Ориентированные на приложения методы хранения XML-данных

Максим Гринев, Иван Щеклеин
maxim@grinev.net, shcheklein@gmail.com

1. Введение

Модель данных XQuery [1] является стандартной моделью данных для работы со слабоструктурированными данными, представленными в формате XML. Поддержка слабоструктурированных данных делает эту модель достаточно универсальной и пригодной для преставления данных различной степени структурированности от регулярных реляционных данных до текстовых документов с размытой структурой.

Оборотной стороной такой универсальности является достаточно низкая эффективность существующих реализаций. На сегодняшний день уже сложился ряд подходов [2, 3, 4, 5] к реализации модели данных, но каждый из этих подходов обладает очевидными преимуществами и недостатками, что делает эти подходы применимыми только для достаточно узких классов приложений. Более того, модель данных XQuery поддерживает возможности, которые являются избыточными для каждого конкретного вида приложения.

Например, предположим, что приложение использует XML для представления реляционных данных. Запросы к таким данным обычно не требуют поддержки таких возможностей модели данных как *братские (sibling)* и *родительские (parent)* оси или *порядок узлов в документе (document order)*.

Другой пример – это запросы к контент-ориентированным XML-данным, таким как энциклопедические статьи [7] или текстовый документ, представленный в формате Microsoft Word XML [6]. Зачастую такие запросы не адресуют XML-элементы, предназначенные для описания способов визуализации данных (примеры такие элементов: *para*, *bold*, *emphasize* и другие, которые составляют, как правило, большую часть элементов в документе), но адресуют семантически значимые элементы, такие как *author*, *дата*, *библиография*. Следовательно, элементы визуализации могут быть представлены на уровне хранения в сжатом незапрашиваемом виде для увеличения скорости операций модификации и сериализации XML данных (под сериализацией здесь и далее мы понимаем процесс трансляции

внутреннего представления данных в строковое представление, соответствующее формату XML).

Приведенные выше рассуждения позволяют нам прийти к выводу, что эффективное внутреннее представление и обработка XML-данных не могут быть достигнуты с использованием какого-либо общего подхода. По нашему мнению, единственно возможным подходом, способным обеспечить высокую эффективность для такой универсальной модели данных, является выбор способов внутреннего представления и методов обработки данных под потребности конкретного приложения. При этом достаточной информацией для описания потребностей является схема XML-данных и рабочая нагрузка в виде возможных запросов и операций модификации данных.

То есть мы предлагаем пойти дальше построения планов выполнения запросов при фиксированных структурах хранения данных, как это делается в большинстве современных систем управления XML-данными, и, кроме того, выбирать структуры хранения данных, необходимые для эффективного выполнения запросов и модификаций для данного приложения. Такой подход позволит поддерживать XQuery модель данных на логическом уровне, но избежать излишних накладных расходов на физическом уровне хранения данных.

С использованием такого подхода можно добиться эффективности обработки регулярных реляционных данных в формате XML, сопоставимой с эффективностью, которая обеспечивается реляционными базами данных. При этом контент-ориентированные данные будут обрабатываться с эффективностью, сопоставимой с эффективностью систем хранения текстовых документов. В данной статье мы описываем наши первые результаты по разработки таких методов хранения и обработки XML-данных.

Статья имеет следующую структуру. В следующем разделе мы рассматриваем примеры, демонстрирующие преимущества предлагаемого подхода. В разд. 3 дается обзорное описание подхода. В разд. 4 описывается физическое представление данных и иллюстрируется на примерах. Разд. 5 посвящен обзору близких работ и существующих подходов хранения XML-данных. В заключительном, шестом разделе мы намечаем пути дальнейших исследований.

