услугах, ценах на них, спросе на товары и т. п. для анализа конкретной производственной или рыночной ситуации. Такие преобразования в организации управленческого труда стали возможны благодаря существенным качественным изменениям в его технологии. Автоматизация потоков информации, применение экономико-математических методов обработки данных, внедрение в практику работы баз данных и баз знаний — все это приняло совершенно новые, конкретные способы формирования, подготовки управленческих решений и их реализации. Произошло смещение акцентов и в формулировании критериев эффективности информационных систем управления.

Правильный выбор программного продукта и фирмы-разработчика — это первый и определяющий этап автоматизации предприятия. В настоящее время проблема выбора информационной системы из специфической задачи превращается в стандартную процедуру, которую предлагают различные внедренцы и консалтинговые фирмы.

Естественно, каждое предприятие имеет свою уникальную организационную специфику, и при внедрении могут возникать различные нюансы, которые требуют дополнительного рассмотрения и поиска методов их решения. Для этого существуют профессиональные бизнес-консультанты и консалтинговые фирмы, количество которых увеличивается с каждым годом. На настоящий момент существует достаточно широкий спектр ИС, призванных удовлетворить самые разнообразные нужды, как небольших компаний, так и компаний-гигантов. Очевидно, что рынок информационных технологий и систем в России будет активно развиваться, и программные продукты, представленные на нем уже сегодня, в полной мере охватывают все аспекты деятельности предприятий, от менеджмента, логистики, маркетинга, производства, сбыта до бухгалтерского учета и управления персоналом.

СПИСОК ЛИТЕРАТУРЫ

1. Барановская Т. П. и др. Информационные системы и технологии в экономике.— М.: Финансы и статистика, 2003. — 365 с.

2. Емельянова Н. З. и др. Основы построения автоматизированных информационных систем: Учебное пособие.— М : Форум: Инфра-М, 2005. 425 с.

3. www.cnews.ru — Портал с аналитикой по различным рынкам.

4. Крошилин С. В., Медведева Е. И. Компьютерные технологии в маркетинге и маркетинговых исследованиях: Методические указания для практических занятий студентов экономических специальностей Коломна: КГПИ, 2007.— 36 с.

5. Угрюмова А. А., Медведева Е. И. Экономика и организация торгового предприятия: учебно-методическое пособие / Министерство образования Московской области. Коломенский государственный педагогический институт. — Коломна: КГПИ, 2007. 108 с.

6. Мхитарян С. В. Маркетинговая информационная система. — М.: Эксмо, 2006. — 487 с.

7. Черкашин П. А. Готовы ли Вы к войне за клиента?: Стратегия управления взаимоотношениями с клиентами (CRM). - М.: Интернет-университет информационных технологий ИНТУИТ.ру, 2004. 380 с.

8. Крошилин С. В. Маркетинговые информационные системы // Математическое и программное обеспечение вычислительных систем: Межвуз. сб. науч. тр. Рязань: РГРТА, 2002.- С. 46 52.

9. Крошилин С. В. Принцип построения маркетинговых информационных систем для службы маркетинга предприятия с индивидуальным изготовлением продукции // Новые информационные технологии: Сб. науч. тр. — Рязань: РГРТА, 2001. - С. 100 102.

10. Медведева Е. И., Крошилин С. В. Информационно-аналитическая технология в маркетинговой деятельности // Интеллектуальные технологии в образовании, экономике и управлении: Сборник статей 2-й Международной конференции. Воронеж, 2005. С. 35 38.

Материал поступил в редакцию 21.03.08.

УДК 025.4 026.036:004.6

О. Л. Олейникова, А. Н. Шогин

Использование "обратного" индексирования в ИПС с собственным хранилищем данных

Рассматриваются вопросы ускорения обработки контекстных запросов в информационно-поисковых системах (ИПС) с собственным хранилищем данных. Предложен способ индексирования полей с небольшим разбросом значений и показана эффективность его использования на реальных базах реферативной информации. Приводится метод использования выражений языка SQL внутри собственного языка ИПС, а также реализация обработки комбинированных запросов.

