**XQuery: язык запросов XML. Выражения. Последовательности. Конструкторы элементов. Выражение FLWOR.**

**XQuery** — язык запросов и функциональный язык программирования, разработанный для обработки данных в формате XML, простого текста, JSON или других предметно-специфичных форматах. XQuery использует XML как свою модель данных. Предназначен для запроса и преобразования коллекций структурированных и неструктурированных данных.

XQuery 1.0 был разработан рабочей группой XML Query в составе организации W3C. Эта работа координируется другой рабочей группой, работающей над XSLT 2.0. Эти две группы разделяют ответственность за XPath 2.0, входящий в состав XQuery 1.0. 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 для свободного поиска по тексту и для внесения изменений в XML документы и базы данных, а также для процедурных операций.

XQuery версии 3.0 принят W3C 8 апреля 2014 года.

XQuery версии 3.1 принят W3C 21 марта 2017 года.

Несколько примеров **применения**, где используется XQuery:

1. Выборка информации из баз данных с помощью веб-сервисов
2. Формирование отчётов на основе данных в XML базах данных
3. Поиск информации в текстовых документах
4. Выборка и преобразование данных XML в XHTML формат для публикации в вебе
5. Сбор данных из нескольких баз данных для интеграционных приложений
6. Разделение документа XML на несколько частей для выполнения отдельных множественных операций.

**Модель данных**

Формально входные и выходные данные XQuery определяются в терминах модели данных, описанной [здесь](http://www.w3.org/TR/query-datamodel). Запросная модель данных обеспечивает абстрактное представление одного или нескольких документах или фрагментов XML-документов. Модель данных опирается на понятие *последовательности*. **Последовательность** (sequence) - это упорядоченный набор нулевого или большего числа объектов. **Объект** (item) может быть узлом или атомарным значением. **Атомарное значение** (atomic value) - экземпляр одного из встроенных типов данных, определенных в XML Schema, таких как строки, целые и десятичные числа, даты и т.п. **Узел** (node) соответствует одному из семи видов: элементы, атрибуты, тексты, документы, комментарии, команды обработки и пространства имен. Узел может иметь другие узлы в качестве потомков, что позволяет образовывать одну или несколько иерархий узлов. Некоторые виды узлов, такие как элементы и атрибуты, имеют имена или типизированные значения, либо и то, и другое. **Типизированное значение** (typed value) - это последовательность из нуля или большего числа атомарных значений. Узлы индивидуальны (т. е. два узла можно различить, даже если они имеют одинаковые имена и значения), но атомарные величины такой индивидуальностью не обладают. Для всех узлов иерархии имеется полный порядок, называемый порядком документа (document order), в соответствии с которым каждый узел предшествует своему потомку. Порядок документа соответствует порядку, в котором следовали бы узлы, если бы иерархия узлов представлялась в формате XML. Порядок документа между узлами в разных иерархиях определяется в реализации, но он должен быть последовательным, т.е. все узлы одной иерархии должны располагаться либо до, либо после всех узлов другой иерархии.

Последовательности могут быть неоднородными, т.е. могут содержать смесь узлов и атомарных значений разного типа. Однако последовательность никогда не может быть объектом в другой последовательности. Все операции, создающие последовательность, определены так, что результат операции - одноуровневая последовательность. Не проводится различие между объектом и последовательностью единичной длины, т.е. узел и атомарное значение величины считаются идентичными последовательности единичной длины, содержащей эти узел или атомарное значение.

Допускаются последовательности нулевой длины, и иногда они используются для представления отсутствующей или неизвестной информации, во многом так же, как в реляционных системах используются неопределенные значения.

Помимо последовательностей в запросной модели данных определяется специальное значение, называемое значением ошибки (error value), которое является результатом вычисления выражения, содержащего ошибку. Значение ошибки не может присутствовать в последовательности вместе с каким-либо другим значением.

Входные XML-документы могут быть преобразованы в запросную модель данных с помощью процесса, называемого проверкой корректности по схеме (schema validation). Этот процесс выполняет грамматический разбор документа, проверяет его корректность в соответствии с некоторой схемой и представляет документ в виде иерархии узлов и атомарных значений, помеченных типом полученным из схемы. Если входной документ не имеет схемы, проверка его корректности выполняется в соответствии с используемой по умолчанию рекомендательной схемой, которая присваивает родовые типы - узлы маркируются как anyType, а атомарные величины - как anySimpleType.

Результат запроса может быть преобразован из запросной модели данных запросов в XML-представление с помощью процесса, называемого сериализацией (serialization). Следует отметить, что результат запроса не всегда является правильно построенным XML-документом. Например, запрос может возвращать атомарное значение или последовательность элементов, не имеющих общего предка.

**Данные для примеров**

Чтобы проиллюстрировать запросную модель данных и обеспечить основу для последующих примеров, рассмотрим небольшую базу данных, содержащую данные интерактивного аукциона и основанную на Use Case R. Эта база данных содержит два XML-документа с именами items.xml и bids.xml.

Документ items.xml содержит корневой элемент с именем items, который, в свою очередь, содержит элемент item для каждого из товаров, предложенных к продаже на аукционе. Каждый элемент item имеет атрибут status и подэлементы с именами itemno, seller, description, reserve-price и end-date. Элемент reserve-price указывает минимальную продажную цену, установленную владельцем товара, а end-date определяет дату окончания торгов.

