**XQuery: язык запросов XML. Условные выражения. Выражения some и every. Действия над множествами. Сравнения узлов.**

**Арифметика.**

В XQuery обеспечиваются обычные арифметические операции: **+, - , \*, div** и **mod**, а также агрегатные функции **sum, avg, count, max** и **min**, которые применяются к последовательности чисел и возвращают числовой результат. Оператор деления в XQuery называется **div**, чтобы его можно было отличить от слэша. Если после оператора вычитания следует имя, перед ним должен стоять пробел, который позволяет отличить его от дефиса, поскольку в XML дефис - корректный символ для имени.

Арифметические операции определяются для числовых значений. К числовым значениям относятся значения типов **integer, decimal, float, double** или типов, производных от них. Если типы операндов арифметической операции различны, операнды приводятся к ближайшему общему типу в соответствии с иерархией приведения **integer -> decimal -> float -> double**. Если операнд арифметического оператора является узлом, то автоматически извлекается его типизированное значение.

Важный частный случай - применение арифметических операций к пустым последовательностям. В XQuery пустая последовательность иногда используется для представления отсутствующей или неизвестной информации, во многом подобно тому, как неопределенное значение используется в реляционных системах. По этой причине операции **+, -, \*, div** и **mod** определяются таким образом, что они возвращают пустую последовательность, если любой из операндов - пустая последовательность. Для иллюстрации этого правила предположим, что переменная **$emps** связана с последовательностью элементов emp, каждый из которых представляет сотрудника и содержит элементы **name** и **salary**, а также дополнительные элементы **comission** и **bonus**. Выражение в Q5 преобразует эту последовательность в последовательность элементов **emp**, каждый из которых содержит элементы **name** и **pay**, причем значение pay равно полной заработной плате сотрудника. Для тех сотрудников, комисионные (commission) или премия (bonus) которых не заданы (**$e/commission** или **$e/bonus** - пустая последовательность), генерируемый элемент pay будет пустым.

**(Q5) Задана последовательность 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)

Иногда желательно определять значение по умолчанию, которое может заменять пропущенные операнды в арифметических выражениях. Ниже объясняется, как в этом случае может использоваться функция, определенная пользователем.

**Операции над последовательностями.**

Оператор **intersect** порождает последовательность, в которую включены все узлы, имеющиеся в обоих операндах. Оператор except позволяет получить последовательность, содержащую все узлы, которые есть в первом операнде, но отсутствуют во втором.

Операторы **union, intersect** и **except** возвращают последовательность узлов в порядке документа и удаляют дубликаты из получившихся последовательностей с учетом индивидуальности узлов. Запрос Q6 является примером использования оператора **intersect**.

**(Q6) 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 года.**

<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. Так, запросу Q6 эквивалентен следующий запрос.

<recent-large-bids>

document("bids.xml")/\*/bid

[bid-amount > 1000.00 and bid-date

> date("2002-01-01")]

</recent-large-bids>

Важно помнить о том, что **intersect** и **except** бессмысленно использовать для комбинирования узлов разных документов, поскольку узлы в разных документах никогда не могут быть идентичными. Рассмотрим следующий запрос.

document("items.xml")//itemno

except

document("bids.xml")//itemno

В этом запросе операция **except** применяется к двум последовательностям узлов **itemno**. Поскольку последовательности узлов выбираются из различных документов, ни один узел во второй последовательности не может быть идентичен узлу из первой последовательности. Результатом запроса будет последовательность всех узлов **itemno** документа **items.xml**. Если предполагалось с помощью этого запроса получить список элементов **itemno** для товаров, которые не имеют ставок, то можно добиться этого воспользовавшись библиотечной функцией **empty**, которая возвращает **true**, если ее операнд - пустая последовательность.

for $i in document("items.xml")//item

where empty(document("bids.xml")

//bid [itemno eq $i/itemno])

return $i/itemno

В этом примере предикат **itemno eq $i/itemno** сравнивает два узла **itemno**, извлекая и сравнивая их содержимое, а не проверяя их идентичность.

Операция |, оставленная для совместимости с XPath 1.0, эквивалентна операции **union**. Эти операции иногда применяются в шагах выражения пути. Например, следующее выражение пути находит объединение всех потомков b и потомков c узлов в последовательности, связанной с **$a**; узлы в этом объединении затем используются в качестве контекстных узлов для следующего шага в пути.

