SXTM: Высокопроизводительный менеджер управления XML-транзакциями*

Петр Плешачков
peter@ispras.ru

Аннотация. В настоящее время XML повсеместно используется как формат для описания слабоструктурированных данных. Для поддержки приложений, нуждающихся в обработке больших объемов XML-данных, ведущие производители реляционных систем управления базами данных (РСУБД) включили в свои системы дополнительные компоненты (XML-расширения), в которых реализуются стандарты XML.

Серия экспериментов, проведенных с существующими XML-расширениями, выявила низкую производительность РСУБД при выполнении большого количества параллельных запросов на выборку и изменения частей XML-документов. Анализ данной проблемы показал, что существующие механизмы блокировок в РСУБД приводят к большому количеству псевдоконфликтов между XML-транзакциями, вследствие чего параллелизм XML-транзакций серьезно ограничивается.

В статье предлагается новый семантический метод управления XML-транзакциями, который реализуется над РСУБД в виде дополнительного компонента – SXTM (Semantic XML Transaction Manager). Ключевым компонентом SXTM является планировщик XML-транзакций, основанный на протоколе XDGL.

Семантический менеджер транзакций SXTM был реализован как надстройка для Microsoft SQL Server 2005. Экспериментальная оценка SXTM показала существенное уменьшение времени выполнения XML-транзакций при большом количестве параллельных запросов на выборку и модификацию.

1. Введение

Расширяемый язык разметки XML [25] в настоящее время стал стандартным средством моделирования слабоструктурированных данных. Кроме того, язык XML широко используется в приложениях, в которых определяющими являются требования надежности и согласованности данных. Пользователи таких приложений, как правило, вводят запросы[1] на выборку или изменения в параллельном режиме.

* Работа поддержана грантами РФФИ 04-07-08003 и 05-07-90204.

[1] Далее в этой статье термин *запрос* без указания его типа будет означать запрос на выборку или модификацию данных.

На сегодняшний день в большинстве развитых коммерческих реляционных СУБД (РСУБД), таких как IBM DB2, MS SQL Server 2000/2005, Oracle 10g, встроена поддержка стандартов XML. Эта функциональность реализуется в виде дополнительного компонента – XML-расширения. Помимо этого, во всех вышеуказанных РСУБД имеется эффективная поддержка транзакций. В совокупности эти факторы обуславливают высокую привлекательность РСУБД для обработки XML-запросов в параллельном режиме.

Вышеупомянутые XML-расширения поддерживают XML на основе метода STORED [5]. Основная идея метода заключается в следующем: XML-документ хранится в большом объекте типа BLOB (MS SQL Server 2005) или CLOB (IBM DB2 v 8.2, MS SQL Server 2000), а процессор запросов расширяется поддержкой запросов на выборку и изменения XML-документов (см. рис. 1). Таким образом, XML-документы хранятся в столбцах некоторой таблицы, которую мы будем называть *таблицей документов*(*document table*).

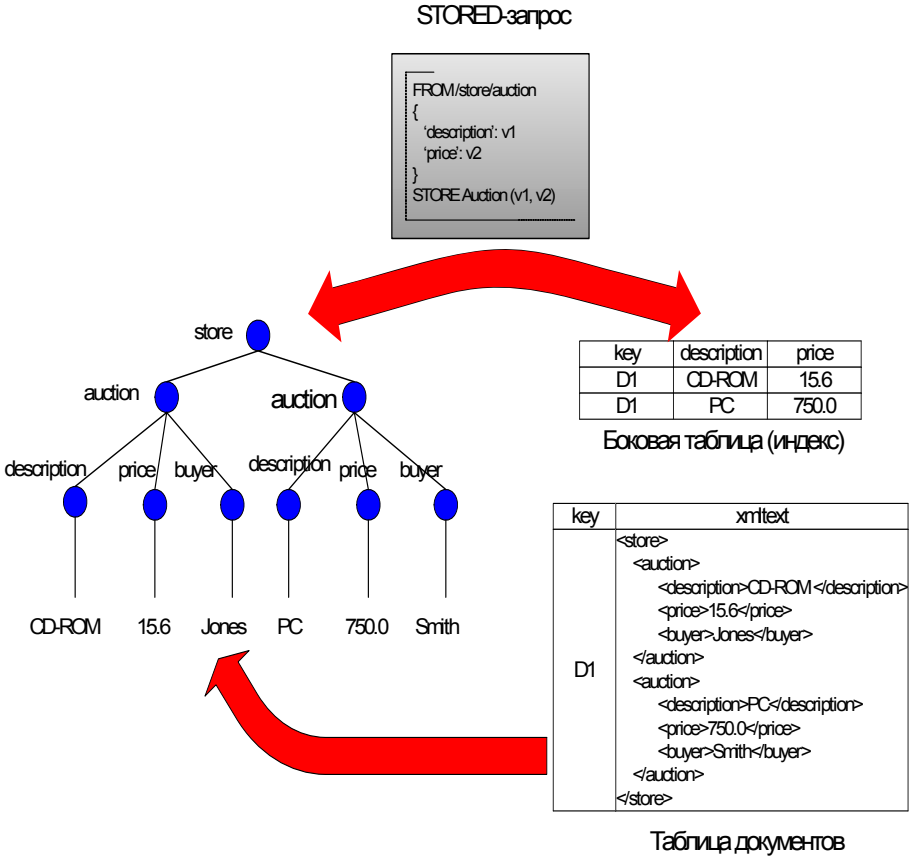

Рис. 1. Метод STORED хранения XML-документов в РСУБД

Для увеличения скорости выборки данных вводятся дополнительные индексы, которые, как правило, реализуются в виде дополнительных таблиц (*side tables*). В дальнейшем мы будем называть эти таблицы *боковыми*. Для задания частей XML-документа, которые должны храниться в боковой таблице, проектировщик базы данных должен определить STORED-запрос. Это позволяет существенно увеличить скорость выполнения XML-запросов на выборку. Кроме того, например, в MS SQL Server 2005 для поддержки доступа к XML существуют специальные типы индексов: путевые индексы, индексы по значениям, индексы по значениям и путям в XML-документе.

При изменении XML-документа необходимо, чтобы боковые таблицы (индексы) и таблицы документов находились в согласованном состоянии. Например, в IBM DB2 v 8.2 согласованность данных поддерживается на основе триггеров: при изменении XML-элемента (или атрибута), который находится как в боковой таблице, так и в таблице документов, автоматически выполняется триггер, который осуществляет удаление старого значения в боковой таблице и вставку нового значения.

В этой работе мы изучаем возможности эффективной обработки большого количества XML-запросов[2] в параллельном режиме с использованием коммерческих XML-расширений.

Проведя ряд экспериментов с XML-расширениями, мы выявили, что производительность системы существенно деградирует при большом количестве XML-транзакций. Дело в том, что изоляция транзакций в РСУБД реализуется на основе механизма блокировок данных, в котором не учитывается ни иерархическая природа XML-данных, ни семантика XML-операций.

Пример 1 (псевдоконфликты XML-транзакций в РСУБД) В качестве примера XML-данных использовался документ XMark [20], который описывает систему аукционов. Рассмотрим две конкурентные транзакции, выполняющие операции над XML-данными. Предположим, что первая транзакция выбирает все элементы *item*, в то время как вторая транзакция изменяет значения всех элементов *price*. Очевидно, что транзакции не должны конфликтовать, поскольку они работают с разными данными. Но планировщик транзакций в РСУБД заблокирует одну из транзакций до окончания второй, если элементы *price* и *item* хранятся в одном документе. Аналогичная ситуация возникнет в случае, когда читающая транзакция выполняется на основе боковых таблиц: изменяющая транзакция заблокирует боковую таблицу на запись с целью поддержки XML-документа и его боковых таблиц в согласованном состоянии и тем самым заблокирует первую транзакцию (или наоборот). Описанная ситуация в литературе [9] известна как псевдоконфликт. Большое количество псевдоконфликтов и приводит к низкой степени параллелизма XML-транзакций.

[2] Этим критерием мы и будем руководствоваться при оценке производительности системы.

В этой статье мы описываем метод, позволяющий устранить отмеченные недостатки. В предлагаемом методе используется двухуровневая модель транзакций [22-23]. В двухуровневой модели транзакций все транзакции второго уровня реализуются в виде набора транзакций первого уровня. В нашем случае XML-транзакция реализуется в виде набора транзакций над РСУБД.

