УДК 004.622

Е.А. Новохатская, аспирант
А.Б. Кунгурцев, канд. техн. наук
Одесский национальный политехнический университет, г. Одесса, Украина

ФОРМИРОВАНИЕ ЛЕКСЕМ ПРИ ГРУППИРОВКЕ ЗАПРОСОВ В МЕТОДЕ ИНКРЕМЕНТАЛЬНОГО ОБНОВЛЕНИЯ МП

К.А. Новохатська, аспірант
О.Б. Кунгурцев, канд. техн. наук
Одеський національний політехнічний університет, м. Одеса, Україна

ФОРМУВАННЯ ЛЕКСЕМ ПРИ ГРУПУВАННІ ЗАПИТІВ У МЕТОДІ ІНКРЕМЕНТАЛЬНОГО ОНОВЛЕННЯ МП

Yekaterina Novokhatska, PhD student
Aleksey Kungurtsev, PhD inTechnical Sciences
Odessa National Polytechnic University, Odessa, Ukraine

FORMATION OF TOKENS IN QUERY GROUPING IN THE METHOD OF INCREMENTAL MAINTENANCE OF MATERIALIZED VIEWS

Рассмотрены проблемы группировки запросов в методе инкрементального обновления материализованных представлений. Предложен алгоритм формирования лексем, учитывающий синтаксические особенности языка SQL. Снижена ресурсоемкость методики группировки запросов путем перехода от текстовых алгоритмов сравнения запросов к числовым.

***Ключевые слова**: группировка запросов, формирование лексем, инкрементальное обновление, материализованное представление.*

Розглянуто проблеми групування запитів у методі інкрементального оновлення матеріалізованих представлень. Запропоновано алгоритм формування лексем, що враховує синтаксичні особливості мови SQL. Знижено ресурсоємність методики групування запитів завдяки переходу від текстових алгоритмів порівняння запитів до числових.

***Ключові слова**: групування запитів, формування лексем, інкрементальне оновлення, матеріалізоване представлення.*

The paper presents questions of query grouping in the method of incremental update of materialized views. The algorithm of token formation that takes into account syntactic features of SQL is offered. Resource consumption of query grouping technique is lowered by transition from text comparison algorithms to numeric.

***Key words**: query grouping, tokens formation, incremental update, materialized view.*

Постановка проблемы. С ростом объемов данных, содержащихся в современных СУБД, понадобились новые подходы к предоставлению эффективного доступа к информации. Одним из способов повышения производительности запросов стало использование материализованных представлений (МП), которые представляют собой физические объекты базы данных, содержащие предварительно вычисленные результаты выполнения запроса.

Для достижения большей эффективности МП их стали использовать не только для оптимизации одиночных ресурсоемких запросов, но и для целых групп запросов, схожих по синтаксису и обрабатываемым данным. Так появился вопрос о создании автоматизированных инструментов, которые бы позволили проанализировать журнал транзакций СУБД и определить схожие группы запросов-кандидатов на материализацию.

Анализ последних исследований и публикаций. Одной из подзадач при группировке запросов является их представление к виду, удобному для сравнения и дальнейшей обработки. В работе [1] было предложено разбивать запрос на следующие множества:

1) набор полей фразы FROM;
2) набор полей фразы SELECT и WHERE;
3) условия выборки полей фразы WHERE, приведенные к дизъюнктивной либо конъюнктивной форме.

Разбиение запроса осуществлялось посредством использования регулярных выражений. Полученные множества сравнивались и по результатам сравнения формировались группы.

Преимуществом данной методики является ее простота. Однако в ней не принимаются во внимание многие конструкции современного языка SQL, что делает группировку неточной. Алгоритм не учитывает иерархические, коррелированные запросы, подзапросы, фразы WITH и т. д., что существенно сокращает область его использования. При дальнейшей обработке необходимо оперировать строковыми данными, что неудобно и требует лишних ресурсов. Так как сравниваются только поля, таблицы и логическая структура запроса, группу могут формировать запросы, результат которых существенно различается.

Выделение не решенных ранее частей общей проблемы. Нами предлагается разбиение запроса на лексемы с учетом грамматики языка SQL и их представление в виде числовых векторов, что при дальнейшей группировке позволит повысить качество сформированных групп и снизить ресурсоемкость методики путем оперирования только числовыми данными.

