

XML-native DB. XQuery



Александр Календарев

В прошлом номере был разговор про XML-native DB, в частности описывалась разработка Института системного программирования XML-native DB Sedna. Для доступа в XML-ориентированных базах данных консорциумом W3C рекомендован язык XQuery. Он является такой же неотъемлемой частью XML-native базы данных, как SQL в реляционных СУБД.

XQuery. Немного истории

Недавно группа XML консорциума W3C отпраздновала свое десятилетие. Вскоре, после выхода первой спецификации XML, появилась потребность в организации хранения информации в формате XML, и соответственно появилась потребность в разработке алгоритмов и методов ее извлечения. Поэтому было предложено разрабатывать «язык запросов», и уже в сентябре 1998 года появился проект спецификации XQL (XML Query Language, <http://www.w3.org/TR/NOTE-xml-ql>) – языка запросов для XML. Данная работа шла параллельно с работой группы XPath, которая занималась разработкой спецификации XPath – языка позиционирования XML-узлов (<http://www.w3.org/TR/xpath>). В связи с тем что разработка обеих групп пересекалась, группа XQL была расформирована, а сама спецификация XQL была признана неудачной.

В августе 1999 года была сформирована группа XQuery в составе 39 человек под руководством Пауля Коттона

(Paul Cotton). А группа XPath впоследствии вошла в состав рабочей группы XSL (<http://www.w3.org/Style/XSL>).

Первые требования спецификации модели данных (Data Model) XQuery вышли в январе 2000 года, а уже в 2001 году публикуется проект спецификации описания языка XQuery.

3 ноября 2005 года XQuery 1.0 получил статус W3C Candidate Recommendation, а 23 января 2007 года XQuery 1.0, одновременно с XSLT 2.0 и XPath 2.0, получил статус официальной рекомендации (W3C Recommendation).

В настоящее время ведутся работы по развитию этого стандарта, с добавлением выражений полнотекстового поиска и для внесения изменений в XML-документы и базы данных (XQuery Update), а также для процедурных операций.

Группой XQuery консорциума W3C подготовлены следующие спецификации:

- XML Path Language (XPath) 2.0;
- XQuery and XPath Data Model;

- XQuery and XPath Functions and Operators;
- XQuery Formal Semantics;
- XML Query Requirements;
- XML Query Use Cases;
- XSLT and XQuery Serialization;
- XML Syntax for XQuery (XQueryX);
- XQuery and XPath Full-Text Requirements;
- XQuery and XPath Full-Text Use Cases.

Сам язык запросов XQuery не был бы так интересен, если бы не появились его практические реализации. На примере Senda мы и рассмотрим, как осуществляется работа с NXD.

XQuery. Основные понятия

В основе XQuery лежит XPath 2.0. Но с другой стороны, XQuery расширяет абстракцию формирования данных с помощью FLWOR-выражений. Акроним «выражения FLWOR» был назван по первым буквам операторов:

- **for** – связывает одну или более переменных, создавая набор кортежей;

- **let** – связывает переменные с результатом вычисления выражения;
- **where** – осуществляет фильтрацию кортежей по заданному условию;
- **order by** – отвечает за сортировку;
- **return** – отвечает за создание выходного выражения для данного кортежа.

XQuery вводит понятие «последовательность». Последовательность – это список, содержащий множество объектов или включающий в себя одну или несколько иных последовательностей.

Каждый объект может представлять либо узел, либо атомарное значение.

Под списком узлов понимается множество узлов, выбранных по имени, типу или иному критерию. В отличие от представления XML DOM-модели, модель данных в XQuery представлена в виде дерева узлов. Допускаются следующие типы узлов: документ, элемент, атрибут, текст, пространство имен, команда обработки (process-instruction) и комментарий. Каждый узел считается уникальным, даже если узлы идентичны.

