МОДЕЛЬ ПОЛУСТРУКТУРИРОВАННЫХ ДАННЫХ, ЯЗЫК ЗАПРОСОВ XQUERY И XML-СУБД

Система управления знанием требует структурного подхода к незнанию.
А. Юркин

Объектные базы данных

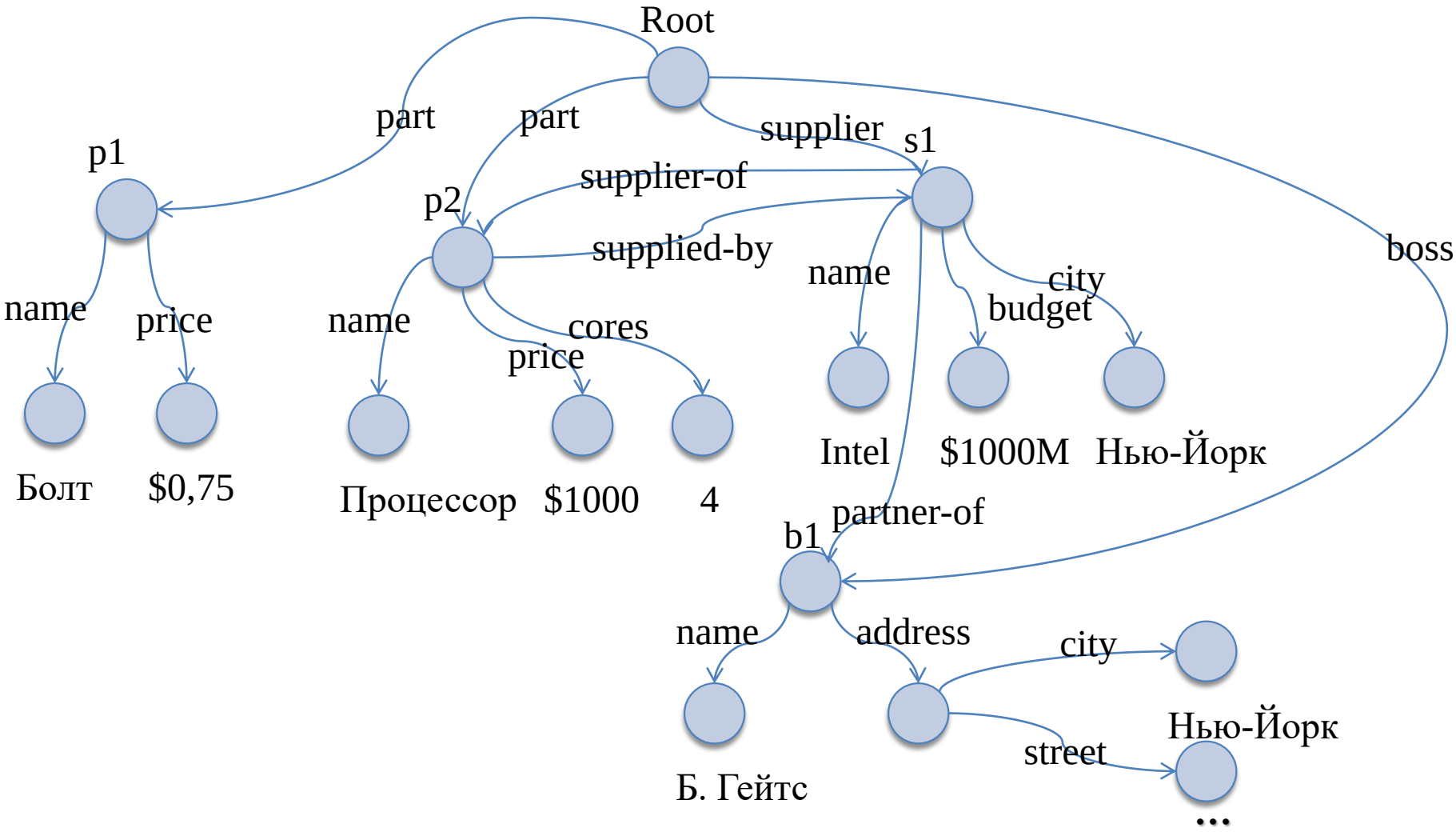
Содержание

- Модель полуструктурированных данных
- Поддержка XML данных в СУБД Oracle
- Язык запросов XQuery
- XML СУБД Sedna

