УДК 004.89

ЭФФЕКТИВНЫЙ ЛОГИЧЕСКИЙ ВЫВОД В ПРОДУКЦИОННОЙ МОДЕЛИ С ПОМОЩЬЮ БАЗ ДАННЫХ С ВЕРТИКАЛЬНОЙ АРХИТЕКТУРОЙ

Р.С. Катериненко

Предлагается идея применения вертикальных (column-oriented) баз данных для логического вывода в продукционной модели знаний. Рассматриваются потенциальные преимущества использования указанного типа архитектуры по сравнению с традиционными реляционными базами данных для специфичных SQL-запросов, возникающих в продукционной модели. Приводятся результаты сравнительного эксперимента с вертикальной базой данных KDB+ и реляционной Oracle XE.

Ключевые слова: продукционная модель, логический вывод, вертикальная база данных, векторные языки программирования.

Введение

По своей природе логический вывод имеет большую алгоритмическую трудоемкость. Но в некоторых моделях, таких как дескрипционная логика и продукционные системы, трудоемкость ниже. Благодаря этому возможно практическое применение этих моделей. С целью повышения скорости логического вывода предлагается использовать базы данных (БД) с вертикальной архитектурой. В последнее время сообщество профессионалов все чаще обращает свое внимание на нереляционные БД [1, 2], но неизвестно ни одного применения этого типа для логического вывода. Предлагаемый подход является развитием идеи использования реляционных систем управления базами данных (СУБД) для логического вывода в продукционной модели знаний, предложенной в работе [3].

Трансляция в SQL

В работе [3] предлагается алгоритм трансляции процедуры логического вывода в SQL-запросы к реляционной БД. В продукционной модели знания описываются с помощью фактов и правил (продукций). Факты представляют собой атомарные формулы или, другими словами, реализацию какого-либо предиката. Например, факты SubClass(Fruct, Lemon) или Parent(John, Bob), являются реализациями предикатов SubClass(*X, Y*) и Parent(*X, Y*) соответственно. Основной объем базы знаний заключен в фактах. Их количество, как правило, превосходит количество правил. Предикаты могут быть произвольной арности. В предлагаемом подходе предикаты хранятся в таблицах БД. В столбцах таблицы, соответствующей предикату, находятся термы, при которых он принимает истинное значение. Логический вывод осуществляется путем последовательного применения продукций к фактам и порождения новых фактов до тех пор, пока продолжают выводиться ранее неизвестные факты (Fixpoint iteration).

С помощью SQL-запросов осуществляется поиск фактов, при которых искомый предикат является истинным. В контексте концепции закрытого мира (Closed World Assumption) это означает, что истинны те и только те факты, которые занесены в БД или могут быть получены логическим выводом из них. Например, если запрос пользователя имеет вид «$P(x)$», то должны быть выданы все факты с предикатом P, подтверждающие его истинность. При этом должны быть учтены факты, порожденные применением продукций.

При загрузке продукций анализируется их зависимость между собой по предикатам. Две продукции $P1$ и $P2$ называются зависимыми, если выполняется хотя бы одно условие:

– предикат из условия продукции $P1$ совпадает с предикатом из следствия продукции $P2$;
– предикат из следствия продукции $P1$ совпадает с предикатом из условия продукции $P2$.

По критерию зависимости строится граф (Program graph). Это позволяет найти для искомого предиката все необходимые продукции (цепочку продукций или дерево), выполнение которых может повлиять на значение искомого предиката. Найденное подмножество, как правило, содержит значительно меньшее количество продукций, чем вся база знаний. Далее по найденным продукциям генерируется SQL-запрос, возвращающий множество фактов для искомого предиката. Каждый предикат вида $b(x)$ определяет множество фактов:

```
S = {SELECT * FROM Predicate_B};
```

Продукция, содержащая предикаты $a1$ и $a2$, соединенные конъюнкцией по общей переменной, например, $a(y) :- a1(y)$, $a2(y)$, задает множество вида

```
S = {SELECT * FROM S1 JOIN S2}.
```

Две и более продукции с общим выводимым предикатом, например, $a1(t) :- a2(t)$, $a1(t) :- a3(t)$, задают дизъюнкцию множеств фактов $S1$ и $S2$ и объединяются в одно множество с предикатом $a1$:

```
S = {SELECT * FROM S1 UNION S2}.
```

Благодаря эффективности современных СУБД SQL-запросы выполняются достаточно быстро, и достигается ускорение логического вывода.