Узлы можно выбрать по Xpath-выражению, например, для выбора всех названий регионов демонстрационной БД аукцион применима следующая конструкция:

Пример 1. Простое использование XPath

```
doc("auction")/site/regions/*/name()
```

В результате мы получим следующий список:

```
africa,asia,australia,europe,namerica,samerica
```

Использование этого Xpath-выражения аналогично использованию простого оператора:

```
SELECT * FROM TABLE
```

Можно средствами Xpath наложить фильтр, например, выбрать все элементы <item>, которые входят в регион «asia»:

Пример 2. Усложненное использование XPath

```
doc("auction")/site/regions/*[name()='asia']/item
```

Аналогом фильтра в SQL является конструкция WHERE в операторе SELECT, т.е. аналогичное SQL-выражение:

```
SELECT * FROM TABLE regions WHERE name='asia'
```

Синтаксис XQuery в своем арсенале имеет выражения FLWOR. Так, **пример 1** можно преобразовать следующим образом:

Пример 3. Использование выражений FLWOR

```
for $reg in doc("auction")/site/regions/*
return $reg/name()
```

Мы преобразовали запрос, но при этом сами результаты выполнения запроса изменяться не будут.

Используя запрос из **примера 2**, с помощью операторов FLWOR его можно преобразовать в следующий вид:

Пример 4. Использование выражений FLWOR

```
for $reg in doc("auction")/site/regions/*
where $reg/name()='asia'
return $reg/item
```

В **примерах 3 и 4** используется оператор for, синтаксис которого интуитивно понятен и очень похож на оператор цикла в процедурных языках.

Как упоминалось ранее, XQuery так же, как и XSLT, позволяет осуществлять преобразования результата запроса, например, чтобы оформить результат **примера 1** в следующий XML-документ:

```
<region>africa</region>
<region>asia</region>
<region>australia</region>
<region>europe</region>
<region>namerica</region>
<region>samerica</region>
```

Для этого в выражении return необходимо составить шаблон выходного выражения:

```
<region>{$reg/name()}</region>
```

где выражение, заключенное в фигурные скобки {}, вычисляется и осуществляется подстановка. В результате получится следующий запрос:

Пример 5. Использование конструктора выражений

```
for $reg in doc("auction")/site/regions/*
return <region>{$reg/name()}</region>
```

Пусть нам необходимо результат запроса из **примера 5** заключить в теги <regions>{результат}</regions>, тогда данное преобразование должно принять следующий вид:

Пример 6. Более расширенное использование конструктора выражений

```
for $reg in doc("auction")/site/regions
return
<regions>
{
  for $r in $reg/*
  return <region>{$r/name()}</region>
}
</regions>
```

Как вы видите, с помощью XQuery можно формировать структуру XML-документа и производить практически любую трансформацию выходного документа, наподобие того, как это делается в XSLT. Сам SQL в отличие от XQuery формирует только набор строк.

Упорядочивание выборки

Рассмотрим более сложный пример:

Пример 7. Выборка и упорядочивание

```
for $reg in doc("auction")/site/regions/*
return
<region name="{ $reg/name() }">
{
  for $loc in $reg/item
```

```

order by $loc/location
return <item location="{ $loc/location}" ⌋
      id="{ $loc/@id}" ⌋/>
}
</region>

```

В **примере 7** осуществляется выборка всех лотов по регионам и упорядочивание лотов по расположению лота (значение тега <location>, выражение «order by \$loc/location»). Как вы уже догадались, аналогом в SQL является конструкция ORDER BY в операторе SELECT.

Результат выполнения **примера 7**:

```

<region name="africa">
  <item location="United States" id="item0"/>
</region>
<region name="asia">
  <item location="Denmark" id="item2"/>
  <item location="United States" id="item1"/>
</region>
<region name="australia">
  <item location="United States" id="item4"/>
  <item location="Uzbekistan" id="item3"/>
</region>
...

```

Группировка

На следующем примере рассмотрим группировку. Выведем всех участников аукциона и страну их проживания:

Пример 8. Группировка

```

let $person := doc("auction")/site/people/person

for $p in $person
return <person country="{ $p/address/country}" ⌋
      >{ $p/name/text() }</person>

```

Первое, на что хочется обратить внимание в **примере 8**, — это использование последовательности, т.е. присвоение оператором let переменной \$person некой последовательности узлов, значение которой определяется Xpath-выражением: «(doc("auction")/site/people/person)».

