УДК 004.658.2

МЕТОДИКА ГЕНЕРАЦИИ ФУНКЦИЙ ОБНОВЛЕНИЯ В МЕТОДЕ ИНКРЕМЕНТАЛЬНОГО ОБНОВЛЕНИЯ МАТЕРИАЛИЗОВАННЫХ ПРЕДСТАВЛЕНИЙ

Е. А. Новохатская, Ю. Н. Возовиков,
Одесский национальный политехнический университет, г. Одесса

Одной из подзадач в методе инкрементального обновления материализованных представлений является генерация функций обновления на основе синтаксического разбора текста запроса. В статье предложена методика создания подобных функций, проанализированы случаи, когда существует необходимость использования таблиц обновлений, и выведены коэффициенты, позволяющие оценить эффективность применения МП на стадии эксплуатации.

Ключевые слова: *материализованное представление, инкрементальное обновление.*

Однією з підзадач у методі інкрементального оновлення матеріалізованих уявлень є генерація функцій оновлення на основі синтаксичного розбору тексту запиту. У статті запропоновано методику створення подібних функцій, проаналізовано випадки, коли існує необхідність використання таблиць оновлень і виведені коефіцієнти, що дозволяють оцінити ефективність застосування МП на стадії експлуатації.

Ключові слова*: матеріалізоване уявлення, інкрементальне оновлення.*

1 ВВЕДЕНИЕ

Одним из способов повышения производительности работы баз данных является использование материализованных представлений (МП) – сохраненных результатов выполнения запросов. Они обеспечивают значительное сокращение времени выполнения запроса за счет устранения большого числа операций выборки, объединения, а также агрегирования. Это дает существенный выигрыш в производительности, однако, в свою очередь, приводит к сложностям обслуживания данных и дополнительным затратам ресурсов.

Одной из основных проблем при использовании МП является обеспечение достоверности данных при их изменениях в основных таблицах. Для ее решения при каждой операции вставки, удаления и изменения данных необходимо выполнить пересчет всего МП, что является неэффективной, ресурсоемкой и в целом дорогостоящей операцией, особенно в случаях, когда работа ведется с разделенными базами данных, а также хранилищами.

Альтернативой этого метода является выполнение инкрементальных обновлений МП. За последние годы было предложено несколько алгоритмов [1-3], однако большинство из них затрагивает узкие классы задач. В [1] была предложена техника использования таблиц изменений (change-table technique), значительно повышающая производительность при обновлении агрегаций и объединений (joins). Недостатком этого метода является его реализация на языке логической выборки (deductive query language). В [3] рассматривается алгоритм для вычисления SPJ-запросов, не затрагивающий реализацию агрегаций, и т. д.

В [2] разработан математический аппарат для обслуживания МП, однако он не применим к операциям DELETE или DISTINCT.

2 ПОСТАНОВКА ЗАДАЧИ

Целью данной работы являются разработка методики генерации функций обновления МП, анализ необходимости использования ТО для

различных видов запросов и вывод числовых коэффициентов, позволяющих оценить эффективность применения ТО.

3 ИСХОДНЫЕ ДАННЫЕ

Пусть в результате некоторого анализа ИС было принято решение о материализации запросов $S_i<text, f_i, t_i, \Delta f_i, \Delta t_i>$, $i = \overline{1,N}$, где:

f_i – частота появления запроса;

t_i – время выполнения запроса;

Δf_i – частота изменения данных в ИТ, используемых в запросе;

Δt_i – время, необходимое для обновления данных в ИТ.

Критерии поиска запросов-кандидатов на материализацию – отдельная нетривиальная задача, которая не рассматривается в данной статье.

На основании лексического анализа предлагается классифицировать запросы по следующим группам:

1) простой запрос на выборку – обращение идет к одной таблице, возможно использование функций агрегирования;

2) простой запрос с группировкой;

3) простой многотабличный запрос – данные выбираются из нескольких таблиц, однако не выполняется объединение по внешнему ключу;

4) подчиненные запросы на выборку; коррелированные запросы;

5) простое объединение по равенству;

6) многотабличный запрос с объединением по условию и функциям агрегирования.

На практике встречаются более сложные спецификации запросов, однако большинство из них могут быть разделены на простые блоки, которые, в свою очередь, попадают под предложенную классификацию.

Представим запросы S как функции трех переменных

$$f(\pi, T, P), \text{ или } f(f_{вн}(\pi), T, P) \qquad (1)$$

где π – набор запрашиваемых полей;

T – множество таблиц, участвующих в запросе;

P – множество условных предикатов фразы WHERE;

$f_{вн}$ – внутренняя функция (COUNT, DISTINCT, и т. д.).

Создадим для них МП M_i, $i = \overline{1,N}$ как результаты выполнения этих запросов.

4 ТАБЛИЦА ОБНОВЛЕНИЙ

Рассмотрим некоторое МП M, содержащее поля m_i, $i = \overline{1,n}$. Зададим для данного МП таблицу обновлений (ТО) ΔM с полями m_j, $j = \overline{1,k}$ из спецификации запроса с указанием вида функции обновления (*Insert*, *Update*, *Delete*). Спецификацией запроса будем называть объединение множества $M = \bigcup_{i=1}^{n} m_i$ с множеством полей из P.

Тогда для каждой операции добавления, удаления и обновления данных в ИТ, вместо непосредственной модификации МП, необходимо выполнить соответствующую запись в ΔM с указанием старого (OLD) и обновленного (NEW) значений полей.

Проанализируем последнее утверждение. В [1] предложено создавать ТО для всех видов запросов. Однако на практике в некоторых случаях использование ТО является менее эффективным решением, чем непосредственное обновление МП. Это связано с тем, что время

обслуживания ТО может быть сопоставимо или даже больше, чем модификация МП.

Рассмотрим эффективность использования ТО для разных видов запросов. Для этого нам потребуются следующие временные характеристики:

– $t_{refresh}$ – время, необходимое для непосредственного обновления МП при модификации ИТ;

– $t_{МП}$ – время чтения результата из МП;

– Δt_{write} – время записи в ТО;

– $\Delta t_{refresh}$ – время, необходимое для обновления МП на основании данных из ТО.

