Оптимизация представления XML документов в реляционной базе данных

Д.В. Барашев, Е.А. Горшкова, Б.А. Новиков
Санкт-Петербургский государственный университет
dbarashev@mail.ru, cathy@meta.math.spbu.ru, borisnov@acm.org

Аннотация

В данной работе мы исследуем задачу представления массива XML данных в реляционной СУБД и оптимизации этого представления по отношению к набору запросов. Мы рассматриваем различные способы хранения XML данных в таблицах, предлагаем функцию оценки эффективности реляционной схемы и пытаемся найти схему, наиболее эффективную для данного набора запросов.

1 ВВЕДЕНИЕ

С ростом количества информации, происходящей из одной предметной области, но из разных источников, растет и хаос в ее структуре. Характерным примером служит Интернет: сайты, посвященные единой тематике, могут иметь и имеют различные логические структуры, скрытые в путанице HTML тегов. Более того, даже на одном сайте, интегрирующем информацию из разных источников, может не быть четкой логической структуры.

Информация, структура которой четко не определена, но все же имеет некоторые черты, получила название *слабоструктурированной*. Такой информации достаточно много в окружающем мире. Слабоструктурированная информация может появиться не только на сайтах, интегрирующих различные источники данных, но и в едином источнике, описывающем данные одной природы, но сильно отличающиеся по содержанию. Так, например, собрания критики произведений двух различных авторов, скорее всего, будут сильно отличаться, если одно из произведений более актуально, нежели второе.

Наиболее распространенным способом представления слабоструктурированных данных является XML, относительно новый язык разметки текста [1], который достаточно быстро завоевывает популярность как в Интернет, так и в качестве языка передачи данных между приложениями. Наиболее привлекательной особенностью XML, позволяющей использовать его в электронных библиотеках, является возможность отделения данных от их визуального представления и использования тегов разметки для определения логической структуры текста. Кроме того, в настоящее время разрабатываются простые и мощные способы преобразования XML в более привычные форматы [2]. Это позволяет авторам XML документов разделить работу по составлению и оформлению документа, определять структуру документов через *определения типов (DTD)* и гибко менять его визуальное представление. Потребители XML документов получают возможность проводить поиск требуемых данных, подобно поиску в базе данных, используя информацию о структуре текста.

За последние годы было разработано множество языков запросов для поиска информации в XML данных ([3]). Как правило, все языки в качестве результата выполнения запроса возвращают документы или их части, удовлетворяющие некоторым ограничениям структуры или содержания. Результаты этих запросов, в свою очередь, могут быть использованы для построения новых документов.

Языки запросов для XML можно условно разделить на две группы. К первой относятся языки, ориентированные на текстовое представление XML документа. Запросы в них строятся в виде набора шаблонов, которым должен удовлетворять искомый участок документа, соединенных логическими условиями. Представителем этой группы является, например, язык XML-QL [4]

Языки второй группы оперируют с графовыми моделями XML документов [9]. Основным понятием в таких языках является *путь* в графе, выходящий из корня. Последовательность меток дуг, образующих путь, используется в запросах для обозначения всех объектов, достижимых по путям с такими метками. В таких языках часто используется конструкция `select-from-where`, указывающая пути к искомым объектам и накладывающая ограничения выборки в виде ограничений на объекты, достижимые по некоторым путям. Типичными представителями этой группы являются языки Lorel [5] и SOQL[6].

Параллельно с разработкой языков запросов велись поиски наилучшего способа хранения XML документов. Было предложено несколько вариантов хранилищ: файловая система, реляционная СУБД, объектно-ориентированная

©Вторая Всероссийская научная конференция
ЭЛЕКТРОННЫЕ БИБЛИОТЕКИ:
ПЕРСПЕКТИВНЫЕ МЕТОДЫ И ТЕХНОЛОГИИ,
ЭЛЕКТРОННЫЕ КОЛЛЕКЦИИ
26-28 сентября 2000г., Протвино

СУБД, специализированное хранилище для XML. Каждая из этих альтернатив имеет свои достоинства и недостатки. Файловая система кажется наиболее естественным решением и идеально подходит для хранения статических XML документов, которые целиком выдаются в качестве результата запроса. Таковыми документами являются, например, статические Web-страницы. В то же время, при необходимости конструирования документа динамически или выполнения хоть какого-нибудь поиска, файловая система резко снижает производительность.

