УДК 004.652

Емельченков Е.П., Макаров А.И.

Смоленский государственный университет, г. Смоленск, Россия

ОДИН СПОСОБ ПРЕДСТАВЛЕНИЯ ОПЕРАЦИОННЫХ ДАННЫХ В РЕЛЯЦИОННОЙ МОДЕЛИ

АННОТАЦИЯ

В статье рассматривается один способ ускорения выполнения унарных операций манипулирования данными в реляционных базах данных и приводится его математическое обоснование.

КЛЮЧЕВЫЕ СЛОВА

Реляционные базы данных; оперативные данные; операции манипулирования данными; ISAM.

Emelchenkov E.P., Makarov A.I.

Smolensk State University, Smolensk, Russia

ONE WAY OF PRESENTING OPERATION DATA IN A RELATIONAL MODEL

ABSTRACT

In the article one way to accelerate the implementation of the unary operations for manipulating data in relational databases is discussed and its mathematical foundation is provided.

KEYWORDS

Relational database; operational data; data manipulation operations; ISAM.

Статья посвящена решению проблемы ускорения выполнения унарных операций манипулирования данными: SELECT, INSERT, DELETE, UPDATE в реляционных базах данных. В основу решения проблемы положен метод проецирования хранящихся в базе данных на списковые структуры. Для использования этого метода нужно определить какой вид данных в наибольшей степени для него подходит. Это обусловлено тем, что в теории реляционных баз данных существует понятие перманентных данных, заменившее ранее использовавшееся «операционные данные». Однако, в современных системах управления базами данных это, скорее, два различных термина.

Перманентные данные – это данные, использующиеся в приложениях поддержки принятия решения, в которых большее значение имеют операции реляционной алгебры, в особенности декартово произведение (в языке SQL ему соответствует операция JOIN).

Операционные данные, хранимые для оперативных или производственных приложений баз данных, т.е. рутинных, часто выполняющихся приложений для поддержки повседневной работы предприятия. Для таких сред используется понятие «Оперативная обработка транзакций» (On-Line Transaction Processing – OLTP) и имеет наибольшее значение скорость выполнения операций манипулирования данными, таких как SELECT, INSERT, DELETE, UPDATE, далее операции манипулирования оперативными данными.

Но следует заметить, что очень часто перманентные данные получаются или обновляются с определенной периодичностью из операционных данных, хранимых в другой базе. Следовательно, объемы данных обоих типов одинаковы, но в современном понятии Big Data рассматриваются приоритетно перманентные данные, а операционные отходят на второй план.

Реляционная модель данных имеет 4 ограничения для отношений (таблиц)[2]:

1. Каждый кортеж содержит точно одно значение (соответствующего типа) для каждого атрибута;
2. Атрибуты не характеризуются каким-либо упорядочением (например, слева на право);
3. Кортежи не характеризуются каким-либо упорядочением (например, сверху вниз);
4. В отношении отсутствуют дубликаты кортежей.

Нормализованность отношений предполагает, что каждый кортеж для каждого из своих атрибутов имеет точно одно значение. Если же допускать любое множество элементов с фиксированной структурой и заданными на нем операциями, как возможный домен атрибутов, все возможные отношения находятся в 1 нормальной форме.

В конечном представлении в виде таблицы, атрибуты (столбцы) имеют строгий порядок, но следует учитывать, что этот порядок необходим только для представления информации и не имеет значения в теории баз данных.

Реляционная модель данных предполагает отсутствие упорядоченности кортежей. Утверждается [1], что «То обстоятельство, что тело отношения является множеством кортежей, облегчает построение полного механизма реляционной модели данных, включая базовые средства манипулирования данными – реляционные алгебру и исчисление». А также «Отсутствие требования к поддержанию порядка на множестве кортежей отношения придает СУБД дополнительную гибкость при хранении баз данных во внешней памяти и при выполнении запросов к базе данных». Однако, из-за отсутствия упорядоченности сложность операций манипулирования оперативными данными возрастает до n – количества записей в базе.