Для анализа прироста производительности введем три числовых коэффициента, позволяющие сравнить долю времени, затрачиваемую системой на выполнение запроса:

– без МП: $$K = t \cdot f + \Delta t \cdot \Delta f; \quad (2)$$

– с МП без использования ТО: $$K_{МП} = t_{refresh} \cdot \Delta f + t_{МП} \cdot f; \quad (3)$$

– с МП, обслуживаемым ТО: $$K_{TO} = \Delta t_{write} \cdot \Delta f + t_{МП} \cdot f + \Delta t_{refresh} \cdot f. \quad (4)$$

Из сопоставления вышеперечисленных коэффициентов можно сделать следующие выводы:

– если $K \approx K_{МП}$ и $K_{TO} << K_{МП}$, то для обновления МП следует использовать ТО;

– если $K_{TO} \approx K_{МП}$, то от ТО следует отказаться и непосредственно модифицировать МП;

– если $K \approx K_{МП} \approx K_{TO}$, то использование данного МП не дает прироста производительности.

В результате анализа различных групп запросов нами были составлены рекомендации, приведенные в табл. 1.

Таблица 1 – Рекомендации по использованию ТО

Не использовать ТО	Использовать ТО
1) простой запрос 2) простой запрос с группировкой при низкой селективности 3) многотабличный запрос без объединения 4) некоторые группы простых объединений	1) простой запрос с группировкой при высокой селективности 2) подчиненные и коррелированные запросы 3) многотабличные запросы с объединением и агрегацией

Под селективностью понимается отношение числа уникальных значений некоторого поля ИТ к общему числу строк таблицы.

5 ЛОГИЧЕСКАЯ ЗАПИСЬ ФУНКЦИИ ОБНОВЛЕНИЯ

Для упрощения создания функции обновления МП в терминах SQL удобно перед генерацией триггера составить логическую запись функции обновления. Для этого введем следующие обозначения:

$\in_p$ – вхождение элемента предиката в условие P фразы WHERE;

$\underset{join}{=}$ – внутреннее объединение (inner join);

$\underset{join}{\Leftarrow}$ – левое объединение (left join);

$\underset{join}{\Rightarrow}$ – правое объединение (right join);

$\underset{join}{\Leftrightarrow}$ – внешнее объединение (outer join);

$\prod$ – объединение столбцов в единую строку, если условие истинно;

$\bigcup$ – объединение строк;

:= – присваивание;

/ – удаление элемента из множества.

Для обновления данных МП введем оператор $\overset{\Delta}{\leftarrow}_{\beta}^{\alpha}$, где α – условный предикат ($\alpha \in_p P$), а β – функция обновления. Данный оператор предложен в [1], однако мы модифицируем его функционал с целью достижения универсальности математического аппарата:

$$New(M)=M \overset{\Delta}{\leftarrow}_{\beta}^{\alpha} \Delta M. \tag{5}$$

Рассмотрим параметры α и β подробнее. Зададим α как условный предикат вида

$$\text{IF } \alpha\text{=TRUE THEN } \beta str_i,\ \bar{}\ str_j,\ i=\overline{1,k},\ j=\overline{1,m} \text{ ELSE RETURN FALSE,} \tag{6}$$

истинность которого проверяется для каждой пары строк $\langle str_i,\ \Delta str_j \rangle$, $i=\overline{1,k}$, $j=\overline{1,m}$, принадлежащих таблицам M и ΔM соответственно. Если для какой-либо строки Δstr_j нашлась такая строка str_i, что условие α истинно, то необходимо обновить найденную строку в соответствии с оператором β.

Строка str_i, согласующаяся со строкой Δstr_j в соответствии с условием α, обновляется следующим образом. Зададим функцию β как тройку вида

$$\langle C, op1(x), op2(x) \rangle \tag{7}$$

соответствующую предикату

$$\text{IF CONDITION=TRUE THEN } op1(x),\ \text{ELSE } op2(x) \tag{8}$$

где x, $x \in str_i$ – некоторая атомарная запись, хранящаяся в МП. Операция $\langle C, op1(x), op2(x) \rangle$ является математическим примитивом – оператором, необходимым для логической записи запроса. Для различных спецификаций вид функции β будет различен. Приведем полученные нами выражения в табл. 2.

Таблица 2 – Вид функции β для различных операций

	INSERT	UPDATE	DELETE
SELECT COUNT	<*true**,++,*x***>	<*true, x, x*>	<*true*, --, *x*>
SELECT SUM	<*true*,+,*x*>	<*true*,+*new-old*,*x*>	<*true*, -, *x*>
SELECT MIN	$<<\cdot, \doteq, x>$	$<<\cdot, \doteq, x>$	$<\neq, x, MIN(T)>$
SELECT MAX	$<\cdot>, \doteq, x>$	$<\cdot>, \doteq, x>$	$<\neq, x, MAX(T)>$
SELECT AVG	SUM/COUNT		
SELECT	<*true*, $\bigcup$, *x*>	<*true*, $\doteq$, *x*>	<*true*, /, *x*>
SELECT DISTINCT	$<\in M, x, \bigcup>$	$<\in M, x, \bigcup>$	$<\in M, /, x>$
IS***	<*p*,*true*,*false*>		

* – условие всегда истинно; ** – значение не изменяется; *** – функция IS необходима для применения условия к столбцам, отсутствующим в списке итоговых полей, но содержащихся в общей спецификации запроса

Рассмотрим несколько характерных случаев:

1. Если для одного из полей $m_i \in str_i$ применяется оператор U, то создается новая строка результата и x:= NULL.

2. Если создается новая строка результата и запрос содержит функцию MIN или MAX, то x: = текущему значению поля.

3. Если запрос содержит функцию COUNT, SUM или AVG и в итоговой выборке содержится несколько столбцов, то вся строка удаляется только в том случае, когда агрегирующее значение равно нулю.