Далее, для того чтобы сгруппировать всех участников по странам проживания, необходимо выделить последовательность, которая содержит узлы с именем названий стран:

```
let $country := distinct-values($person/address/country)
```

Для того чтобы пропустить одноименные узлы, используется функция distinct-values().

Если сравнивать с SQL, то данное выражение аналогично оператору «SELECT DISTINCT ...».

Далее все очень просто: проходимся оператором for по всем странам и выбираем имена тех person, которые проживают в данных странах:

Пример 9. Группировка

```

let $person := doc("auction")/site/people/person
let $country := distinct-values($person/address/country)
for $c in $country return
<country name="{ $c}">
  { for $p in $person[address/country=$c ]
    return
    <person>{ $p/name }</person>
  }
</country>

```

Результат:

```

<country name="United States">
  <person <name>Huei Demke</name> </person>
  <person <name>Daishiro Juric</name> </person>
  ...
</country>
<country name="Cook Islands">
  <person <name>Shooichi Oerlemans</name> </person>
</country>
<country name="Greenland">
  <person <name>Nestoras Gausemeier</name> </person>
</country>

```

Конечно, SQL-конструкция:

```
SELECT name, country FROM table GROUP BY country
```

выглядит наглядней, но XQuery более гибкий при построении структур данных.

Пересечения и выборки

XQuery не был бы полным языком, если бы с его помощью нельзя было бы делать объединения и пересечения множеств. Пусть нам необходимо выбрать имена всех участников и сгруппировать их по открытым аукционам. По сути, это запрос «SELECT INNER JOIN ...». Например, данный запрос выдает только ссылки на имена участников:

Пример 10. Выборка по одному множеству

```

let $auc := doc("auction")/site/open_auctions
for $a in $auc/open_auction
return
<auction >{
  for $p in $a/bidder/personref/@person
  return <person>{ $p }</person>
}</auction>

```

И результат этого запроса необходимо пересечь с запросом:

```
let $person := doc("auction")/site/people/person
```

В итоге получаем запрос:

Пример 11. Пересечение выборок

```

let $auc := doc("auction")/site/open_auctions
let $person := doc("auction")/site/people/person
for $a in $auc/open_auction
return
<auction id="{ $a/@id}">{
  for $p in $person[@id = $a/bidder/personref/@person]
  return <person>{ $p/name/text() }</person>
}</auction>

```

Красным цветом выделены изменения. В данном запросе связывающей конструкцией WHERE, аналога SQL, является Xpath-выражение:

```
$person[@id=$a/bidder/personref/@person]
```

Хотелось бы отметить необходимость использования функции text() в возвращаемом выражении: \$p/name/text(). Функция text() возвращает значение текстового узла name. В противном случае мы получили бы значение всего узла, включая обрамляющие теги:

```
<person><name>Huei Demke</name></person>
```

Функцию `text()` можно заменить на функцию `data()`, которую необходимо уже применять ко всему Xpath-выражению: `data($p/name)`.

Результаты выполнения нашего запроса (см. **пример 11**):

```
<auction id="open_auction0">
  <person>Huei Demke</person>
  <person>Laurian Grass</person>
  . . .
</auction>
<auction id="open_auction1">
  <person>Jamaludin Kleiser</person>
  <person>Eliana Rueemmler</person>
  . . .
</auction>
<auction> . . . </auction>
```

Из набора FLWOR-выражений было рассмотрено все, кроме выражения `IF`. Данному выражению трудно найти аналог в SQL-запросах, хотя его можно сравнить с оператором `IF`, используемым в хранимых процедурах, но с более ограниченными возможностями.

Пусть нам необходимо выбрать стоимость лотов из всех аукционов, присвоив каждому лоту признак-attribute, в зависимости от цены – `small` для лотов меньше \$30 и `normal` для всех остальных лотов.