Отсутствие дубликатов кортежей следует из отсутствия необходимости хранения уже имеющихся данных.

Эти четыре требования являются фундаментальными свойствами отношений в реляционной модели баз данных и действительно облегчают работу с перманентными данными. Однако, как уже было замечено, для оперативных данных, большое значение имеют операций манипулирования оперативными данными, на которые третье требование накладывает ограничение на сложность, а значит и на время выполнения.

Если же отказаться в технологических целях от допущения неупорядоченности кортежей, то можно рассматривать динамически упорядоченную модель данных, которую можно связать с реляционной моделью, установив соответствие между данными и операциями в обеих моделях. Таким образом предлагаемая модель будет промежуточной моделью между высокоуровневой, используемой на этапе проектирования реляционной моделью, и низкоуровневой моделью вычислений, которая определяется характером и структурой вычислительной системы [3]. Но это допущение следует из определения отношения, а значит для создания новой модели придется дать новое определение этому термину. Согласно С.Д.Кузнецову: “Для уточнения термина отношение выделяются понятия заголовка отношения, значения отношения и переменной отношения. Кроме того, требуется вспомогательное понятие кортежа” [1].

Заголовком (или схемой) отношения r (Hr) называется конечное множество упорядоченных пар вида <A, T>, где A называется именем атрибута, а T обозначает имя домена, произвольного типа с фиксированной структурой. По определению требуется, чтобы все имена атрибутов в заголовке отношения были различны.

Кортежем tr, соответствующим заголовку Hr, называется множество упорядоченных триплетов вида <A, T, v>, по одному такому триплету для каждого атрибута в Hr.

Телом Br отношения r называется произвольное множество кортежей tr.

Значением Vr отношения r называется пара из множества Hr и Br.

Переменной VARr называется именованный контейнер, который может содержать любое допустимое значение Vr.

Перед рассмотрением операций манипулирования оперативными данными в новой модели, следует описать структуру самих данных. Все кортежи хранятся в физической памяти вычислительной системы, не занимая место в ее оперативной памяти, обращение к ним происходит только на заключительных этапах операций, с целью исключения потери времени на считывание. В оперативной памяти хранятся двухуровневые индексы, созданные на основе индексно-последовательного метода доступа (ISAM). Таблица главных индексов представляет собой проекцию всей базы на двухсвязный список, где каждая запись состоит из значения атрибута A_i соответствующей записи базы, ее адреса в физической памяти (что является уникальным ключом), и двух полей указателей, необходимых для организации двухсвязного списка. Основным требованием к данной таблице является ее упорядоченность по значению атрибутов A_i. Таблица вторичных индексов строится по всем правилам реляционной модели, каждый кортеж состоит из значения атрибута A_i, каждой k-ой записи, начиная отсчитывать с конца таблицы первичных индексов и адреса этой записи в оперативной памяти. Таблица вторичных индексов получается упорядоченной в обратном таблице первичных индексов порядке за счет упорядоченности последней и начала отсчета с конца. Таких пар таблиц должно содержаться по одной на каждый, участвующий в построении предикатов, используемых в запросах, атрибут или набор атрибутов, для которых можно задать отношение порядка. Одним из оптимальных значений k является значение $\sqrt[3]{n^2}$, если n -общее число записей в базе данных. Тогда размер таблицы вторичных индексов будет $\sqrt[3]{n} - 1$.

В новой модели операция SELECT является основной среди операций манипулирования оперативными данными, так как будет задействована при выполнении любой другой операции. В

зависимости от предикатов, заданных в запросе, будет происходить обращение к одной или более парам индексных таблиц. Далее рассмотрим случай использования одного предиката, легко обобщаемый на большее их количество.