$a/(b | c)/d

**Условные выражения.**

Условные выражения обеспечивают возможность выполнения одного из двух выражений в зависимости от значения третьего выражения. Это записывается в знакомом формате if...then...else, поддерживаемом во многих языках. Требуется наличие всех трех условий (**if, then** и **else**), а выражение в условии if должно быть заключено в скобки. Результат всего условного выражения зависит от значения выражения в условии if, называемого выражением проверки (test expression). Правила таковы.

Если значением выражения проверки являются булевское значение **true** или непустая последовательность (используемая как <проверка существования>), то выполняется оператор then.

Если значением выражения проверки являются булевское значение **false** или пустая последовательность, то выполняется оператор else.

В противном случае условное выражение возвращает значение ошибки.

Следующее простое условное выражение может быть использовано для получения стоимости товара, в зависимости от существования атрибута с именем **discounted**.

if ($part/@discounted) then $part/wholesale

else $part/retail

Q7, представленный на *Рис. 3*, - пример более сложного запроса, содержащего условное выражение. Этот запрос также иллюстрирует несколько уровней вложенности выражений FLWR и конструкторов элементов.

**Рис.3.**

Q7) Создать отчет, описывающий состояние

ставок для различных товаров.

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.**

Кванторные выражения позволяют проверить некоторое условие, устанавливая, истинно ли оно для некоторого значения последовательности (называется квантором существования) или для всех значений последовательности (называется квантором всеобщности). Результатом всегда является **true** или **false**.

Как и FLWR-выражение, кванторное выражение позволяет переменной выполнять итерацию над объектами в последовательности; выполняется поочередное связывание этой переменной с каждым элементом последовательности. Для каждого связывания переменной вычисляется проверочное выражение. Кванторное выражение, которое начинается с **some**, возвращает значение **true**, если выражение проверки истинно для некоторого связывания переменной.

some $n in (5,7,9,11)

satisfies $n > 10

Кванторное выражение, начинающееся с **every**, возвращает значение **true**, если выражение проверки истинно для всех связываний переменной. Например, следующее кванторное выражение возвращает значение **false**, поскольку выражение проверки истинно не для всех связываний.

every $n in (5,7,9,11)

satisfies $n > 10

Использование кванторных выражений иллюстрируется запросом Q8.

**(Q8) 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>

#### Функции

В XQuery предусмотрена библиотека предопределенных функций, а также предоставляется возможность определения пользователями их собственных функций. 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.

В следующем примере определена 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)

}

highbid(document("items.xml")

//item [itemno = "1234"])

Типы, используемые при объявлении типов аргументов и результата в определении функции, могут быть простыми, как **decimal**, или более сложными типами, например, элементами или атрибутами.

В XQuery не поддерживается перегрузка функций, определенных пользователем, т. е. не допускается наличие двух функций с одинаковыми именами. Тем не менее, некоторые встроенные функции являются перегруженными. Например, функция **string** может преобразовывать в строку аргумент почти любого типа.

Аргументы при вызове функции должны соответствовать объявленным типам параметров функции. С этой целью аргумент функции числового типа может быть приведен к объявленному типу параметра с помощью иерархии приведения **integer -> decimal -> float -> double**. Аргумент также считается удовлетворяющим условию вызова, если тип этого аргумента является производным типом (т.е. подтипом) объявленного типа параметра. Если функция, ожидающая атомарное значение в качестве параметра, вызывается с аргументом, являющимся элементом, то до передачи аргумента функции из него извлекается типизированное значение элемента и проверяется на совместимость с ожидаемым типом параметра. Значение, производимое телом функции, должно также соответствовать возвращаемому типу, объявленному в определении; используются обычные правила проверки соответствия типов параметров.

