Модел. и анализ информ. систем. Т. **16**, № **4** (2009) 22–33

УДК 519.677

Новые алгоритмы лексической оптимизации запросов

Кузнецов С. Д., Мендкович Н. А.

Институт Системного Программирования РАН

e-mail: mend@f-group.ru

получена 22 сентября 2009

Ключевые слова: оптимизация запросов, лексическая оптимизация.

Предлагаются новые алгоритмы модификации запроса. Эти алгоритмы включают лексическую оптимизацию, основанную на математических преобразованиях, которые ранее не использовались для задач, связанных с преобразованием запросов.

1. Введение

Важным направлением исследований в области систем управления базами данных (СУБД) является повышение эффективности выполнения запросов путем их оптимизации, минимизации времени отклика и/или потребляемых технических ресурсов при обработке запросов к базе данных. Для дальнейшего изложения следует условиться о значении используемых терминов. Будем называть *запросом* любое языковое выражение, которое описывает совокупность данных в базе данных (БД), подлежащих выборке или обновлению. В запросе мы выделяем *ограничение* — часть запроса, содержащую все множество условий, описывающих выборку, выдачей которой должно завершаться его исполнение. Условие — часть ограничения запроса, описываемая с помощью множества атрибутов запроса и констант и содержащая не более чем одну операцию сравнения. Алгоритм оптимизации, обрабатывающий тот или иной запрос, должен определить множество путей или планов его осуществления и выбрать из их числа множество, в наибольшей мере отвечающее выбранным критериям оптимальности. Подобный алгоритм неизбежно решает несколько подзадач и может быть разделен на ряд этапов. В современной литературе, посвященной проблемам оптимизации [1-2], принято выделять четыре таких этапа или класса операций оптимизационного алгоритма:

1. преобразование запроса в стандартную внутреннюю форму;
2. модификация запроса, т.е. приведение его к форме, при которой выполнение наиболее эффективно;

3. анализ запроса, выбор потенциальных низкоуровневых процедур, используемых при его выполнении;

4. генерация и сравнительный анализ различных планов доступа, возможных способов выполнения запроса.

На практике четвертый класс операций воспринимается как основа оптимизационного алгоритма, а прочие операции — как подчиненные, направленные на подготовку генерации планов доступа, в то время как на четвертом этапе оптимизатор использует эти планы, оценивает и выбирает лучший из них для выполнения запроса [3]. Однако, несмотря на подчиненную роль, которую играет модификация запросов, рассмотрение новых подходов к решению данной задачи представляется вполне актуальным, так как оптимизатор по-прежнему нуждается в генерации оптимальных, в рамках того или иного критерия, планов доступа. Среди множества видов алгоритмов модификации запросов мы выделим так называемые алгоритмы лексической оптимизации или перезаписи запроса, на основе анализа самого текста запроса как лексической конструкции, по итогам которой запрос приводится к иному виду. Будем считать, что *цель данной оптимизации* — ликвидация избыточности и приведение запроса к наиболее лаконичному виду, хотя к решению проблемы лексической оптимизации существуют и иные подходы, которые будут описаны ниже. В представленных ниже примерах мы показываем, что в ряде случаев выбранный нами подход приводит к упрощению обработки запроса, что должно способствовать ускорению его реализации.

В разделе 2 обсуждаются существующие алгоритмы, практика решения задачи модернизации запроса и соответствие алгоритмов сформулированному критерию оптимальности. В разд. 3 рассматривается внутреннее представление запроса. В разд. 4 предлагается оптимизация на основе «алгоритма поглощения» алгебраического алгоритма Квайна, позволяющая сократить наиболее сложные логические выражения. В заключении анализируется место предложенных алгоритмов в существующей практике оптимизации.

2. Приемы оптимизации запросов и ее практика в современных приложениях

С момента формулирования проблемы оптимизации запросов в этом направлении была проведена значительная работа, и сколь-либо полный обзор публикаций по теме потребовал бы значительного объема текста, его проведение не является задачей данной статьи. К числу известных авторам подробных обзоров по данной теме в настоящий момент относятся [1-2], [4-5].