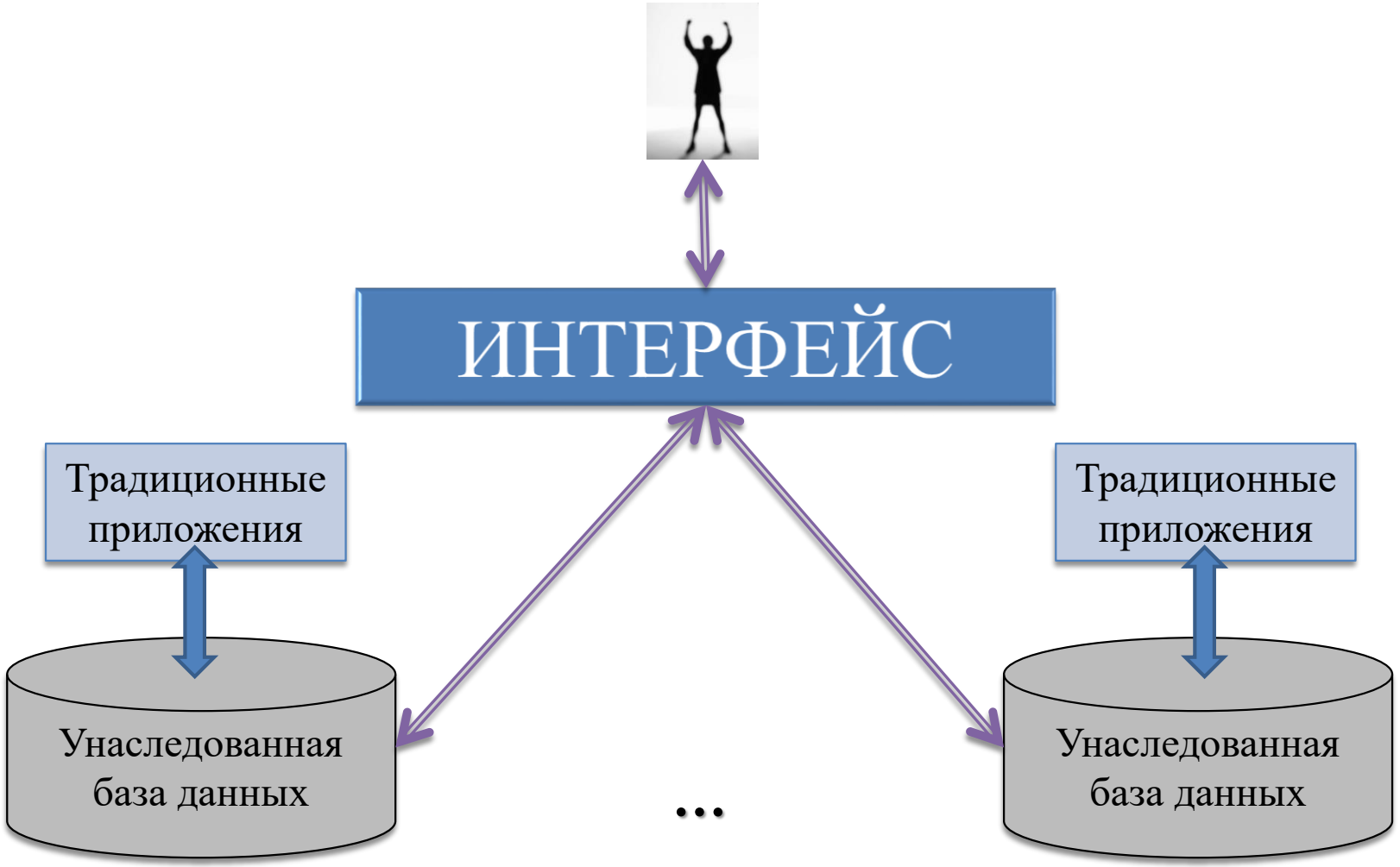
Модель полуструктурированных данных

- В модели полуструктурированных (semistructured) данных отсутствует фиксированная схема данных – в отличии от реляционной, ОО и ОР моделей.
  - Данные сами по себе несут информацию о своем представлении.
  - Представление данных может изменяться произвольно (со временем и/или в контексте конкретной базы данных)
- Основные функции модели
  - Интеграция данных
  - Представление XML документов