Следующий пример иллюстрирует, как пользователь может написать функцию, которая предоставляет значение по умолчанию для отсутствующих данных. Функция с именем **defaulted** 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 defaulted

(element? $e, anySimpleType $d)

returns anySimpleType

{

if (empty($e))then $d

else if (empty($e/\_)then $d

else data($e)

}

С помощь этой функции запрос Q5 можно переписать (здесь отсутствующие или пустые элементы commission и bonus считаются равными нулю).

for $e in $emps

return

<emp>

{

$e/name,

<pay> | {$e/salary

+ defaulted ($e/commission,0)

+ defaulted ($e/bonus,0)}

</pay>

}

</emp>

sortby(pay)

Функция, в теле которой присутствует вызов самой себя, называется рекурсивной (recursive), и две функции, в теле каждой из которых присутствует вызов другой функции пары, называются взаимно рекурсивными (mutually recursive). В следующем примере рекурсивная функция **depth** может быть вызвана для элемента и возвращает глубину элемента в иерархии, начинающейся с аргумента вызова. Если у элемент-аргумента отсутствуют потомки, глубина иерархии равна единице. Иначе глубина иерархии на единицу больше максимального значения глубины всех иерархий, корнем которых является потомок элемента-аргумента. Это значение вычисляется путем рекурсивного вызова функции **depth**.

define function depth(element $e)

returns integer

{

if (empty($e/\*)) then 1

else 1 + max

(for $c in $e/\* return depth($c))

}

depth(document("bids.xml"))

#### Типы

При создании запроса иногда необходимо сослаться на некоторый тип. Например, как уже было отмечено, при определении функции требуется описать типы параметров функции и ее результата. В других видах выражений XQuery также требуется возможность ссылаться на некоторые типы.

Один из способов сослаться на тип - это указать его квалифицированное имя, или QName. QName может указывать на встроенный тип, такой как **xs:integer**, или на тип, который определен в некоторой схеме, такой как **abc:address**. Если в QName имеется префикс пространства имен (часть, расположенная слева от двоеточия), этот префикс должен быть привязан к некоторому идентификатору пространства имен. Это связывание достигается путем описываемого в следующем разделе объявления пространства имен в прологе запроса.

Еще один способ сослаться на тип - сделать это с помощью родового ключевого слова, такого как **element** или **attribute**. За этим ключевым словом может следовать QName, которое в большей степени ограничивает имя или тип узла. Например, **element** обозначает любой элемент, **element shipto** - любой элемент с именем **shipto**; и **element of type abc:address** означает элемент, тип которого - **address**, объявленный в пространстве имен **abc**. Ключевое слово **attribute** обозначает любой атрибут, **node** - любой узел, а **item** - любой объект (узел или атомарное значение).

В XQuery также предусмотрен дополнительный синтаксис, который позволяет ссылаться на другие виды узлов и на типы элементов, которые определены в локальной части схемы. Например, **element city in customer/address** указывает на элемент с именем **city**, как это определено в контексте схемы **customer/address**.

За ссылкой на тип может следовать один из трех индикаторов присутствия: <\*> означает <ноль или больше>, <+> означает <один или больше>, а означает <ноль или один>. Отсутствие индикатора присутствия означает присутствие ровно одного экземпляра указанного типа. Проиллюстрируем использование индикаторов присутствия.

|  |  |
| --- | --- |
| **element memo?** | означает возможное появление элемента с именем memo |
| **element of type order+** | означает один или несколько элементов с типом order |
| **element\*** | означает любое число любых элементов |
| **attribute?** | означает необязательный атрибут с любым именем и типом |

Ссылки на тип появляются не только в определениях функции, но и других местах. Одно из таких мест - второй операнд операции **instance of**, которая возвращает **true**, если ее первый операнд является экземпляром типа, указанного во втором операнде. Следующие примеры иллюстрируют использование операции **instance of** (предполагается, что префикс **xs** привязан к пространству имен схемы [*http://www.w3.org/2001/XMLSchema*](http://www.w3.org/2001/XMLSchema)).

|  |  |
| --- | --- |
| **49 instance of xs:integer** | возвращает **true** |
| **"Hello" instance of xs:integer** | возвращает **false** |
| **<partno>369</partno> instance of element\*** | возвращает **true** |
| **$a instance of element shipto** | возвращает **true**, если **$a** привязана к элементу с именем **shipto** |

Первая часть **typeswitch** состоит из выражения, тип которого проверяется (выражение операнда, operand expression), и необязательной переменной, связываемой со значением выражения операнда. Далее следуют одно или несколько операторов case, каждый из которых содержит тип и выражение. Операнд выражения по очереди проверяется на соответствие типу, указанному в каждом из условий case. Первый оператор case, для которого операнд выражения является экземпляром заданного типа, называется действующим случаем (effective case); выражение в этом операторе case вычисляется и служит результатом **typeswitch**. Если выражение операнда не соответствует ни одному из типов, указанных в условиях case, результат **typeswitch** берется из последнего оператора, действующего по умолчанию.