Примером хранилища, разработанного специально для XML, может служить СУБД Lore [7]. Такие хранилища используют свои собственные методы управления дисковой памятью, хранения данных и.т.п. Возможно, этот подход является наилучшим, однако его реализация, пригодная для промышленного использования, очевидно, потребует большого количества времени. В настоящее время подобные системы остаются на уровне экспериментальных прототипов.

Реляционные и объектно-реляционные СУБД уже в настоящее время имеют мощные средства управления памятью, транзакциями, репликацией данных, не говоря уже об индексировании и поиске информации. Однако отображение полуструктурированной модели в реляционную достаточно затруднено. Этой проблеме было посвящено большое количество исследований, в которых были предложены разнообразные способы отображения XML в таблицы ([8]). Однако ясно, что не существует идеальной реляционной схемы, одинаково хорошо подходящей для хранения произвольных XML документов.

Сравнение производительности описанных выше подходов говорит о том, что реляционная СУБД, даже при использовании не самого эффективного отображения, не сильно отстает в производительности от объектных баз данных. Это дает основание надеяться на то, что реляционная БД, схема которой оптимизирована с учетом особенностей хранимых данных и выполняемых запросов, будет способна конкурировать с объектными СУБД и специальными хранилищами, предоставляя также свои дополнительные возможности по обработке транзакций, масштабируемости и т.д.

2 ПОСТАНОВКА ЗАДАЧИ

2.1 Определения

В этой работе будут использоваться следующие термины:

Фиксированный массив XML данных: набор XML документов, изменения в котором происходят редко и не меняют его количественных характеристик, таких как частота включения какого-либо XML элемента в массив.

Метка: общее обозначение для имен элементов и атрибутов документа.

Перекрестная ссылка: атрибут типа `IDREF`.

(Прямые) потомки элемента: все атрибуты этого элемента, все вложенные элементы и все текстовые данные.

Родительский элемент данного атрибута или элемента: элемент, прямым потомком которого является данный атрибут или элемент.

Продолжаемый логический путь в XML документе:

- Имя корневого элемента является продолжаемым путем
- Конструкция $P.E$, где P является продолжаемым путем, а E — именем элемента–прямого потомка последнего элемента пути P, тоже является продолжаемым путем.
- Конструкция $P.L$, где P является продолжаемым путем, а L — именем перекрестной ссылки–прямого потомка последнего элемента пути P, тоже является продолжаемым путем.

Непродолжаемый логический путь в XML документе: конструкция $P.@A$, где P является продолжаемым путем, а A — именем атрибута–прямого потомка последнего элемента пути P.

(Логический) путь в XML документе: любой продолжаемый или непродолжаемый путь.

(Физический) экземпляр (логического) пути P: последовательность XML элементов, имена и отношения "родитель-потомок" которых удовлетворяют записи логического пути P.

Восстановление экземпляров пути P: процесс отыскания всех экземпляров пути P в массиве данных.

Экземпляр метки L: элемент или атрибут, имя которого совпадает с меткой.

2.2 Входные данные

Входными данными задачи является массив XML данных и набор запросов, выполняемых над этим массивом. Каждый запрос имеет *тело*, и два параметра: *интенсивность* и *относительный приоритет*. Под интенсивностью запроса понимается интенсивность в смысле Пуассоновского процесса, то есть среднее число поступающих за единицу времени dt экземпляров запросов. Относительные приоритеты определяют правило выбора следующего запроса, если одновременно несколько запросов ожидают выполнения. Мы предполагаем, что относительный приоритет является вещественным числом между 0 и 1.

2.3 Цель задачи

Формально целью является минимизация некоторой функции стоимости, оценивающей качество схемы БД в терминах эффективности выполнения запросов в совокупности. Требования к функции стоимости следующие: она должна принимать во внимание структуру БД, интенсивности запросов и их относительные приоритеты. Можно построить разные варианты этой функции, и один из них будет дан в разделе "Способ сравнения реляционных схем хранения XML данных".