Постановка эксперимента

После небольшого введения в трансляцию продукций в SQL рассмотрим особенности генерируемого SQL. Наличие предикатов с общей переменной, связанных конъюнкцией, трансформируется в операцию соединения (join). Кроме того, как минимум одна конъюнкция имеется почти в каждой продукции. Таким образом, для логического вывода очень критична скорость выполнения соединения. В реляционной алгебре эта операция принимает на вход предикат. Путем вычисления истинности предиката устанавливается, должны ли обрабатываемые строки входить в результирующее множество. При трансляции продукций предикат всегда вырожденный – это равенство соответствующих термов.

Еще одной особенностью продукционной модели является то, что чаще всего предикаты имеют небольшую арность, но большое количество реализаций. При этом не все термы предиката связаны переменными, т.е. не все столбцы соответствующей предикату таблицы участвуют в вычислении запроса. Это наводит на мысль об использовании такой СУБД, в которой бы имелась возможность проводить операции не над строками, а над отдельными столбцами. Иначе говоря, целесообразно применение вместо реляционной БД вертикальной, ключевым свойством которой является возможность эффективно запросить значение отдельного столбца таблицы, а не всей строки, и тем самым избежать считывания лишней, нерелевантной информации и ограничить множество используемых в запросе столбцов [1]. Рассмотрим основные априорные свойства систем этого класса.

Особенности архитектуры вертикальных СУБД позволяют функционировать, занимая меньший объем хранилища по сравнению с традиционными строко-ориентированными аналогами. Это достигается благодаря более эффективной работе с архивированными столбцами в вертикальных БД. Некоторые системы этого класса реализуют свои операторы для работы с архивированными столбцами и используют их напрямую в плане вычисления запроса [2]. Например, рассматриваемая в эксперименте система KDB+ [4] позволяет хранить каждый столбец таблицы в отдельном файле и считывать его в память (выгружать из памяти) по мере необходимости. Кэширование осуществляется по столбцам, а не по строкам. Чем больше обращений к столбцу, тем выше вероятность нахождения его в кэше. Благодаря постолбцовому хранению значительно сокращается количество модифицируемых при удалении/добавлении структур данных [5].

Как уже упоминалось ранее, увеличив скорость выполнения соединения по равенству, можно добиться существенного прироста производительности. Вертикальная архитектура СУБД позволяет сделать подобное соединение очень эффективным. В некоторых системах применяется BAT-алгебра (BAT-Algebra, Binary Associated Table) [4].

Вертикальные БД ориентированы на высокопроизводительное чтение данных, а не на запись. Это позволяет поддерживать сложные структуры хранилища с трудоемкой операцией вставки, но быстрым чтением. При работе с продукционной базой знаний возникает схожая ситуация: факты редко меняются, но при этом часто требуется логический вывод, порождающий громоздкие запросы к базе. Типичный пример – экспертная система.

Учитывая вышеописанные предпосылки и априорные свойства вертикальных БД, было решено провести сравнительный эксперимент по скорости выполнения запроса. В качестве представителя систем указанного класса была взята система KDB+. Поскольку данная работа является первой работой автора в этом направлении, то не ставилась цель сделать исчерпывающий обзор вертикальных БД и выявить лучшую. KDB+ представляет собой СУБД, базирующуюся на векторном языке программирования Q. Данная технология нашла широкое применение в биржевой торговле для анализа больших массивов котировок биржевых инструментов. Синтаксис и семантика языка Q очень сильно отличается от SQL, от процедурных и объектных языков программирования. Базовые объекты языка – это функция и список, т.е. упорядоченный набор объектов. Соответственно, в контексте языка Q таблица является совокупностью списков, по одному для каждого столбца.