2. Мотивирующие примеры

Для демонстрации основных преимуществ и различных аспектов предлагаемого подхода мы выбрали упрощенную версию приложения, которое используется для создания электронной версии Большой Российской Энциклопедии (БРЭ) [7]. Иллюстрации 1 и 2 показывают фрагменты XML-документа, содержащего статью энциклопедии, и его описывающей схемы соответственно. По определению [9] описывающая схема содержит ровно

один путь для каждого пути в документе, и каждый путь в описывающей схеме является путем хотя бы в одном из документов. В этом примере документ представляет собой том энциклопедии, который содержит, по крайней мере, три статьи. Каждая статья состоит из заголовка, списка авторов и тела, которое содержит текст статьи.

```
<volume>
  <article id="2">
    <title>Cyclotron resonance</title>
    <authors>
      <author>Century S.Edelman.</author>
      <author>I. Kaganov</author>
    </authors>
    <body>
      <p>
        <b>Cyclotron resonance</b> Selective absorption of
electromagnetic...
        <link idref="1">Effective weight</link>
        <p> ... <i> ... </i> ... <b> ... </b> ... </p>
        <link idref="6">Lorentz force</link>...
      </p>
    </body>
  </article>
  ...
  <article id="3">
    <title>Dorfman Jacob Grigorevich</title>
    <authors>
      <author>I. Ivanov</author>
    </authors>
    <body>
      <p>
        <b>Dorfman Jacob Grigorevich</b> the Soviet physicist, the
doctor...
        <link idref="2">Cyclotron resonance</link>
        ... <i> ... </i>...
      </p>
    </body>
  </article>
  ...
  <article id="1">
    <title>Effective weight</title>
    <authors>
      <author>I. Kaganov</author>
    </authors>
    <body> ... </body>
  </article>
 </volume>
```

Рис. 1. Фрагмент Большой Российской Энциклопедии.

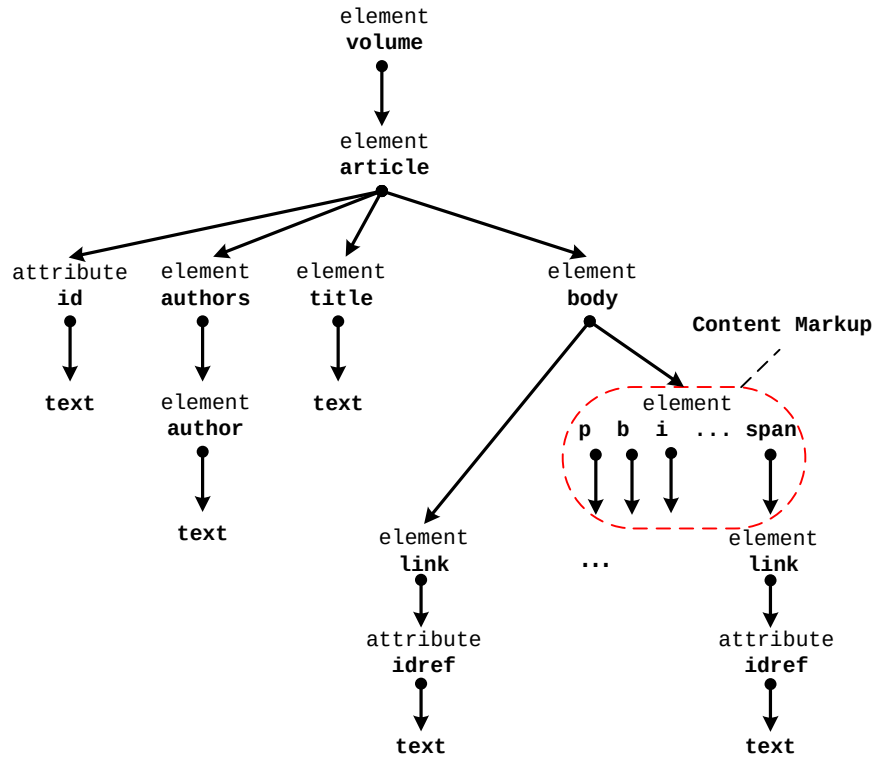

Рис. 2. Описывающая схема Большой Российской Энциклопедии

Для обработки энциклопедии в приложении используются набор запросов, которые являются предопределенными и могут изменяться только при переходе к новой версии системы. Ниже приводится список основных запросов.