Для реализации второго (семантического) уровня вводится дополнительный менеджер управления транзакциями SXTM (Sematic XML Transaction Manager). В результате на втором уровне планировщик конкурентных транзакций учитывает семантику выполняемых операций при определении конфликта между транзакциями. Транзакция второго уровня реализуется как набор субтранзакций нижнего уровня, которые освобождают установленные ими блокировки при своем завершении. В результате “семантические” блокировки на втором уровне устанавливаются на все время выполнения XML-транзакции, а “грубые” блокировки в РСУБД, устанавливаемые субтранзакциями, освобождаются гораздо раньше. За счет этого производительность системы может существенно повыситься, особенно, если несколько транзакций одновременно читают и изменяют один и тот же документ. Рис. 2 иллюстрирует общую архитектуру системы при использовании двухуровневой (слева) и одноуровневой (справа) моделей управления XML-транзакциями. В дальнейшем мы будем называть транзакции, поддерживаемые РСУБД, DB-транзакциями.

Ключевым компонентом SXTM является новый протокол изоляции XML-транзакций XDGL (XPath-based DataGuide Locking), в котором учитывается семантика XML- операций при определении конфликтов между транзакциями. В этой работе мы приводим только обзор этого протокола, а его формальное описание и обоснование можно найти в работе [16]. Важнейшей отличительной чертой XDGL является его независимость от метода хранения XML- документа. Это достигается за счет того, что в XDGL используется не сам XML-документ, а описывающая схема (DataGuide) [7] этого XML-документа.

При делении XML-транзакции на субтранзакции необходимо сохранить свойство атомарности транзакций, когда все изменения данных, произведенные транзакцией, должны либо полностью присутствовать в БД, либо полностью отсутствовать в БД. В этой работе мы рассматриваем различные способы обеспечения атомарности XML-транзакций при использовании двухуровневой модели транзакций.

Важным дополнительным преимуществом SXTM является то, что DB-транзакции могут запускаться на втором уровне изоляции транзакций (READ COMMITTED) в соответствии со стандартом SQL, поскольку протокол XDGL сам гарантирует сериализацию XML-транзакций. Это позволяет увеличить параллелизм DB-транзакций.

Для экспериментальной оценки, SXTM был реализован в виде дополнительного компонента для системы Microsoft SQL Server 2005. Мы провели оценку производительности системы при использовании SXTM (рис. 2 слева) и

плоской модели транзакций (рис. 2 справа) и выявили существенное повышение производительности при использовании SXTM.

Структура статьи. В разд. 2 описываются используемые языки запросов и изменений XML-данных. В разд. 3 мы описываем общую архитектуру SXTM и обсуждаем ее преимущества. Разд. 4 посвящен обсуждению семантических особенностей XML-операций выборки и модификации, которые влияют на выбор типов блокировок для изоляции XML-операций. В разд. 5 рассматривается новый протокол изоляции XML-транзакций XDGL, приводится несколько примеров, демонстрирующих применение XDGL. Атомарность XML-транзакций обсуждается в разд. 6. Описание проведенных экспериментов с SXTM и анализ полученных результатов приводятся в разд. 7. В разд. 8 и 9 мы приводим обзор родственных работ по теме исследования и делаем заключительные выводы.

2. Языки запросов и изменений XML-документов

2.1. XQuery

В качестве языка запросов мы используем язык XQuery [3]. На сегодняшний день процесс стандартизации XQuery практически завершен, и в скором времени будет выпущена первая официальная версия языка. Большинство компаний, производящих коммерческие РСУБД, заявило о запланированной поддержке XQuery в ближайших версиях своих продуктов.

Неотъемлемой частью XQuery является язык XPath [1], который предназначен для адресации частей XML-документов. Помимо XQuery, язык XPath также широко применяется в другом популярном языке обработке XML документов – XSLT [14].

Основной конструкцией XQuery является FLWR-выражение, обладающее следующим синтаксисом:

```
( FOR $var1 IN expr1 | LET $var2 := expr2 )+
WHERE expr3
RETURN expr4
```

FLWR-выражение состоит из четырех составляющих: разделов FOR, LET, WHERE и RETURN. В разделе FOR определяется переменная $var1, которая последовательно связывается с каждым из элементов последовательности, получаемой в результате вычисления выражения expr1. В разделе LET переменная $var2 связывается со всем результатом вычисления выражения expr2. Допускается несколько разделов FOR или LET. В разделе WHERE значения переменных фильтруются на основе предиката expr3. Наконец, раздел RETURN определяет результат FLWR-выражения: выражение expr4 вычисляется для каждого связывания переменных, для которого результатом вычисления предиката в разделе WHERE секции является «истина». Результатом всего FLWR-выражения является последовательность узлов, полученных после вычисления раздела RETURN.

При помощи FLWR-выражений на XQuery можно выражать операции соединения, трансформации, группировки и т. д. Особую роль играет операция трансформации, поскольку она очень часто используется в Web-приложениях при построении динамических HTML-страниц. В XQuery операция трансформации выражается при помощи конструкторов элемента и атрибута. Примером трансформации является следующий запрос, результатом которого является новое представление книг с названием “Data on the Web” и автором “Serge Abiteboul”:

```
<result>
{
  FOR $book IN //book
  WHERE $book/author = 'Serge Abiteboul' AND
        $book/title = 'Data on the Web'
  RETURN
    <book>
    {
       $book/title,
       $book/author,
       $book/isbn,
       $book/price
    }
    </book>
}
</result>
```

XQuery является функциональным языком, поэтому в нем выражения могут быть вложены друг в друга произвольным образом. Например, в разделе FOR FLWR-выражения может находиться другое FLWR-выражение и т. д. с любым уровнем вложенности. Помимо этого, на XQuery пользователь может определять и использовать свои собственные функции. Также существует достаточно большое количество встроенных стандартных функций [15].

Далее в статье мы будем использовать обозначения *Q* и *LP* для ссылок на XQuery-запросы и путевые выражения XPath соответственно.

2.2. XUpdate

К настоящему времени рабочая группа XQuery в W3C еще не разработала стандартный язык изменения частей XML-документов. В нашей работе мы используем подмножество языка изменения частей XML-документов, который был предложен в статье И. Татаринова [21]. В дальнейшем язык изменений мы будем называть XUpdate. Синтаксис этого языка выглядит следующим образом:

```
update := 'InsertInto' '(' constr1 ',' locpath ')' |
          'InsertBefore' '(' constr2 ',' locpath ')' |
          'InsertAfter' '(' constr2 ',' locpath ')' |
```

```
          'Delete' '(' locpath ')'
          'Rename' '(' locpath, QName ')'
constr1:= 'element' '{' QName '}'  content |
          'attribute' '{' QName '}' content
constr2 := 'element' '{' QName '}'  content
content    := '{' PCDATA '}' | '{' '}'
```

Каждая операция принимает на вход несколько аргументов (*constr1*, *constr2* и *locpath*). Выражения *constr1* и *constr2* определяют новые узлы. Выражение *locpath1* определяет путь адресации узлов в XML-документе, которые являются целевыми узлами операции изменения. Ниже приводится описание каждой операции.

- *InsertInto(constr1, locpath)*: вставляет новый узел (элемент или атрибут) в каждый целевой узел в позицию последнего дочернего узла.
- *InsertBefore(constr2, locpath)*: вставляет новый узел (только элемент) для каждого целевого узла в позицию предшествующего брата.
  InsertAfter(constr2, locpath) вставляет новый узел (только элемент) для каждого целевого узла в позицию последующего брата .
- *Delete(locpath)*: осуществляет глубокое удаление узлов (вместе со всеми потомками), определяемых *locpath*.
- *Rename(locapth, QName)*: присваивает новое имя QName целевым узлам операции.

Далее в статье мы будем использовать обозначения I_I, I_A, I_B, D и RN для ссылок на операции *InsertInto*, *InsertAfter*, *InsertBefore*, *Delete* и *Rename* соответственно. Кроме того, мы будем использовать обозначение I_* для ссылки на произвольную операцию вставки.

3. Общая архитектура SXTM

Транзакции являются ключевым механизмом для обеспечения надежности и согласованности данных в информационных системах, когда существует вероятность одновременного доступа нескольких пользователей к одним и тем же данным на чтение и изменение или вероятность краха системы. Исследователи в области баз данных за последние двадцать лет разработали ряд методов, которые позволяют реализовать транзакции эффективно.

Когда XML используется в приложениях, в которых требования надежности и согласованности данных играют определяющую роль, необходимо, чтобы все операции над базой данных со стороны приложений выполнялись в рамках транзакций. Существуют два способа обеспечения транзакционных требований для XML-приложений.

Первый способ является наиболее простым и заключается в использовании транзакционной функциональности РСУБД. Это позволяет прикладному программисту сгруппировать набор запросов к базе данных, которые должны обладать свойствами атомарности и изолированности, в одну единую

транзакцию. В результате РСУБД обеспечивает транзакционные свойства для целой группы запросов. Рис. 2 (справа) иллюстрирует этот подход, который мы будем в дальнейшем называть *плоской моделью транзакций*. Этот подход не требует дополнительных усилий для реализации транзакционных свойств XML-приложений, поскольку вся транзакционная функциональность сосредоточена в РСУБД.