4. Если запрос содержит функцию MIN или MAX и в итоговой выборке содержится несколько столбцов, то операция SELECT (DISTINCT) игнорируется.

5. При выполнении операции SELECT DISTINCT условие CONDITION оператора β применяется не к отдельной строке множества M, а ко всему МП.

6. Если выполняется операция UPDATE для SELECT DISTINCT и создается новая строка, то остальные поля старой строки обновляются в соответствии с операцией DELETE, а поля новой строки – INSERT.

В табл. 3 приведены приоритеты выполнения операций для случаев INSERT, UPDATE и DELETE.

Таблица 3 – Очередность выполнения операций

	INSERT	UPDATE	DELETE
SELECT COUNT (SUM, MIN, MAX, AVG)	3	3	2
SELECT (DISTINCT)	2	2	3
IS**	1	1	1

Рассмотрим работу оператора на примере.

Пример 1. Обновление результата выполнения функции MAX.

Предположим, что задано некоторое M, в одном из полей которого хранится значение функции MAX. Рассмотрим, как сработает оператор β при различных модификациях ИТ. Сведем результаты в табл. 4.

Таблица 4 – Пример работы оператора β для операции MAX

M	Модификация ИТ	Операция обновления
100	-	-
100	INSERT 50	$50 \cdot > 100$=false $\rightarrow$ max=x=100
100	INSERT 100	$100 \cdot > 100$=false $\rightarrow$ max=x
100	DELETE 50	$50 \cdot > 100$=false $\rightarrow$ max=x
150	INSERT 150	$150 \cdot > 100$=true $\rightarrow$ max=new=150
?	DELETE 150	(150==150)=true$\rightarrow$пересчет МП

Приведем еще несколько дополнительных комментариев. В табл. 2 сказано, что если МП содержит функцию *MIN, MAX* и строка Δstr_j пересекается со строкой str_s, то необходимо выполнить пересчет МП. Однако на практике, когда работа ведется с большими хранилищами данных на тысячи записей, существует возможность пренебречь точностью данных ради повышения производительности. В таком случае пользовательские изменения ИТ можно не учитывать и дождаться полного обновления МП. Тут следует учесть, что при таких наборах данных существует большая вероятность того, что в ИТ хранится несколько записей с минимальным (максимальным) значением, поэтому при единичных удалениях данные в МП останутся достоверными.

Если к системе предъявляется требование достоверности данных, то при возникновении критической ситуации потребуется пересчет МП. Так как эта операция достаточно длительная, то возможным вариантом решения проблемы является ввод счетчика. В этом случае необходимо знать не только, какое значение является минимальным (максимальным), но и сколько записей с этим значением присутствует в таблицах. Тогда пересчет МП потребуется при обнулении счетчика.

6 ГЕНЕРАЦИЯ ФУНКЦИИ ОБНОВЛЕНИЯ ДЛЯ РАЗЛИЧНЫХ ГРУПП ЗАПРОСОВ

Рассмотрим эффективность применения ТО и достоверность метода для разных групп запросов, записанных в терминах PostgreSQL.

6.1. Простой запрос на выборку (запрос к одной таблице; возможно использование функций агрегирования)

Затраты времени на выполнение такого запроса незначительны и сопоставимы со временем $\Delta T_{refresh}$. Поэтому использование ТО в таком случае является достаточно ресурсоемкой задачей, понижающей производительность. Для решения этой проблемы мы предлагаем отказаться от использования ТО и выполнять обновление МП без промежуточных действий.

Рассмотрим общую структуру запроса.

$$\begin{array}{l}\text{SELECT [DISTINCT] } поле_1, поле_2, ..., поле_n \\ \text{FROM } T_1 \text{ WHERE } p_1, p_2, ..., p_k.\end{array} \qquad (9)$$

В таком случае МП представляет собой функцию:

$$\begin{array}{l}\text{МП} = f(<поле_1, поле_2, ..., поле_n>, T_1, \{ p_1, p_2, ..., p_k \}), \text{ или} \\ \text{МП} = f(<f_{вн}(поле_1), f_{вн}(поле_2), ..., f_{вн}(поле_n)>, T_1, \{ p_1, p_2, ..., p_k \}).\end{array} \qquad (10)$$

Для удобства обозначим выражение (10) как МП = $f(C)$. При изменении ИТ новое значение МП будет получено следующим образом:

$$\text{NEW(МП)} = f(C+\Delta C). \qquad (11)$$

На основании того, что C и ΔC являются множествами, а оператор $\overset{\Delta}{\leftarrow}_{\beta}^{\alpha}$ обладает свойством линейности [2], справедливо записать

$$f(C+\Delta C) = f(C) \overset{\Delta}{\leftarrow}_{\beta}^{\alpha} f(\Delta C);$$

$$f(C+\Delta C) = f(C) \overset{\Delta}{\leftarrow}_{\beta}^{\alpha} f(\forall \Delta str_j \in \Delta C \mid j = \overline{1,l}),$$

где l – число модифицированных строк;

$$f(C+\Delta C) = f(C) \overset{\Delta}{\leftarrow}_{\beta}^{\alpha} \bigcup_{j=1}^{l} f(\Delta str_j), \forall \Delta str_j \in \Delta C;$$

$$f(C+\Delta C) = f(C) \overset{\Delta}{\leftarrow}_{\beta}^{\alpha} \bigcup_{j=1}^{l} (\forall \Delta str_j [поле_i] \mid j = \overline{1,l}; i = \overline{1,n});$$

$$f(C+\Delta C) = f(C) \overset{\Delta}{\leftarrow}_{\beta}^{\alpha} \bigcup_{j=1}^{l} \prod_{i=1}^{n} \Delta str_j [поле_i].$$

Аналогичным образом раскрываем $f(C)$:

$$f(C+\Delta C)= \bigcup_{s=1}^{m}\prod_{i=1}^{n} str_s[поле_i] \overset{\Delta}{\leftarrow}_{\beta}^{\alpha} \bigcup_{j=1}^{l}\prod_{i=1}^{n} \Delta str_j[поле_i],$$