3 СПОСОБ ПРЕДСТАВЛЕНИЯ XML ДОКУМЕНТА В РЕЛЯЦИОННОЙ БД

3.1 Графовая модель

В качестве модели для XML данных удобно брать графовую модель, различные варианты которой были описаны в литературе ([9]). В этом разделе мы коротко опишем используемую нами графовую модель.

Вершинами графа являются элементы и атрибуты XML документа. Дуги в графе представляют отношение "элемент-потомок" и помечены метками по следующему правилу: если элемент (атрибут) E_1 является потомком элемента E_2 то меткой дуги из E_2 в E_1 является название элемента (атрибута) E_1.

Вершины, соответствующие атрибутам, являются листьями графа и содержат в себе значение атрибута. Несколько иначе представлены перекрестные ссылки. Для каждой перекрестной ссылки A вместо отдельной вершины в графе создается дуга из родительского элемента для A в родительский элемент соответствующего атрибута типа `ID`. Дуга помечена именем перекрестной ссылки A.

Помимо вложенных элементов и атрибутов, элементы XML могут содержать текстовые данные. Для каждого куска текстовых данных создается вершина в графе и входящая в нее дуга из соответствующего родительского элемента. Дуге присваивается какое-нибудь фиктивное имя. Таким образом, текстовые данные можно рассматривать как атрибуты содержащего их элемента, имеющие фиктивные имена.

Дополнив граф фиктивным корнем, мы получаем возможность представить несколько XML документов в одном графе.

Наконец, в целях восстановления XML документа из графа, необходимо поддерживать очередность вложенных элементов и атрибутов в графе. Таким образом, в графовой модели выходящие из вершины дуги считаются упорядоченными.

3.2 Базовая схема

Для хранения любого XML документа достаточно двух описанных ниже реляционных таблиц. Такое представление мы будем называть *базовым*.

В таблице `ELEMLINKS` хранятся экземпляры всех найденных в документе элементов и перекрестных ссылок. В поле `object_id` хранится уникальный идентификатор элемента. Перекрестные ссылки уникальных идентификаторов не имеют, и для них в поле `object_id` хранится идентификатор элемента, содержащего соответствующий атрибут типа ID. Как для элементов, так и для ссылок, в поле `name` хранится их имя, в поле `parent_id` — идентификатор родительского элемента, а в поле `ord_num` — порядковый номер данного элемента или ссылки среди потомков родительского элемента. Для ссылок в поле `target_ord_num` хранится порядковый номер соответствующего атрибута типа ID среди потомков его родительского элемента, а для элементов значение этого поля равно `NULL`.

ELEMLINKS	
parent_id	NUMERIC
object_id	NUMERIC
name	VARCHAR
ord_num	INTEGER
target_ord_num	INTEGER

В таблице `ATTRIBUTES` хранятся экземпляры всех встречаемых в документе атрибутов, исключая перекрестные ссылки. Поля `parent_id`, `name` и `ord_num` имеют тот же смысл, что и одноименные поля в таблице `ELEMLINKS`. В полях `str_value` и `num_value` хранятся, соответственно, символьное и численное значения атрибута. Нужно сделать замечание о том, что в текущей спецификации XML [1] не существует числового типа для значений атрибутов. По существу, все атрибуты имеют строковый тип. Тем не менее, численное значение необходимо для корректного выполнения арифметических операций и операций сравнения. Если в дальнейшем в спецификации XML появятся какие-либо новые типы значений атрибутов (например, даты), то их можно будет учесть добавлением колонок в таблицу `ATTRIBUTES`. Булевское поле `is_string` используется для построения битовых индексов по атрибутам с символьным значением.

ATTRIBUTES	
parent_id	NUMERIC
name	VARCHAR
ord_num	INTEGER
num_value	NUMERIC
str_value	VARCHAR
is_string	SMALLINT

4 ОПТИМИЗАЦИЯ РЕЛЯЦИОННОГО ПРЕДСТАВЛЕНИЯ

Несмотря на то, что базовое представление позволяет хранить любой XML документ, такое представление крайне неэффективно. Поскольку для выполнения запроса нужно восстановить экземпляры всех используемых в нем логических путей, а базовое представление фактически отображает отношение “родитель-потомок” для одной пары, то для каждого используемого в запросе логического пути из N меток потребуется выполнить $N-1$ соединение таблицы `ELEMLINKS` с самой собой или с таблицей `ATTRIBUTES`. Учитывая возможно большой объем массива данных, это требует больших временных ресурсов.