Простейшее решение продемонстрировано в **примере 12**:

Пример 12. Выборка с условием

```
let $price := doc("auction")/site/closed_auctions/
closed_auction/price
for $p in $price
return
  if ( $p < 30 ) then
    <lot type="small" >{data($p)}</lot>
  else
    <lot type="normal" >{data($p)}</lot>
```

Результат выполнения запроса:

```
<lot type="normal">62.07</lot>
<lot type="normal">61.60</lot>
<lot type="small">19.59</lot>
<lot type="small">9.41</lot>
<lot type="small">21.75</lot>
```

В отличие от своих собратьев XSLT и SQL XQuery, благодаря включению FLWOR-выражений, более приближен к процедурному языку, и он проще в понимании. Еще больше его приближает наличие в нем функций. Вызов функции состоит из списка выражений, разделенных запятыми, которые являются аргументами функции. В **примере 13** показывается обращение к встроенной функции `concat()` – склеивание строк:

Пример 13. Использование встроенных функций

```
let $name := doc("auction")/site/regions/*/name()
for $n in $name
return concat( $n, " is region tag")
```

Однако XQuery не был бы мощным средством, если бы не имел функций, определяемых пользователем. Каждая пользовательская функция, как и в любом процедурном

языке, состоит из объявления и тела. Для примера определим функцию `inc()`, которая увеличивает значение счетчика на 1:

Пример 14. Объявление функции

```
declare function math:increment($num as xs:decimal) as
  xs:decimal {
  $num + 1
};
```

Следует отметить, что входные и выходные параметры у функций должны быть типизированы. Допускаются как стандартные типы, определенные спецификацией XML Schema:

- `xs:string`
- `xs:integer`
- `xs:decimal`
- `xs:float`
- `xs:date`
- `xs:QName`
- `xs:anyURI`

Так и простые типы:

- `element`
- `node`
- `attribute`

XQuery также еще позволяет определять пользовательский тип:

```
define type user {
  attribute id of xs:ID
  element rating ?
}
```

Необходимо отметить, что имя функции должно иметь ранее определенный префикс пространства имен (namespace prefix). Под пространством имен понимается идентифицируемая с помощью ссылки URI (RFC2396) коллекция имен, используемых в XML-документах для обозначения типов элементов и именования атрибутов. В данном случае с помощью `uri` будет идентифицироваться набор имен, определяемых пользователем функций. W3C рекомендовано, чтобы использовался уникальный домен разработчика, например `http://mycompany.ru/xquery/fn`. Определяется пространство имен выражением:

```
declare namespace fn="http://nycompany.ru/Sedna/fn";
```

В случае если пространство имен не объявлено, то должен быть объявлен локальный префикс: `local:increment()`.

Спецификация XQuery не определяет операторы изменения данных в XML-документах. Однако любая БД должна не только извлекать данные из хранилища, но и иметь возможность манипулировать ими. В DML (Data Manipulation Language), являющимся подмножеством SQL, существуют операторы: `INSERT`, `UPDATE` и `DELETE`, выполняющие функции вставка/изменение/удаление. XML-DB-консорциум выступил с инициативой XUpdate и предложил свой синтаксис. Данный синтаксис очень похож на синтаксис операторов DML.

Например, для добавления данных необходимо выполнить следующий запрос:

```
Пример 15. Выражение Update. Вставка узла

UPDATE
insert <person id="person25">
  <name>John Smith</name>
  <phone>223-322-223-322</phone>
  <creditcard>3454 3656 2344 6767</creditcard>
</person>
into document("auction")/site/people
```

А для удаления этих же данных необходимо выполнить запрос:

```
Пример 16. Выражение Update. Удаление узла

UPDATE
delete document("auction")/site/people/ ⌋
  person[name/text()="John Smith"]
```

Можно выполнить запрос на замену некоторых узлов:

```
Пример 17. Выражение Update. Замена узла

UPDATE replace
$g in document("auction")/site/people/ ⌋
  person[@id="person25"]/name with
  <name>Ivanov Ivan</name>
```

XQuery. Ближе к практике

В одном веб-проекте, который представляет собой интернет-каталог, необходимо собрать информацию с нескольких XML-источников и опубликовать ее на веб-сайте. Сбором информации занимается программа-загрузчик XML-документов, которая вызывается по расписанию. Далее на основе собранных документов строится обобщенный XML-документ. Пусть обобщенная информация имеет следующую структуру (см. **рисунок**).