Цель статьи. Главной целью данной работы является:

1. Снижение ресурсоемкости методики группировки запросов путем оперирования числовыми данными.
2. Повышение качества формирования лексем с учетом синтаксиса языка SQL.
3. Повышение качества группировки с помощью расширения критериев сравнения запросов.

Входные данные. Пусть в результате работы некоторой информационной системы за некоторый период времени τ был сформирован журнал транзакций, содержащий множество запросов $S = \bigcup_{k=1}^{K} s_k \langle query \rangle$, где K – число записей в журнале транзакций, *query* – текст запросов вида *SELECT*.

Обработка данных. Введем понятие атомарной лексемы. Под атомарной лексемой *l* будем понимать одно или более выражений языка SQL такие, как названия полей, имена и псевдонимы таблиц, константы, функции, операторы, являющиеся минимальными смысловыми единицами при формировании фраз *SELECT, FROM, WHERE,* условий сортировки, группировки и т. д.

Словарем лексем *V* будем называть множество уникальных лексем $L = \bigcup_{n=1}^{N} l_n$, где N – общее число найденных лексем при разборе множества запросов *S*. Каждая запись словаря *V* описывается двойкой вида $<l_n, n>$, $n=1..N$, где l_n – текущая лексема, n – ее идентификатор (порядковый номер) в словаре *V*.

Для каждого запроса s_k, $k=1..K$ составим вектор D_k, содержащий множество лексем из словаря *V*, найденных при разборе данного запроса. Для простоты обработки данных вместо лексем при составлении вектора D_k будем оперировать их порядковыми номерами из словаря *V*. Число вхождений лексемы *l* в вектор D_k может быть больше 1.

Опишем алгоритм разбора запроса s_k, $k=1..K$. Словарь лексем $V = \{\varnothing\}$, $N = 0$.

Шаг 0. Инициализация.

Инициализируем коллекцию запросов $T=\{s_k\}$. *J* – число элементов коллекции T – равно единице. $D_k = \{\varnothing\}$.

Шаг 1. Подготовка запросов к разбору.

Для каждого запроса t_j, $j=1..J$ из множества T выполним следующие действия:

1.1. Удаление комментариев.

1.2. Замена числовых констант на шаблон @NUMBER.

1.3. Замена строковых констант и дат на шаблон @LITERAL.

1.4. Замена переменных на шаблон @BINDING.

1.5. Исключение операторов DISTINCT.

1.6. Удаление фраз ORDER BY.

1.7. Удаление псевдонимов таблиц.

Данный шаг позволяет на первом же этапе идентифицировать одинаковые по синтаксической структуре запросы, отличающиеся лишь значениями констант и переменных.

Фразы ORDER BY исключаются из разбора, т. к. способ сортировки результирующих данных не интересен с точки зрения формирования МП.

Два одинаковых запроса, различающихся только наличием DISTINCT, можно считать условно равнозначными, т. к. МП для запроса без фильтрации уникальных значений полностью покроет запросы с фильтрацией. Таким образом, фраза DISTINCT также не принимается во внимание при группировке запросов.

Шаг 2. Поиск конструкций *WITH* и блоков, объединенных операциями над множествами.

Посредством регулярных выражений для каждого запроса t_j, $j=1,J$ выделим фразу *WITH*, воспользовавшись следующей грамматической конструкцией, записанной в РБНФ:

t_j := *WITH* <*alias*> *AS* '(' <sq_1> ')' [{, <*alias*> *AS* '(' <sq_i> ')'}$_{i=2}^{H}$] *SELECT*…,

где sq_i, $i=1..H$ – текущий подзапрос, *alias* – имя подзапроса, *H* – число подзапросов в фразе *WITH*.

Исключим конструкцию *WITH* из исходного запроса t_j. Найденные подзапросы sq_i, $i=1..H$ добавим в коллекцию *T*:

$T = T \bigcup \{sq_i \mid i = \overline{1,H}\}$, $J = J+H$.

Аналогичным образом выделим независимые блоки запросов, объединенные операциями над множествами:

t_j := <sq_1> {(*UNION ALL*|*UNION*|*INTERSECT*|*MINUS*) <sq_i> }$_{i=2}^{W}$,

где *W* – число запросов, участвующих в объединении.