где m – количество строк в МП;

$$f(C+\Delta C)= \bigcup_{\substack{s=1 \\ j=1}}^{m,l}\prod_{i=1}^{n} (f(str_s[поле_i]) \overset{\Delta}{\leftarrow}_{\beta}^{\alpha} f(\Delta str_j[поле_i]));$$

$$f(C+\Delta C)= \bigcup_{\substack{s=1 \\ j=1}}^{m,l}\prod_{i=1}^{n} (f(str_s[поле_i] \overset{\Delta}{\leftarrow}_{\beta}^{\alpha} \Delta str_j[поле_i])). \qquad (12)$$

Далее раскроем предикат α как множество { p_1, p_2,..., p_k }. При этом условие p_1 накладывает ограничение на поле $поле_1$ таблицы T_1, p_2 - на $поле_2$ и т.д. Данное утверждение не строгое – одно условие может накладывать ограничение на несколько столбцов.

Тогда оператор $\overset{\Delta}{\leftarrow}_{\beta}^{\alpha}$ будет подразумевать следующее:

– IF p_i =TRUE THEN β_i <$str_s[поле_i]$, $\Delta str_j[поле_i]$>, $s=\overline{1,m}$, $i=\overline{1,n}$, $j=\overline{1,l}$ ELSE RETURN FALSE – в случае, если для i-го поля выборки задано некоторое условие (<p_i, β_i, FALSE>);

– IF TRUE THEN β_i <$str_s[поле_i]$, $\Delta str_j[поле_i]$>, $s=\overline{1,m}$, $i=\overline{1,n}$, $j=\overline{1,l}$ ELSE RETURN FALSE – если для i-го поля нет ограничения (<TRUE, β_i, FALSE>);

– IF p_i =TRUE THEN IS<$str_s[поле_i]$, $\Delta str_j[поле_i]$>, $s=\overline{1,m}$, $i=\overline{1,n}$, $j=\overline{1,l}$ ELSE RETURN FALSE – в случае, если значение i-го поля не вошло в результирующую выборку, но для него задано некоторое условие (<p_i, IS, FALSE>).

Для всей строки Δstr_j множества ΔC:

$$< p_1,\beta_1,\text{FALSE} > \prod < p_2,\beta_2,\text{FALSE} > \prod \ldots \prod < p_k,\beta_k,\text{FALSE} > . \qquad (13)$$

Т.е. если для всех полей строки Δstr_j выполняется соответствующее условие α_i, то вся строка обновляется путем применения оператора β_i к значению поля $str_s[поле_i]$.

Таким образом, на основании формулы (13) составляются функции обновления для МП, построенных на простых запросах. Такая логическая запись легко преобразуется в триггер, что продемонстрировано на примере 2.

Пример 2. Составление функции обновления для простого запроса

Рассмотрим следующий запрос:

SELECT *Имя, Фамилия* FROM *Рабочий* WHERE *Ставка>40*.

Создадим для него МП и ТО следующего вида:

МП = SELECT(<*Имя, Фамилия*>, *Рабочий, Ставка>40*)

ΔМП=Δ*Рабочий*{*OpType, NEW.Имя, NEW.Фамилия, NEW.Ставка, OLD.Имя, OLD.Фамилия, OLD.Ставка*}

Предположим, что в начальный момент времени МП = {∅}.

По формуле (13) получаем следующие выражения для функций обновления:

$\overset{\Delta}{\leftarrow}_{\beta INSERT}^{\alpha} =$

Имя<true,select,false>∏*Фамилия*<true,select,false>∏*Ставка*<true,is,false>

$\overset{\Delta}{\leftarrow}_{\beta DELETE}^{\alpha} =$

Имя<old,select,false>∏*Фамилия*<old,select,false>∏*Ставка*<true,is,false>

$\overset{\Delta}{\leftarrow}_{\beta UPDATE}^{\alpha} =$

Имя<old,select,false>∏*Фамилия*<old,select,false>∏*Ставка*<true,is,false>

Таблица 5 – Обновление простого запроса на выборку

МП	ΔРабочий	Обновление
	{INSERT, *Андрей, Ильинский, 21, null, null, null*}	$\{\varnothing\}\overset{\Delta}{\leftarrow}_{\beta}^{\alpha}{}_{INSERT}$ {Андрей, Ильинский, 21} α : true∏true∏false = false
Имя – Александр Фамилия – Иванов Ставка – 55	{INSERT, *Александр, Иванов, 55, null, null, null*}	$\{\varnothing\}\overset{\Delta}{\leftarrow}_{\beta}^{\alpha}{}_{INSERT}$ {Александр, Иванов, 55} α : true∏true∏true = true {∅}⋃Александр {∅}⋃Иванов {∅}⋃55
	{DELETE, *null, null, null, Андрей, Ильинский, 21*}	{Александр, Иванов, 55} $\overset{\Delta}{\leftarrow}_{\beta}^{\alpha}{}_{DELETE}$ {Андрей, Ильинский, 21} α : false∏false∏false = false
Имя – Алексей Фамилия – Иванов Ставка – 55	{UDDATE, *Алексей, Иванов, 55, Александр, null, null*}	{Александр, Иванов, 55} $\overset{\Delta}{\leftarrow}_{\beta}^{\alpha}{}_{UPDATE}$ {Алексей, Иванов, 55}, old={Александр, Иванов, 55} α : true∏true∏true = true Имя ≒ Алексей Фамилия ≒ И
{∅}	{DELETE, *null, null, null, Алексей, Иванов, 55*}	{Алексей, Иванов, 55} $\overset{\Delta}{\leftarrow}_{\beta}^{\alpha}{}_{DELETE}$ {Алексей, Иванов, 55} α : true∏true∏true = true {Алексей}/Александр={∅} {Иванов}/Иванов={∅} {55}/55={∅}