Для сравнительного эксперимента мы выбрали продукцию вида

```
Q(x) :- A(q,X) ^ C(m,X,Y,n) ^ B(Y,p),
```

в которой четырехместный предикат C связан переменной X с двухместным предикатом A и переменной Y с двухместным предикатом B. Каждый определяется соответствующей таблице, содержащий все его реализации:

```
create table tableA (
t0 char(20),
t1 char(20),
primary key (t0)
);
create table tableB (
t0 char(20),
t1 char(20),
primary key (t0)
);
```

```
create table tableC (
t0 char(20),
t1 char(20),
t2 char(20),
t3 char(20),
primary key (t0)
);
```

SQL-запрос для Oracle, соответствующий продукции, выглядит следующим образом:

```
select count(ta.t1)
from tableA ta, tableB tb,  tableC tc
where ta.t1 = tc.t1 and tb.t1 = tc.t2;
```

При этом выполняется только подсчет выводимых фактов, чтобы исключить задержки, связанные с выводом на терминал. Как уже говорилось, синтаксис и семантика языка Q сильно отличается от SQL, поэтому аналогичный Q-запрос выглядит по-другому:

```
\t select cnt:count(i)
from tableC
where (tableC.t1 in tableA.t1) and (tableC.t2 in tableB.t1).
```

Анализ результатов эксперимента

Ниже приведены результаты эксперимента. Графики описывают время выполнения запроса в зависимости от количества фактов в каждой таблице. Шкала по оси *Y* выбрана логарифмической, так как наибольший интерес представляет время исполнения запроса, нежели шаг по оси *X*. Графикам Oracle соответствуют линии с нечетными номерами, графикам KDB+ – линии с четными номерами.

На рис. 1 приведены графики максимальных и минимальных времен для обеих СУБД. Хорошо заметно, что минимальное время KDB+ никогда не превосходит минимальное время Oracle. Максимальные времена (выбросы) чаще всего соответствуют первому запуску запроса. Это связано с отсутствием информации в кэше. На больших объемах (больше 350 тысяч фактов) у Oracle наблюдается значительный рост времени первого исполнения запроса.

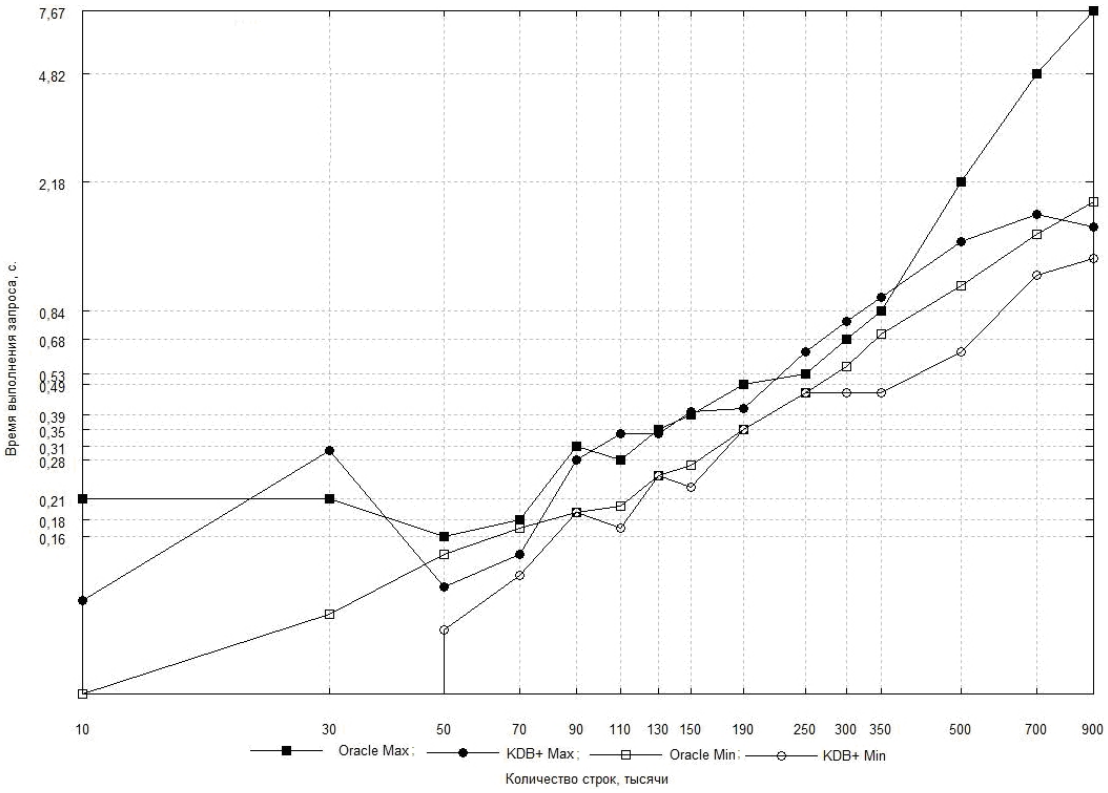

Рис. 1. Максимальное и минимальное время выполнения запроса

На рис. 2 приведены графики усредненных времен с исключенными выбросами. В среднем KDB+ показывает стабильный результат, в отличие от Oracle: при числе фактов до 90 000 время выполнения запроса намного превосходит KDB+, затем, при числе фактов до 300 000, приближается к нему и иногда оказывается меньшим.