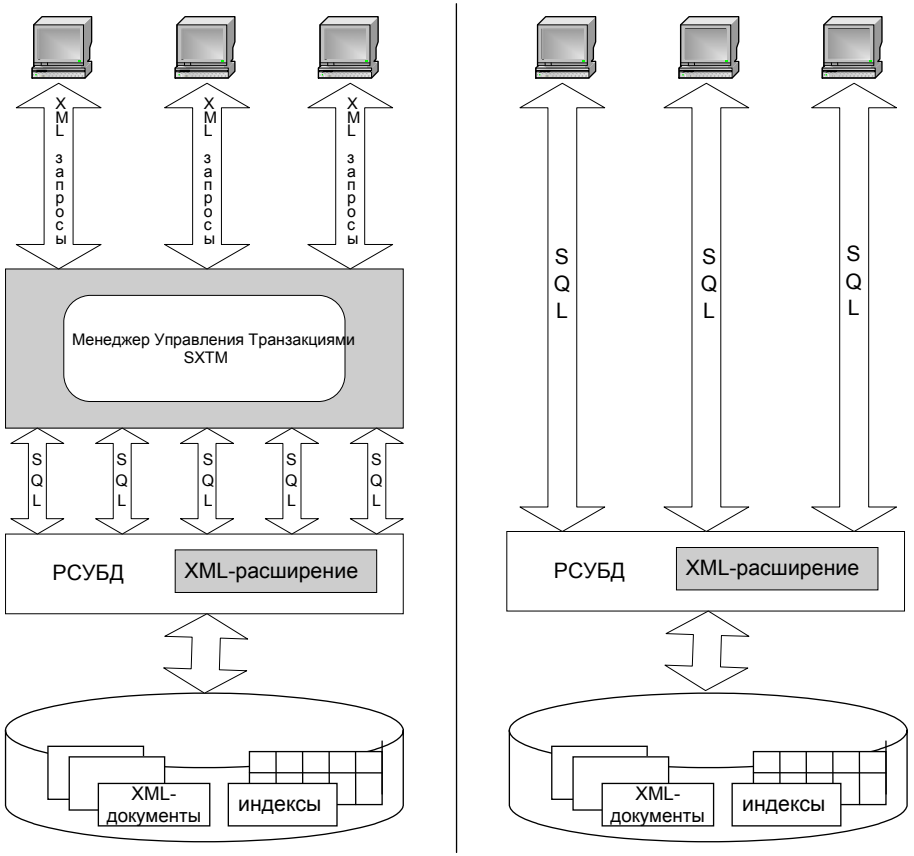

Рис. 2. Двухуровневая (слева) и одноуровневая (справа) модели управления XML транзакциями

Во втором способе используется *двухуровневая модель транзакций* [22]. В этой модели набор запросов пользователя, для которых требуется наличие свойств атомарности и изолированности, формирует *глобальную транзакцию* (такие транзакции мы будем называть XML-транзакциями). Дополнительный менеджер транзакций, построенный над РСУБД, декомпозирует исходную глобальную транзакцию на набор субтранзакций к РСУБД, или DB-транзакций.

При использовании второго способа мы получаем очень важное преимущество. На втором уровне менеджер XML-транзакций при принятии решения о конфликте между транзакциями может учитывать семантику XML-операций и устанавливать “семантические” блокировки до конца транзакции. В свою очередь, DB-транзакция завершается и освобождает “грубые” блокировки гораздо раньше завершения глобальной транзакции. Этот подход иллюстрирует рис. 2 (слева). За счет этого можно увеличить параллелизм конкурентных транзакций.

Для сравнения двух вышеописанных подходов в контексте их применения для XML-транзакций мы разработали и реализовали SXTM – дополнительный семантический менеджер управления XML-транзакциями. В SXTM глобальными транзакциями являются XML-транзакции, которые состоят из наборов XML-запросов. SXTM перехватывает все запросы клиента и обращает их в субтранзакции к РСУБД. DB-транзакции могут запускаться на пониженных уровнях изоляции за счет того, что SXTM гарантирует сериализацию XML-транзакций. Эта оптимизация, в совокупности с быстрым освобождением блокировок в РСУБД, приводит к существенному повышению параллелизма XML-операций.

4. Семантические особенности операций XQuery и XUpdate

В этом разделе мы обсуждаем семантические особенности XQuery и XUpdate операций, которые необходимо учитывать при разработке протокола изоляции XML-транзакций. Эти особенности влияют на выбор типов синхронизационных блокировок, требуемых в протоколе изоляции XML-транзакций, на котором основывается SXTM. Семантические особенности операций состоят в следующем:

- *LP*. Путевое выражение (*locpath*) является базовой конструкцией, используемой как в XQuery, так и в XUpdate. Поэтому для получения наилучшей гранулированности блокировок в протоколе должна максимально учитываться семантика *locpath*-выражений. *Locpath* состоит из последовательности шагов, на каждом из которых требуются различные типы блокировок. Например, на последнем шаге выбираются узлы, содержимое которых сериализуется [3] и передается пользователю. Поэтому для таких узлов необходимо установить блокировку на чтение для поддеревьев (*глубокую* блокировку) с корнями в этих узлах. С другой стороны, для узлов, выбираемых на промежуточных шагах, достаточно установить блокировку на чтение только самого узла (*неглубокую* блокировку). В дополнение к этому, необходимо гарантировать, что не удерживаются глубокие блокировки на запись на поддеревьях, включающих узлы или поддеревья, которые мы хотим заблокировать на чтение. Эта проблема решается при помощи блокировок *намерения* (*intention*), которые выставляются на каждого предка рассматриваемого узла до установки блокировки на

сам узел. Если в шаге есть предикат с операциями (например =, <, >), для которых нужно производить атомизацию [3], то атомизируемые узлы также должны подвергаться глубокой блокировке на чтение.

- *Q*. Запросы на XQuery обращаются к данным при помощи путевых выражений. Поэтому блокировки должны выставляться на основе путевых выражений (их семантические особенности обсуждались в предыдущем пункте). Но при этом они не должны рассматриваться изолированно от всего выражения XQuery, поскольку иначе теряется дополнительная семантика, которая позволяет улучшить гранулированность блокировок. Дело в том, что в выражении XQuery не для всех узлов, выбираемых *locpath*, требуется блокировать все поддерево. Например, в выражении *fn:count(//person)* необходимо гарантировать неизменяемость количества узлов *person*, но при этом потомки элементов *person* могут изменяться произвольным образом другими транзакциями. В качестве еще одного примера рассмотрим FLWR-выражение: *for $v in //person return $a/name*. Здесь блокировки не обязаны обеспечивать неизменяемость всех поддеревьев *person*. Вместо этого достаточно гарантировать, что сами элементы *person* не будут никак изменены другими транзакциями (при этом потомки могут изменяться), и что не будут изменены поддеревья *name*.
- I_I. Операция вставки нового узла в позицию последнего дочернего узла каждого целевого узла вызывается с двумя аргументами: выражение *locpath* специфицирует целевые узлы, а выражение *constr2* определяет новый узел. Блокировки для *locpath* мы обсуждали выше. Для новых узлов необходимо установить неглубокую блокировку на запись. Например, для нового узла *<person/>* требуется установить монопольную блокировку только на узел *person*, но при этом не нужно блокировать узлы, находящиеся ниже в иерархии. Для узла *<name>John</name>* имеются два варианта блокировок: можно установить монопольную неглубокую блокировку на узлы *name* и *text* (ребенок *name*) либо монопольную глубокую блокировку на узел *name*. Дополнительной семантической особенностью операции I_I является следующее: при вставке нового узла в позицию последнего дочернего узла целевого узла необходимо гарантировать, что другие транзакции не будут вставлять другие узлы на это же место. Очевидно, что можно установить глубокую блокировку (на чтение) на узел, в который вставляется новый узел, и тем самым гарантировать, что в него не будут вставлены новые узлы, но это слишком грубое решение. Здесь лучше ввести новый тип блокировок: неглубокую блокировку на чтение, которая дополнительно предотвращает вставку новых узлы *в* блокируемый узел. Аналогично для операций I_A и I_B требуется ввести неглубокие блокировки на чтение, которые дополнительно предотвращают вставку новых узлов *после* или *перед* блокируемым узлом соответственно. В остальном к операциям I_A и I_B применимы те же рассуждения, что и к операции I_I.