Каждый запрос sq_i, $i=1..W$ также в дальнейшем будем интерпретировать как самостоятельные запросы, добавив их в коллекцию T. Сам запрос t_j, который был разбит на несколько запросов sq_i, $i=1..W$, должен быть исключен из множества T:

$T = T \bigcup \left\{ sq_i \mid i = \overline{1,W} \right\}$, $T = T \setminus \{t_j\}$, $J = J+W-1$.

Шаг 3. Поиск коррелированных запросов во фразе SELECT.

Нередко в запросах используются коррелированные подзапросы следующего вида:

SELECT s.object_id AS tree_id,

(SELECT root FROM trees WHERE tree_id = s.obj_id AND rownum = 1) AS root_id

FROM objects s.

В общем виде:

t_j := *SELECT … [{'(' <*sq_i*>')'}*$_{i=1}^{X}$*] … FROM…*

Добавим найденное множество подзапросов sq_i, $i=1..X$ во множество T. Сами подзапросы в запросе t_j заменим шаблоном “@SUBQUERY”.

Шаг 4. Разбор фразы FROM.

Для каждого запроса t_j, $j=1..J$, принадлежащего множеству J, выделим фразу FROM:

t_j := *…FROM <from_clause> '('SELECT|WHERE|GROUP BY|ORDER BY|;')'…;*

<from_clause> := $<l_1>$*[{,<*l_i*>}*$_{i=2}^{Y}$*].*

Проверяем наличие найденных лексем l_i, $i = 1..Y$ в словаре данных V. Если лексемы не были найдены, добавляем их в виде двойки $<l_i, N+i>$. Полученные идентификаторы $N+i$ добавляем в вектор D_k.

Шаг 5. Разбиение фраз JOIN на лексемы.

При разборе FROM фраз воспользуемся следующими языковыми конструкциями для анализа JOIN выражений:

t_j :=*…<* l_1*>[INNER] JOIN <* l_2*>*

(ON <l_3*>[{(AND|OR) <*l_i*>}*$_{i=4}^{Z}$*]|USING <*l_3*>[{,<*l_i*>}*$_{i=4}^{Z}$*]).…;*

t_j :=*…<* l_1*>(CROSS|NATURAL [INNER|OUTER]) JOIN <* l_2*>…;*

t_j :=*…<* l_1*>(FULL|LEFT|RIGHT) [OUTER] JOIN <* l_2*>*

(ON <l_3*>[{(AND|OR)<*l_i*>}*$_{i=4}^{Z}$*] | USING <*l_3*>[{,<*l_i*>}*$_{i=4}^{Z}$*]).…*

Аналогично предыдущему шагу анализируем найденные лексемы l_i, $i = 1..Z$, добавим их в словарь V и заполним вектор D_k.

Шаг 6. Анализ фраз WHERE.

Разобьем найденные фразы WHERE на множество простых условных предикатов и добавим их в словарь данных V. Сформируем вектор D_k:

t_j *:=…WHERE <w_clause> ('('SELECT|ORDER BY|GROUP BY|WITH|;')' AS|START WITH)…*

<w_clause> := $<l_1>$*[{(AND|OR)<*l_i*>}*$_{i=2}^{U}$*].*

Шаг 7. Разбиение иерархической функции на атомарные составляющие.

Выделим лексемы из иерархических запросов, используя данное выражение:

t_j *:=…START WITH* $<l_1>$*[{(AND|OR)<*l_i*>}*$_{i=2}^{A}$*] CONNECT BY [NOCYCLE] [PRIOR]*

$<l_{A+1}>$*[{(AND|OR)<*l_j*>}*$_{j=A+2}^{A+B}$*] } (;|')'|WHERE|ORDER BY)…*

Аналогичным образом обработаем найденные лексемы l_n, $n = 1..A+B$.

Шаг 8. Выделение лексем из фраз GROUP BY и HAVING.

t_j *:=…GROUP BY* $<l_1>$*[{,<*l_i*>}*$_{i=2}^{E}$*][HAVING* $<l_{E+1}>$*[{(AND|OR)<*l_j*>}*$_{j=E+2}^{E+F}$*]].*

В случае отсутствия найденных лексем l_i, $i = 1..E+F$ в словаре данных V добавляем их в виде двойки $<l_i, N+i>$. Полученные идентификаторы $N+i$ добавляем в вектор D_k.