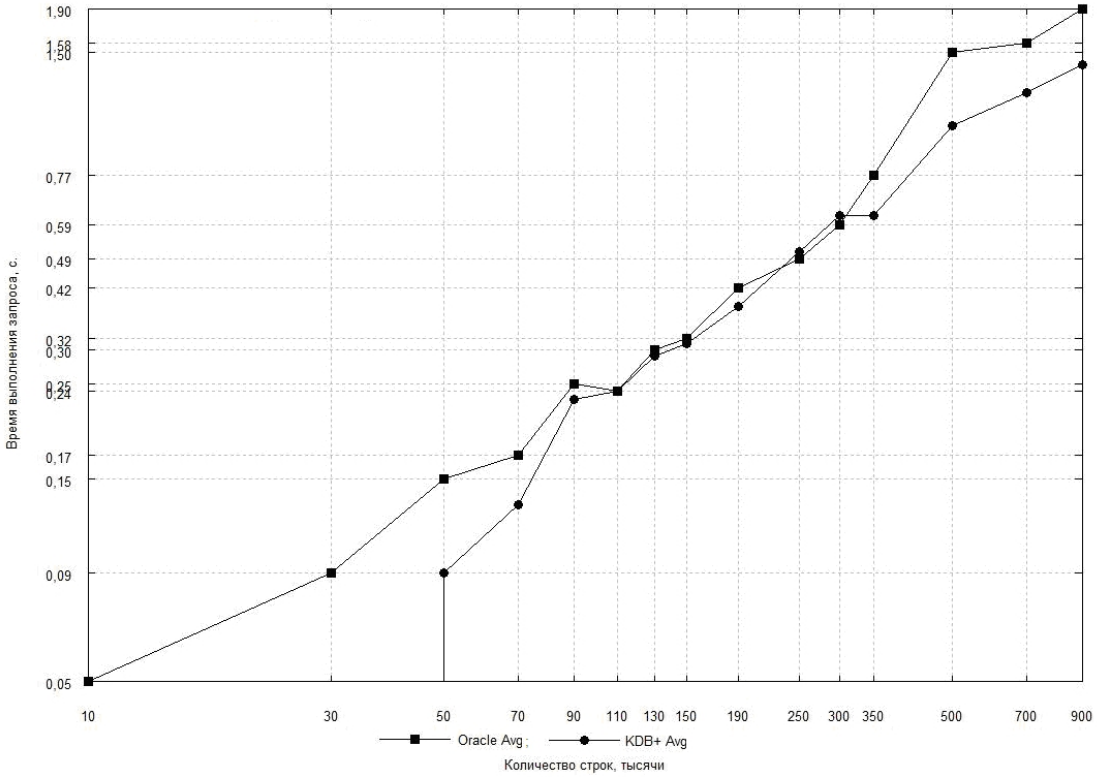

Рис. 2. Среднее время выполнения запроса

```
SQL> select count(ta.t1) from tableA ta, tableB tb,  tableC tc where ta.t1 = tc.t1 and tb.t1 = tc.t2

COUNT(TA.T1)
------------
      175000

Elapsed: 00:00:00.84

Execution Plan
----------------------------------------------------------
Plan hash value: 2254415514

-------------------------------------------------------------------------------------

| Id  | Operation            | Name    | Rows  | Bytes |TempSpc| Cost (%CPU)|
          |

-------------------------------------------------------------------------------------

|   0 | SELECT STATEMENT     |         |     1 |    88 |       |  5780   (2)|
01:10 |
|   1 |  SORT AGGREGATE      |         |     1 |    88 |       |            |
          |
|*  2 |   HASH JOIN          |         |   301K|    25M|    13M|  5780   (2)|
01:10 |
|   3 |    TABLE ACCESS FULL | TABLEB  |   425K|  9145K|       |   646   (3)|
00:08 |
|*  4 |    HASH JOIN         |         |   301K|    18M|    13M|  3315   (2)|
00:40 |
|   5 |     TABLE ACCESS FULL| TABLEA  |   408K|  8780K|       |   645   (3)|
00:08 |
|   6 |     TABLE ACCESS FULL| TABLEC  |   301K|    12M|       |  1192   (2)|
00:15 |

-------------------------------------------------------------------------------------


Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("TB"."T1"="TC"."T2")
   4 - access("TA"."T1"="TC"."T1")
```