Синим цветом на **рисунке** изображен иерархический каталог товаров, который определен тегами:

```
<category id="12" name="пылесосы" ... >
```

Зеленым цветом представлена иерархия товаров, распределенная по брендам.

Например, товар «пылесос THOMAS TWNtt». Пылесос относится к элементу каталога: «Бытовая техника – пылесосы», имеет бренд – марку производителя – THOMAS и является моделью TWNtt. Тег бренда будет:

```
<brand id="23" name="THOMAS" >
```

```
<category @name @id >
├── <category>
├── <brand @name @id >
│   ├── <description>
│   └── <item @name @category >
│       ├── <description>
│       └── <offer @price @shop_id @url>
```

Структура обобщенной информации

Каждый бренд – множество моделей, каждая из которых описывается тегом item:

```
<item id="34521" name="TWNtt" category="12">
  <description>описание модели</description>
</item>
```

Тег <item> имеет атрибут category, который ссылается на элемент каталога (на **рисунке** это изображено красной стрелкой). По каждой модели есть набор предложений от разных магазинов. Каждое предложение описывается тегом:

```
<offer price="12500" shop_id="12" ⌋
  url="http://texnoshock.ru/12343" />
```

Для выбора всех товаров данной категории достаточно сделать следующий запрос:

```
Пример 18. Запрос на выборку товаров по всей категории «пылесосы»

let $catalog := doc("catalog")/category[@name = "пылесосы"]
let $goods := doc("goods")/brand/item

for $g in $goods
where $catalog/@id = $g/@category
return $g
```

В нашем примере конструкция WHERE осуществляет связывание элементов документов doc("catalog") и doc("goods") наподобие конструкции WHERE при «джойне» таблиц в SQL. Для поиска конкретного товара по названию достаточно выполнить следующий запрос:

```
Пример 19. Запрос на выборку товаров по имени товара

let $goods := doc("items2")/goods/brand/item
for $g in $goods
where @name="TWNtt"
return $
```

Иногда при поиске мы не знаем точного имени, а знаем только часть, тогда можно использовать встроенные строковые функции: fn:contains, fn:start-with, fn:ends-with, fn:substring-before, fn:substring-after:

```
Пример 20. Запрос на выборку товаров по части имени:

let $goods := doc("items2")/goods/brand/item
for $g in $goods
where fn:contains($g/@name, "S80")
return $g
```

В заключение хочется отметить, что определенные технологии надо использовать строго по назначению, для чего они разрабатывались. Если в проекте большая часть данных обрабатывается в XML-формате, то и использование специально адаптированных под XML средств даст положительный эффект.

XQuery и WEB

В последнее время хоть и растет популярность использования XML-технологий, но в ближайшее время массового использования XML-native DB в WEB, на мой взгляд, не предвидится. Как говорится, нет спроса – нет и предложения. Соответственно хостеры и не спешат устанавливать XML DB.

Существуют отдельные проекты, которые используют XML-native DB, и число таких проектов постепенно растет. Авторы проекта Sedna реализовали поисковую систему WikiXMLDB (<http://wikixmlb.dyndns.org>), которая включает 24 Гб информации, собранной в Википедии (<http://en.wikipedia.org>). Поиск информации осуществляется довольно-таки быстро. В данном проекте использован индекс full-text.

Авторы проекта Sedna для использования построения индекса full-text используют внешний коммерческий компонент dt_search (<http://www.dtsearch.com>). В ближайшем будущем авторы проекта обещают внедрить внутренний механизм построения индекса full-text.

Использование Sedna в качестве хранилища информации целесообразно в случае работы с XML-документами. Например, в таких проектах, которые интенсивно обмениваются XML-документами или SOAP-сообщениями. Один проект закачивает и обрабатывает из разных источников 600-700 XML-сообщений. В другом известном мне проекте сделан импорт базы данных «1С» – склад в электронный магазин в XML-формате. Но такие проекты – скорее всего исключение, вернее сказать, попытки освоить новые технологии.

Если зашел разговор о технологиях, то, сравнивая с наиболее популярной РСБД MySQL, можно сказать, что по скорости выбора данных Sedna соизмерима с MySQL и даже на некоторых запросах обгоняет его. Чтобы быть более объективным, необходимо добавить, что при тестировании не учитывалась загрузка системы, т.е. тестирование проходило на «чистой» машине.