Пусть дан предикат П. Включающий в себя значение атрибута A_i. Необходимо среди вторичных индексов соответствующих данному атрибуту найти запись или записи, максимально близкие к истинному предикату, но не обращающие его в истинное тождество (к примеру, для предиката “Год выпуска” < 1890 и “Год выпуска” > 1700 – найти наименьшую из записей , с атрибутом “Год выпуска” не меньшим 1890, либо наибольшую не меньшую 1700). Это возможно организовать с помощью дихотомического поиска, соответственно сложность этой части запроса составляет $O(Log_2(m))$, где m – количество записей в метаданных. Затем получив из найденных записей адреса соответствующих кортежей первичных индексов, двигаться по списку в направлении значений, дающих истинный предикат, пока не будет обнаружена первая удовлетворяющая ему запись, начиная с нее и до первой, не удовлетворяющей предикату, по адресу в физической памяти можно получать все указанные в запросе атрибуты. Либо считать k записей дойдя соответственно до следующей записи в метаданных. В данной части алгоритма, требуется прочитать не более k записей для нахождения первого совпадения, соответственно ее сложность O(k). Суммарная теоретическая сложность получается $O(k + Log_2(m))$.

Второй из основных операций, является операция INSERT. Второй она рассматривается, потому что опирается на поиск записи в файле, при реализации же она имеет возможно первостепенное значение. При добавлении новой записи, она записывается на свободное место в физической памяти, но в каждую из пар индексных таблиц необходимо внести сведения об изменении базы. Так, в таблице первичных индексов появляется новая запись, меняются ссылки двух соседних записей, так как основой структуры является двухсвязный список, а в таблице вторичных производится перерасчет. Поиск подходящего места для записи схож с запросом SELECT по предикату $A_i = A'_i$ новой записи, с небольшим изменением: считывание происходит до первого совпадения, либо первой записи, превзошедшей по порядку значение A_i. Пересчет происходит из требований к количеству записей, разделяющих записи метаданных, их должно быть k штук. Для его реализации необходимо изменить все вторичные индексы, значения атрибута A_i которых не превосходит соответствующее значение добавленной записи, на соседние для них первичные индексы, в порядке возрастания. Сложность операции INSERT будет складываться из сложности поиска и максимальной сложности пересчета: $O(k + Log_2(m)) + O(m)$.

Однако следует заметить, что k и m могут принимать только целочисленные значения, а из их назначения $k * m = n$. Получается, что невозможно получить такие вторичные индексы, которые будут удовлетворять требованию: между записями, соответствующим двум соседним вторичным индексам, расположено строго k записей. Максимально приближенным вариантом будет, когда при росте n, растут соответственно k и m.

Одним из способов решения данной проблемы является создание внеиндексной (для вторичных индексов) части таблицы, в которой будут накапливаться записи до определенного количества, а потом при пересчете, они будут включаться в существующие группы из k записей, увеличивая значение k на один, либо образовывать новую группу увеличивая количество записей в таблице вторичных индексов m на один. Таким образом может образоваться внеиндексная часть базы размером k или m записей. Это увеличивает наибольшую сложность поиска, а следовательно, и добавления до $O(\max(k,m) + Log_2(m))$ и $O(\max(k,m) + Log_2(m)) + O(m) + O((m^2+m)/2)$, так как сложность увеличения количества записей вторичных индексов сводится к добавлению адреса и значения наименьшего по порядку значения атрибута A_i и его адреса, а увеличение размера записей к считыванию $\sum_{i=1}^{m} i = (m^2 + m)/2$ записей: первый вторичный индекс сместится на одну первичную запись, второй – на две и так далее. Из сложности добавления легко заметить, что оптимальные значения удовлетворяют условию k > m. Тогда получаем сложности $O(k + Log_2(m))$ и $O(k + Log_2(m) + (m^2+3m)/2)$. Из сложностей поиска и добавления, и смысла переменных k и m получаем систему ограничений:

$$\begin{cases} k + Log_2(m) \to min \\ k + Log_2(m) + (m^2 + 3m)/2 \to min. \\ k * (m + 1) \geq n \end{cases}$$