Данные логические операции будут соответствовать следующему триггеру:

Листинг 1. Пример триггера для обновления МП, построенного на базе простого запроса на выборку.

```
SELECT Имя, Фамилия FROM Рабочий WHERE Ставка >40
CREATE FUNCTION refresh_func()
RETURNS TRIGGER AS $new_mp$
BEGIN
   IF (TG_OP='INSERT') THEN
      INSERT INTO MP VALUES (Имя, Фамилия) = (SELECT NEW.Имя,
      NEW.Фамилия FROM Рабочий WHERE NEW.Ставка>40);
      RETURN NEW;
   ELSEIF (TG_OP='UPDATE') THEN
      UPDATE MP SET (Имя, Фамилия) = (SELECT NEW.Имя,
      NEW.Фамилия FROM Рабочий)
      WHERE MP.Имя = OLD.Имя, MP.Фамилия =OLD.Фамилия;
      RETURN NEW;
   ELSEIF (TG_OP='DELETE') THEN
      DELETE FROM MP WHERE
      MP.Имя =OLD.Имя AND MP.Фамилия =OLD.Фамилия;
      RETURN NEW;
   ENDIF;
   RETURN NULL;
END;
$new.mp$ LANGUAGE plpgsql;
CREATE TRIGGER AFTER INSERT OR UPDATE OR DELETE ON Рабочий
EXECUTE PROCEDURE refresh_func();
```

6.2 Простой запрос с группировкой

При использовании группировки (операция GROUP BY) необходимо проанализировать предметную область и, по возможности, предсказать количество сгруппированных строк результата, т.е. оценить коэффициент селективности. Если оно будет невелико или поиск в МП оптимизирован (ввод индекса и пр.), то рационально обойтись без использования вспомогательных таблиц обновлений. Если такой возможности нет, то следует как можно более минимизировать затраты на ТО.

Логическая запись запроса с группировкой также претерпевает изменения. Для поля, по которому выполняется группировка, операция SELECT заменяется на SELECT DISTINCT.

Рассмотрим подробнее операцию обновления простого запроса с группировкой на конкретном примере.

Пример 3. Обновление МП, созданного как результат выполнения запроса с группировкой

SELECT *Спец,* COUNT*(Фамилия)* AS *Res* FROM *Рабочий* GROUP BY *Спец*
МП = SELECT(<DISTINCT(*Спец*), COUNT*(Фамилия)*>, *Рабочий,* TRUE)
ΔМП=Δ*Рабочий*{*OpType,* NEW.*Спец, OLD.Спец*}

$$\leftarrow^{\Delta}_{\beta INSERT}{}^{\alpha} = Специальность\text{<true,select distinct,false>}\prod Фамилия\text{<true,count,false>}$$

$$\leftarrow^{\Delta}_{\beta DELETE}{}^{\alpha} = Специальность\text{<true,select distinct,false>}\prod Фамилия\text{<true,count,false>}$$

$$\leftarrow^{\Delta}_{\beta UPDATE}{}^{\alpha} = Специальность\text{<true,select distinct,false>}\prod Фамилия\text{<true,count,false>}$$

Таблица 6 – Обновление простого запроса на выборку

МП	ΔРабочий	Обновление
Спец | *Res* Строитель | 10 Сварщик | 3	–	–
Спец | *Res* Строитель | 11 Сварщик | 3	{INSERT, Строитель, null}	$\{<\text{Строитель},10>\} \overset{\Delta\ \alpha}{\leftarrow}_{\beta INSERT} \{\text{Строитель}\}$ $\alpha : true \prod true = true$ $\{\text{Строитель}\} \in \{\text{Строитель}\} = true \rightarrow$ *Специальность=x; Res++;* $\{<\text{Сварщик},3>\} \overset{\Delta\ \alpha}{\leftarrow}_{\beta INSERT} \{\text{Строитель}\}$ $\alpha : false \prod true = false$
Спец | *Res* Строитель | 11 Сварщик | 2	{DELETE, null, Сварочник}	$\{<\text{Строитель},11>\} \overset{\Delta\ \alpha}{\leftarrow}_{\beta DELETE} \{\text{Сварщик}\}$ $\alpha : false \prod true = false$ $\{<\text{Сварщик},3>\} \overset{\Delta\ \alpha}{\leftarrow}_{\beta DELETE} \{\text{Сварщик}\}$ $\alpha : true \prod true = true$ *Res--;* $Res \neq NULL \rightarrow$ *Специальность=x;*
Спец | *Res* Строитель | 11 Сварщик | 1 Сантехник | 1	{UPDATE, Сантехник, Сварочник}	$\{<\text{Строитель},11>\} \overset{\Delta\ \alpha}{\leftarrow}_{\beta UPDATE}$ {new=Сантехник, old=Сварщик}; $\alpha : false \prod true = false$ $\{<\text{Сварщик},2>\} \overset{\Delta\ \alpha}{\leftarrow}_{\beta UPDATE}$ {new=Сантехник, old=Сварщик}; $\alpha : true \prod true = true$ МП$\cup${Сантехник, 0}; *old.Res--;* *new.Res++;*
Спец | *Res* Строитель | 11 Сварщик | 1 Сантехник | 1 Маляр | 1	{INSERT, Маляр, null}	{<Строитель,11>, <Сварщик,1>, <Сантехник,1>} $\overset{\Delta\ \alpha}{\leftarrow}_{\beta INSERT}$ {Маляр} $\{\text{Маляр}\} \notin \{\text{Строитель, Сварщик, Сантехник}\} \rightarrow$ МП$\cup${Маляр, 0}; *new.Res++;*

6.3 Простой многотабличный запрос (данные выбираются из нескольких таблиц, однако в запросе не используется связь по внешнему ключу)

В таких запросах новое значение МП зависит только от пользовательских изменений ΔC и не требует повторного перебора строк множества C.

$$\text{SELECT [DISTINCT] } T_1.\text{поле}_1, T_1.\text{поле}_2, \ldots, T_1.\text{поле}_n, T_2.\text{поле}_{n+1}, T_2.\text{поле}_{n+2}, \ldots, T_2.\text{поле}_{n+m}, \ldots \text{ FROM } T_1, T_2, \ldots \text{ WHERE } p_1, p_2, \ldots, p_k \quad (14)$$

Обновление МП возможно без использования ТО и выполняется аналогично алгоритму в п. 4.1.