(**Q1**) Получение списка названий статей

```
declare ordering unordered;
volume/article/title
```

(**Q2**) Получение статьи по идентификатору

```
declare ordering unordered;
volume/article[@id eq “...”]
```

(**Q3**) Получение статьи по названию

```
declare ordering unordered;
volume/article[title eq “...”]
```

(**Q4**) Перечислить названия статей, на которые ссылается статья с идентификатором равным 1

```
declare ordering unordered;
for $i in volume/article
          [@id eq "1"]//link
return volume/article
       [@id eq $i/@idref]/title
```

(**Q5**) Перечислить звания статей, которые ссылаются на статью с названием «Атом»

```
declare ordering unordered;
let $j := volume/article
          [title eq "atom"]/@id
for $i in volume/article
where $i//link[@idref eq $j]
return $i/title
```

Рассматривая этот пример, мы можем выделить несколько интересных моментов, которые являются общими для многих приложений.

1. *Элементы визуализации.* Контент-ориентированные XML документы часто содержат большое количество XML-элементов, которые обрабатываются исключительно front-end-приложениями (такими как браузер или текстовый процессор) при отображении документа. В приведенном выше примере к таким элементам относятся *p*, *i*, *b*. Такие элементы, как правило, не адресуются запросами. Однако при хранении XML-документов с использование любого общего подхода такие элементы будут представляться таким же образом, как и семантически значимые элементы.
2. *Реляционные данные.* Помимо элементов визуализации в приведенном примере можно выделить элементы и атрибуты с простыми значениями (например, атрибуты *id, idef* и *title element*), которые адресуются запросами. При этом значения этих элементов и атрибутов используются только как промежуточные данные при вычислении запросов в том смысле, что это эти элементы не извлекаются сами по себе, а только как часть другого элемента (например, атрибут *id* используется для нахождения статьи и извлекается из базы данных только как часть статьи).
3. *Порядок узлов документа (document order).* По умолчанию результат вычисления запроса неявно сортируется в порядке узлов в документе. Однако очень часто этот порядок не имеет никакого значения для приложения. Например, в рассматриваемом примере не имеет смысла взаимный порядок следования названия статьи и авторов. В приведенных запросах неявная сортировка выключается в прологе.
4. *Известные наперед запросы (рабочая нагрузка).* В приведенном приложении все запросы известны еще на этапе его создания, то есть система не поддерживает ad hoc запросов к данным. Это позволяет нам, в

частности, построить список путевых выражений, которые составляют основу всех этих запросов: *volume/article*, *volume/article/link*, *volume/article/title*.

Далее в статье мы покажем, как приведенные выше соображения могут быть использованы для выбора структур хранения и планов выполнения запросов.

3. Описание подхода

Для реализации предлагаемого подхода необходимо решить две основные задачи: разработать методы выбора структур хранения для наперед известной и неизменной рабочей нагрузки; разработать методы реорганизации структур хранения на случай изменения рабочей нагрузки. Как мы уже отмечали выше, предполагается, что для большинства приложений рабочая нагрузка (то есть запросы и операции модификации) известна заранее и не подвержена частым изменениям. В следующих подразделах мы рассматриваем каждый из этих методов.