Не стоит забывать, что Big Data подразумевает собой огромное количество записей, следовательно, $Log_2(m)$ имеет много меньший порядок чем k или m, а значит этим значением можно пренебречь при вычислениях.

$$\begin{cases} k \to min \\ k + (m^2 + 3m)/2 \to min. \\ k * (m + 1) \geq n \end{cases}$$

Из неравенства $k * (m + 1) \geq n$ легко получить нижнюю границу значения k, а одно из ограничений системы утверждает, что k должно быть минимальным.

$$\begin{cases} k \to min \\ k + (m^2 + 3m)/2 \to min \\ k \geq n/(m+1) \end{cases} \sim \begin{cases} k = n/(m+1) \\ \frac{n}{m+1} + (m^2 + 3m)/2 \to min \end{cases}.$$

Остается найти оптимальное значение m. Рассмотрим функцию f(m)=$\frac{n}{m+1} + (m^2 + 3m)/2$. Она непрерывна на D(f), f'(m)=$\frac{-n}{(m+1)^2} + m + 1.5$. Функция убывает на (-∞;x) где x – решение уравнения $\frac{-n}{(m+1)^2} + m + 1.5 = 0$. Снова учтя цели решения данной задачи, пренебрежем числом 0.5, оставив уравнение вида $\frac{-n}{(m+1)^2} + m + 1 = 0$. Его решением является m=$\sqrt[3]{n} - 1$, тогда k=$\sqrt[3]{n^2}$.

Таким образом мы получили близкие к оптимальным значения k и m. И можем выразить сложности операций SELECT и INSERT через n учитывая все предыдущие допущения: $O(\sqrt[3]{n^2})$ – сложность операции SELECT, $O(1.5\sqrt[3]{n^2} + 0.5\sqrt[3]{n})$ – сложность операции INSERT. Следует заметить, что это сложности в самых «неблагоприятных» случаях, когда искомой записи нет или, когда требуется изменение количества записей, разделяющих вторичные индексы.

Операция DELETE может рассматриваться как обратная к INSERT. Следовательно, выполнять она должна те же шаги: искать нужную запись, менять ссылки соседних элементов в таблице первичных индексов, производить пересчет в таблице вторичных индексов, а также в определенных случаях изменять значения k и m в соответствии с изменяющимся n. Поиск происходит аналогично операции SELECT. Перерасчет представляет собой замену всех вторичных индексов, значение атрибута A_i которых не превосходит удаляемое, на соседние первичные в порядке убывания, если в таком случае один из вторичных индексов должен выйти за пределы таблицы, то он удаляется, уменьшая значение m на один. Уменьшать значение k можно путем считывания $\sum_{i=1}^{m} i = (m^2 + m)/2$ записей: первый вторичный индекс сместится на одну первичную запись, второй – на две, и так далее. Учитывая все описанные шаги – сложность удаления записи получилась равной сложности добавления.

Последняя из не рассмотренных унарных операций манипулирования оперативными данными - UPDATE. Она состоит из поиска записи в которую необходимо внести изменения, собственно изменения и перемещения записи на соответствующее ее новым значениям место в таблица индексов. Поиск записи ничем не отличается от предыдущих. А вот изменение таблиц индексов является слиянием пересчета при операции DELETE и операции INSERT. После изменения данных необходимо удалить со старого места и изменить ссылки соседних элементов таблицы первичных индексов, и произвести пересчет таблицы вторичных, единственным отличием от выше описанной операции является отсутствие необходимости в уменьшении значения k. Затем следует добавить «новую» запись на подходящее место операцией INSERT, опять же без необходимости изменять k, так как оно сохранено предполагающим наличие этой записи. Сложность операции получается равной $O(2\sqrt[3]{n^2} + 2\sqrt[3]{n})$. Однако, если не должны изменяться значения атрибута A_i, для которых построены таблицы индексов, то сложность сводится к обыкновенному поиску записи - $O(\sqrt[3]{n^2})$.