4.1 Способы оптимизации реляционной схемы

Оптимизацию представления массива данных можно проводить в нескольких направлениях:

- Уменьшение размера основных таблиц `ELEMLINKS` и `ATTRIBUTES`
- Построение навигационных таблиц
- Выделение типизированных частей схемы

Далее мы рассмотрим все эти направления.

4.1.1 *Сокращение размеров таблиц*

Размер основных таблиц можно уменьшить путем *выделения* часто встречаемых в документе меток в отдельные таблицы. В процессе выделения метки создается новая таблица, которой дается имя выделяемой метки с префиксом, указывающим, является ли эта метка атрибутом или элементом. Последнее необходимо, так как могут встретиться атрибуты и элементы с одинаковыми именами. Новая таблица содержит все поля соответствующей основной таблицы, за исключением поля `name`. После этого экземпляры всех меток из соответствующей основной таблицы, имеющие имя выделяемой метки, перемещаются в новую таблицу. Этот подход не сокращает количества соединений, требуемых для восстановления экземпляров путей, но сокращает время, затрачиваемое на выполнение соединений за счет уменьшения количества строк и отсутствия необходимости поиска выделенных меток в основных таблицах.

4.1.2 *Навигационные таблицы*

Используя внутренние идентификаторы элементов, *экземпляры* логических путей можно представить в виде последовательностей идентификаторов составляющих их элементов. Для путей, часто используемых в запросах, можно построить вспомогательные таблицы, называемые *навигационными*, в которых хранятся экземпляры путей. Для каждого логического пути создается своя навигационная таблица, хранящая его экземпляры. Такие таблицы позволяют избежать соединений при восстановлении соответствующих путей.

4.1.3 *Выделение типизированных частей*

Вполне возможно, что исходный набор XML данных изначально имел какую-то более или менее определенную структуру. Это возможно, например, в том случае, если он был получен с веб сайта, содержание которого генерируется из реляционной или объектно-ориентированной базы данных. В таких наборах данных мы имеем шансы найти структурированные (типизированные) части. Разработаны различные методы выявления структуры в XML документах: одни выполняют поиск *строго типизированных частей*, содержащие объекты с единой структурой [10], другие ищут *слабо типизированные части*, содержащие объекты, имеющие в целом схожую структуру, но, возможно, отличающиеся в небольших деталях [11].

Для объектов, принадлежащих одному типу, имеет смысл создавать таблицы, содержащие в себе все поля этого типа. Такой подход позволяет избежать лишних соединений при выборке нескольких полей объектов этого типа.

Помимо уменьшения числа соединений, выделение типов позволяет рассматривать информацию в существующих реляционных базах данных как массив XML данных со строгой типизацией и таким образом осуществить в случае надобности переход от реляционной к XML модели без больших затрат.

5 СПОСОБ СРАВНЕНИЯ СХЕМ

Формальной целью задачи является минимизация некоторой функции, оценивающей качество реляционной схемы. В качестве таковой функции мы предлагаем избрать одну из характеристик *системы массового обслуживания (СМО)*, построенной на основе входного набора запросов и оптимизатора реляционной СУБД. В частности, мы будем рассматривать *среднее время пребывания заявки в очереди*. Выбор именно этой характеристики обусловлен тем, что пользователи, как правило, заинтересованы не в минимизации размера очереди или каких то других характеристик СМО, а в том, чтобы их запросы обрабатывались как можно меньшее время. Уменьшая среднее время ожидания запросов в очереди, мы проводим более или менее справедливую политику.

5.1 Построение системы массового обслуживания

Для построения СМО нужно задать входящие потоки требований, обслуживающие приборы и дисциплину обслуживания [12].