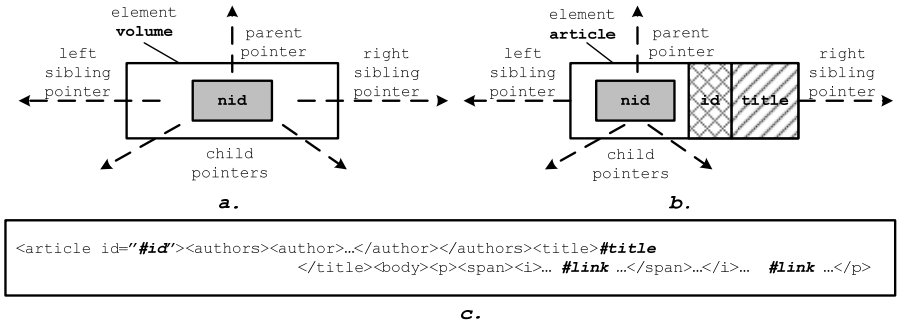

Рис. 3. Различные способы представления данных

3.1. Выбор структур хранения ориентированных на приложение

Имея заранее известную рабочую нагрузку, мы компилируем планы выполнения запросов и соответствующий план хранения данных для заданной нагрузки. В этих планах возможности языка XQuery, которые являются избыточными для поддержки требуемых запросов, не поддерживаются при выполнении. Для построения плана хранения мы используем следующие основные методы:

1) *Комбинирование структурного и текстового представления данных*. Как уже упоминалось выше, большинство элементов в контент-ориентированных XML документах никогда не адресуются запросами. Нами был разработан метод анализа запросов, позволяющий выявить узлы документов, которые необходимо хранить в структурном

представлении (т.е. поддерживая необходимые указатели для навигации между узлами документов) для возможности вычислить все запросы эффективным способом. Все остальные элементы (как правило, это элементы визуализации) сохраняются в текстовом представлении в виде текстовых узлов XML модели данных.

Разработанный метод является достаточно гибким и не ограничивается сохранением всего XML поддерева в виде текста. Элементы со структурным представлением могут иметь в качестве детей элементы с текстовым представлением, которые в свою очередь содержат в качестве детей элементы со структурным представлением. Мы также расширили этот подход для хранения в текстовом представлении некоторых элементов, адресуемых запросом, но по которым не производится поиск. Например, рассмотрим запрос «найти имена все сотрудников старше 60 лет». В этом запросе поиск производится по элементу «возраст» и этот элемент имеет смысл хранить в структурном виде. В то же время элемент «имя» можно хранить в текстовом виде как часть элемента «сотрудник», поскольку предполагается, что в результате поиска обычно возвращается небольшое число элементов, и для них извлечение имени из текстового представления через разбор на лету не является дорогостоящей операцией для всех найденных сотрудников. Для эффективного разбора на лету предлагается использовать методы потоковой обработки XML [18].

2) *Комбинирование методов кластеризации узлов в блоках внешней памяти*. Кластеризация по описывающей схеме, используемая в Sedna XML Database [9], может быть использована совместно с методом хранения рядом детей и родителей, который используется в системах Natix и DB2 [2, 3]. Выбор способа кластеризации для различных групп узлов производится на основе анализа запросов. Комбинирование этих двух методов должно дать существенный выигрыш в производительности.

3) *Исключение избыточных структур и выравнивание вложенности*. Анализируя запросы, можно установить, какие структуры являются избыточными для их выполнения. Главным образом, можно удалять излишние указатели и группирующие элементы. Например, реляционные XML-данные могут быть сохранены в виде компактных записей близко к тому, как они хранятся в реляционных системах. Такие записи не имеют такую же строгую структуру, как в реляционных системах, поскольку необходимо поддерживать возможность нерегулярности в данных, но группировка элементов в записи существенно повышает эффективность системы, так как сокращается количество блоков, которые необходимо прочитать.

   Подобное «уплощение» данных можно проводить и в ряде других случаев для исключения промежуточных элементов. Например, если элемент «сотрудник» содержит элемент «адрес», содержащий, в свою очередь, элементы «улицы» и «дом», то элемент адрес может быть исключен, если

на его уровне нет элементов с именем «улицы» и «дом», которые могут значить что-либо другое.

Обратим внимание, что метод не приводит к потере или размножению данных, а только исключает излишние структурные элементы, однако он может быть естественным образом расширен использованием проекции данных выполняемой при загрузке или модификации данных или поддержкой материализованных представлений. Проекции могут быть построены по анализу путевых выражений или предикатов с константами. Кроме того, исключение избыточных структур (особенно указателей) имеет еще и потенциальное преимущество, связанное с тем, что узлы становятся менее связанными, что не только повышает скорость выполнения запросов и модификаций, но и открывает новые перспективы в улучшении гранулярности транзакционных блокировок и построении распределенных баз данных. Например, это создает предпосылки для реализации параллелизма по данным (data parallelism) на архитектуре shared-nothing.