- *D*. Операция *D* выполняет глубокое удаление целевых узлов. Поэтому необходимо запретить чтение или модификацию удаляемых поддеревьев другими транзакциями. Следовательно, на удаляемые узлы необходимо установить монопольную глубокую блокировку.
- *RN*. Операция переименования присваивает новое имя узлу, но при этом не изменяет его потомков. Поэтому нужно установить монопольную неглубокую блокировку на изменяемый узел, а также на узел с новым именем. При этом не требуется устанавливать блокировки на потомков переименовываемого узла. В результате, например, транзакции *Rename (/doc/person, person2)* и */doc//name* не будут конфликтовать по блокировкам.

5. Протокол XDGL

В SXTM используется новый протокол XDGL [17], основанный на двухфазном протоколе синхронизационных блокировок (2PL) [6].

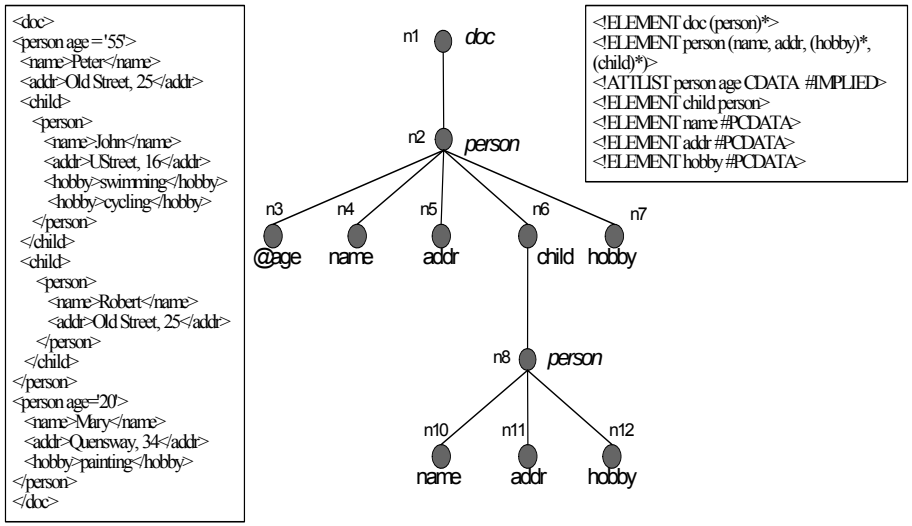

Рис. 3: XML-документ GTree, его описывающая и предписывающая схемы

При определении блокировок необходимо учитывать, что для SXTM узлы XML-документа физически недоступны. Поэтому в SXTM блокировки устанавливаются не на узлах самого XML-документа, а на узлах дополнительной структуры, которая в литературе [7] называется описывающей схемой XML-документа. Описывающая схема[3] представляет собой структурное обобщение XML-документа. В схеме представлены все пути в

[3] Далее в статье неуточняемый термин *схема* будет обозначать описывающую схему XML-документа.

XML-документе, и только они. Поскольку одному пути в XML-документе, как правило, соответствуют несколько узлов, обычно размер описывающей схемы гораздо меньше размера соответствующего XML-документа. Например, схема гигабайтного XML-документа XMark [20] занимает всего 10 килобайт. Поэтому SXTM может полностью кэшировать описывающую схему в основной памяти, что существенно сокращает накладные расходы, связанные с установкой блокировок на схеме. Другой особенностью схемы является ее относительно редкая изменчивость. Последнее обстоятельство существенно облегчает задачу поддержку согласованности схемы и соответствующего ей XML-документа.

В XDGL поддерживаются три вида блокировок на схеме XML-документа: (1) структурные блокировки, (2) предикатные блокировки и (3) логические блокировки. Следующие подразделы посвящены описанию этих типов блокировок.

5.1. Структурные блокировки

В XDGL поддерживаются следующие виды структурных блокировок:

- Блокировка *SI* (*shared into*). Эта блокировка используется в операции I_i. Блокировка устанавливается на узел схемы, который соответствует целевым узлам этой операции. Блокировка *SI* запрещает какие-либо модификации узла, а также вставку других узлов *в* блокируемый узел. Аналогично определяются блокировки *SA* (*shared after*) и *SB* (*shared before*).
- Блокировка *X* (*exclusive*) . Эта блокировка используется в операциях I_* и *RN*. Для операций I_* эта блокировка устанавливается на узел схемы, который соответствует новым (вставляемым) узлам. Для операции *RN* блокировка устанавливается на переименовываемый узел, а также на узел схемы, соответствующий новому узлу (узлу с новым именем). *X*-блокировка запрещает какие-либо чтения или модификации нового узла операциями *Q*, I_*, *D* и *RN*.
- Блокировка *S* (*shared*). Эта блокировка используется во всех операциях, в которых используются путевые выражения XPath. Например, она устанавливается на узел схемы, соответствующий промежуточным результатам вычисления XPath. Эта блокировка запрещает какие либо изменения узла, включая его переименование. При этом блокировка не накладывает какие либо ограничения на потомков узла.
- Блокировка *ST* (*shared tree*). Эта блокировка используется в операциях *Q*, I_* и *D*. Она устанавливается на узел схемы и неявно блокирует в совместном режиме всех потомков этого узла. Таким образом, эта блокировка предотвращает какие-либо изменения внутри целого поддерева.
- Блокировка *XT* (*exclusive tree*). Эта блокировка используется в операции *D*. Она устанавливается на узел схемы и неявно блокирует в монопольном режиме всех потомков этого узла.

- Блокировка *IS* (*intention shared*). Эту блокировку требуется установить на каждого предка узла, который должен быть заблокирован в одном из разделяемых режимов. *IS*-блокировка гарантирует отсутствие блокировок на поддеревьях на верхних уровнях. Аналогично определяется блокировка *IX* (*intention exclusive*).

Таблица совместимости структурных блокировок показана на рис. 4. Символ '+' на рисунке обозначает совместимость блокировок. Символ 'P' обозначает условную несовместимость типов блокировок, и решение о наличии или отсутствии конфликта должно приниматься на основе предикатов, которые мы обсудим в следующем подразделе.

	Установленная блокировка								
Запрашиваемая блокировка	SI	SA	SB	S	X	ST	XT	IS	IX
SI	P	+	+	+	P	+	P	+	+
SA	+	P	+	+	P	+	P	+	+
SB	+	+	P	+	P	+	P	+	+
S	+	+	+	+	P	+	P	+	+
X	P	P	P	P	P	P	P	+	+
ST	+	+	+	+	P	+	P	+	P
XT	P	P	P	P	P	P	P	P	P
IS	+	+	+	+	+	+	P	+	+
IX	+	+	+	+	+	P	P	+	+

Рис. 4. Матрица совместимости структурных блокировок XDGL

5.2. Предикатные блокировки

Блокировка узла схемы в каком-либо режиме неявно приводит к блокировке в этом режиме всех узлов в XML-документе, которые соответствуют данному узлу схемы. Очевидно, что во многих случаях это будет приводить к блокировке лишних узлов XML-документа. Например, для запроса */price[. = 100]* не нужно блокировать все узлы *price*, а достаточно заблокировать узлы *price*, значение которых равно *100*.

Для решения указанной проблемы мы с каждой структурной блокировкой ассоциируем предикат. Этот предикат накладывает ограничение на значение узла в XML-документе. Таким образом, блокировка в XDGL состоит из двух частей: типа блокировки и предиката. Две блокировки *(lock-type1, pred1)* и *(lock-type2, pred2)* совместимы, если выполняется одно из условий: (1) тип блокировки *lock−type1* совместим с *lock−type2* согласно таблице совместимости, (2) конъюнкция предикатов *pred1* и *pred2* тождественно равна *false*.

Вообще говоря, проверка совместимости предикатов является NP-трудной задачей [12], и поэтому в XDGL мы проверяем на совместимость только простые предикаты (сравнения с константами) для которых существуют полиномиальные алгоритмы проверки на совместимость [18], а остальные предикаты отождествляем с *true*.

5.3. Логические блокировки

В этом подразделе мы описываем логические блокировки, которые в XDGL используются для предотвращения фантомов.

Прежде всего рассмотрим, в каких ситуациях могут появляться фантомы. Предположим, что транзакция T_1 читает все атрибуты *age* в XML-документе *GTree* (см. рис. 3), используя запрос *//@age*. В это же время транзакция T_2 вставляет новый атрибут *age,* используя операцию: *InsertInto(attribute age {30}, /doc/person/child/person)*. При повторном чтении всех атрибутов *age* транзакция T_1 прочитает новый атрибут-фантом *age*, вставленный второй транзакцией. В общем случае, фантомы могут появляться в случае, если (1) операция модификации расширяет схему (добавляет новый путь), (2) эта модификация приводит к изменению целевых узлов предыдущих операций чтения или модификации других транзакций.