Современные информационно-поисковые системы (ИПС) используют в качестве хранилища данных либо широко распространенные реляционные СУБД (Oracle, MS SQL Server, MySQL, ...), либо собственные механизмы работы с данными. Оба эти подхода имеют очевидные преимущества и недостатки. В первом случае разработчику нет необходимости заботиться о том, как работать с данными, как строить индексы, но поиск информации сводится к построению SQL запроса или PL-SQL программы. К сожалению, языки ИПС по вполне понятным причинам гораздо ближе к естественным

языкам, чем к SQL и, по этой причине, трансляция "язык ИПС — SQL" приводит к очень сложным и, как следствие, времязатратным запросам. Во втором случае разработчик сам определяет наиболее подходящие для обработки запросов структуры хранения данных, что влечет резкое усложнение самой ИПС, но позволяет оптимизировать время выполнения запросов, не оглядываясь на какие-либо языки.

ИПС "Сокол", созданная в ВИНИТИ РАН и использующаяся как основная при онлайновом доступе к банку данных и при распространении баз данных на CD ROM, относится ко второму типу ИПС в указанном выше смысле. Структуры данных ИПС "Сокол" подробно описаны в [1–5]. Из этих публикаций следует, что ИПС содержит как статические, так и динамические инверсные структуры, которые не являются индексами в классическом смысле. Однако именно эти структуры обеспечивают в большинстве случаев очень быструю обработку контекстных запросов на языке, близком к языку классических систем типа Dialog (нынешний владелец ICI Thompson).

Сутью обработки запросов являются контекстные булевы операции над хит-листами, которые представляют собой первичные (терминальные) элементы запроса. Более точно, любому термину (в том числе и с символами усечения) соответствует линейный массив элементов, каждый из которых описывает точное положение конкретного слова в базе данных. Эти элементы состоят из идентификатора записи, идентификатора поля и (относительного) положения слова в поле. В реальности ситуация конечно сложнее, поскольку необходимо определить понятия идентификаторов записей и полей, определить, что такое "положение" слова в поле и, наконец, минимизировать размер элемента.

Тем не менее, любые булевы контекстные операции над хит-листами приводят к хит-листам и дают, в конечном счете, список релевантных документов. Однако операции объединения хит-листов требуют обязательной предварительной их сортировки. Это и является узким местом всей обработки запроса с точки зрения затрат времени.

Несмотря на то, что структура хит-листа достаточно жестко определена, экспериментальная проверка всех известных в настоящее время алгоритмов сортировки [6] не позволила найти абсолютно оптимального. Более того, даже разработка оригинального, специально оптимизированного алгоритма [5], давала результаты примерно на 10–20% лучше, чем стандартные. Конечно, неприемлемые результаты (более 1–2 минут) получаются лишь при обработке наиболее "одиозных" терминов, таких как 1$ (все слова, начинающиеся с символа единицы), но, тем не менее, уже запросы вида TEST AND 1$ IN PY сильно тормозят их выполнение. Но такие запросы уже вполне реальны!

Существует еще одна ситуация, когда использование хит-листов практически невозможно, — это статистический анализ баз данных. Вообще говоря, такой анализ в случае СУБД, не основанных на языке SQL, является предметом отдельного рассмотрения. Здесь отметим лишь, что статистический анализ требует привлечения фактографических операторов, которые принципиально нереализуемы для инверсных структур.

Таким образом, можно сделать вывод о том, что оптимизация выполнения некоторых контекстных запросов и трансляция запросов статистического анализа баз данных (по сути SQL запросов) упираются в отсутствие некоторых дополнительных структур хранения ИПС.

Естественно, что в первую очередь была рассмотрена возможность использования классических индексов в виде B-деревьев или хэш-функций. Как было показано выше, первичной (терминальной) единицей при обработке запроса является хит-лист. Это означает, что в узле дерева или значением хэш-функции должен быть элемент хит-листа в указанном выше смысле. Понятно, что а) с учетом накладных расходов такая структура будет занимать гораздо больше места, чем инверсная структура, и б) результатом поиска по условию будет в некоторых случаях огромный хит-лист. В любом случае, время выборки результирующего хит-листа будет намного больше, чем при выборке его же, но из инверсной структуры в силу простоты последней. Очевидным плюсом такого решения является априорная упорядоченность хит-листа в силу, например, структуры B-дерева.