Теоретическую основу лексической оптимизации, алгоритмам которой посвящена настоящая статья, составили одна из работ П. Холла [6], в которой рассматривается применение методик реляционной алгебры по преобразованию логических выражений в целях оптимизации запросов. Большинство ранее опубликованных работ не были посвящены анализу методов лексической оптимизации, однако использовали подобные методы применительно к определенному кругу задач или как

составную часть более сложного метода оптимизации. Это касается, в частности, алгоритмов оптимизации «вложенных подзапросов». Так, ряд приемов из работы [7] может быть отнесен к лексическим, а предложенное позднее решение этой проблемы также включало лексическое преобразование запроса в эквивалентный запрос без вложений [8].

Большинство алгоритмов лексической оптимизации ориентировано на сокращение запросов путем удаления избыточных условий, обработка которых СУБД не влияет на получаемый в итоге набор данных. Например, в [3] рассмотрен ряд приемов лексической оптимизации запросов, применяемых в СУБД Oracle 10g и сводящихся к поиску и удалению семантически избыточных условий, в частности т.н. алгоритм поиска «частых подвыражений», но оценка их места в работе оптимизатора и подробное описание их реализации отсутствуют. Известно существование и других подходов к оптимизации, основанных на перезаписи запроса, исключающих генерацию нерелевантных кортежей путем вычисления дополнительных «магических» таблиц, которые действуют как фильтры [9-10]. Данный вид алгоритмов, известных как «алгоритмы магических множеств», относится к числу сложных для анализа и понимания. В частности [4] характеризует его как слишком «интуитивный» и «таинственный». Судя по всему, в настоящее время способам генерации планов на основе лексического подхода к оптимизации уделяется мало внимания. Так, оптимизатор СУБД компании Oracle, согласно доступным нам описаниям [3], ориентирован в первую очередь на сравнительный «оценочный» анализ существующих путей доступа. А во многих обзорах, посвященных оптимизатору данной СУБД, методы лексической оптимизации не упоминаются вовсе [11]. В оптимизаторах Starburst и DB2 основное внимание уделяется семантической оптимизации запроса и усовершенствованию системы выбора планов их выполнения [12].

Не ясно, чем обусловлен явно недостаточный интерес к этому виду методов модернизации запросов. Большинство примеров модернизации, которые можно отнести к лексическим и синтаксическим методам трансформации, описанных в процитированных выше релизах и известных авторам опубликованных работах, практически никогда не основываются на сложных математических алгоритмах. Более того, использование таковых в рамках решения задач оптимизации практически не обсуждалось.

Представляется небезынтересным предпринять попытку привлечь существующие в рамках линейной и булевой алгебр методики преобразования выражений, описанные ниже, для генерации возможных планов доступа при решении задачи оптимизации запросов и попытаться оценить границы их применимости и степень полезности.

В предлагаемом алгоритме оптимизации выделяются три этапа:

- стандартизация условий, позволяющая организовать автоматизированный процесс их сравнения и обработки;
- оптимизация с помощью алгоритма «поглощения» избыточных условий;
- оптимизация набора условий с использованием алгоритмов минимизации булевой алгебры.

Краткое теоретическое описание алгоритмов уже было представлено авторами

в докладе [13]. В настоящей статье обсуждаются способы реализации этих алгоритмов.

3. Стандартизация и логическая минимизация

Учитывая, что в литературе отсутствует описание общепринятого алгоритма стандартизации условий, вкратце опишем тот алгоритм, который используется в описываемом в этой статье подходе. Заметим, что здесь рассматриваются только условия, заданные в виде $arith-expr1\ comp-op\ arith-expr2$, где $arith-expr1$ и $arith-expr2$ — это выражения, построенные на именах столбцов и константах, а $comp-op$ — операция сравнения.

Для этого необходимо выполнение следующих действий:

- открытие всех скобок, что снижает лаконичность каждого из условий на начальном этапе стандартизации, но значительно упрощает разработку алгоритма стандартизации, так как при использования скобок абсолютно тождественные по смыслу выражения могут быть представлены множеством совершенно различных способов, что значительно затрудняет их сравнение и анализ;
- лексикографическое упорядочение атрибутов и констант в левой части условий, чтобы программа могла однозначно идентифицировать совпадение или несовпадение написания условий;
- объединение, суммирование или вычисление произведения, одинаковых элементов. Например, произведение $a * x * x$ примет вид $a * x^2$;
- суммирование констант в правой части;
- приведение всех операций сравнения к одному виду; будем считать, что есть лишь три разновидности сравнений: «больше» ($>$), «меньше» ($<$) и «равно» ($=$); в случае, например, если встречается ограничение «меньше или равно», то оно преобразуется в логическую сумму ограничений «$<$» и «$=$»;
- приведение ограничения к ДНФ. Выбор именно дизъюнктивной формы связан с тем, что большинство существующих математических алгоритмов минимизации ориентировано на сокращение функций, представленных в ДНФ[1].