Чтобы избежать появление узлов-фантомов, мы вводим две дополнительные блокировки: *L* и *IN*. *L* (*logical*) блокировка устанавливается на узел схемы для предотвращения появления фантомов в поддереве, соответствующем этому узлу. При требовании установки *L*-блокировки специфицируется набор *свойств* для узлов в документе. Свойства узлов документа выражаются при помощи условий на имя узла и его значение. *L*-блокировка запрещает вставку новых узлов в поддерево, если вставка приводит к расширению схемы, и вставляется узел со свойствами, указанными в *L* блокировке. В свою очередь, транзакция, которая расширяет схему, должна установить блокировку *IN* (*insert new node*) для каждого предка вставляемого узла. При требовании установки *IN*- блокировки также специфицируется набор свойств вставляемого узла.

Ниже указаны все возможные комбинации для свойств *L*-блокировки (*relop* – это операция сравнения):

(1) *node-name='name1'* (например, для запроса *//person*);
(2) *node-name='name1', node-value relop 'val1'* (например, для запроса *//name[.≠'John']*);
(3) *node-name='name1', child-name='name2', child-value relop 'val1'* (например, для запроса *//person[name ≠ 'John']*).

Для проверки того, что свойства вставляемого узла не конфликтуют со свойствами *L*-блокировки, установленной другой транзакцией, в *IN*-блокировке необходимо указать три свойства вставляемого узла: *new-node-parent-name*, new-node-name, *new-node-value*. *L*- и *IN*-блокировки не совместимы в каждом из следующих случаев:

(1) *new-node-name IN*-блокировки равно *node-name L*-блокировки, и при этом *L* блокировка не содержит других свойств (случай (1)).
(2) свойства *new-node-name* и *new-node-value IN*-блокировки соответствуют значениям свойств *L*-блокировки (которых должно быть два); иными словами, *node-name=new-node-name* и *new-node-value relop 'val1'* ≠ *#f* (случай (2)).
(3) все три свойства *IN*-блокировки соответствуют трем свойствам *L*-блокировки; иными словами, *node-name=new-node-parent-name*, *child-name=new-node-name* и *new-node-value relop 'val1'* ≠ *#f* (случай (3)).

Заметим, что если в *L*-блокировке вместо имени узла указано '*', то это имя узла считается соответствующим произвольному имени.

5.4. XDGL планировщик

В этом подразделе мы опишем алгоритм, в соответствии с которым действует XDGL для операции $a(op_i, t)$. (t обозначает идентификатор XML-транзакции).

(1) Вычислить множество всех путей в схеме, приводящих к данным, которые читаются или изменяются операцией $a(op_i, t_j)$. Назовем это множество *DP* (*data-path-set*).
(2) Вычислить множество узлов n_j схемы, которые соответствуют конечным узлам путей из *DP*. В случае, если на узел налагается предикат p_j, ассоциировать его с этим узлом. Обозначим полученное множество через $NP=\{(n_j, p_j)\}$ (*node-predicate-set*).
(3) Вычислить множество всех узлов n_j схемы (и их свойства $properties_j$), в поддеревьях которых могут появиться фантомы. Обозначим полученное множество через $PH=\{(n_j, properties_j)\}$ (*phantom-set*).
(4) Если op_i расширяет схему, то вычислить свойство $properties_i$ нового узла.
(5) Вычислить множество узлов схемы, которые прочитываются в результате выполнение операций *locpath* в $a(op_i, t_j)$ (это множество включает в себя узлы, прочитанные *locpath* на промежуточных шагах). Назовем это множество *RS* (*read-set*).
(6) Для каждого $n_j \in \{RS \setminus DP\}$ установить блокировки (S, *#t*) и (IS, *#t*) на n_j и всех предках n_j соответственно.
(7) По следующим правилам установить для op_i структурные блокировки:
- Пусть op_i – это операция Q. Для каждого $n_j \in NP$:
  (a) если n_j соответствует узлу, атомизируемому при выполнении op_i, то установить на n_j блокировку (ST, p_j), иначе установить на n_j блокировку (S, p_j);
  (b) для каждого предка n_j установить блокировку (IS, *#t*).
- Пусть op_i – это операция I_I. Для каждого $n_j \in NP$:
  (a) если n_j соответствует целевым узлам I_I, то установить блокировку (SI, p_j) на узел n_j и блокировку (IS, *#t*) на всех его предков;
  (b) если n_j соответствует дополнительным ветвям *locpath* (необходимым для указания предиката) операции I_I, то установить на узел n_j блокировку (ST, p_j) (либо (S, p_j), если для проверки предиката не требуется атомизация узла) и блокировку (IS, *#t*) на его предков;
  (c) если n_j соответствует новому узлу, который вставляется I_I операцией, то установить на узел n_j блокировку (X, p_j) и блокировку (IX, *#t*) на его предков.
- Для операций I_A или I_B. выполнить шаги, аналогичные предыдущему пункту.
- Пусть op_i – это операция *D*. Для каждого $n_j \in NP$:
  (a) если n_j соответствует целевым узлам *D*, то установить на узел n_j блокировку (XT, p_j) и блокировку (IX, *#t*) на его предков;
  (b) если n_j соответствует дополнительным ветвям *locpath* (необходимым для указания предиката) операции *D*, то установить на узел n_j блокировку (ST, p_j) (либо (S, p_j), если для проверки предиката не требуется атомизация узла) и блокировку (IS, *#t*) на его предков.
- Пусть op_i – это операция *RN*. Для каждого $n_j \in NP$:
  (a) если n_j соответствует целевым узлам *RN* (переименовываемому узлу и новому узлу, полученному при переименовании), то установить на узел n_j блокировку (X, p_j) и блокировку (IX, *#t*) на его предков;
  (b) если n_j соответствует дополнительным ветвям *locpath* (необходимым для указания предиката) операции *RN*, то установить на узел n_j блокировку (ST, p_j) (либо (S, p_j), если для проверки предиката не требуется атомизация узла) и блокировку (IS, *#t*) на его предков.

(8) Для каждого $n_j \in PH$ установить блокировку (L, $properties_j$).
(9) Если op_i расширяет схему, то установить блокировку (IN, $properties_i$) для всех предков узла, расширяющего схему.
(10) Если требуемую блокировку установить невозможно (она не совместима с уже установленной блокировкой), то блокировать выполнение операции $a(op_i, t)$ до тех пор, пока не будет освобождена конфликтующая блокировка.

5.5. Дополнительные оптимизации в XDGL

В этом подразделе обсуждаются дополнительные оптимизации в XDGL, которые возможны при наличии *предписывающей* схемы XML-документа. Мы будем считать что предписывающая схема представляется в виде DTD, но на самом деле оптимизация возможна и при использовании других языков описания схемы. В описывающей схеме теряется очень важная информация о документа – порядок следования узлов, в то время как из DTD такую информацию можно получить. Например, фрагмент DTD

```
<!ELEMENT person (name, age?, sex, salary?)>
```

показывает, что все дочерние узлы элемента *person* строго упорядочены: *age* всегда следует после *name*, *sex* – строго после *age* (или *name*, если *age* отсутствует), а необязательный элемент *salary* всегда завершает список дочерних узлов элемента *person*.

Информацию о порядке дочерних узлов можно использовать для оптимизации блокировок для осей *preceding-sibling* и *following-sibling*. Дело в том, что для шага *preceding-sibling::node-name* необходимо установить блокировки *S* для всех “братьев” узла *node-name*, поскольку описывающая схема не обеспечивает информацию о том, какие узлы следуют за *node-name*. При наличии же информации о порядке узлов можно блокировать не всех “братьев” *node-name*. Например, для запроса *count(/person/sex/preceding-sibling::*)* на последнем шаге достаточно заблокировать в режиме *S* только *sex* и *salary*, а *name* и *age* можно не блокировать.

5.6. Примеры использования XDGL

В этом подразделе мы рассмотрим несколько примеров, на которых демонстрируется протокол XDGL. Все примеры основываются на XML-документе GTree (рис. 3). На рисунках для примеров мы показываем только часть узлов схемы, на которых транзакциями устанавливаются блокировки.

Пример 2 (чтение и удаление узлов-братьев) Предположим, что транзакция T_1 намерена прочитать все элементы *name*, выбираемые путевым выражением */doc/person/name*. В это же время транзакция T_2 хочет удалить все элементы *hobby*, определяемые путем доступа: */doc/person/hobby*: *Delete(/doc/person/hobby)*. Очевидно, что логического конфликта между транзакциями нет, поскольку удаление узлов второй транзакцией не влияет на результат первой. Таким образом, транзакции T_1 и T_2, вообще говоря, могут выполняться одновременно. На рис. 5 мы показываем, что протокол XDGL действительно позволяет выполнять транзакции T_1 и T_2 одновременно, поскольку между блокировками, необходимыми для выполнения T_1 и T_2, нет конфликтов.