Есть еще одно, менее очевидное, "но" против использования классических индексов для обработки контекстных запросов: они устанавливают соответствие "ключ" — "значение", где "ключом" является некоторая функция от записи, а "значением" — номер или идентификатор записи. Любые другие способы использования индексов требуют более или менее значительной модификации стандартных алгоритмов индексации, причем зачастую с непредсказуемыми последствиями. Но использование стандартной индексации не дает хит-листа, что означает невозможность ее использования в качестве терминального элемента запроса, т. е. выражение типа:

TERM op IDXEXPR,

где **TERM** — обычный поисковый термин или поисковое выражение, **op** — контекстная операция, а **IDXEXPR** — выражение с индексированными терминами, не может рассматриваться как корректное выражение поискового запроса к ИПС.

С другой стороны, в языке запросов к ИПС есть вполне аналогичное выражение, определяющее ограничение действия хит-листа на список полей базы данных:

TERM IN FIELDLIST,

где **TERM** — поисковый термин или поисковое выражение, **IN** — операция ограничения, а **FIELDLIST** — список полей, на которые ограничивается хит-лист термина или выражения **TERM**.

Аналогия здесь вполне уместна, поскольку операция имеет смысл только в случае **AND**. Другие контекстные операторы этой же группы (**SAME**, **ADJ**) отпадают, поскольку привязаны к полям или даже позициям в полях, а оператор **OR** не имеет семантического смысла (если, конечно, не использовать в качестве индексов заведомо неразумные функции типа "весь реферат" или "все авторы").

Таким образом, мы приходим к тому, что наиболее разумной синтаксической конструкцией использования индексов является конструкция вида:

TERM WHERE (LIMITEXPR),

где **TERM** имеет тот же смысл, что и выше, **(LIMITEXPR)** — некое выражение с использованием индексов, а **WHERE** — новое ключевое слово для облегчения восприятия синтаксиса и упрощения анализа запроса. Обязательность скобок в данном случае диктуется тем, что семантический смысл многих операторов в самом **LIMITEXPR** сильно отличается от того, что используется в поисковом запросе и анализ запроса, например такого вида:

TERM WHERE PY<2000 AND TERM1,

не дает однозначного результата. это может быть и

TERM WHERE (PY<2000 AND TERM1)

с одним смыслом, и:

TERM WHERE (PY<2000) AND TERM1

с совершенно другим смыслом, т. е. **TERM1** в первом случае используется как индекс, а во втором случае — как обычный термин.

Итак, поскольку теперь уже установлено синтаксическое использование индексов, рассмотрим, какие структуры возможно использовать для вычисления оператора **WHERE**.

Мы имеем обычный бинарный оператор, в левой части которого находится терминальная структура типа "хит-лист", а в правой части должен находиться "ограничитель" — аналог списка полей. Последнее означает, что для каждого элемента хит-листа должно быть вычислено, является ли он "истинным" или "ложным". Почему именно в такой последовательности? Дело в том, что "разумные" хит-листы существенно меньше (на порядки), чем объем базы данных. В противном случае такие поля, для которых хит-листы сравнимы с размерами базы, имеет смысл именно индексировать, а не помещать в инверсные списки. Примеры очевидны — год публикации, язык, страна и т. п. Следовательно, разумно именно "ограничивать" хит-лист, а не пересекать его с индексом.

Таким образом, возникает понимание следующего утверждения: быстро вычислить значение некой (вообще говоря, SQL) функции для конкретной записи базы данных. Строго говоря, это утверждение никак не связано с классическим пониманием индекса — там ситуация прямо противоположная — вычисление совокупности идентификаторов записей по ключу.