Рис. 3. План выполнения запроса

После 350 000 фактов время запроса Oracle опять начинает расти. Представляется, что это связано с расходами на создание дополнительных структур при операции соединения, как видно из плана исполнения SQL-запроса на рис. 3. Соединение реализовано алгоритмом «Hash join». Для одной из таблиц, участвующей в соединении, создается хэш-таблица. Далее при последовательном проходе по другой таблице, применяя хэш-функцию, находятся совпадения.

Таким образом, на маленьких объемах построение хэш-таблицы не окупалось быстрым поиском по хэш-функции, но дало хороший результат на средних объемах. На больших объемах данных (приближающихся к 1000000) хэш-таблица увеличивается в размере настолько, что затраченное на ее создание время впоследствии не компенсируется быстрым поиском. К сожалению, пока не удалось найти информацию, какой алгоритм используется для операции соединения в KDB+. На рис. 3 также можно заметить, что реляционная БД создает временные структуры размером 100–200% от начального размера таблиц. Для вертикальной БД эта цифра составляет около 10%.

Заключение

При использовании СУБД для логического вывода в продукционной модели знаний происходит генерация специфичных SQL-запросов. Специфика заключается в потенциально большом количестве операций соединения по равенству и операций объединения. Чаще всего в запросах используется обращение только к небольшому количеству столбцов таблицы. Для эффективного использования этих особенностей может быть целесообразным использование вертикальной базы данных вместо традиционной строко-ориентированной базы данных. Преимущество заключается в эффективной реализации требуемых операций, исключающей считывание нерелевантных столбцов и оптимизации занимаемого объема хранилища.

На сравнительных экспериментах вертикальная база данных продемонстрировала лучшее время исполнения запроса при меньшем потреблении памяти. СУБД Oracle также предоставляет большие возможности по оптимизации: сбор статистики с целью изменения плана запроса, построение индексов по столбцам и т.д. Этими средствами можно улучшить производительность, но при этом будет затрачена память на дополнительные структуры. На данный момент автор не располагает информацией о способах настройки производительности KDB+. Таким образом, можно говорить о преимуществе вертикальной базы данных для логического вывода в продукционной модели, только если обе взяты с настройками по умолчанию, без оптимизации структуры хранения и запросов.

Литература

1. Stonebraker M., Bear Chuck, Cetintemel Ugur, Cherniack Mitch, Hackem Tingjian Ge Nabil, Harizapoulos Stavros, Lifter John, Rogers Jennie, Zdonik Stan. One Size Fits All? Part 2: Benchmarking Results // Proceedings of Conference on Innovative Database Research (CIDR). – 2007. – P. 173–184.
2. Stonebraker M., Abadi Daniel J., Batkin Adam, Chen Xuedong, Cherniak Mitch, Ferreiram Miguel, Lau Edmond, Lin Amerson, Madden Sam, O'Neil Elizabeth, O'Neil Pat, Rasin Alex, Tran Nga, Zdonik Stan. CStore: A Column-Oriented DBMS // Proceedings of the 31st International Conference on Very Large Data Bases. – 2005. – P. 553–564.
3. Bessmertny I., Katerinenko R. Inference acceleration in production model of knowledge // Programming and Computer Software. – 2011. – V. 42. – P. 197–199.
4. Kx Systems. Система управления базой данных KDB+. Документация [Электронный ресурс]. – Режим доступа: http://www.kx.com/products/database.php, св. Яз. англ. (дата обращения 10.10 2011).
5. Svensson Per, Boncz Peter, Ivanova Milena, Kersten Martin, Nes Niels. Emerging database systems in support of scientific data // Scientific data management: challenges technology and deployment. – 2010. – P. 235–281.

Катериненко Роман Сергеевич – Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, аспирант, innuWorks@gmail.com