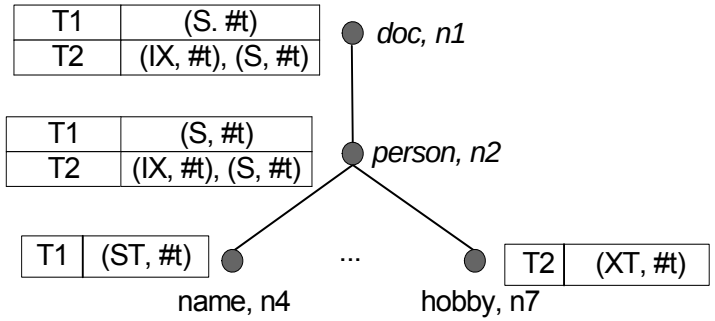

Рис. 5. Блокировки на схеме для примера 2

Пример 3 (конфликт двух операций вставки) Предположим, что транзакция T_1 намерена вставить новый элемент: *InsertInto (<child/>, /doc/person)*. В это же время транзакция T_2 хочет вставить другой элемент: *InsertInto(<hobby/> , /doc/person)*. Транзакции T_1 и T_2 не могут выполняться одновременно, поскольку они обе вставляют узлы в позицию последнего дочернего узла узлов *person*. Таким образом, возникает конфликт, связанный с упорядоченностью узлов в XML-документе. На рис. 6 мы демонстрируем, что XDGL справляется с подобными конфликтами.

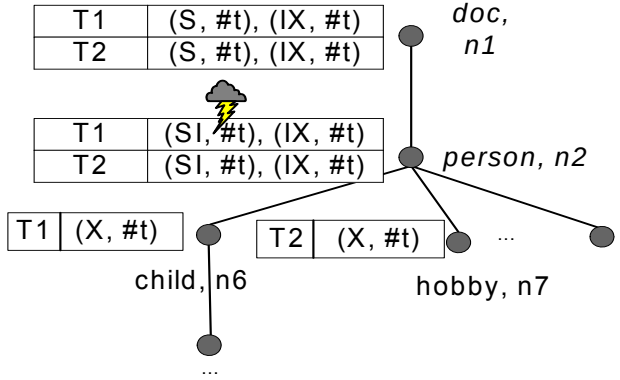

Рис. 6. Блокировки на схеме для примера 3

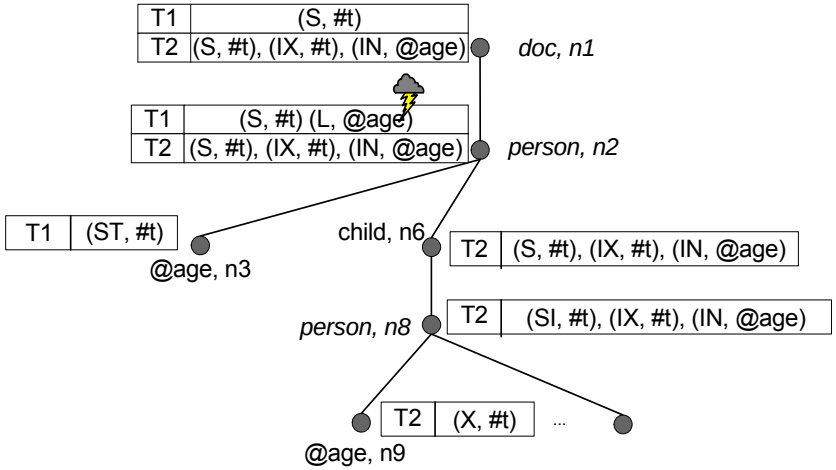

Рис. 7. Блокировки на схеме для примера 4

Пример 4 (вставка узла-фантома) Предположим, что транзакция T_1 намерена прочитать все атрибуты *age*, определяемые путевым выражением */doc/person//@age*. В это же время другая транзакция *T2* намерена вставить

новый атрибут *age*: *InsertInto(attribute age {'54'}, /doc/person/ child/person)*. При одновременном выполнении транзакций возможно появление узла-фантома *age* для T_1. На рис. 7 мы демонстрируем, что XDGL запрещает вставку узла-фантома *age* транзакцией T_2.

6. Атомарность XML-транзакций

Свойство атомарности транзакций означает, что все изменения, произведенные транзакцией над базой данных, либо будут после фиксации транзакции полностью присутствовать в базе данных и будут видны другим пользователям, либо ни одно изменение не будет внесено в базу данных. При декомпозиции XML-транзакции на субтранзакции SXTM необходимо гарантировать ее атомарность.

Если в РСУБД поддерживается модель вложенных транзакций ONT (Open Nested Transactions) [23], то атомарность XML-транзакции можно реализовать средствами РСУБД. Основной характеристикой этой модели транзакций является то, что глобальная транзакция состоит из субтранзакций нижнего уровня, и при фиксации каждой субтранзакции полностью *освобождаются* ее блокировки. Тогда XML-транзакцию можно запускать в РСУБД как глобальную ONT-транзакцию, а отдельные операции XML выполнять как субтранзакции. В результате при откате XML-транзакции (или ее повторном выполнении) РСУБД по журналу может произвести все необходимые действия для обеспечения атомарности XML-транзакции.

Если РСУБД не поддерживает модель ONT, то SXTM разбивает исходную транзакцию на набор *независимых* DB-транзакций. В этом случае в SXTM должен присутствовать дополнительный менеджер восстановления XML-транзакций, поскольку в РСУБД ничего не известно о глобальной транзакции, и средствами РСУБД нельзя ее откатить (или повторно выполнить после сбоя). Менеджер восстановления в SXTM можно реализовать следующим образом.

SXTM должен заносить в журнал все изменения, производимые операциями модификации XML-документа. При фиксации каждой глобальной транзакции в журнал должна попадать соответствующая запись COMMIT. Журнал можно реализовать в виде дополнительной таблицы реляционной базы данных со следующей схемой:

```
SXTM-Log(LSN INT PRIMARY KEY, Transaction-ID INT,
         Operation VARCHAR(3), Parameters CLOB)
```

Первичным ключом таблицы *SXTM-Log* является столбец *LSN* (*Log Sequence Number*), значения которого уникально идентифицируют записи в журнале. В SXTM *LSN* можно реализовать на основе обычного счетчика, значение которого увеличивается на единицу при вставке очередной записи в журнал. По *LSN* можно восстановить порядок следования операций модификации в глобальной транзакции. Столбец *Transaction-ID* содержит идентификатор глобальной XML-транзакции, к которой относится операция. В столбце *Operation* хранится тип операции модификации. Наконец, столбец *Parameters* хранит все параметры операции изменения.

Вставка в журнал *SXTM-Log* записи об операции модификации op_i должна выполняться в рамках DB-транзакции, в которой выполняется эта операция. Таким образом, в случае мягкого сбоя системы (теряется содержимое основной памяти, но не повреждаются данные на диске) после восстановления базы данных средствами РСУБД в журнале *SXTM-Log* окажутся только записи для зафиксированных DB-транзакций. Других записей не будет, поскольку РСУБД гарантирует атомарность DB-транзакций.

Важным следствием этого свойства является то, что менеджеру восстановления SXTM никогда не требуется производить *повторное выполнение* (*REDO*) XML-операций, поскольку при выполнении операции фиксации XML-транзакции, когда производится вставка в журнал записи COMMIT для этой транзакции, происходит фиксация последней DB-транзакции. Таким образом, тот факт, что SXMT успешно зафиксировал XML-транзакцию, означает, что РСУБД успешно зафиксировала все субтранзакции, и поэтому при восстановлении после сбоя повторное выполнение всех субтранзакций будет выполнять РСУБД.

Другим важным обстоятельством является то, что для последней DB-транзакции в журнал не нужно помещать запись об XML-операции, выполняемой в этой транзакции, а достаточно вставить запись COMMIT. Это связано с тем, что SXTM никогда не откатывает последнюю DB-транзакцию (в случае сбоя эта транзакция всегда откатывается РСУБД).

Рассмотрим, какую информацию необходимо сохранять в столбце *Parameters*, чтобы можно было корректно выполнить *отката* (*UNDO*) для любой XML-операции модификации op_i.