Итак, сформулируем задачу построения необходимой структуры данных следующим образом: за минимальное время вычислить некоторое выражение с одним параметром — идентификатором записи, т. е. фактически дискретную функцию $F(n)$, где n — номер записи. Единственно разумным способом такого вычисления является предварительное построение образа данной функции. Поскольку значением функции является псевдослучайная величина, то наиболее подходящим представлением для нее является таблица, т. е. массив пар "аргумент — значение". При этом "аргумент" не обязательно должен быть идентификатором записи, а "значение" не обязательно должно быть значением функции.

Существует возможность минимизации размеров данной таблицы, если функция является ступенчатой с малым количеством перепадов, как на рис. 1.

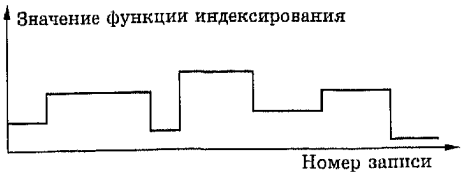

Рис. 1. Таблица массива пар "аргумент — значение"

В этом случае в качестве аргумента функции можно выбрать точки перепадов и в качестве значений — величины перепадов. Однако анализ реальных функций, для которых, казалось бы, количество перепадов должно быть относительно невелико, показал, что это не так. Например, для политематической реферативной базы за пять лет с общим количеством записей 3 396 550 количество перепадов составило:

а) индексация по году публикации — 1 399 280,

б) индексация по виду документа — 832 098;

в) индексация по стране публикации — 1 818 159;

г) индексация по языку публикации — 1 221 292.

Таким образом, использование в качестве аргумента функции точек перепада в реальных условиях неоправданно, поскольку объем таблицы значений снижается всего в 2–3 раза, но при этом резко возрастают затраты на вычисление функции.

Другим аспектом, ограничивающим использование предварительно рассчитанных дискретных функций, является количество их значений. Если это количество сравнимо с количеством записей в базе данных, то очевидно, что использование инверсных структур намного более эффективно, поскольку размер хит-листа в этом случае минимален и не превышает нескольких тысяч и, следовательно, поддается очень быстрой сортировке. А именно сортировка, как было показано выше, является узким местом при обработке запроса.

Итак, предположим, что количество значений функции индексирования ограничено и относительно невелико. Попробуем вычислить размер этого ограничения. Поскольку очевидно, что в таблице значений функции индексирования реально должны использоваться не сами значения, а ссылки на них, т. е. номера позиций в таблице значений, то разумно предположить, что размер такой ссылки равен либо одному, либо двум, либо четырем байтам (т. е. байт, короткое целое и целое числа). Максимальное значение целого числа равно $2^{16} = 4\,294\,967\,296$, т. е. для баз данных ВИНИТИ РАН это число сравнимо с размерами многолетних политематических баз данных, что противоречит утверждению о несравнимости количества значений функции индексирования и размера базы. С другой стороны, один байт может представить всего 256 значений функции, что очевидно недостаточно. Таким образом, приходим к тому, что размер таблицы значений (количество пар "аргумент — значение") должен быть равен 65 536 или, для простоты алгоритмов, 32 768

Соответствующий алгоритм вычисления значения функции для конкретной записи выглядит следующим образом (рис. 2):

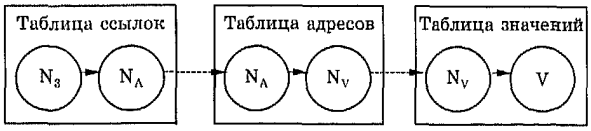

Рис. 2. Алгоритм вычисления значения функции для конкретной записи

Здесь вычисление позиции в таблице ссылок происходит просто сдвигом на ($N_3 \times 2$) байт относительно ее начала (N_3 — номер записи), а значение в этой таблице указывает на позицию в таблице адресов реальных значений функции.