Здесь мы не будем подробно останавливаться на реализации описанных действий. Полученное в итоге выражение представим в виде суммы множества произведений или единичных элементов, констант или атрибутов, выступающих в этом случае в роли переменных выражения. Все произведения и элементы, не содержащие имен атрибутов, концентрируются в правой части условия (т.е. в правой части оказывается константное выражение), прочие — в левой (разумеется, с необходимой заменой знаков на противоположенные). Константное выражение вычисляется, и получается условие вида $arith - expr\ comp - op\ const$, где $const$ — это некоторое значение. В результате этих преобразований мы придем к однозначному способу записи всех условий, что поможет их анализу.

4. Алгоритмы перезаписи запроса

1. Алгоритм поглощения

Одной из «ключевых технологий трансформации» запросов в оптимизаторе СУБД Oracle является алгоритм «исключения общих подвыражений», сводящийся к достаточно тривиальному поиску избыточных условий, удаление одного из которых не изменяет смысла ограничения, но делает его более лаконичным [3].

В этом разделе описывается алгоритм, позволяющий находить и сокращать более широкий набор условий, являющихся избыточными, но не совпадающих текстуально даже после стандартизации выражения. Для этого требуется перейти от уровня формального синтаксиса выражений до их семантики, сравнивая их части между собой. Избыточность может иметь следующие виды:

- один из конъюнктов ограничения, записанного в ДНФ, избыточен по отношению к другому, так как все значения, описанные первым, входят в число значений, описанных вторым;
- условия группы конъюнктов описывают все множество значений одного из атрибутов, причем прочие условия, входящие в конъюнкты, совпадают.

Для обнаружения описанных избыточностей производится поиск условий с общей левой частью. Если такие условия обнаруживаются, и в них используется одна и та же операция сравнения, производится удаление того условия, которое обладает более «узкими границами». Рассмотрим, например, запрос, целью которого является получение идентификационных номеров всех сотрудников, зарплата которых превышает 500, сумма зарплаты и бонуса менее 1000, а значение идентификационного номера больше 0, или тех, чья зарплата более 400.

Пример 1.

```
SELECT ID
FROM EMP
WHERE (ID>0 AND Salary>500 AND Salary+Bonus<1000) OR (Salary>400);
```

В данном случае первые три условия лишены смысла, поскольку множество строк таблицы «Пользователи», выбранное в соответствии с ними, будет отвечать и условию «`Salary>400`» (так как если жалование больше 500, то оно, естественно, больше чем 400). Следовательно, запросу 1 чисто логически соответствует более короткая запись из примера 2.

Пример 2.

```
SELECT ID_пользователя
FROM Пользователи
WHERE Salary>400;
```

Теперь сравним левые части ограничений. Нашей целью является выявление ограничений с одинаковыми левой частью и операцией сравнения. Т.е., если в пределах одного конъюнкта имеются условия $A > B$ и $A > C$:

$$A > B \texttt{ AND } A > C = A > C, \quad \text{где} \quad B > C \tag{1}$$

Назовем этот случай поглощением условия $A > B$ условием $A > C$. Если же в некотором конъюнкте имеется множество условий, поглощающих часть условий другого конъюнкта, то второй конъюнкт подлежит удалению, как семантически избыточный по отношению к первому. Существует также ряд других ситуаций, предусмотренных в правилах булевой алгебры, которые позволяют сделать запись ограничения более лаконичной.

Представим вариант алгоритма, осуществляющего поиск подобных повторов. В целях формализации процесса поиска опишем ограничение с помощью табл. 1.

Таблица 1. Внутреннее представление ограничений запроса из примера 1

	ID	Salary	Bonus+Salary
1	$>$	$>$	$<$
2	0	$>$	0

Здесь каждой строке соответствует конъюнкт ограничения, каждому столбцу — левая часть одного из условий, содержащихся в ограничении. Каждый элемент таблицы содержит три бита: признак вхождения в конъюнкт (ПВК), соответствующий данной строке таблицы, и код условия (два бита, КУ; для определенности будем считать, что КУ равно 0, если ПВК равно 0; значение КУ, равное 1, соответствует операции «=», 2 — «>», 3 — «<»). Легко увидеть, что приведение к данному виду любого ограничения, находящегося в ДНФ, легко осуществимо на основе любого его «разумного» внутреннего представления, выбранного в конкретной системе.