Данные определения данных, их структуры и операций над ними полностью описывают модель представления оперативных данных в реляционной алгебре. Далее следует рассмотреть преимущества и недостатки использования именно такой модели в СУБД.

Одно из первых преимуществ – скорость поиска. За счет сокращения его сложности нет необходимости последовательного перебора всех данных базы, либо сортировки таблиц после добавления новых записей. Выигрыш по сравнению с обычным поиском приблизительно в $\sqrt[3]{n}$ раз. Единственным ограничением является необходимость хранения таблиц первичных индексов в оперативной памяти. Так как они представляют собой двухсвязные списки использование ссылок вне динамической памяти значительно замедлит операции, за счет трат время на считывание данных из физической памяти.

Тут же следует рассмотреть недостаток: при выполнении операции SELECT по более чем одному атрибуту необходимо задействовать в определенной количество раз большее производительной мощности, так как индексные таблицы никак не связаны между собой и поиск будет происходить по каждому и них, а затем будут выбираться общие найденные записи. Решение этой проблемы может быть создание индексной таблицы, имеющей вместо поля данных атрибута A_i поле данных для нового атрибута $A_{i1..ip}$ представляющего совокупность данных Ai,…A_p с заданным для них отношением порядка.

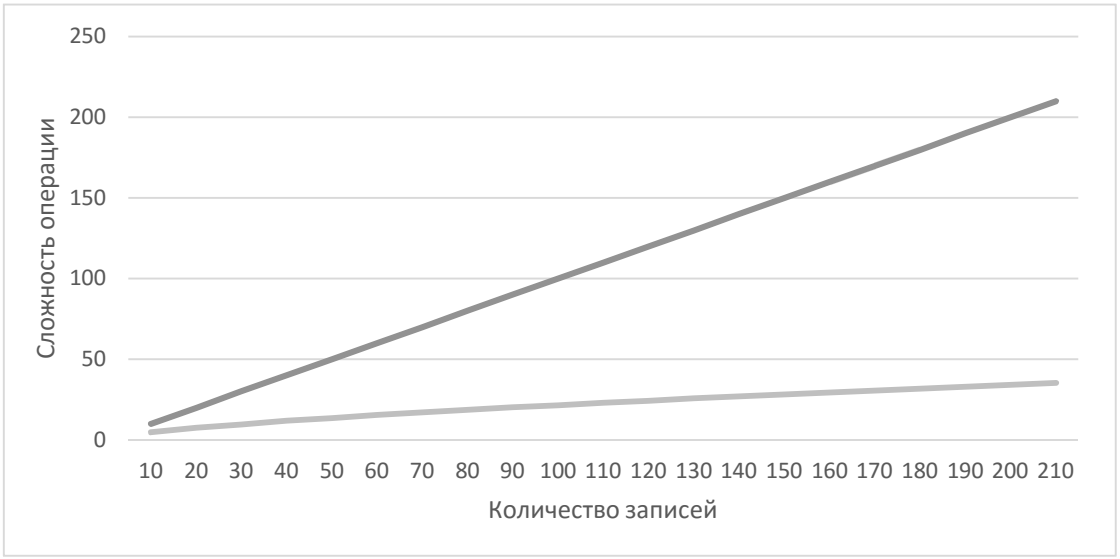

Рис.1. Сравнение скорости поиска в данной модели и в классической

Второе преимущество – полная упорядоченность сразу по нескольким значениям атрибутов. В отличии от классической реляционной модели любая база данной модели упорядочена и не нуждается в сортировке ни для каких операций или выгрузки данных в удобных для пользователя виде. Достаточно просто считывать последовательно записи по одной из таблиц первичных индексов.

Недостаток относящийся к упорядоченности схож с предыдущим – если пользователю необходимо отсортировать по нескольким значениям атрибутов, расставив при этом приоритеты – этого сделать не удастся, так как таблицы индексов никак между собой не связаны. Решением опять же может стать новая пара таблиц индексов для атрибута $A_{i1..ip}$ представляющего совокупность данных $A_i,…A_p$ с заданным для них отношением порядка.