Входящими потоками требований в нашей системе будут потоки экземпляров запросов. Каждый запрос, указанный во входном наборе $Q := \{q_r\}_{r=1}^{k}$ запросов нашей задачи, образует один входящий пуассоновский поток с интенсивностью λ_r, равной интенсивности запроса. Как известно, несколько входящих пуассоновских потоков можно объединить в один поток, который также будет пуассоновским [12] с суммарной интенсивностью $\lambda = \sum_{r=1}^{k} \lambda_r$. Однако требования в этом потоке не будут одинаковыми, поскольку у запросов разные относительные приоритеты. На это мы обратим внимание позже.

Обслуживающим прибором в СМО будет процессор запросов реляционной СУБД. Как правило, современные СУБД позволяют получить время, потраченное на выполнение запроса. Этой возможностью мы воспользуемся для того, чтобы получить среднее время e_r выполнения каждого запроса , полученное после достаточно большого количества испытаний, которое примем за *точное время обслуживания требований данного типа*. Таким образом, мы получаем СМО с одним пуассоновским входным потоком, требования в котором разделяются по типам и для каждого типа известно время обслуживания требований и их относительный приоритет.

Для среднего времени $\overline{\omega}$ пребывания требования в очереди известна общая формула:

$$\overline{\omega} = \int_0^\infty \omega dF(\omega), \tag{1}$$

где $F(\omega)$ есть функция распределения времени ожидания требования в очереди.

Обозначим за $K(r)$ функцию распределения вероятности того, что прибывающий в систему запрос имеет приоритет ниже r, то есть $K(r) = P\{R \le r\}$, где R — приоритет запроса. С учетом этой функции, формула (1) перепишется в виде:

$$\overline{\omega} = \sum_{r=1}^{k} \overline{\omega_r} dK(r) = \sum_{r=1}^{k} \overline{\omega_r} \frac{\lambda_r}{\lambda} \tag{2}$$

где $\overline{\omega_r}$ — среднее время ожидания требований с приоритетом r, а $dK(r)$ — вероятность того, что прибывающее в систему требование имеет приоритет r.

Можно показать [12], что

$$\overline{\omega_r} = \frac{\frac{\lambda}{2}\left(\sigma^2 + \frac{1}{\mu^2}\right)}{\left(1 - \Psi_{r^-}\right)\left(1 - \Psi_{r^+}\right)}, \qquad (3)$$

где

$\mu = \sum_{r=1}^{k} \mu_r dK(r)$ — математическое ожидание интенсивности обслуживания требований в совокупности

$\mu_r = \frac{1}{e_r}$ — интенсивность обслуживания требований с приоритетом r

$\sigma^2 = \sum_{r=1}^{k} {\mu_r}^2 dK(r) - \mu^2$ — дисперсия интенсивности обслуживания требований в совокупности

$\Psi_r = \sum_{n=1}^{r} \frac{\lambda dK(n)}{\mu_n}$ — показатель насыщенности системы с входящим потоком требований с приоритетом меньше или равным r

r^- — наибольший приоритет $r' : r' < r$, или $-\infty$, если такового нет

r^+ — наименьший приоритет $r' : r' > r$, или ∞, если такового нет

$\Psi_{-\infty} = 0$

$\Psi_{\infty} = \Psi = \frac{\lambda}{\mu}$

и тогда формула (2) перепишется в виде:

$$\overline{\omega} = \frac{1}{2}\sum_{r=1}^{k}\left(\sigma^2 + \frac{1}{\mu^2}\right)\frac{\lambda_r}{\left(1 - \Psi_{r^-}\right)\left(1 - \Psi_{r^+}\right)}. \qquad (4)$$

Формула (4) и является функцией стоимости в нашей задаче. Положим $\Omega(S) = \frac{1}{\overline{\omega}}$, где S — схема БД и $\overline{\omega}$ вычислено для этой схемы. Функцию $\Omega(S)$ назовем *эффективностью* схемы S.