Пример 4. Обновление многотабличного запроса

Зададим таблицу *Девушки(Ид, Имя, Город)* и *Юноши (Ид, Имя, Город)* и проанализируем следующий запрос:

SELECT *Имя* FROM *Девушки*, *Юноши* WHERE *Город*=”Одесса”

МП = SELECT(<*Имя*>, *Девушки*, *Юноши, Город*=”Одесса”)

$\overset{\Delta}{\leftarrow}_{\beta INSERT}^{\alpha}$ = *Имя*<true,select,false>∏*Город*<=="Одесса",is,false>

$\overset{\Delta}{\leftarrow}_{\beta DELETE}^{\alpha}$ =*Имя*<==old,select,false>∏*Город*<=="Одесса",is,false>

$\overset{\Delta}{\leftarrow}_{\beta UPDATE}^{\alpha}$ =*Имя*<==old,select,false>∏*Город*<=="Одесса",is,false>

6.4 Подчиненные запросы на выборку

В подобных случаях мы предлагаем материализовать подзапросы, которые послужат таблицами обновлений для главного запроса. Т.е. при обращении к основному МП необходимо обработать данные только из подчиненных ТО, не затрагивая ИТ, что значительно сократит объёмы перебора строк, особенно в случае использования коррелированных запросов. Кроме того, такой метод упрощает схему обслуживания самих ТО.

Пример 5.

Простым примером использования подзапроса может служить следующий оператор:

```
SELECT * from tbl1 WHERE f2=(SELECT f2 FROM tbl2 WHERE f1=1);
```

Создадим МП2 для подзапроса:

```
SELECT f2 FROM tbl2 WHERE f1=1;
```

Тогда основное МП запишем как:

```
SELECT * from tbl1, MP2 WHERE tbl1.f2= MP2.f2;
```

Пример 6.

```
SELECT f1, COUNT(*), SUM(f2) from tbl1 t1 GROUP BY f1
HAVING SUM(f2) > (SELECT MIN(f2)*4 FROM tbl1 t1_in
WHERE t1.f1=t1_in.f1);
MP2 = SELECT f1, MIN(f2)*4 AS Result FROM tbl1 GROUP BY f1;
MP =  SELECT f1, COUNT(*), SUM(f2) from tbl1 GROUP BY f1
HAVING SUM(f2)> (SELECT Result FROM MP2 WHERE tbl1.f1=MP2.f1);
```

6.5 Простое объединение по равенству

При проектировании БД часто возникают случаи, когда для отображения связи «М:М» создается таблица-пересечение, в которую включаются атрибуты, соответствующие ключу первой и второй таблиц. Для получения агрегированных данных из этих таблиц необходимо выполнить их объединение по равенству внешних ключей. В таких случаях решающее значение имеет спецификация запроса.

Приведем пример.

Пример 7.

```
SELECT Адрес COUNT (*) AS Result FROM Здание, Назначение, Рабочий
WHERE Здание.ИдЗдания = Назначение.ИдЗдания AND Рабочий.ИдРаб=
= Назначение.ИдРаб GROUP BY Здание.Адрес
```

Таблица 7 – Пример обновления объединения по равенству

Здание	INSERT	INSERT в МП
	DELETE	DELETE из МП
	UPDATE (*Адрес*)	UPDATE (*Адрес*)
Рабочий	INSERT	–
	DELETE	–
	UPDATE	–
Назначение	INSERT	++ *Result* WHERE *Адрес* = New.*Адрес*
	DELETE	-- *Result* WHERE *Адрес* = Old.*Адрес*
	UPDATE (ИдЗдания) UPDATE (ИдРаб)	-- *Result* WHERE *Адрес* = Old.*Адрес* AND ++ *Result* WHERE Адрес=New.*Адрес*

Из этого примера следует, что в некоторых случаях обновление запросов на объединение – достаточно тривиальная задача и нет необходимости использовать ТО.

Однако на практике встречаются более сложные случаи объединений (JOINS), обновление которых требует сложных вычислений.

6.6 Многотабличный запрос с объединением по условию и функциями агрегирования

При инкрементальном обновлении МП, построенного на базе многотабличного запроса, необходимо проанализировать структуру таблиц результата, спецификацию исходного запроса, а также частоту и характер обновления ИТ. На основании анализа запроса и опыта работы с системой выбирается оптимальное решение в рамках конкретной предметной области.

Рассмотрим общую структуру запроса:

$$\text{SELECT [DISTINCT] } T_1.\text{поле}_1, T_1.\text{поле}_2, ..., T_1.\text{поле}_n, T_2.\text{поле}_1, T_2.\text{поле}_2, ..., T_2.\text{поле}_k \text{ FROM } T_1 \text{ [FULL|LEFT|RIGHT[OUTER]] JOIN } T_2 \text{ ON } T_1p_1, T_1p_2,..., T_1p_l \text{ AND } T_2p_1, T_2p_2,..., T_2p_m \text{ AND } (T_1 \cup T_2)p_1, (T_1 \cup T_2)p_2,..., (T_1 \cup T_2)p_c \quad (15)$$

где T_1p_i и T_2p_j, $i=\overline{1,l}$, $j=\overline{1,m}$ – условные предикаты, которые содержат поля только из таблиц T_1 и T_2 соответственно, а $(T_1 \cup T_2)p_y$, $y=\overline{1,c}$ – сложные условия, в которых участвуют поля как таблицы T_1, так и T_2.