Шаг 9. Разбиение фраз SELECT на лексемы.

Чтобы разбить фразу SELECT на множество названий полей, констант и функций воспользуемся данной языковой конструкцией:

t_j := *SELECT* $<l_1>[\{,<l_i>\}_{i=2}^{C}]$ *FROM…*

Пополняем словарь данных *V* и вектор D_k, добавив в них сформированные лексемы l_i, $i = 1..C$.

Таким образом, для запроса s_k, $k=1..K$ составлен вектор D_k. Инкрементируем значение *k* и возвращаемся к шагу 0.

Работу предложенного алгоритма рассмотрим на примере запроса S_1:

```
select    s1.rt object_id, s1.reference, np.object_id value
from      params np,
          (select connect_by_root nr.object_id rt, nr.reference
          from    references nr
          where  nr.attr_id = 1
          start with nr.object_id in (2, 3, 4) and nr.attr_id = 5
          connect by prior  nr.reference = nr.object_id and nr.attr_id = 6
          order by  LEVEL) s1
          where np.object_id = s1.reference and np.attr_id = 7 and rownum=8;
```

Словарь лексем *V*, сформированных для S_1, представлен в табл. 1. Для наглядности к каждой лексеме был добавлен префикс в виде названия фразы, в которой эта лексема была найдена (“select”, “from” и т. д.).

Таблица 1

Словарь лексем для запроса S_1

N	*l*
1	*select rt*
2	*select reference*
3	*select object_id*
4	*from params*
5	*select connect_by_root object_id*
6	*from references*
7	*where attr_id = @NUMBER*
8	*start with object_id in (@NUMBER, @NUMBER, @NUMBER)*
9	*start with attr_id = @NUMBER*
10	*connect by prior reference = object_id*
11	*connect by prior attr_id = @NUMBER*
12	*order by LEVEL*
13	*where object_id = reference*
14	*where attr_id = @NUMBER*
15	*where rownum=@NUMBER*

В результате анализа запроса получен следующий числовой вектор:

D_1 = *{1, 2 (2), 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}.*

Как можно заметить, лексема L_2 <"*select reference*"> встречается в векторе D_1 дважды.

Аналогично для запроса S_2:

```
select    object_id
from      references
where     attr_id = 9
          and rownum = 10.
```

Используя уже существующий словарь *V*, получаем следующий вектор:

D_2 = *{3, 6, 14, 15}.*

Как видно из приведенного выше примера, словарь лексем минимизирует затраты ресурсов на хранение запросов, т. к. содержит только уникальные лексемы. Представ-

ление запросов в виде числовых векторов, упрощает операцию их сравнения, а также расширяет разнообразие алгоритмов, которые могут быть применены на следующем шаге для получения групп синтаксически близких запросов. Помимо текстовых алгоритмов поиска дубликатов в программном коде к ним, например, добавляются алгоритмы кластеризации категорийных данных, такие как CLOPE [2], Cobweb [3] и т. д.

Сравнительный анализ алгоритмов. Для проведения сравнительного анализа предшествующего алгоритма [1] (далее «Алгоритм 1») и предложенного алгоритма (далее «Алгоритм 2») было сгенерировано множество запросов по классификации, описанной в работе [4]:

– простые запросы на выборку;

– простые запросы с группировкой;

– простые многотабличные запросы;

– запросы, использующие подзапросы и коррелированные запросы (в том числе блоки WITH);

– запросы с простым объединением по равенству;

– многотабличные запросы с объединениями и агрегациями;

– иерархические запросы;

– запросы с аналитическими функциями.

При генерации запросов использовалась грамматика языка Oracle SQL.

1. Ресурсоемкость алгоритмов. Анализ потребляемых ресурсов был проведен для журнала транзакций Oracle СУБД в 64-битной JVM. Для хранения данных использовались примитивные типы данных и массивы. Количество запросов в журнале транзакций – 1389 записей, из них – 524 уникальных запроса. Результаты анализа сведены в табл. 2.

Таблица 2

Сравнительный анализ алгоритмов по ресурсоемкости

Параметр	Алгоритм 1, байт	Алгоритм 2, байт
Размер словаря лексем	–	42 230
Размер структуры	482 100	43 415
Общий размер потребляемой памяти	482 100	85 645

Таким образом, за счет использования Алгоритма 2 размер памяти, потребляемой для хранения результатов разбора запросов, сократился в 5,6 раз.