6 НАСТРОЙКА БАЗЫ ДАННЫХ И ОПТИМИЗАЦИЯ ЗАПРОСОВ

Все вышеописанные методы позволяют ускорить выполнение запросов над XML данными. Однако иногда они могут быть неприменимы (например, выделение типов не даст результатов, если в XML документе нет типизованных частей) или приводят к большим накладным расходам (например, создание навигационных таблиц для всех встречающихся в документе путей приведет к большим расходам на поддержание вспомогательных структур). Очевидно, не существует универсального решения, значительно увеличивающего эффективность выполнения любых запросов над любым набором данных. Именно поэтому появилась исходная задача: оптимизировать представление конкретного набора данных для конкретного набора запросов.

Процесс оптимизации проходит несколько шагов. На первом шаге анализируется набор данных и из него выделяются типизированные части. В этом могут помочь DTD. Подсчитывается также статистика того, насколько часто та или иная метка встречается в документе вообще и в типизированных частях в частности.

На втором шаге анализируется набор запросов и выясняется, насколько часто и с каким приоритетом понадобится доступ к экземплярам той или иной метки или пути документа.

На третьем шаге выявляются и отбрасываются заведомо неприемлемые варианты изменения базовой схемы. Так, например, выделение отдельной таблицы для метки, экземпляры которой встречаются очень редко и практически не используются в запросах, скорее всего не приведет к повышению производительности.

Далее мы начинаем последовательно изменять базовую схему, принимая во внимание полученную на первых двух шагах информацию. Сначала проводится общая настройка базы данных: создаются типизованные таблицы, выделяются часто встречаемые метки, создаются индексы. Потом начинается оптимизация с учетом набора запросов: для активно используемых путей создаются навигационные таблицы, возможно выделяются еще некоторые метки. Полученные схемы сравниваются между собой по эффективности и постепенно мы получаем схему, не являющуюся, возможно, оптимальной, но значительно повышающую эффективность выполнения запросов.

7 РЕАЛИЗАЦИЯ ПРОТОТИПА

Для практических экспериментов нами разрабатывается прототип, предоставляющий возможность выполнения запросов над хранимым в реляционной БД массивом XML данных и возможность оптимизации схемы БД одним из описанных способов (в настоящее время реализовано только выделение меток). В качестве реляционной СУБД используется Microsoft SQL Server. Web сервер и middleware слой, осуществляющий преобразование запросов с Lorel на SQL, работают на RedHat Linux 6.2

8 ЗАКЛЮЧЕНИЕ

Проводимые нами теоретические и практические исследования дают надежду на то, что использование реляционных СУБД для хранения XML данных достаточно перспективно.

Список литературы

[1] Extensible Markup Language
http://www.w3c.org/XML/Overview.html

[2] Extensible Stylesheet Language
http://www.w3.org/TR/xsl/

[3] The W3C Query Languages Workshop
http://www.w3c.org/TandS/QL/QL98/Overview.html

[4] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, Dan Suciu
XML-QL: A Query Language for XML
http://www.w3c.org/TR/NOTE-xml-ql

[5] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom,Janet L. Wiener
The Lorel Query Language for Semistructured Data
Journal on Digital Libraries, volume 1 number 1, 1996.

[6] Осипов М.В., Мачульский О.Л., Калиниченко Л.А.
Отображение модели данных XML в объектную модель языка СИНТЕЗ
Электронные библиотеки, Санкт-Петербург, 1999

[7] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, Jennifer Widom
Lore: A Database Management System for Semistructured Data.
SIGMOD Record 26(3): 54-66 (1997)

[8] Daniela Florescu, Donald Kossmann
A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database
Rapport de Recherche No. 3680 INRIA, Rocquencourt, France, May 1999

[9] Roy Goldman, Jason McHugh, Jennifer Widom
From Semistructured Data to XML: Migrating the Lore Data Model and Query Language
WebDB Workshop, Philadelphia, USA, 1999

[10] P.Buneman, S.Davidson, M.Fernandez, D.Suciu.
Adding stucture to unstructured data
Proceedings of ICDT, p.336-350, Delphi, Greece, 1997.

[11] Svetlozar Nestorov, Serge Abiteboul, Rajeev Motwani
Extracting Schema from Semistructured Data.
SIGMOD Conference 1998: 295-306.

[12] А. Кофман, Р.Крюон
Массовое обслуживание. Теория и приложения.
Издательство “Мир”, Москва, 1965.