Раскроем выражение (11) для случая обновления внешнего объединения:

$$\begin{aligned}
&\text{NEW(МП)} = (T_1 \cup \Delta T_1) \underset{join}{\Leftrightarrow} (T_2 \cup \Delta T_2) \\
&\text{NEW(МП)} = (T_1 \underset{join}{\Leftrightarrow} T_2) \cup (T_1 \underset{join}{\Leftrightarrow} \Delta T_2) \cup (\Delta T_1 \underset{join}{\Leftrightarrow} T_2) \bigcup (\Delta T_1 \underset{join}{\Leftrightarrow} \Delta T_2)^{*} \quad (16) \\
&\text{NEW(МП)} = \text{МП} \cup (T_1 \underset{join}{\Leftrightarrow} \Delta T_2) \cup (\Delta T_1 \underset{join}{\Leftrightarrow} T_2) \cup (\Delta T_1 \underset{join}{\Leftrightarrow} \Delta T_2)
\end{aligned}$$

* - данное выражение также анализировалось в [1].

В случае модификации только одной из таблиц выражение (16) упрощается следующим образом:

$$\text{NEW(МП)} =) \; T_1 \underset{join}{\Leftrightarrow} (T_2 \cup \Delta T_2)$$

$$\text{NEW(МП)} = \; (T_1 \underset{join}{\Leftrightarrow} T_2) \cup (T_1 \underset{join}{\Leftrightarrow} \Delta T_2)$$

$$\text{NEW(МП)} = \text{МП} \cup (T_1 \underset{join}{\Leftrightarrow} \Delta T_2) \qquad (17)$$

и аналогично получаем:

$$\text{NEW(МП)} = \text{МП} \cup (T_2 \underset{join}{\Leftrightarrow} \Delta T_1)$$

Проанализируем последнее выражение. Пусть таблицы T_1 и T_2 имеют мощность равную N. Тогда для вывода результата исходного запроса СУБД потребуется выполнить $N * N$ операций перебора строк. При использовании МП и ТО единичное изменение в одной из ИТ потребует N операций перебора, что дает прирост производительности в N раз.

Однако, если ИТ модифицируются часто, каждая из таблиц имеет большую размерность и спецификация запроса содержит сложные конструкции (внутренние функции, большое число условий или таблиц, участвующих в запросе), то достигнутого прироста производительности будет недостаточно. Операции $(T_1 \underset{join}{\Leftrightarrow} \Delta T_2)$ и $(\Delta T_1 \underset{join}{\Leftrightarrow} T_2)$ все еще потребуют больших затрат времени.

Для решения данной проблемы построим дерево запроса (рис. 1).

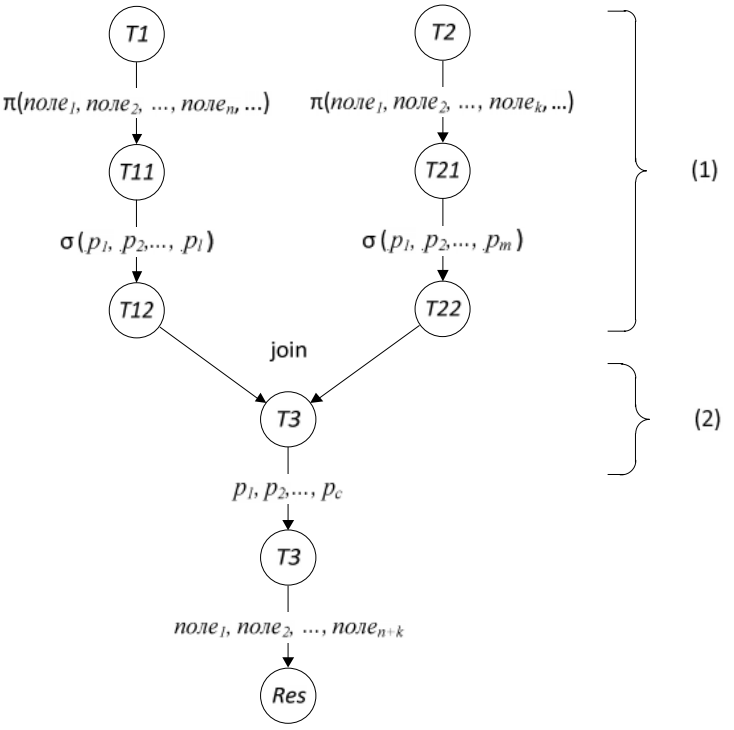

Рисунок 1 - Дерево запроса на объединение

Возможны следующие решения:

1. Материализация данных для этапа 1 (см. рис. 1):

Запрос (15) разбивается на два подзапроса по следующему алгоритму:

Этап 1. Формирование множества полей. Для каждой таблицы T_1 и T_2 множество полей, попадающих в результирующую таблицу T_1, формируется путем объединения полей из фразы SELECT и подмножества полей из сложных условий фразы ON.

$$\begin{aligned}&\text{Для таблицы } T_1\text{: } T_1.поле_1 \cup T_1.поле_2 \cup \ldots \cup T_1.поле_n \cup T_1.поле_{n+1} \cup \\ &\qquad T_1.поле_{n+2} \cup T_1.поле_{n+c}, \text{ где } T_1.поле_{n+1} \subset (T_1 \cup T_2)p_1, \\ &\qquad T_1.поле_{n+2} \subset (T_1 \cup T_2)p_2, \ldots, T_1.поле_{n+c} \subset (T_1 \cup T_2)p_c \\ &\text{Для таблицы } T_2\text{: } T_2.поле_1 \cup T_2.поле_2 \cup \ldots \cup T_2.поле_k \cup T_2.поле_{k+1} \cup \\ &\qquad T_2.поле_{k+2} \cup T_2.поле_{k+c}, \\ &\text{где } T_2.поле_{k+1} \subset (T_1 \cup T_2)p_1,\ T_2.поле_{k+2} \subset (T_1 \cup T_2)p_2, \ldots, \\ &\quad T_2.поле_{k+c} \subset (T_1 \cup T_2)p_c\end{aligned} \tag{18}$$