2. Покрытие синтаксиса языка SQL. В результате экспериментального анализа было установлено, что Алгоритм 1 не поддерживает разбор запросов, содержащих блоки WITH, иерархии и аналитические функции. Для сложных многотабличных запросов и подзапросов также были получены неудовлетворительные результаты – часть из них была обработана с ошибками, остальные попали в список исключений и не прошли разбор. Результат анализа сведен в табл. 3.

Таблица 3

Сравнительный анализ алгоритмов по покрытию синтаксиса языка SQL

Вид запроса	Алгоритм 1	Алгоритм 2
Простые запросы на выборку	+	+
Простые запросы с группировкой	+	+
Простые многотабличные запросы	+	+
Запросы с подзапросами	частично	+
Запросы с блоками WITH	-	+
Запросы с простым объединением по равенству	+	+
Многотабличные запросы с объединениями и агрегациями	частично	+
Иерархические запросы	-	+
Запросы с аналитическими функциями	-	+

3. Качество сформированных групп определялось путем их сравнения с эталонами, сформированными вручную. При вычислении данной метрики использовались только те запросы, которые были покрыты обоими алгоритмами.

Поскольку на выходе анализируемых алгоритмов получены данные различной структуры, на шаге группировки запросов также применялись разные методики. Для Алгоритма 1 использовалось решение, предложенное в работе [1]. Запросы сопоставлялись путем сравнения их логической структуры, а именно дизъюнктивной либо конъюнктивной канонической формы.

Лексемы, полученные с помощью Алгоритма 2, обрабатывались с помощью алгоритма кластеризации CLOPE [5], основанного на использовании гистограмм.

Полученные с помощью двух методик группы запросов были сопоставлены с эталонными значениями. Процент ошибки был рассчитан как отношение числа запросов, попавших в эталонную группу, к общему числу запросов.

За счет учета синтаксических особенностей запросов помимо анализа их логической структуры нам удалось на 20 % улучшить качество сформированных групп по сравнению с предыдущей методикой. Однако данная метрика является не точной, т. к. не учитывает ошибку ручного формирования эталонных групп и влияние разных алгоритмов группировки.

Выводы. Предложенный алгоритм формирования лексем позволил существенно сократить ресурсоемкость методики группировки запросов путем оперирования числовыми данными. Размер памяти, потребляемой для хранения результатов разбора запросов, сократился в 5,6 раз.

Учет синтаксиса языка SQL при формировании лексем позволил применить методику группировки к запросам, содержащим блоки WITH, иерархии и аналитические функции. Также было достигнуто улучшение качества разбора запросов, содержащих подзапросы.

За счет учета синтаксических особенностей запросов помимо анализа их логической структуры удалось на 20 % улучшить качество сформированных групп.

Список использованных источников

1. *Нгуен Чан Куок Винь*. Повышение эффективности применения материализованных представлений в автоматизированных системах с реляционными базами данных : дис. канд. техн. наук / Нгуен Чан Куок Винь. – Одесса, 2005. – 192 с.

2. *Yang Y*. CLOPE: A Fast and Effective Clustering Algorithm for Transactional Data / Y. Yang, X. Guan, J. You // Knowledge Discovery and Data Mining. – 2002. – P. 682–687.

3. *Fisher D*. Knowledge Acquisition Via Incremental Conceptual Clustering / D. Fisher // Machine Learning 2. – 1987. – P. 139–172.

4. *Новохатская Е. А*. Методика генерации функций обновления в методе инкрементального обновления материализованных представлений / Е. А. Новохатская, Ю. Н. Возовиков // Вісник СумДУ. Серія «Технічні науки». – № 3. – 2011. – С. 82–96.

5. *Новохатська К. А*. Зниження обчислювальної складності задачі групування запитів у методі інкрементального оновлення МП із застосуванням алгоритму CLOPE / К. А. Новохатська, А. Б. Кіпер // Інтелектуальні технології в системному програмуванні : збірник наукових праць Всеукраїнської науково-практичної конференції молодих учених та студентів. – Хмельницький, 2013. – С. 120–121.