Для обнаружения избыточности первого вида организуется двойной цикл по строкам таблицы, и выполняется проверка потенциальной избыточности одной из строк очередной пары. На первом шаге этой проверки производится логическое побитовое умножение строк очередной пары с участием маски, выделяющей в каждой строке биты ПВК (обозначим соответствующую константную строку через МПВ). Если в результате получается строка из всех нулей, то это означает, что в соответствующих конъюнктах нет ни одной пары условий с одинаковой левой частью, и алгоритм переходит к проверке следующей пары строк.

Иначе на следующем шаге проверяется, содержит ли какая-либо строка из очередной пары строк все ненулевые значения, присутствующие в другой строке той же пары, т.е. включает ли второй конъюнкт только такие условия, что все условия с той же левой частью входят в первый. Если обозначить строки проверяемой пары через a и b, то для этой проверки нужно выполнить две поразрядных логических операции: `NOT` a `AND` b `AND` МПВ и `NOT` b `AND` a `AND` МПВ. Если обе операции производят результат, состоящий не из всех нулей, то проверка не удается, и алгоритм переходит к обработке следующей пары строк. Если нулевой результат выдает операция `NOT` a `AND` b `AND` МПВ, то потенциально избыточным является конъюнкт, соответствующий b, если `NOT` b `AND` a `AND` МПВ, — то a. Пусть для определенности имеет место первый случай, т.е. пусть конъюнкт, соответствующий строке b, включает все условия с той же левой частью, которую имеют те, что входят в конъюнкт, соответствующий строке a. Причем конъюнкт, соответствующий строке b, не должен включать другие условия. Далее проверяется, поглощает ли каждое условие второго конъюнкта соответствующее условие первого конъюнкта. Эту проверку проще всего производить, используя систему правил, представленную в таблице 2, где первая

строка соответствует поглощающему условию, первый столбец - поглощаемому, а в каждом из полей представлено необходимое и достаточное условие для указанного поглощения. Прочерк означает невозможность поглощения в данном случае.

Таблица 2. Правила поглощения условий.

	$y == const2$	$y > const2$	$y < const2$
$x == const1$	$const2 == const1$	$const1 > const2$	$const2 > const1$
$x > const1$	–	$const1 > const2$	–
$x < const1$	–	–	$const2 < const1$

Если проверка поглощения не удается хотя бы для одного условия второго конъюнкта, алгоритм переходит к обработке следующей пары строк таблицы. Иначе строка b таблицы полностью обнуляется, и алгоритм переходит к обработке следующей пары строк. После завершения обработки последней пары строк алгоритм модифицирует исходное внутреннее представление на основе строк таблицы, оставшихся ненулевыми. В случае примера 1 алгоритм работает достаточно просто. Логическое произведение двух строк равно 010, что означает, что два рассматриваемых конъюнкта содержат лишь по одному условию с общей левой частью. Потенциально избыточным является второй конъюнкт. Условия с общей левой частью имеют вид `Salary>500` и `Salary>400`. Поскольку $500 > 400$, второе условие поглощает первое, и первый конъюнкт является избыточным. В итоге запрос принимает вид, показанный в примере 2.

2. Алгоритм минимизации

С целью выявления второго вида избыточности и ее устранения будем использовать алгоритмы минимизации булевой алгебры. При этом будем считать, что ограничение представлено так же, как и в случае, рассмотренном ранее. Имеются две разновидности сокращения функции алгебры логики: приведение функции к кратчайшей форме (минимизация числа элементарных конъюнкций) и ее приведение к минимальной форме (минимизация общего числа символов) [14]. Алгоритм Квайна [15-16], вариант которого описывается ниже, ориентирован на минимизацию первого вида, так как сокращает функцию, прежде всего, за счет удаления избыточных конъюнктов. Возможность использования данного алгоритма при оптимизации запросов ранее рассматривалась в литературе [17], однако описания примеров его реализации в данных целях нам обнаружить не удалось.