Проиллюстрируем использование **typeswitch**. Это выражение может появиться в цикле, где переменная **$customer** итерируется над множеством элементов customer, каждый из которых имеет подэлемент с именем **billing-address**. Подэлементы **billing-address** могут относиться к нескольким различным типам, каждый из которых требуется обрабатывать особым образом. Переменная **$a** связывается с **billing-address**, а затем вычисляется одно из нескольких выражений, в зависимости от динамического типа **$a**. В каждом операторе case **$a** имеет особый тип, например, в первом условии case типом **$a** должен быть **element** of type **USAddress**. Если выясняется, что элемент **billing-address** не соответствует ни одному из ожидаемых типов, результатом выражения является.

typeswitch ($customer/billing-address) as $a

case element of type USAddress

return $a/state

case element of type CanadaAddress

return $a/province

case element of type JapanAddress

return $a/prefecture

default return "unknown"

Имена типов также используются в трех внешне похожих выражениях XQuery, называемых **cast, treat** и **assert**. Каждое из этих выражений содержит ключевое слово, ссылку на тип и выражение, заключенное в скобки.

Выражение **cast** служит для преобразования результата выражения к одному из встроенных типов XML Schema. Поддерживается предопределенный набор преобразований. Например, результат выражения **$x div 5** может быть приведен к типу **xs:double** с помощью выражения **cast as xs:double($x div 5)**. В случае неудачного выполнения операция приведения типа может вернуть значение ошибки. Например, выполнение **cast as xs:integer($mystring)** будет успешным, если **$mystring** - строковое представление **integer**, но вернет ошибку, если **$mystring** имеет значение. Выражение cast не может использоваться для приведения значения к типу, определенному пользователем; для этого ему следует написать специальную функцию.

Выражение **treat** позволяет гарантировать, что динамический (времени исполнения) тип выражения соответствует предполагаемому типу. Например, предположим, что статическим(времени компиляции) типом выражения **$customer/shipping-address** является **Address**. Некоторое подвыражение может иметь смысл только для значений, соответствующих подтипу **Address**, такому как **USAddress**. Создатель подвыражения может использовать выражение **treat** для объявления ожидаемого типа подвыражения.

treat as USAddress($customer/billing-address)

В отличие от **cast**, выражение **treat** на самом деле не меняет тип операнда. Выполнение происходит в два этапа: (1) операнду присваивается некоторый статический тип, который может использоваться для проверки типа при компиляции запроса; (2) во время выполнения, если реальное значение выражения не соответствует указанному типу, возвращается значение ошибки.

Чтобы понять, как процессор запросов мог бы использовать информацию, предоставляемую выражением **treat**, рассмотрим следующий пример.

$customer/billing-address/zipcode

Компилятор XQuery, проверяющий типы, мог бы решить, что в этом примере имеется ошибка типа, поскольку статическим типом **$customer/billing-address** является **Address**, а тип **Address**, в общем случае, не имеет подэлемента **zipcode**. Однако в следующей формулировке статический тип выражения меняется на **USAddress**, у которого есть подэлемент **zipcode**, и ошибка типа исчезает.

(treat as USAddress

($customer/billing-address))/zipcode

Как и **treat**, выражение **assert** используется для предоставления процессору запросов информации, которая может оказаться полезной для проверки типов. Выражение **assert** сообщает процессору запросов, что его выражение операнда имеет некоторый статический тип. Если процессор производит статическую проверку типов, он может породить ошибку, если окажется не в состоянии проверить, что данное выражение соответствует заявленному типу. Выражение **assert** является более строгим, чем выражение **treat**, поскольку оно относится к статическому типу выражения и, следовательно, не зависит от входных данных и может быть проверено перед выполнением запроса. С другой стороны, выражение **treat** относится к динамическому типу выражения и, как следствие, зависит от входных данных и может быть проверено только при выполнении запроса.

В следующем примере будет генерироваться ошибка типа времени компиляции, если статическим типом **$customer/billing-address** является **Address**.

(assert as USAddress

($customer/billing-address))/zipcode

В XQuery не требуется, чтобы в реализации поддерживалась статическая проверка типов. Для процессора запросов, который не обеспечивает статическую проверку, **assert** эквивалентно **treat**.