Надо отметить, что нет методики сравнения XML BD и реляционных БД, поэтому для сравнения были загружены в MySQL и Sedna одни и те же данные. Сравнивалась скорость извлечения однотипных данных.

Скорость выборки данных во многом зависит от правильной организации структуры данных. Надо отметить, что простое копирование структуры таблиц РСБД здесь не подходит и это отдельная тема для статьи. Однако Sedna имеет небольшой недостаток. Это относительно значительное время, необходимое на первоначальную загрузку данных (команда LOAD). Данный недостаток отсекает часть проектов, в использовании которых необходимо оперативно в реальном времени вводить большие объемы XML-данных и сразу же их обрабатывать.

Если сравнивать проекты, то в MySQL хорошо задействованы механизмы кэширования запросов. И если первый запрос выполняется относительно медленно, то все последующие – довольно-таки быстро. Однако механизм кэширования эффективен, если спектр запросов узок. В Sedna кэширование результатов запроса или его части реализовано менее эффективно. Поэтому при сравнении времени исполнения выборок одних и тех же запросов результаты разные, в зависимости от загрузки серверов в данное время другими запросами.

Раз затронута тема WEB, то необходимо упомянуть, что разработчиками проекта был написан модуль mod_sedna для веб-сервера Apache 2.0. Данный модуль позволяет использовать Sedna не только как хранилище, но и как модуль обработки логики.

Используются расширения файлов .xqy и .xquery, заданные по умолчанию. Для обработки этих файлов необходимо в httpd.conf добавить следующие строки:

```
LoadModule sedna_module modules/mod_sedna.so
LoadModule apreq_module modules/mod_apreq.so
AddHandler sedna-handler .xqy .xquery
```

В .xqy-файле пишется запрос, результаты которого отдаются сервером веб-клиенту. Раз есть исходный запрос, то должны быть и параметры запроса. Они передаются как тело POST-запроса. Например:

```
<data><person id="1">Alexandre</person></data>
```

Эти данные доступны в теле запроса как элементы документа request_parameters:

```
let $id = document("request_parameters")/data/person/@id;
let $name = document("request_parameters")/data/person/text();
```


Пара слов о доступе. Sedna – это полноценная БД с разделенным доступом. Все данные соединения содержатся в следующих HTTP-переменных (могут быть как GET, так и POST):

- **se_url** – URL на котором запущена Sedna;
- **se_db** – имя БД;
- **se_login** – login;
- **se_password** – пароль.

Не знаю, из каких принципов исходили разработчики, но, на мой взгляд, это является не совсем правильным с точки зрения защиты веб-приложений. Эти данные необходимо прописать либо в конфигурационном файле httpd.conf, либо в тексте xqy-программы.

Один их вариантов использования Sedna – это формирование SOAP-ответа для веб-служб или формирование непосредственно HTML для дальнейшей публикации данных в WEB, например, с использованием mod_sedna.

Чего не хватает в проекте Sedna, так это хорошего инструмента профилирования. При построении и оптимизации запросов можно полагаться только на свою интуицию и здравый смысл. Нет инструмента определения, как быстро исполнилась та или иная часть запроса, какие индексы были задействованы, каково значение промежуточных переменных.

Остается надеяться, что разработчики будут постоянно совершенствовать свой продукт и в скором времени доведут его до конкурентоспособного состояния. 

1. Официальный сайт проекта Sedna – <http://modis.ispras.ru/sedna/index.htm>.
2. Sedna Programmer's Guide. Документация, входящая в дистрибутив.
3. Консорциум W3C. XQuery – <http://www.w3.org/TR/xquery>.
4. XQuery в Википедии – <http://ru.wikipedia.org/wiki/XQuery>.
5. XQuery tutorial – <http://www.w3schools.com/xquery/default.asp>.
6. XUpdate – <http://xmlb.org.sourceforge.net/xupdate/xupdate-wd.html>.
7. Говард Кац. XQuery от экспертов. Кудиц-образ. М. 2005 г.