3.2. Реорганизация структур хранения при изменении приложения

В случае изменения приложения (то есть изменения рабочей нагрузки) в общем случае необходимо полностью перестраивать хранимые данные с целью изменения структуры их хранения. При этом политика реорганизации может быть достаточно гибкой. Во-первых, частота, с которой производится реорганизация, может быть сделан зависимой от выбранного уровня оптимизации, который задается в приложении. Во-вторых, реорганизация будет требоваться не так часто, как это может показаться на первый взгляд. В самом деле, мало вероятно, что новые запросы, которые обращены к «реляционным» XML-данным начнут использовать ссылки на братьев (sibling pointers), которые были удалены на этапе компиляции плана хранения.

Тем не менее, в общем случае реорганизация все равно может потребоваться. Реорганизация может проводиться следующим образом. Вся база данных целиком может быть перестроена с использование массивно-параллельных распределенных вычислений. На современном оборудовании такое перестроение базы данных небольшого и даже среднего объема может быть осуществлено за приемлемое время, равное одному прочтению базы данных с диска.

Как отмечалось выше, «уплощение» структур хранения облегчает создание распределенных баз данных. Если база данных является распределенной, то такая перестройка может быть произведена еще быстрее. В простом случае (когда не были использованы методы оптимизации распределенных баз данных, например, collocated join [10]) перестройка может быть выполнена параллельно и независимо для каждого узла распределенной базы данных. В более сложном случае (данные распределены по узлам для возможности

использовать collocated join) могут быть применены методы, подобные map-reduce [11], для перераспределения данных.

При реорганизации базы данных существует две основные альтернативы. Во-первых, простое решение состоит в остановке базы данных на время перестройки. Если предположить, что для малых и средних баз данных перестройка не займет много времени, то такое решение приемлемо для многих приложений и может осуществляться в ночное время. Более продвинутое решение состоит в использовании механизма теневых страниц (или snapshot isolation [12]) для реорганизации базы данных без ее остановки.

4. Представление данных физического уровня

В этом разделе мы даем обзор основных идей для создания перестраиваемого внутреннего представления XML-данных, оптимизированного под заданную рабочую нагрузку.

Вернемся к примеру, приведенному в разд. 2. Описывающая схема используется для группировки XML-узлов по путям в документе. Для каждой группы узлов оптимальный способ хранения выбирается с учетом нагрузки (рис. 3, a-c):

- *Дискриптор узла (node descriptor)*. Каждый узел в группе может быть сохранен как дескриптор узла, который имеет прямые указатели на детей, родителей и братьев. Это дает возможность эффективной навигации для вычисления структурных путевых выражений [9]. Кроме того, в этом подходе может быть использована нумерующая схема [11, 9]. Каждый дескриптор узла содержит метку номерующей схемы (*nid*). Основное преимущество использования нумерующей схемы состоит в быстром определении связей предок-потомок между любой парой узлов. Нумерующая схема может быть также использована для определения связей, заданных порядком узлов в документе (document order relationship).
- *Значения, упакованные в дескриптор узла*. Для некоторых узлов могут быть использованы структуры, схожие с записями, используемыми при хранении реляционных данных [20]. Запись упаковывается в дескриптор узла (как значения *id* и *title*, показанные на рисунке 3, *b*). Такое “уплощение” данных дает несколько преимуществ. Во-первых, мы получаем практически максимально компактное представление за счет избавления от лишних указателей. Во-вторых, ускоряется выполнение путевых выражений. Особенно тех, которые накладывают условия на упакованные узлы (например, *//article[@id eq “1”]*). В-третьих, существенно ускоряется скорость сериализации данных.
- *Узлы, упакованные в текст*. Узлы, которые не адресуются запросами (например, элементы визуализации) могут храниться в сериализованном текстовом виде. Это позволяет существенно

сэкономить место и увеличить скорость сериализации. Как упоминалось в разд. 3, этот подход не исключает полностью возможность запрашивать элементы из текстового представления, поскольку может быть использован механизм разбора текстового представления на лету. Тестовое представление может содержать заглушки (placeholders) для ссылок на узлы, сохраненные с использование двух приведенных выше методов.