Полуструктурированные данные



Интеграция данных на основе модели полуструктурированных данных



Поддержка XML данных в СУБД

- XML данные как тип BLOB в СУБД
  - СУБД не знает семантику BLOB-поля и не умеет его обрабатывать
- API для работы СУБД с XML данными
  - Формирование реляционной схемы по XML схеме или DTD; преобразование запросов к XML данным в запросы к реляционной схеме
- XML данные как встроенный тип XMLTYPE в СУБД
  - СУБД умеет обрабатывать XML данные (проверка по XML схеме, преобразование текста в XML и обратно)
- XML СУБД
  - XML-документы вместо реляционных таблиц, специальный язык запросов вместо SQL

Поддержка XML данных в СУБД Oracle

- Тип данных XMLTYPE
  - Поле реляционной таблицы может иметь тип XMLTYPE
  - Таблица может иметь (объектный) тип XMLTYPE
  - Контроль соответствия данных типа XMLTYPE определению типа (DTD) или схеме (XML Schema) документа
- Функции PL/SQL и методы объектного типа XMLTYPE для работы с XML данными
  - Преобразование текста в XML элемент и обратно
  - Отбор XML данных в соответствии с условием XPath

Тип данных XMLTYPE

```
create table books (
   bookno number primary key,
   description XMLTYPE);
insert into books values (
   100, XMLTYPE('
        <book>
            <title> Язык программирования Си </title>
                  <author> Б. Керниган </author>
            <author> Д. Ритчи </author>
            <pages>271</pages>
        </book> '));
```

Тип данных XMLTYPE

```
insert into books values (
  101, XMLTYPE('
    <?xml version="1.0"?>
    <!DOCTYPE book [
      <!ELEMENT book (title, author*, pages)>
      <!ELEMENT title (#PCDATA)>
      <!ELEMENT author (#PCDATA)>
      <!ELEMENT pages (#PCDATA)> ]>
      <book>
        <title> Незнайка на Луне </title>
        <author> Н. Носов </author>
        <pages> 103 </pages>
      </book> '));
```

Функции для работы с XMLTYPE-данными

- ExtractValue – извлечение значения XML элемента
  - select bookno, ExtractValue(description, 'book/title') from books;
- ExistsNode – отбор XML данных по условиям XPath
  - select bookno, b.description.XMLDATA from books b where b.description.ExistsNode('book[author="Н. Носов"]')=1;

XMLTYPE как объектный тип

```
create table xmlbooks of XMLTYPE;
insert into xmlbooks values (XMLTYPE('…'));
select * from xmlbooks b; -- XML элементы
select value(x) from xmlbooks x; -- текст
select XMLDATA from xmlbooks; -- текст
```

Функции преобразования XML данных

- XML→текст
  - select bookno, b.description.Extract('/book/author') xdoc from books b;
  - select bookno, ExtractValue(b.description.Extract('/book/title'), '/title') xdoc from books b;
- текст→XML
  - select XMLelement("part", PName) from P;
  - select XMLelement("part", XMLattributes(Pname as "part-name", P# as "part-no") from P;