#### Проверка корректности

Процесс проверки корректности может выполняться применительно к документу XML или к части документа, например, отдельному элементу.

В запросной модели данных с каждым узлом-элементом ассоциируется аннотация типа. Аннотация типа свидетельствует о том, что данный элемент прошел проверку на соответствие определенному заявленному типу. Элементы, которые не прошли проверку или не соответствуют заявленному типу, получают аннотацию родового типа **anyType**. Например, элемент, создаваемый конструктором элементов, имеет аннотацию **anyType** до тех пор, пока не получит более конкретный тип с помощью выражения **validate**. Ниже конструируется элемент и проверяется в соответствии со схемой (схемами), которые указываются в прологе запроса.

validate {<shipto>

<street>123 Elm St. </street>

<city>Elko, NV</city>

<zipcode>85039</zipcode>

</shipto>}

Аннотация типа используется в выражениях, проверяющих тип элемента, таких как instance of и typeswitch, и в выражениях, требующих элемента конкретного типа, таких как вызовы функций. Так, проверка элемента shipto может присвоить ему аннотацию типа USAddress, которая может позволить использовать его в качестве аргумента вызова функции, типом параметра которой является element of type USAddress.

#### Структура запроса

В XQuery запрос состоит из двух частей, называемых прологом запроса (query prolog) и телом запроса (query body). Пролог состоит из серии объявлений, которые определяют среду для обработки тела. Тело запроса - просто выражение, чье значение определяет результат запроса.

Пролог нужен только в том случае, если тело зависит от одного или нескольких пространств имен, схем или функций. Если такая зависимость существует, объекты, от которых зависит тело запроса должны быть объявлены в прологе запроса. Мы обсудим по отдельности объявления для пространства имен, схем и функций.

В объявлении пространства имен определяется префикс пространства имен и указывается его привязка к URI пространства имен. Префикс может быть любым идентификатором. В следующем объявлении пространства имен определяется префикс **xyz** и указывается его привязка.

namespace xyz

= "http://www.xyz.com/example/names"

Это объявление позволяет использовать префикс xyz в именах QName в теле запроса. Префикс связывается с URI некоторого пространства имен и служит в качестве уникального квалификатора для имен элементов, атрибутов и типов. Например, **xyz:billing-address** может уникально идентифицировать элемент **billing-address**, определенный в пространстве имен http://www.xyz.com/example/names. С одним пространством имен можно связать несколько префиксов.

В прологе запроса можно объявить пространство имен по умолчанию, применяемое ко всем неквалифицированным именам элементов и топов, и еще одно пространство имен по умолчанию, применяемое ко всем неквалифицированным именам функций. Ниже иллюстрируется синтаксис объявления пространств имен по умолчанию.

default element namespace

= "http://www.xyz.com/example/names"

default function namespace

= "http://www.xyz.com/example/functions"

Если пространства имен по умолчанию не введены, то неквалифицированные имена элементов, типов или функций считаются не относящимися к какому-либо пространству имен. Неквалифицированные имена атрибутов всегда считаются не относящимися к какому-либо пространству имен.

Помимо объявлений пространств имен пролог запроса может содержать одно или несколько объявлений импорта схемы. При объявлении импорта схемы указывается URI схемы, а также может быть указан второй URI, определяющий место, где может быть найден файл схемы. Цель импорта заключается в том, чтобы предоставить процессору запросов определения элементов, атрибутов и типов, которые объявлены в указанной схеме. Процессор запросов может использовать эти определения для проверки вновь сконструированных элементов, для оптимизации и для проведения статического анализа типов в запросе.

В схеме набор элементов, атрибутов и типов обычно определяется в некотором пространстве имен, называемом целевым пространством имен (target namespace) схемы, но префикс пространства имен не определяется. Поэтому при импорте схемы можно указать префикс пространства имен, привязанный к целевому пространству имен этой схемы. В следующем объявлении импорта схемы префикс пространства имен **xhtml** связывается с целевым пространством имен некоторой схемы, а также системе предоставляется отдельная <подсказка>, позволяющая найти эту схему.

schema namespace xhtml

= "http://www.w3.org/1999/xhtml"

at "http://www.w3.org/1999/xhtml/xhtml.xsd"

Помимо объявлений пространства имен и импорта схем пролог запроса может содержать одно или несколько определений функции. Функции, определенные в прологе запроса, могут использоваться в теле запроса или в телах других функций.