Для выполнения алгоритма Квайна требуется выделить в ограничении группу конъюнктов, содержащих одинаковое количество условий (например, N). Затем проводится попарное сравнение конъюнктов и «склейка» тех пар, которые имеют вид A `AND` X и B `AND` X, где X — некий конъюнкт условий размерностью $n-1$, для которых A `AND` B = `Истина`. Конъюнкты X именуются *первичными импликантами* и записываются отдельно. Затем для всех выбранных X повторяется процедура попарного анализа и «склейки» с целью получения конъюнктов размерностью $n-2$, и так до тех пор, пока удается обнаруживать пары конъюнктов описанного вида.

В алгоритме Квайна целью дальнейшего анализа является определение минимального множества импликант, соответствующего максимальному множеству конъюнктов, с целью их замены в конечном представлении. Этот алгоритм основывался

на логическом тождестве:

$$A \texttt{ AND } X \texttt{ OR } B \texttt{ AND } X = X \tag{2}$$

если A OR $B =$ Истина. В случае, если A OR $B =$ Истина, верно также

$$A \texttt{ OR } (B \texttt{ AND } X) = A \texttt{ OR } (A \texttt{ AND } X) \texttt{ OR } (B \texttt{ AND } X) = A \texttt{ OR } X. \tag{3}$$

Возможность сокращения запроса с помощью равенства (3) осуществляется в рамках проверки избыточности «алгоритмом поглощения» после проверки потенциально избыточных конъюнктов на соответствие условиям таблицы 2. Если после этого ни один из сравниваемых конъюнктов не был удален, то проводится проверка на соответствие равенству (3) групп условий с одинаковой левой частью из обоих конъюнктов. Проверка производится в рамках описанного выше «алгоритма поглощения» в случае обнаружения потенциально избыточных строк, не отвечающих ни одному из правил таблицы 2. Если соотношение условий с одинаковой левой частью соответствует равенству (3), то условие B удаляется из потенциально избыточного конъюнкта.

Для реализации алгоритма Квайна образуется таблица (см. ниже табл. 4), где строкам соответствуют первичные импликанты всех размерностей, а столбцам — конъюнкты исходного представления ограничения. Элемент таблицы имеет значение 0, если первичная импликанта данной строки не входит в конъюнкт, соответствующий столбцу, и 1 — если входит.

Для определения искомого набора импликант производятся шесть этапов обработки этой таблицы:

1. Из дальнейшего анализа исключаются столбцы, содержащие только нули. Они в любом случае входят в конечное представление, поскольку не покрываются найденными импликантами.
2. Производится поиск существенных импликант, т.е. строк, одно из единичных значений которых является единственным в столбце. Обнаруженные существенные импликанты и соответствующие им конъюнкты исключаются из дальнейшего анализа.
3. В видоизмененной таблице выделяются одинаковые столбцы, содержащие единицы в одних и тех же строках. Все столбцы-«дубликаты», кроме одного, исключаются из таблицы.
4. Исключаются все строки, не содержащие единичных значений (их образование возможно после завершения предыдущего этапа).
5. Из числа оставшихся строк выбирается набор импликант, включающий единицы во всех столбцах. Если имеется несколько вариантов, то выбирается тот, который содержит минимальное число условий.
6. Формируется итоговое представление ограничения, состоящие из конъюнктов, выбранных на этапе 1, и импликант, выбранных на этапах 2 и 5.

Рассмотрим работу алгоритма Квайна на следующем примере. Пусть требуется отобрать авторов, имеющих различные соотношения чисел опубликованных статей, книг и цитирований их работ другими авторами.

Пример 3.

```
...WHERE Articles>25 AND Books>2 AND Citations=32
         OR NOT (Articles>25) AND Books>2 AND Citations=32
         OR NOT (Articles>25) AND Books>2 AND NOT(Citations=32)
         OR Articles>25 AND NOT(Books>2) AND NOT(Citations=32)
```

Стандартное представление этого ограничения будет иметь следующий вид:

```
...WHERE Articles>25 AND Books>2 AND Citations=32
         OR Articles=25 AND Books>2 AND Citations=32
         OR Articles<25 AND Books>2 AND Citations=32
         OR Articles<25 AND Books>2 AND Citations>32
         OR Articles<25 AND Books>2 AND Citations<32
         OR Articles=25 AND Books>2 AND Citations>32
         OR Articles=25 AND Books>2 AND Citations<32
         OR Articles>25 AND Books=2 AND Citations<32
         OR Articles>25 AND Books=2 AND Citations>32
         OR Articles>25 AND Books<2 AND Citations<32
         OR Articles>25 AND Books<2 AND Citations>32
```

Внутреннее представление ограничения представлено на табл. 3. Выполнение представленного выше алгоритма обнаружения избыточности первого вида в данном случае ничего не дает. При выполнении данного алгоритма создается табл. 4. Исходные конъюнкты в ней представлены номерами.