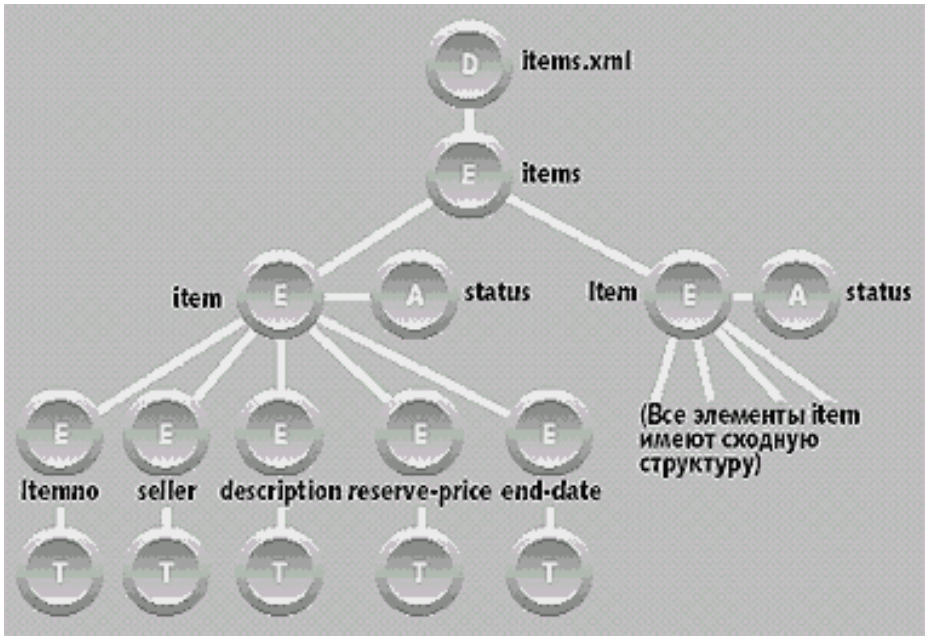
Язык запросов XQuery

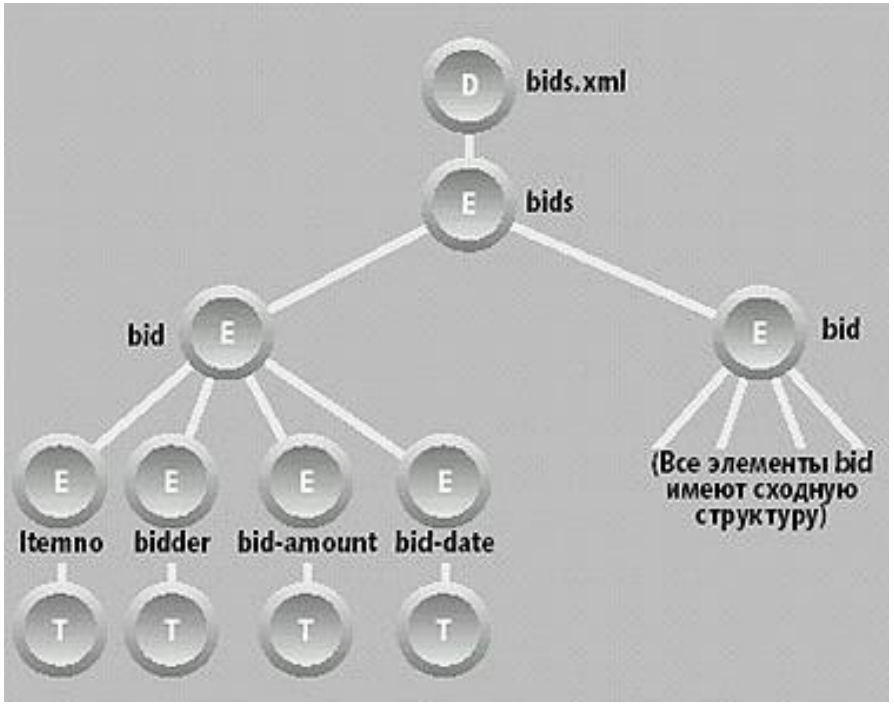
- XQuery – язык запросов к XML данным, разработанный рабочей группой консорциума W3C. Стандартизирован в 2007 г.
  - XML Query Working Group: http://w3.org/XML/Query
  - Описание языка: http://w3.org/TR/xquery
  - Обучение на примерах: http://w3.org/TR/xquery-use-cases
- XQuery – функциональный язык
  - состоит из нескольких типов выражений, которые могут использоваться в различных сочетаниях
  - основан на системе типов XML Schema
  - совместим с другими стандартами XML.