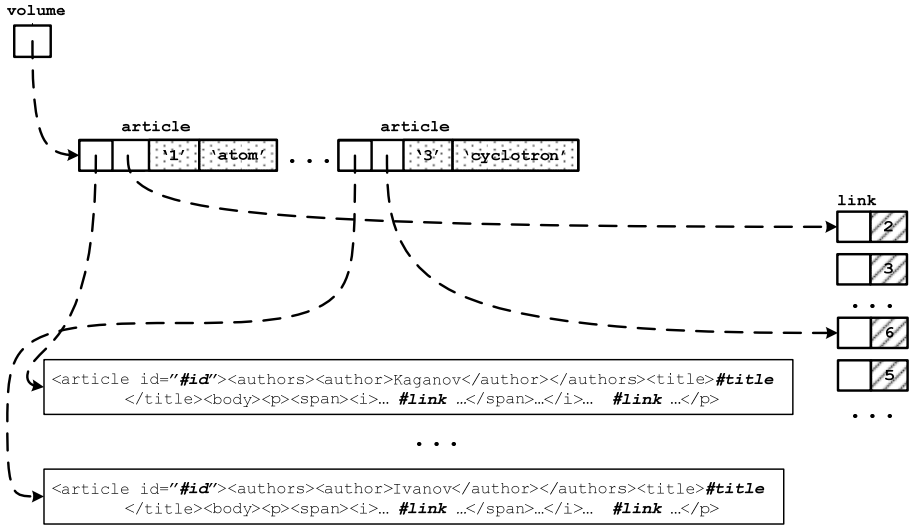

Рис. 4: *Пример внутреннего представления данных.*

На рис. 4 показан план хранения для примера, приведенного в разд. 2. В соответствии с этим планом получается следующее:

- Узлы *article* и *link* представляются в структурном виде с использованием дескрипторов узлов. Это связано с тем, что эти узлы напрямую запрашиваются и сериализуются в запросах Q1-Q5. При этом метки описывающей схемы и указатели на братьев не хранятся, поскольку они не используются для выполнения запросов.
- Атрибуты *id* и *idref* упакованы в дескрипторы родительских узлов, поскольку по ним производится поиск для извлечения этих родительских узлов.
- Узел *title* запрашивается и сериализуется в путевых выражениях запросов Q1-Q5, поэтому он также «уплощается».

- Узлы *author* и другие дети узла *article* упакованы в текстовые представления. При сериализации узла *article*, заглушки *#id*, *#title* и *#link* заменяются полями *id*, *title* и *link* соответственно.

5. Близкие работы

Нам известно очень небольшое число работ посвященных методам хранения XML данных с возможность оптимизации под приложение. В системе OrientStore [13] был предложен подход, в котором комбинируются способы представления данных, предложенные в системах Natix [2] и Senda [4]. Однако подход системы OrientStore не предполагает анализ запросов и основан только на анализе схемы данных. Кроме того, внутреннее представление поддерживает все возможности модели данных XQuery, что зачастую является избыточным, как мы показали в этой статье.

Подходы к хранению данных, предложенные в системах LegoDB [14, 15] или XCacheDB [21], очень близки к подходам, предложенным в этой статье. Тем не менее, эти системы полностью построены над реляционными системами, что накладывает свои ограничения и вызывает дополнительные накладные расходы.