И самый большой из недостатков данной модели тоже относится к требованию упорядоченности: усложнение, а как следствие и замедление, операций INSERT, DELETE и UPDATE. Для поддержания упорядоченной структуры базы необходимо тратить лишние ресурсы на эти операции по сравнению с классической моделью. Тут нет возможности предложить решение проблемы, можно ставить только задачи оптимизации выполнения этих операций, путем построения других алгоритмов, распараллеливания существующих, создания других структур данных. Однако, следует заметить, что выигрыш в поиске компенсирует добавление записей, а значит в целом если количество операций SELECT будет не на много меньше количества остальных операций, то их продолжительность не будет заметна.

Самым большим плюсом является возможность организации многопоточного доступа к базе. Опираясь на идеи ISAM, если база не будет ограничена от такой возможности на физическом уровне, легко организовывается множественный доступ. Не одна из операций над одним атрибутом или набором атрибутов не требует создания более одного потока вычислений. Пересекаться потоки могут только при пересчете вторичных индексов, что может быть исключено уже при непосредственной реализации данной модели и не будет оказывать большого влияния на ее работу. Таким образом можно будет исключить время «простоя» базы, пока оперативные данные переводятся в другую базу и становятся перманентными. А значит помимо ускорения работы увеличивается само время доступное для работы.

Еще одним плюсом является отсутствие необходимости в хранении данных в открытом виде. Для полноценного функционирования базе достаточно таблиц первичных и вторичных индексов, в которых в «открытом» виде хранятся значения только некоторых полей. Остальные же могут оставаться зашифрованными до момента их обработки. Что повышает защищенность данных от несанкционированного доступа, потому что даже достигнув оперативной памяти невозможно получить полную информацию о базе, для этого надо обращаться к физической памяти и нет помех для создания шифрования этого обращения. В то же время за счет «незащищенности» отдельных полей база не теряет в производительности.

Подводя итог, еще раз обращаю внимание, что в данной статье рассматривается представление именно оперативных данных и операции свойственные именно этому виду данных. Остальные операции реляционной модели могут выполняться и на этой модели, так как она

является наследницей реляционной, но в большинстве случаев из-за особой структуры данных и описания унарных операций они будут выполняться намного медленнее, чем во многих других моделях. Но нет никаких помех при переводе оперативные данные из базы в данной модели в перманентные данные другой, где уже в свою очередь и производить необходимые операции, а при необходимости загрузить обработанные данные обратно.

Литература

1. С.Д. Кузнецов. Основы баз данных. Учебное пособие 2-е издание, исправленное. Москва 2007. Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. – 484 с.
2. Дейт К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильяме", 2005. — 1328 с.
3. Мунерман В.И. Построение архитектур программно-аппаратных комплексов для повышения эффективности массовой обработки данных. – Системы высокой доступности, № 4, 2014, т.10, с. 3-16.

References

1. S.D. Kuznetsov. Osnovy baz dannykh. Uchebnoe posobie 2-e izdanie, ispravlennoe. Moskva 2007. Internet-Universitet Informatsionnykh Tekhnologiy; BINOM. Laboratoriya znaniy, 2007. – 484 s.
2. Deyt K. Dzh. Vvedenie v sistemy baz dannykh, 8-e izdanie.: Per. s angl. — M.: Izdatel'skiy dom "Vil'yame", 2005. — 1328 s.
3. Munerman V.I. Postroenie arkhitektur programmno-apparatnykh kompleksov dlya povysheniya effektivnosti massovoy obrabotki dannykh. – Sistemy vysokoy dostupnosti, № 4, 2014, t.10, s. 3-16.

Поступила 12.10.2016

Об авторах:

Емельченков Евгений Петрович, заведующий кафедрой информатики Смоленского государственного университета, кандидат физико-математических наук, ypy1101@gmail.com;

Макаров Александр Ильич, студент физико-математического факультета Смоленского государственного университета, al.makarov8@gmail.com.