Таблица 3. Внутреннее представление запроса из примера 3

	Articles	Books	Citations
1	$>$	$>$	$=$
2	$=$	$>$	$=$
3	$<$	$>$	$=$
4	$<$	$>$	$>$
5	$<$	$>$	$<$
6	$=$	$>$	$>$
7	$=$	$>$	$<$
8	$>$	$=$	$>$
9	$>$	$=$	$<$
10	$>$	$<$	$>$
11	$>$	$<$	$<$

Для каждого конъюнкта производится поиск потенциально избыточных строк, где все условия, кроме одного, полностью совпадают, а у оставшегося условия во всех таких случаях совпадает правая часть. Столбец, соответствующий этому условию, назовем *имплицирующим*. Поиск производится путем последовательного сравнения строк. Если для рассматриваемой строки ни одного потенциально избыточного конъюнкта не найдено, то мы переходим к обработке следующей строки. Если же

в результате поиска обнаружена группа потенциально избыточных строк, то производится логическое сложение условий из имплицирующего столбца. Если логическая сумма условий имплицирующего столбца тождественно истинна, то конъюнкция всех условий первой строки, кроме того, что соответствует имплицирующему столбцу, образуют первичную импликанту. В табл. 4 заносится строка, соответствующая этой имликанте, столбцам которой, соответствующим выбранным конъюнктам, присваиваются единичные значения. Например, первым трем строкам табл. 3 соответствует первичная импликанта Books>2 `AND` Citations=32. Для нее формируется первая строка табл. 4, в которой первое, второе и третье поле содержат значение 1.

В этом случае, как и при исполнении предыдущего алгоритма, можно прибегнуть к операциям над битовыми строками. Каждая пара строк, хотя бы в одном столбце которых значение бита ПВУ равно 1, складывается по модулю два. Если битовая строка суммы содержит не более одной ненулевой группы КУ, то это означает, что конъюнкты содержат только одно условие с несовпадающим кодом операции. Чтобы проверить, так ли это, в строке производится поиск любой из единиц. При ее обнаружении соответствующая группа КУ обнуляется. Если после этого строка не становится нулевой, то это значит, что она содержит еще минимум одно ненулевое значение в другой паре битов. Следовательно, рассматриваемые строки содержат более одного несовпадающего условия.

При работе с примером 3 этот алгоритм позволил выделить следующие первичные импликанты: Books>2 `AND` Citations=32; Articles=25 `AND` Books>2; Articles<25 `AND` Books>2. При их поиске построена табл. 4.

Таблица 4. Существенные импликанты и соответствующие конъюнкты для примера 3

	1	2	3	4	5	6	7	8	9	10	11
Books>2 `AND` Citations=32	1	1	1	0	0	0	0	0	0	0	0
Articles=25 `AND` Books>2	0	1	0	0	0	1	1	0	0	0	0
Articles<25 `AND` Books>2	0	0	1	1	1	0	0	0	0	0	0

Затем проводятся заключительные действия. Четыре последние столбца не могут быть представлены с помощью созданных импликант и автоматически включаются в конечное представление. Все три представленные импликанты являются существенными, так что все они включаются в конечное представление. На этом выполнение алгоритма заканчивается.

Итоговое представление Примера 3 выглядит следующим образом:

```
Books>2 AND Citations=32
OR Articles=25 AND Books>2
OR Articles<25 AND Books>2
OR Articles>25 AND Books=2 AND Citations<32
OR Articles>25 AND Books=2 AND Citations>32
OR Articles>25 AND Books<2 AND Citations<32
OR Articles>25 AND Books<2 AND Citations>32
```

При использовании отрицаний оно может быть записано более лаконично:

```
Books>2 AND Citations=32
OR NOT(Articles>25) AND Books>2
OR Articles>25 AND NOT(Books>2) AND NOT(Citations=32)
```

Именно это представление является наиболее лаконичным из возможных с точки зрения алгоритма Квайна. Его выполнение в описанном выше случае приводит к удалению 58,3% избыточных условий.