Модель данных

- Последовательность – упорядоченный набор из 0 или большего числа объектов.
- Объект – узел или атомарное значение.
- Атомарное значение – экземпляр одного из типов данных, определенных в XML Schema.
- Узел – одно из: элемент, атрибут, текст, документ, комментарий, команда обработки, пространство имен.
  - Узел может иметь другие узлы в качестве потомков, что позволяет образовывать одну или более иерархий узлов.
  - Элемент и атрибут могут имеют имя или/и типизированное значение (последовательность из 0 или большего числа атомарных значений).
- Последовательность может быть неоднородной (узлы и атомарные значения разного типа). Последовательность не может быть объектом в другой последовательности.
- Порядок документа: каждый узел предшествует своему потомку.

Данные для примеров





Последовательности

- *Последовательность* – набор значений, объединенных оператором ",".
  - 1, 2, 3 ; (1, 2, 3) ; ((1, 2), 3) ; 1 to 3
- *Переменная* – имя, которое начинается с "$", связано со значением и используется в выражении для представления этого значения.
  - let $start := 1, $stop := 3 return $start to $stop
  - Результат – последовательность 1, 2, 3.

Выражения пути

- *Выражение пути* – серия *шагов*, разделенных "/".
  - Результат шага – последовательность узлов. Значение выражения пути - последовательность узлов, которая формируется на последнем шаге.
  - Шаг вычисляется в контексте некоторого узла, называемого контекстным узлом.
- Осевой шаг – перемещение от контекстного узла по иерархии узлов в некотором направлении (по оси).
  - При перемещении по указанной оси осевой шаг выбирает узлы, которые удовлетворяют критерию выбора (на основе одного из следующих: имен узлов, положения узлов по отношению к контекстному узлу, предикат на основе значения узла).
  - Оси: **child, descendant, parent, attribute, self** и **descendant-or-self**.

Выражения пути

- Пример: *перечислить описания всех товаров, предлагаемых к продаже Смитом*
  - document("items.xml")/child::*
    /child::item [child::seller = "Smith"]
    /child::description
- Спецификатор оси может быть пропущен в том случае, когда используется ось **child**
  - document("items.xml") /*/item[seller = "Smith"]/description

Выражения пути

- Разделитель шагов "//"означает, что второй шаг может выполнять поиск в нескольких уровнях иерархии, используя для этого ось **descendants**, а не одноуровневую ось **child**.
- Пример: *перечислить все элементы описания товаров, имеющиеся в документе* ***items.xml***
  - document()//description
  - Результат – это последовательность узлов-элементов, которые могут быть найдены на различных уровнях иерархии узлов (хотя в примере все узлы **description** находятся на одном и том же уровне).

Выражения пути

- Ось **self** (или ".") указывает на контекстный узел, ось **axes** (или "..") – на предка контекстного узла.
- Ось **attribute** (или префикс "@" в имени узла) означает имя узла-атрибута.
- Пример: *найти атрибут статуса для товара, который является предком данного описания товара*
  - $description/../@status
  - Поиск начинается с узла, связанного с переменной **$description**, вдоль оси **parent** к родительскому узлу **item**, а затем - вдоль оси **attribute** в поисках атрибута с именем **status**. Результатом является единственный узел-атрибут.

Предикаты

- *Предикат* – заключенное в квадратные скобки выражение, которое используется для фильтрации последовательности значений. Предикаты применяются в шагах выражения пути и вычисляются в контексте объекта-кандидата.
  - item[**seller ="Jones"**]
  - item [reserve-price > 1000]
  - item [5]
  - item [reserve-price]

Вычисление предикатных выражений

- Правила отбора объектов-кандидатов
  - Если в результате вычисления предикатного выражения получается булевское значение, то объект-кандидат выбирается в том случае, если значение предикатного выражение равно **ИСТИНА**.
    - item [reserve-price > 1000]
  - Если результатом вычисления предикатного выражения является число, то объект-кандидат выбирается в том случае, если его порядковый номер в списке объектов-кандидатов равен этому числу.
    - item [5]
  - Если в результате вычисления предикатного выражения получается пустая последовательность, объект-кандидат отвергается. Однако если результат вычисления предикатного выражения представляет собой последовательность, содержащую хотя бы один узел, объект-кандидат выбирается (такая форма предиката применяется для проверки существования узла-потомка, удовлетворяющего некоторому условию).
    - item [reserve-price]