- Пусть op_i – это операция I_I. Для операции вставки узла обратной является операция удаления этого узла. В соответствии с семантикой операции I_I новый узел вставляется на место последнего ребенка целевого узла. Следовательно, обратная операция должна удалять последнего ребенка целевого узла. Например, для операции вставки *InsertInto(<name/>, /doc/person)* в столбец *Parameters* нужно записать информацию об операции *Delete/doc/person/name[position() =last()]*. При выполнении *UNDO* необходимо выполнить эту операцию удаления.
- Пусть op_i – это операция I_B. Аналогично, в столбец *Parameters* нужно записать информацию об операции удаления вставленного узла. Например, для операции вставки *InsertBefore(<name/>, /doc/person/hobby)* в столбец *Patrameters* необходимо записать информацию об операции *Delete(/doc/person/hobby::preceding-sibling [position()=1])*. Для операции I_A в последнем выражении необходимо заменить ось *preceding-sibling* на *following-sibling*.
- Пусть op_i – это операция *D*. Для операции удаления узла обратной является операция вставки этого узла. Поэтому удаляемый узел необходимо сохранить в журнале, чтобы при откате можно было его вставить. При этом требуется запомнить не только сам удаляемый

узел, но и его позицию в XML-документе. Для этого перед выполнением операции удаления нужно выполнить предварительный запрос, результатом которого будут удаляемые узлы и их позиции в XML-документе. Например, для операции *Delete(/doc/person[name='John'])* предварительный запрос будет выглядеть так:

```
for $node at $i in /doc/person
where $node/name = 'John'
return ($node, $i)
```

Этот запрос выбирает все узлы *person* с именем “John”, а также их позиции среди всех узлов *person*. Результат запроса сохраняется в столбце *Parameters* таблицы журнала. Кроме того, сохраняется путь в XML-документе, по которому нужно будет вставлять узлы при выполнении отката. Для рассматриваемого примера таким путем является */doc.*

При выполнении обратной операции (*UNDO*) выполняются следующие действия: (1) из столбца *Parameters* выбираются путь *path*, по которому нужно вставлять узлы, а также сами узлы *node* и их позиции *pos* в XML-документе, (2) для каждого узла $node_i$ выполняется операция вставки *InsertAfter*($node_i$, *path*/*node*()[*position*()=pos_i−1]). Для корректного выполнения этих действий необходимо гарантировать, что позиции удаленных и повторно вставляемых узлов в XML-документе не изменятся. Иначе после повторной вставки узел может попасть не на свое место в XML-документе. Для гарантии этого вводятся две дополнительные блокировки *CD* и *LM*, которые не совместимы одна с другой и обладают следующей семантикой.

Блокировка *CD* (*child delete*) используется в операции *D*. Эта блокировка устанавливается на узел схемы, соответствующий родителю удаляемого узла. *CD*-блокировка, установленная на узел *n*, предотвращает какие-либо вставки или удаления детей узла *n*. Это гарантирует, что при откате операции *D* удаленные узлы будут вставляться в точности в те же позиции, в которых находились до операции удаления.

Блокировка *LM* (*level modified*) используется в операциях I_*, *D* и *RN*. Она устанавливается на узел схемы, состав дочерних узлов которого будет изменяться. Так, если транзакция вставляет новый узел в узел *n*, то на узел *n*, помимо блокировки *SI* необходимо установить еще и блокировку *LM*. Если транзакция вставляет новый узел перед узлом *n*, то на узел *n* должна быть установлена блокировка *SB*, а на родителя узла *n* – блокировка *LM*. При этом блокировка *LM* конфликтует с блокировкой *CD*.

- Пусть op_i – это операция *RN*. При выполнении этой операции происходит как удаление, так и вставка узла. Поэтому в журнал необходимо заносить информацию для повторной вставки переименовываемого узла и для удаления нового узла.

В заключение раздела отметим, что при использовании описанного подхода к реализации в SXTM дополнительного менеджера восстановления XML-транзакций (если в РСУБД не поддерживается модель транзакций ONT) для

операций *D* требуется выполнять дополнительный XQuery-запрос, а также приходится устанавливать дополнительные блокировки. Этих накладных расходов можно избежать, если использовать отложенную стратегию выполнения *D* операций.

Основная идея заключается в том, что SXTM может отложить реальное выполнение операции удаления до тех пор, пока в этой же транзакции не понадобится выполнить какую-либо операцию, зависящую от результатов операции удаления, либо не произойдет фиксация транзакции. Если операция удаления выполняется в конце транзакции (при выполнении последней DB-транзакции), то в журнал уже не нужно писать какую-либо информацию по поводу этой операции, поскольку *UNDO* и *REDO* для последней DB-транзакции всегда выполняет РСУБД.

При отложенном выполнении операции *D* SXTM должен установить обычные XDGL-блокировки для операции *D* в момент ее появления в транзакции, а *LM*- и *CD*-блокировки можно устанавливать непосредственно перед фактическим выполнением (возможно отложенном) операции *D*[4].

7. Эксперименты

Рассмотренный семантический менеджер управления XML-транзакциями был реализован на языке C++ в виде расширения к реляционной СУБД MS SQL Server 2005. Для определения временных характеристик РСУБД при использовании SXTM были проведены эксперименты с измерением производительности. Эксперименты определяют пропускную способность РСУБД при использовании плоской модели и двухуровневой модели транзакций (SXTM). При проведении экспериментов параллельно запускалось два потока транзакций: первый поток состоял из читающих транзакций, а второй – из транзакций, содержащих операции модификации.

Для экспериментов использовался 10-мегабайтный XML-документ, сгенерированный утилитой xml-gen [20]. В качестве запросов использовались стандартные запросы XMark [20]. Поскольку в тестовом наборе XMark нет запросов на изменения, мы разработали набор таких запросов, которые отражают основные операции работы с аукционами: операции преобразования открытого аукциона в закрытый, добавление на аукцион нового участника, изменение базовой цены для продаваемого предмета, изменение способа доставки предмета и т. д. Каждая транзакция состоит из нескольких запросов на выборку или изменение. Оба потока транзакций включают по 100 транзакций. Эксперименты проводились на 2-х процессорной системе AMD Athlon MP 2000+, на которой была установлена ОС Microsoft Windows 2003 Server.

На рис. 8 показана пропускная способность РСУБД при использовании SXTM (слева) и плоской модели транзакций (справа). На диаграмме видно, что SXTM

[4] Если операция *D* выполняется в последней DB-транзакции, то блокировки *LM* и *CD* устанавливать вообще не нужно..

позволяет существенно увеличить пропускную способность РСУБД. Наибольший прирост производительности получается для читающих транзакций. Это связано с тем, что в MS SQL Server 2005 существуют специальные путевые индексы и индексы по значениям, применение которых позволяет выполнять запросы эффективно. В результате, если выполнять только читающие транзакции, то они выполнятся очень быстро. С другой стороны, при использовании плоской модели транзакций читающие транзакции большую часть времени ждут освобождения монопольной блокировки всего XML-документа, которую устанавливает изменяющая транзакция. Фактически, в плоской модели читающие и пишущие транзакции выполняются последовательно через одну: после выполнения пишущей транзакции выполняется читающая транзакция, затем снова выполняется пишущая транзакция и т. д. Именно поэтому временные характеристики выполнения потоков транзакций на чтение и изменение в плоской модели практически совпадают. В SXTM же за счет применения семантических блокировок XDGL читающие транзакции конфликтуют с пишущими транзакциями гораздо реже, что и приводит к существенному уменьшению времени выполнения потока читающих транзакций.

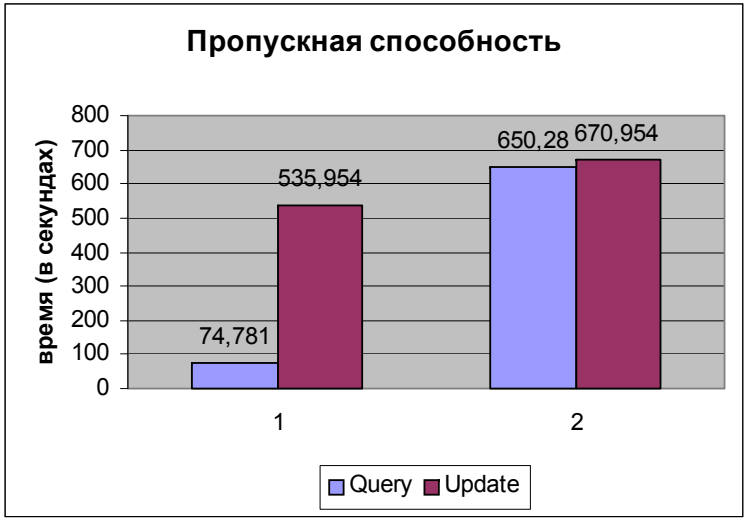

Рис. 8. Экспериментальные результаты: SXTM (1) и плоская модель транзакций (2)

Второй вывод заключается в том, использование SXTM не порождает существенные накладные расходы, которые бы могли ухудшить временные показатели РСУБД.