Предложенные в этой статье идеи не следует путать с так называемым подходом компонентных баз данных (component databases) [16, 17]. По нашему мнению, компонентные база данных являются общим подходом, который не допускает эффективной реализации на практике. В отличие от компонентных баз данных, мы не предлагаем общей системы, предназначенной для принципиально различных классов приложений (таких как OLTP, OLAP и другие) и принципиально различных аппаратных платформ (таких как PDA, персональные компьютеры, серверы и другие). Предложенные в этой статье идеи ограничиваются расширением оптимизации запросов на оптимизацию структур хранения с целью избавления от избыточных свойств модели данных XQuery.

6. Заключение и будущие работы

В данной статье были предложены методы оптимизации структур физического хранения XML данных для эффективного выполнения запросов из предопределенной рабочей нагрузки. Нами были описаны только предварительные результаты, однако предварительные эксперименты подтверждают действенность предложенного подхода. Он позволяет существенно сократить размер внутреннего представления, а также увеличить скорость выполнения запросов. Основным результатом данной работы должно стать создание системы, реализующей модель данных XQuery достаточно эффективно для использования XQuery-систем на практике. В будущих работах мы планируем разработать формальные методы анализа запросов, позволяющие автоматически строить планы хранения данных. Кроме того, мы

планируем создать методы, позволяющие производить эффективную реорганизацию базы данных без необходимости ее остановки.

Литература

[1] XQuery 1.0: An XML Query Language. W3C Recommendation 23 January 2007, w3.org/TR/2007/REC-xquery-20070123

[2] T. Fiebig, S. Helmer et al. Anatomy of a Native XML Base Management System, The VLDB Journal 11/ 4, 2002

[3] M. Nicola, B. van der Linden. Native XML support in DB2 universal database. In Proceedings of the VLDB, Trondheim, Norway, 2005

[4] M. Grinev, A. Fomichev, S. Kuznetsov, K. Antipin, A. Boldakov, D. Lizorkin, L. Novak, M. Rekouts, P. Pleshachkov. Sedna: A Native XML DBMS, www.modis.ispras.ru/sedna

[5] M. Haustein, T. Härder. An efficient infrastructure for native transactional XML processing. Data Knowledge Eng., June 2007

[6] E. Ehrli. Walkthrough: Word 2007 XML Format Microsoft Corporation, June 2006

[7] Издательство “Большая Российская Энциклопедия", http://www.greatbook.ru/

[8] S. Chandrasekaran, R. Bamford. Shared Cache - The Future of Parallel Databases. In Proceedings of the ICDE, 2003.

[9] A. Fomichev, M. Grinev, S Kuznetsov. Descriptive Schema Driven XML Storage. Technical Report, MODIS, Institute for System Programming of the Russian Academy of Sciences, 2004

[10] Join methods in partitioned database environments, IBM DB2 Database Information Center, http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp

[11] J. Dean, S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI, December 2004

[12] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, P. O'Neil. A Critique of ANSI SQL Isolation Levels. SIGMOD International Conference on Management of Data San Jose, May 1995

[13] X. Meng, D. Luo, M. Lee, J. An. OrientStore: A Schema Based Native XML Storage System. In Proceedings of the VLDB, 2003

[14] P. Bohannon, J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, J. Siméon: Bridging the XML Relational Divide with LegoDB. In Proceedings of the ICDE, 2003.

[15] M. Ramanath, J. Freire, J. Haritsa, and P. Roy. Searching for Efficient XML to Relational Mappings. Technical Report, DSL/SERC, Indian Institute of Science, 2003

[16] M. Seltzer. Beyond Relational Databases: There is More to Data Access than SQL, ACM Queue 3/3, April 2005.

[17] S. Chaudhuri, G. Weikum. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. The VLDB Journal, 2000

[18] D. Florescu et al. The BEA Streaming XQuery Processor. The VLDB Journal 13/3, September 2004

[19] Q. Li, B. Moon. Indexing and Querying XML Data for Regular Path Expressions. Proceedings of the VLDB Conference, Roma, Italy, 2001

[20] H. Garcia-Molina, J. Ullman, J. Widom. Database Systems: The Complete Book. Prentice Hall, October 2001