5. Заключение

Итак, в данной статье рассмотрены приемы лексической оптимизации запросов, основанные на алгоритмах булевой алгебры, которые мы условно именуем «алгоритм поглощения» и «алгоритм Квайна». Несмотря на сравнительную простоту описанных выше алгоритмов, нам неизвестны их аналоги, основанные на близких принципах выявления избыточных условий. Сравнение с другими алгоритмами лексической оптимизации, в частности, с алгоритмом поиска «частых подвыражений» [3], представленном в релизе Oracle, показывает, что «алгоритм поглощения» и «алгоритм Квайна» выделяет больше разновидностей условий-дубликатов. Вопрос о большей или меньшей эффективности представленных алгоритмов по сравнению с более ограниченными алгоритмами модификации, существующими в настоящий момент, может быть однозначно решен только экспериментальным путем: при создании множества путей доступа для выборки запросов к реально действующей базе данных.

Список литературы

1. Jarke M., Koch J. Query Optimization in Database Systems // ACM Computer Survey. 1984. V. 16, No. 2.

2. Дейт К. Дж. Введение в системы баз данных. Москва - Санкт-Петербург - Киев: Вильямс, 2001. С. 642.

3. Query Optimization in Oracle Database 10g Release 2. An Oracle White Paper, June 2005.

4. Чаудхари С. Методы оптимизации запросов в реляционных системах // СУБД. 1998. № 3.

5. Кузнецов С. Д. Методы оптимизации выполнения запросов в реляционных СУБД. `http://www.citforum.ru/database/articles/art_26.shtml`

6. Hall P. A. V. Optimization of single expressions in a relational data base system // IBM Journal Res. Devel. 1976. V. 20. № 3.

7. Kim W. On optimizing an SQL-Like Nested Query // ACM TODS, September. 1982. P. 447, 449, 450.

8. Baekgaard L., Mark L. Incremental Computation of Nested Relational Query Expressions // ACM TODS. 1995. № 2. V. 20.

9. Mumick I. S., Finkelsteint S. J., Pirahesh H., Ramakrishnan R. Magic is Relevamt // In Garcia-Molina H. Jagadish, editors. Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data; May 23-25, 1990; Atlantic City, NJ, USA. ACM, 1990. P. 247-258.

10. Faber W., Greco G., Leone N .Magic Sets and their application to data integration // Journal of Computer and System Sciences 2007. №73(4). P. 584-609.

11. Смирнов С. Н., Задворьев И. С. Работаем с Oracle: Учебное пособие. М: Гелиос АРВ, 2002. С. 417-445.

12. Markl V., Lohman G. M., Raman V. LEO: An autonomic query optimizer for DB2 // IBM System Journal. 2003. V. 42. № 1.

13. Mendkovich N., Kuznetcov S. New Algorithms for Lexical Query Optimization // Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces. June 22-25, 2009, Cavtat/Dubrovnik, Croatia. Edited by Vesna Luzar-Stiffler, Iva Jarec, Zoran Bekic. Technical Editor Boris Grinfeld. P. 187-192.

14. Самуйлов К. Е., Севастьянов Л. А., Спесивов С. С. Лекции по дискретной математике. Часть I: логика. М.: изд. РУДН, 2000. С. 26.

15. Quin W. V. O cores and prime implicants of truth functions // American Mathematics Monthly. 1959. V. 66. № 9.

16. Яблонский С. В. Введение в дискретную математику. М., 1979. С. 231.

17. Тарасенко П. Ф., Бухарова М. Ф. Технология “The Reporter” для построения отчетов по базам данных // Вестник Томского Государственного Университета. Апрель, 2002. № 275. С. 171-172.

New algorithms for query modifications

Kuznetsov S. D., Mendkovich N. A.

Keywords: query optimization, lexical optimization.

New algorithms for query modifications are proposed. These algorithms involve the lexical optimization based on the mathematical transformations that have never been used for the query optimization before.

Сведения об авторах: **Сергей Дмитриевич Кузнецов**,
Институт Системного Программирования РАН,
профессор, главный научный сотрудник;
Никита Андреевич Мендкович,
Институт Системного Программирования РАН, аспирант.