании было уменьшено до 10 000. Данные представлены в виде графика на рис. 16.

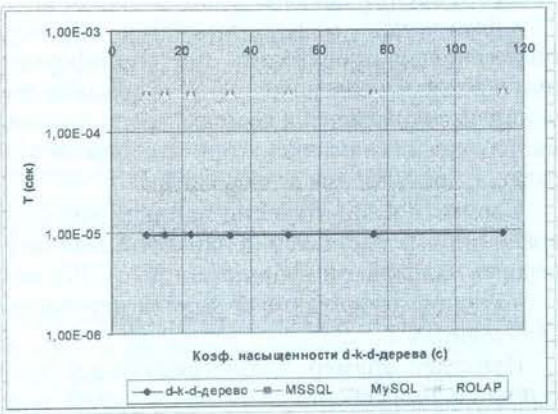

Рис. 16. Зависимости скорости поиска множества элементов в различных структурах от коэффициента насыщенности (для d-k-d-дерева)

Как можно видеть из данного графика, при поиске множества данных d-k-d-деревья быстрее реляционного хранения более чем в 20 раз. Разрыв же с ROLAP-хранением изменился крайне незначительно, так как ROLAP-хранилища подготовлены для поисков больших массивов данных.

Таблица 6

Объем памяти, занимаемый структурами хранения по вторичным ключам

	MSSQL	MySQL	MS ROLAP	d-k-d-дерево
Объем на жестком диске, Гбайт	274,3	310,1	280,2	176,3

Как можно видеть из табл. 6, объем памяти на жестком диске, занимаемый d-k-d-деревом, в 1,5 раза меньше, чем у реляционных реализаций хранилищ. Основной причиной такого преимущества является единая индексная структура поиска по всему объему хранимых данных, в то время как для индексации реляционных данных требуется построение многих индексных структур, которые могут занимать (как в случае MSSQL) до 40% от объема всей БД.

ВЫВОДЫ

В работе предложен подход к построению моделей хранения данных для информационной СРВ, основанный на интегральной их оптимизации. В рамках данного подхода предложены:

- логическая модель представления данных, позволяющая добиться высокой степени устойчивости свойства целостности хранения информации, а также высокой централизации самого хранения данных;
- физическая модель хранения данных, позволяющая производить поиск информации в режиме реального времени при обеспечении высокой степени компактности данных на физических носителях при потоковом характере поступления информации.

Совместное применение данных моделей хранения и представления данных позволяет строить надежные информационные СРВ, использующие расширенный анализ исторических данных.

Приведен пример, иллюстрирующий особенности применения данных моделей для нужд биллинговой системы учета пользовательского трафика.

СПИСОК ЛИТЕРАТУРЫ

1. **Ивлев Д. В., Левков А. А.** Application of bit logical models of information representation for the purpose of organizing effective interaction of a user with multidimensional databases // Тр. Междунар. конф. CSIT-2002. Греция, Патры, 18–24 сент. 2002. С. 49–50.
2. **Левков А. А., Христодуло О. И.** Применение циклических структур для организации эффективного размещения информации на физических носителях в СУБД // Телекоммуникации и информатизация образования. 2003. № 1. С. 43–52.
3. **Павлов С. В.** Системы обработки и хранения информации для контроля и прогнозирования состояния авиакосмических и экологических объектов на основе концепции многомерных баз данных: Дис. ... д-ра техн. наук. Уфа, 1998. 242 с.
4. **Ивлев Д. В.** Многомерная битовая логическая модель представления информации в базах данных: Дис. ... канд. техн. наук. Уфа, 2004. 130 с.
5. **Bayer R., McCreight E. M.** Organization and maintenance of large orderedindices // Acta Informatica. 1972. № 3. С. 173–189.
6. **White M.** N-trees: Large ordered indexes for multi-dimensional space // Techn. Rep.: Application Mathematics Research, Statistical Research Division. US Bureau of the Census, 2002. 134 с.
7. **Архипенков С., Голубев Д., Максименко О.** Хранилища данных. От концепции до внедрения. М.: Диалог-МИФИ, 2002. 126 с.
8. **Сахаров А. А.** Концепции построения и реализации информационных систем, ориентированных на анализ данных // СУБД. 1996. № 4. 56 с.
9. **Ким В.** Три основных недостатка современных хранилищ данных // Открытые системы. 2003. № 2. 82 с.
10. **Галахов И.** Проектирование корпоративной информационно-аналитической системы // Открытые системы. 2003. № 4. 88 с.

ОБ АВТОРАХ

Кабальнов Юрий Степанович, проф., зав. каф. информатики. Дипл. инж. электронной техники (УАИ, 1971). Д-р техн. наук по управлению в технических системах (УГАТУ, 1993). Иссл. в обл. адаптивного и интеллектуального управления.

Левков Александр Александрович, доц. каф. информатики. Дипл. магистр техники и технологий (УГАТУ, 2000). Канд. техн. наук по мат. и прогр. обеспечению (УГАТУ, 2004). Иссл. в обл. организации баз данных и знаний.