Открытым теперь остается лишь вопрос о создании такого "обратного" индекса, точнее, о создании конечной таблицы значений функции индексирования. Очевидно, что при просмотре нескольких миллионов записей нельзя использовать методы динамического наполнения такой таблицы с последующим поиском новых значений в ней. С другой стороны, соответствующие методы давно [7] используются в практике построения компиляторов — это использование хэш-функций. Поскольку размер таблицы априори задан и является степенью двойки, то в качестве хэш-функции использована следующая (c_i – символы исходной строки): $h = (h\hat{}(int)(c_i c_{i+1} c_{i+2} c_{i+3})) << 1$ для всех i, для которых $c_i c_{i+1} c_{i+2} c_{i+3}$ отличны от 0, а в качестве рехэширующего алгоритма – метод Морриса [8].

Для минимизации размера получившейся таблицы значений непосредственно после ее генерации производится сжатие до реального количества значений и замена ссылок в первой таблице (см. рис. 2).

После некоторой оптимизации описанного алгоритма время генерации индекса по политематической пятилетней базе составило около 20 минут на платформе Sun V480.

Объем одного индексного файла составил всего 6,7 Мбайт при общем объеме базы около 6,4 Гбайт, т. е. примерно 0,1%. Это означает, что с точки зрения объемов используемой дисковой памяти введение индексов не составляет сколько-нибудь значимого ее увеличения.

Далее была произведена проверка времени выполнения запросов по классической схеме и с привлечением индексов. В частности, запрос
((анализ$ or оценк$) and (эффективн$ or полнот$ or точност$) and (поиск$ or ИПС or БД)) and (01 in A35 AND (РУС OR АНГЛ) in F08),
использующий только инверсные списки, выполнялся около 30 секунд, из которых обработка выражения (01 in A35 AND (РУС OR АНГЛ) in F08) заняла 27 секунд, а запрос
((анализ$ or оценк$) and (эффективн$ or полнот$ or точност$) and (поиск$ or ИПС or БД)) where (A35=1 AND (F08='РУС' OR F08='АНГЛ')),
использующий комбинацию инверсных списков и "обратных" индексов, выполнялся всего 4 секунды, из которых менее 1 секунды заняла обработка выражения (A35=1 AND (F08='РУС' OR F08='АНГЛ')).

Таким образом, можно сделать вывод о том, что использование "обратных" индексов позволяет значительно ускорить выполнение запросов для больших баз данных.

СПИСОК ЛИТЕРАТУРЫ

1. Леонтьева Т. М., Шогин А. Н. Разработка элементов распределенной схемы хранения и представления в режиме он-лайн гипермедиа информации // НТИ-97: 3-я Международная конференция "Информационные ресурсы. Интеграция. Технология". Тез. докладов.— М.: ВИНИТИ, 26 28 ноября 1997.— С. 145-146.

2. Арский Ю. М., Леонтьева Т. М., Шогин А. Н. Создание инфраструктуры многоуровневой интеграции разнородных данных // НТИ. Сер. 2. - 1997.— № 2.— С. 18-20.

3. Арский Ю. М., Леонтьева Т. М., Шогин А. Н. Современные интернет — интранет технологии в распределенной системе серверов баз данных ВИНИТИ // НТИ-99: 4-я Международная конференция "Интеграция. Информационные технологии. Телекоммуникации". Тез. докладов. — М.: ВИНИТИ, 17 19 марта 1999.— С. 133 134.

4. Леонтьева Т. М., Шогин А. Н. Банк данных ВИНИТИ сегодня и завтра // НТИ-2000: 5-я Международная конференция "Информационное общество. Информационные ресурсы и технологии. Телекоммуникации" Тез. докладов. — М.: ВИНИТИ, 22 24 ноября 2000. – С. 208-210.

5. Арский Ю. М., Леонтьева Т. М., Никольская И. Ю., Шогин А. Н. Банк данных ВИНИТИ: Состояние и перспективы развития. М.: ВИНИТИ, 2006. - 242 с.

6. Кнут Д. Искусство программирования на ЭВМ. - М.: Мир, 1975.

7. Грис Д. Конструирование компиляторов для цифровых вычислительных машин. М.: Мир, 1975.

8. Morris R. Scatter storage techniques, CACM, 11 (Jan. 1968).- P. 38-44.

Материал поступил в редакцию 17.04.08.