Операции в предикатах

- *Операции сравнения значений: eq, ne, lt, le, gt, ge*
  - Могут сравнивать два скалярных значения, но порождают ошибку, если любой из операндов является последовательностью с длиной, большей единицы. Если один из операндов - узел, то прежде, чем выполнить сравнение, операция сравнения значений извлекает его значение.
    - item[reserve-price gt 1000]
    - выбирает узел item, если он имеет в точности один узел-потомок **reserve-price** со значением, большим 1000.
- *Общие операции сравнения: =, !=, >, >=, <, <=*
  - Могут работать с операндами, которые представляются собой последовательности, при условии неявного наличия семантики существования для обоих операндов. Как и операции сравнения значений, общие операции сравнения автоматически извлекают значения узлов.
    - item[reserve-price = 1000]
    - выбирает узел item, если у него имеется хотя бы один узел-потомок со значением, большим 1000.

Операции в предикатах

- *Операции сравнения узлов: is и isnot*
  - Определяют идентичность двух узлов
    - $node1 is $node2
    - принимает значение ИСТИНА, если переменные **$node1** и **$node2** связаны с одним и тем же узлом (т. е. для обеих переменных узел один и тот же).
- *Операции сравнения порядка*
  - сравнивают позиции двух узлов.
    - node1 << $node2
    - принимает значение ИСТИНА, если узел, связанный с **$node1**, в порядке документа встречается раньше, чем узел, связанный с **$node2**.
- Логические операции: and и or
  - могут использоваться для объединения логических условий в предикате. Например, следующий предикат.
    - item [seller eq "Jones" and reserve-price]
    - выбирает узлы **item**, имеющие ровно один элемент-потомок **seller** со значением "Jones", а также, по крайней мере, один элемент-потомок **reserve-price** с любым значением.

Конструкторы элементов

- Имена и значения элементов и атрибутов – константы
  - <highbid status = "pending">
    <itemno>4871</itemno>
    <bid-amount>250.00</bid-amount>
    </highbid>

Конструкторы элементов

- Имена элементов и атрибутов – константы, их значения – вычисляемые выражения
  - <highbid status = "{$s}">
    <itemno> {$i} </itemno>
    <bid-amount>
    {max($bids[itemno = $i]/bid-amount)}
    </bid-amount>
    </highbid>
  - <highbid>
    { $b/@status
    $b/itemno
    $b/bid-amount }
    </highbid>

Конструкторы элементов

- Имена и значения элементов и атрибутов – вычисляемые выражения (конструктор **element**)
  - element
    {name($e)}
    {$e/@*, data($e)*2}
  - Новый элемент имеет то же имя и атрибуты, что и **$e**, и значение, вдвое большее значения **$e**. Функция **data** используется для получения числового значения исходного узла.

Конструктор атрибутов

- Вычисляемый конструктор атрибута **attribute**
  - attribute
    {if $p/sex = "M" then "father" else "mother"}
    {$p/name}

Выражения **FLWR**

- Выражение FLWR – набор операторов **for**, **let**, **where**, **return**.
  - Оператор **for** итерирует каждую переменную над ассоциированной с ней последовательностью, связывая переменную по очереди с каждым объектом последовательности.
  - Оператор **let** связывает каждую переменную сразу со всей ассоциированной последовательностью.
  - Оператор **where** содержит выражение, которое вычисляется для каждого кортежа связывания. Если значением этого выражения являются ИСТИНА, то кортеж включается в результирующую последовательность (иначе отвергается).
  - Оператор **return** вычисляется по очереди и по одному разу для каждого оставшегося после проверки условия **where** кортежа связывания. Результаты вычислений объединяется в последовательность, которая и является результатом выражения FLWR.
- Пример: *для каждого товара, который имеет более десяти ставок, создать элемент* ***popular-item****, содержащий номер товара, описание и число ставок*.
  - **for** $i in document("items.xml")/*/item
    let $b := document("bids.xml")/*/bid[itemno=$i/itemno]
    where count ($b)>10
    return
    <popular-item>
    { $i/itemno,
    $i/description,
    <bid-count> {count ($b)} </bid-count> }
    </popular-item>