Этап 2. Формирование множества условий. Для каждой таблицы T_1 и T_2 множество условий фразы WHERE совпадает с подмножеством простых условий фразы ON.

$$\begin{aligned}&\text{Для таблицы } T_1\text{: } T_1.p_1,\ T_1.p_2, \ldots,\ T_1.p_l \\ &\text{Для таблицы } T_2\text{: } T_2.p_1,\ T_2.p_2, \ldots,\ T_2.p_m\end{aligned} \tag{19}$$

Этап 3. Формирование текста запроса.

$$\begin{aligned}&T_{11}\text{=SELECT } T_1.поле_1,\ T_1.поле_2,\ \ldots,\ T_1.поле_{n+c} \\ &\text{FROM } T_1 \text{ WHERE } T_1p_1,\ T_1p_2, \ldots,\ T_1p_l \\ &T_{21}\text{=SELECT } T_2.поле_1,\ T_2.поле_2,\ \ldots,\ T_2.поле_{k+c} \\ &\text{FROM } T_2 \text{ WHERE } T_2p_1,\ T_2p_2, \ldots,\ T_2p_m\end{aligned} \tag{20}$$

Этап 4. Материализация результатов выполнения запросов (20) и их обслуживание по алгоритму обновления простого запроса (см. п.1).

В результате материализации подзапросов (20) достигается:

– сокращение объемов обрабатываемых данных для обслуживания МП;

– упрощение схемы обновления МП (обновление МП разбивается на несколько более простых подзадач, решение каждой из которых требует незначительных затрат времени).

2. Материализация данных для этапа 2 (см. рис. 1):

В случаях, когда главный запрос содержит большое количество сложных условий, целесообразно сохранить промежуточные результаты для операции простого объединения и обслуживать их по соответствующему алгоритму из п.5.

$$\begin{aligned}&T_3 = \text{SELECT } T_{11}.поле_1,\ T_{11}.поле_2,\ \ldots,\ T_{11}.поле_{n+c}, \\ &T_{21}.поле_1,\ T_{21}.поле_2,\ \ldots,\ T_{21}.поле_{k+c} \text{ FROM } T_{11} \\ &[\text{FULL}|\text{LEFT}|\text{RIGHT}[\text{OUTER}]] \text{ JOIN } T_{21} \text{ ON } (T_1 \cup T_2).k,\end{aligned} \tag{21}$$

где $(T_1 \cup T_2).k$ – ключевое поле, по которому выполняется операция JOIN.

В результате материализации запроса (21) сокращается время, требуемое на обновление основного МП, за счет наличия заранее подготовленных данных для его расчета. Кроме того, схема обслуживания вспомогательного МП проще и производительнее.

3. Преобразование МП при использовании вспомогательных таблиц:

Запрос (15), на основе которого создано МП, при использовании заранее подготовленных данных преобразуется следующим образом:

$$\begin{aligned}&\text{SELECT [DISTINCT] } T_1.поле_1,\ T_1.поле_2,\ \ldots,\ T_1.поле_n, \\ &T_2.поле_1,\ T_2.поле_2,\ \ldots,\ T_2.поле_k \text{ FROM } T_3 \\ &\text{WHERE } (T_1 \cup T_2)p_1,\ (T_1 \cup T_2)p_2, \ldots,\ (T_1 \cup T_2)p_c,\end{aligned} \tag{22}$$

ВЫВОДЫ

В данной работе рассматривается методика генерации функций обновления МП, а также проанализирована необходимость использования ТО для различных видов запросов.

Проведенный анализ выявил, что для некоторых видов запросов использование ТО неэффективно, т.к. требует дополнительных затрат на обслуживание. Были выведены числовые коэффициенты, позволяющие оценить эффективность применения ТО на практике.

Предложенная классификация запросов позволяет распространить рассматриваемый метод на большинство практических приложений. Любой сложный запрос посредством декомпозиции может быть разбит на несколько простых запросов, к которым применимы разработанные алгоритмы.

Преимуществом предложенной методики является универсальность логической записи, которая сводит любой запрос к набору простых операций над атомарными значениями. Такой подход позволяет автоматизировать генерацию триггеров для обновления МП.

Использование инкрементальных обновлений позволяет существенно сократить время обновления МП и тем самым повысить эффективность их применения. Однако на практике время обновления МП будет зависеть от свойств данных, специфики самих запросов и их статистических показателей. Поэтому оптимальное по производительности и ресурсоемкости решение может частично отличаться от предложенных алгоритмов.

SUMMARY

THE METHOD OF UPDATE FUNCTION GENERATION FOR THE INCREMENTAL MAINTENANCE OF MATERIALIZED VIEWS

Novokhatskaya K. A., Vozovikov Y. N.,
Odessa National Polytechnic University, Odessa

A subtask of the incremental maintenance of materialized views is update function generation based upon the query text parsing. In this article we propose the method for creating such functions, analyze necessity to use change-tables and derive the coefficients in order to assess the efficiency of applying materialized views.

Key words: *materialized view, incremental update.*

СПИСОК ЛИТЕРАТУРЫ

1. Gupta H. Incremental maintenance of aggregate and outerjoin expression. / H. Gupta, I. S. Mumick // Information Systems, vol. 31, issue 6. – Sep., 2006.
2. Gluche D. Incremental Updates for Materialized OQL Views / D. Gluche, T. Grust, C. Mainberger, M. H. Scholl // In Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases (DOOD), Switzerland. – Dec., 1997.
3. Blakeley J. Efficiently updating materialized views / J. Blakeley, P. Larson, F. Tompa // Proceedings of the 1986 ACM SIGMOD international conference, vol. 15, is. 2 – Jun., 1986.
4. Кунгурцев А.Б. Методы инкрементальной актуализации материализованных представлений. // А.Б. Кунгурцев, Куок Винь Нгуен Чан // тр. Международной научно-практической конференции „Новые информационные технологии в учебных заведениях Украины”, Одесса, 2005. - С. 128-130.

Поступила в редакцию 21 сентября 2011 г.