8. Родственные работы по теме исследования

За последние несколько лет исследователями по базам данных были предложены различные методы синхронизации операций над XML-данными. В этом разделе мы приведем краткий обзор этих методов, а также обсудим их достоинства и недостатки.

Первая группа методов основана на существующих иерархических (метод гранулированных блокировок) и древовидных протоколах синхронизации [2]. При беглом рассмотрении этих методов можно подумать, что они подходят для синхронизации операций над XML-данными, поскольку соответствуют иерархической структуре XML-документов. Но при тщательном анализе мы пришли к выводу, что данные методы не адекватны специфике XML-данных и операций над ними.

Метод гранулированных блокировок не подходит для XML, поскольку в XML предполагается, что на всех уровнях документа хранятся данные, и при этом типичной операцией является выборка значения узла на некотором уровне, а предки этого узла не рассматриваются. При использовании же гранулированных захватов всегда неявно захватывается все поддерево, что для XML почти всегда является избыточным.

Древовидный протокол позволяет устанавливать блокировки на сами узлы документа. Но этот протокол накладывает слишком строгие ограничения на направление перемещения по XML-документу при переходе от одного узла к другому: можно перемещаться только сверху–вниз. В то же время в XPath существуют оси, которые позволяют перемещаться по XML-документу влево, право, вверх, вниз и т.д. от заданного узла. Таким образом, с использованием древовидного протокола можно поддерживать только ограниченное подмножество XPath.

Вторая группа методов [10-11] была предложена для синхронизации DOM-операций. В этих методах используются различные типы блокировок для захвата узлов в XML-документе на разных уровнях. Кроме того, в работе Хелмера [11] обсуждается возможность применения DTD для повышения уровня параллелизма XML-транзакций. Но эти методы применимы только для DOM-операций и не могут использоваться для путевых выражений XPath и языка XQuery.

Третья группа методов [4, 13] характеризуется тем, что в них основным средством доступа к узлам XML-документа являются путевые выражения. В этих работах для синхронизации XML-операций предлагается использовать *путевые блокировки* (*path locks*). Но, к сожалению, эти методы обладают рядом недостатков. Так, подмножество поддерживаемых выражений XPath очень мало и не отвечает требованиям современных XML-приложений. Кроме того, для больших XML-документов (больше 100Мб) количество блокировок становится очень большим, что приводит к огромным накладным расходам. Заметим, что этим же недостатком обладают также и методы предыдущей группы.

Наконец, в работе Т. Грабса [8] предлагается протокол DGLOCK, основанный на методе гранулированных блокировок для синхронизации доступа к XML-документам. Недостатки этого метода мы обсуждали выше. Кроме того, в [8] рассматривается слишком ограниченное подмножество XPath. Дополнительным недостатком DGLOCK является то, что он не обеспечивает сериализацию транзакций, и при его использовании возможно появление фантомов.

9. Заключение

На сегодняшний день большинство производителей коммерческих РСУБД поддерживает в своих продуктах стандарты XML. Это позволяет хранить XML-данные в реляционных базах данных и работать с XML при помощи декларативных языков запросов и изменений.

В статье описывается метод увеличения параллелизма XML-транзакций в реляционных базах данных, основанный на двухуровневой модели транзакций. В рамках этой модели вводится семантический менеджер управления XML-транзакциями – SXTM. Ключевым компонентом SXTM является протокол изоляции XML-транзакций – XDGL, в котором учитывается семантика XML-операций при определении конфликта между транзакциями. Важнейшим свойством XDGL является его независимость от метода представления XML-документа в РСУБД. Это достигается за счет того, что XDGL устанавливает блокировки не на узлах XML-документа, а на узлах описывающей схемы этого документа.

Кроме того, были рассмотрены вопросы атомарности XML-транзакций при их декомпозиции на субтранзакции к РСУБД. Было предложено два метода, которые гарантируют атомарность XML-транзакций: (1) применение модели транзакций ONT, (2) журналирование всех операций изменения XML-данных. Для оптимизации второго метода была предложена стратегия отложенного выполнения операций удаления, журналирование которых приводит к наибольшим издержкам.

SXTM был реализован в виде надстройки над для Microsoft SQL Server 2005. Экспериментальная оценка SXTM показала существенное уменьшение времени выполнения XML-транзакций при большом количестве параллельных запросов на выборку и модификацию частей XML-документов.

Литература

[1] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, J. Simeon. XML Path Language (XPath) 2.0. W3C Working Draft, http://www.w3.org/TR/2005/WD-xpath20-20050211/, 11 February 2005.

[2] P. Bernstein, V. Hadzilakos, and N. Goodman. Concurrency Control and Recovery in DataBase Systems. Addison-Wesley, 1987.

[3] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie and J. Simeon. XQuery 1.0: An XML Query Language. W3C Working Draft, http://www.w3.org/TR/xquery/, 11 February 2005.

[4] S. Dekeyser, J. Hidders. Conflict Scheduling of Transactions on XML Documents. Proc. ADC 2004, Dunedin, New Zealand.

[5] A. Deutsh, M. Fernandez, D. Suciu. Storing Semistructured Data with STORED. In Proc. of ACM SIGMOD, June, 1999, USA, pp. 431-442.

[6] K. Eswaran, J. Gray, R. Lorie and I. Traiger. The notions of consistency and predicate locks in a database systems. Comm of ACM, Vol. 19, No 11, pp. 624-633, November 1976.

[7] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured DataBases. Proc. VLDB 1997, Athens, Greece.

[8] T. Grabs, K. Bohm and H.-J. Schek. XMLTM: efficient transaction management for XML documents, Proc. ACM CIKM 2002, McLean, Virginia, USA.

[9] J. Gray, A. Reuter. Transaction processing: concepts and techniques. Morgan Kaufmann, 1993.

[10] M. Haustein, T. Harder. taDOM: A Tailored Synchronization Concept with Tunable Lock Granularity for the DOM API. Proc. ADBIS 2003, Dresden, Germany.

[11] S. Helmer, C. Kanne, G. Moerkotte. Lock-based protocols for cooperation on XML documents. Proc. of the 14th Int. Workshop on Database and Expert Systems Applications (DEXA), Prague, Czech Republic.

[12] H. B. Hunt, D. j. Rosenkratz. The complexity of testing predicate locks. Proc. ACM SIGMOD 1979, Boston, Massachusetts.

[13] K. Jea, S. Chen, S. Wang. Concurrency Control in XML Document DataBases: XPath Locking Protocol. Proc. ICPADS 2002, Taiwan, ROC, IEEE, 2002.

[14] M. Kay. XSL Transformations (XSLT) Version 2.0. W3C Working Draft, http://www.w3.org/TR/2005/WD-xslt20-20050211/, 11 February 2005.

[15] A. Malhotra, J. Melton, N. Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators W3C Working Draft, http://www.w3.org/TR/2005/WD-xpath-functions-20050915/, 15 September 2005.

[16] P. Pleshachkov, P. Chardin. S. Kuznetsov. A DataGuide-based Concurrency Control Protocol for Cooperation on XML Data. Proc ADBIS 2005, LNCS 3631.

[17] P. Pleshachkov, P. Chardin. S. Kuznetsov. XDGL: XPath-based Concurrency Control Protocol for XML Data. Proc BNCOD 2005, LNCS 3567.

[18] D. j. Rosenkratz, H. B. Hunt. Processing Conjunctive Predicates and Queries. Proc. VLDB 1980, Montreal, Canada.

[19] M. Rys, M. C. Norie, and H. J. Schek. Intra-transaction Parallelism in the Mapping of an Object Model to a Relational Multi-Processor System. Proc. of VLDB 1996, Morgan Kaufmann.

[20] A. Schmidt, F. Waas, M. Kersten, D. Florescu, I. Manolescu, M. Carey, R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.

[21] I. Tatarinov, Z. Ives, A. Halevy, D. Weld. Updating XML. Proc. ACM SIGMOD 2001, Santa Barbara, California, USA.

[22] G. Weikum. Principles and Realization Strategies of Multilevel Transaction Management. ACM Transactions on Database Systems (TODS), 1991.

[23] G. Weikum, H. J. Schek. Database Transaction Models for Advanced Applications. Morgan Kaufmann, 1992.

[24] G. Weikum, G. Vossen. Transactional Information Systems. Morgan Kaufmann, 2002.

[25] F. Yergeau, T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler. Extensible Markup Language, W3C Recommendation, http://w3.org/TR/2004/REC-xml-20040204, 4th February 2004.