Выражения **sortby**

- Выражения **sortby** используются для переупорядочивания любой последовательности (независимо от того, сгенерирована она выражением FLWR или нет).
  - **for** $i in document("items.xml")/*/item
    let $b := document("bids.xml")/*/bid[itemno = $i/itemno]
    where count ($b) > 10
    return
    <popular-item>
    { $i/itemno,
    $i/description,
    <bid-count> { count ($b) } </bid-count> }
    </popular-item>
    sortby bid-count descending

Арифметические операции

- Арифметические операции
  - обычные: *****, **div**, **mod** **+**, **-** (должна быть отделена от аргументов для отличия от дефиса)
  - агрегатные функции: **sum, avg, count, max**, **min**
  - возвращают пустую последовательность, если один из аргументов – пустая последовательность.
- Пример: *в последовательности элементов **emp** заменить их подэлементы **salary**, **commission** и **bonus** на новый элемент **pay**, содержащий сумму значений исходных элементов, а результирующую последовательность отсортировать по убыванию значений элемента **pay**.*
  - for $e in $emps
    return
    <emp>
    { $e/name, <pay> {$e/salary + $e/commission + $e/bonus} </pay> }
    </emp>
    sortby (pay)
  - Для сотрудников, у которых не заданы комиссионные или премия, генерируемый элемент **pay** будет пустым.

Операции над последовательностями

- Объединение, пересечение, разность: **union, intersect**, **except**
- Пример*: Создать новый элемент с именем **recent-large-bids**, содержащий копии всех элементов **bid** документа **bids.xml**, которые имеют **bid-amount** со значением больше 1000 и **bid-date** со значением позже 1 января 2002 г.*
  - <recent-large-bids>
    document("bids.xml") /*/ bid [bid-amount > 1000.00]
    intersect
    document("bids.xml") /*/ bid [bid-date > date("2002-01-01")]
    </recent-large-bids>
  - Выражения, в которых используются операции **union, intersect** и **except**, часто можно представить в другом виде:
    <recent-large-bids>
    document("bids.xml")/*
    /bid [bid-amount > 1000.00 and bid-date > date("2002-01-01")]
    </recent-large-bids>

Операции над последовательностями

- Важно помнить о том, что Рассмотрим следующий запрос.
- Пример: *получить список элементов* ***itemno*** *для товаров, которые не имеют ставок*.
  - for $i in document("items.xml")//item
    where empty(document("bids.xml")//bid [itemno eq $i/itemno])
    return $i/itemno
  - Функция **empty** возвращает ИСТИНУ, если ее операнд - пустая последовательность.
  - Пример неверного ответа:
    - document("items.xml")//itemno
      except
      document("bids.xml")//itemno
    - **intersect** и **except** бессмысленно использовать для комбинирования узлов разных документов, поскольку узлы в разных документах никогда не могут быть идентичными. Результатом запроса будет последовательность всех узлов **itemno** документа **items.xml**.

Условные выражения

- Пример: *создать отчет, описывающий состояние ставок для различных товаров, в котором пометить каждую ставку статусом* ***<OK>***, ***<too small>*** *или* ***<too late>***. *Поместить отчет в элемент с именем* ***bid-status-report***.
  - <bid-status-report>
    for $i in document ("items.xml")/*/item
    return
    <item>
      {$i/itemno,
        for $b in document ("bids.xml")/*/bid[itemno = $i/itemno]
        return
        <bid>
          { $b/bidder, $b/bid-amount,
          <status>
            { if ($b/bid-date > $i/end-date) then <too late>
              else if ($b/bid-amount < $i/reserve-price) then <too small>
                else <OK> }
          </status> }
        </bid> }
    </item>
  </bid-status-report>

Кванторные выражения

- Позволяют проверить некоторое условие, устанавливая, истинно ли оно для некоторого значения последовательности (квантор существования **some**) или для всех значений последовательности (квантор всеобщности **every**).
- Пример: *найти товары в **items.xml**, для которых все полученные ставки более чем вдвое превысили начальную цену, получить копии всех таких элементов **item**, и поместить их в новый элемент с именем **underpriced-items**.*
  - <underpriced items>
    for $i in document("items.xml")
    where **every** $b in document("bids.xml") /*/bid [itemno = $i/itemno]
    satisfies $b/bid-amount > 2 * $i/reserve-price
    return $i
    </underpriced-items>

Пользовательские функции

- Помимо библиотеки встроенных функций, предоставляется возможность определения пользователями их собственных функций.
  - Типы, используемые при объявлении типов аргументов и результата в определении функции, могут быть простыми, как **decimal**, или более сложными типами, например, элементами или атрибутами.
  - Не поддерживается перегрузка функций, определенных пользователем (не допускается наличие двух функций с одинаковыми именами).
  - При вызове аргументы связываются с параметрами функции, и выполняется ее тело, порождая результат вызова функции. Если тип параметра функции не указан, этот параметр может принимать значения любого типа. Если не указан тип результата функции, то функция может возвращать значение любого типа.
- Пример: *функция с именем **highBid**, в качестве параметра использующая узел-элемент и возвращающая десятичное значение. Функция интерпретирует свой параметр как элемент **item** и извлекает номер товара. Затем она находит и возвращает самую крупную ставку (**bid-amount**), которая была зафиксирована для товара с этим номером.*
  - define function highBid(element $item) returns decimal
    {
     max(document("bids.xml") //bid [itemno = $item/itemno]/bid-amount)
    }
  - Пример вызова: highBid(document("items.xml")//item [itemno = "1234"])

Пользовательские функции

- Пример: *функция, которая предоставляет значение по умолчанию для отсутствующих данных. Функция с именем* ***defaultVal*** *принимает два параметра: узел-элемент (возможно, отсутствующий) и значение по умолчанию. Если элемент присутствует и имеет непустое значение, функция возвращает это значение. Если же элемент отсутствует или пуст, функция возвращает значение по умолчанию.*
  - define function defaultVal(element? $e, anySimpleType $d) returns anySimpleType
    {
     if (empty($e)) then $d
        else if (empty($e/_) then $d
           else data($e)
    }
  - Пример вызова: запрос, рассмотренный ранее (отсутствующие или пустые элементы **commission** и **bonus** полагаются равными нулю).
    - for $e in $emps
      return
      <emp>
            { $e/name,
            <pay>
            union
            {$e/salary + defaultVal($e/commission,0) + defaultVal ($e/bonus,0)}
            </pay> }
      </emp>
      sortby(pay)

Рекурсивные функции

- Пример: функция возвращает глубину элемента в иерархии, начинающейся с аргумента вызова.
  - define function depth(element $e) returns integer
    {
     if (empty($e/*)) then 1
     else
       1 + max (for $c in $e/* return depth($c))
    }
  - Если у указанного элемента отсутствуют потомки, глубина иерархии равна единице. Иначе глубина иерархии на единицу больше максимального значения глубины всех иерархий, корнем которых является потомок указанного элемента.
  - Пример вызова: depth(document("bids.xml"))

Структура запроса на языке XQuery

- *Пролог запроса* состоит из серии объявлений, которые определяют среду для обработки тела запроса.
  - Пролог нужен, если тело запроса зависит от одного или нескольких пространств имен, схем или функций (эти объекты должны быть объявлены в прологе запроса).
    - Пример объявления пространства имен:
      namespace xyz = "http://www.xyz.com/example/names"
    - Пример объявления импорта схемы:
      schema namespace xhtml = "http://www.w3.org/1999/xhtml"
      at "http://www.w3.org/1999/xhtml/xhtml.xsd"
  - Функции, определенные в прологе запроса, могут использоваться в теле запроса или в телах других функций.
- *Тело запроса* – выражение, значение которого определяет результат запроса.

XML СУБД Sedna

- Sedna – свободно распространяемая XML СУБД, созданная в ИСП РАН (Москва) под руководством С.Д. Кузнецова.
  - http://modis.ispras.ru/sedna/
  - Версии для Windows, Linux, MacOS и др.
  - Персистентность хранимых данных, ACID-транзакции, средства безопасности (пользователи, роли и др.), индексы, "горячее" резервное копирование и др.
  - API для Java, C, PHP, Python, Ruby и др.