Документ bids.xml содержит корневой элемент с именем bids, который, в свою очередь, содержит элемент bid для каждой ставки, которая предлагается за товар. Каждый элемент bid имеет подэлементы с именами itemno, bidder, bid-amount и bid-date.

|  |
| --- |
| http://citforum.ru/internet/articles/xqlzxml_1.gif |
| **Рис. 1. Представление модели данных из items.xml** |

На рис. 1 и 2 показаны модельные представления документов items.xml и bids.xml соответственно (включающие только образцы товара и ставки). Круги, помеченные буквами D, E, A и T, обозначают узлы документов, элементов, атрибутов и тестов соответственно.

|  |
| --- |
| http://citforum.ru/internet/articles/xqlzxml_2.jpg |
| **Рис. 2. Представление модели данных из bids.xml** |

**Выражения**

Теперь опишем выражения XQuery.

Основы. Подобно XML и XPath, в XQuery различаются прописные и строчные буквы, а все ключевые слова состоят из строчных букв. Подробные лексические и грамматические правила XQuery описаны [здесь](http://www.w3.org/TR/xquery). Символы, заключенные между **(-** и **-)** считаются комментариями и при обработке запроса игнорируются (конечно, кроме тех случаев, когда они входят в строку, заключенную в кавычки, и считаются частью этой строки).

Простейший вид выражения XQuery - литерал (literal), который представляет атомарное значение.

**47** литерал типа **integer**

**4.7** литерал типа **decimal**,поскольку он содержит десятичную точку

**4.7E3** литерал типа **double**, поскольку он содержит экспоненту

**"47"** литерал типа **string** (внутри строки, заключенной в двойные кавычки, разрешается помещать одинарные кавычки)

Атомарные значения других типов могут создаваться путем вызова конструкторов. Конструктор (constructor) представляет собой функцию, которая создает значение определенного типа на основе строки, содержащей лексическое представление значения этого типа. В общем случае конструктор имеет то же имя, что и тип, значения которого он конструирует. Ниже конструктор используется для создания значения типа date.

date(<2002-5-31>)

Любое выражение XQuery может быть заключено в круглые скобки. Скобки полезны для определения явного порядка вычисления выражения.

Операция запятая (<,>) соединяет два значения в последовательность. Последовательности часто заключаются в скобки, служащие явными разделителями, хотя это не требуется. Пустая пара скобок означает пустую последовательность. Поскольку последовательности не могут быть вложенными, оператор <запятая> создает последовательность, состоящую из всех объектов левого операнда, за которыми следуют все объекты второго операнда. Последовательность также можно создать с помощью операции to, производящую последовательность, которая состоит из всех целых чисел в отрезке от значения левого операнда до значения правого. Следующие примеры иллюстрируют создание последовательностей.

1, 2, 3 последовательность

из трех значений

(1, 2, 3) идентична 1, 2, 3

((1, 2), 3) идентична 1, 2, 3

1 to 3 идентична 1, 2, 3

Переменная (variable) в XQuery - имя, начинающееся со знака доллара. Переменная может быть связана со значением и использоваться в выражении для представления этого значения. Один из способов связывания переменной состоит в использовании выражения LET, которое связывает одну или несколько переменных, а затем вычисляет внутреннее выражение. Значение выражения LET - результат вычисления внутреннего выражения со связанными переменными. Следующий пример иллюстрирует выражение LET, которое возвращает последовательность 1, 2, 3.

let $start := 1, $stop := 3

return $start to $stop

Выражение LET - частный случай выражения FLWOR (for, let, where, order by, return), которое обеспечивает дополнительные способы связывания переменных.

Еще одна простая форма выражений XQuery - вызов функции (function call). XQuery предусматривает наличие базовой библиотеки функций, описанной [здесь](http://www.w3.org/TR/xquery-operators), и описываемый в следующем разделе механизм, позволяющий пользователям определять дополнительные функции. Вызовы функций в XQuery основаны на обычной нотации с заключением в скобки аргументов функции. В следующем примере вызывается функция базовой библиотеки substring, извлекающая из строки первые шесть символов.

substring(, 1, 6)

**Выражения пути.**

Выражения пути в XQuery базируются на синтаксисе XPath. Выражение пути состоит из серии шагов, разделенных символом слэша (/). Результат каждого шага - последовательность узлов. Значение выражения пути - последовательность узлов, которая формируется на последнем шаге.

Каждый шаг вычисляется в контексте некоторого узла, называемого контекстным узлом (context node). В общем случае шаг может быть любым выражением, возвращающим последовательность узлов. Один из важных видов шага, называемый осевым шагом (axis step), можно считать перемещением от контекстного узла по иерархии узлов в некотором направлении, называемом осью (axis). При перемещении по указанной оси осевой шаг выбирает узлы, которые удовлетворяют критерию выбора. Критерий выбора может выбирать узлы на основе их имен, положения по отношению к контекстному узлу или предикату, базирующемуся на значении узла. В XPath определяются 13 осей, и все они поддерживаются в XQuery.

Выражения пути могут быть записаны в полном или в сокращенном синтаксисе. Полный синтаксис для осевого шага предусматривает указание оси и критерия выбора, разделенных парой двоеточий. Q1 иллюстрирует четырехшаговое выражение пути, оформленное в полном синтаксисе. На первом шаге вызывается встроенная функция document, которая возвращает узел-документ документа items.xml. Второй шаг - осевой шаг, который находит всех потомков узла-документа (<\*> выбирает все узлы на данной оси; в данном случае будет выбран единственный узел-элемент с именем items). Третий шаг снова выполняет поиск вдоль оси child, чтобы найти на следующем уровне все элементы-потомки с именем item, которые, в свою очередь, имеют потомков с именем seller и значением . Результатом третьего шага является последовательность узлов-элементов item. Каждый из этих узлов item служит контекстным узлом для четвертого шага, который опять предусматривает поиск по оси child элементов description, являющихся потомками данного item. Окончательный результат выражения пути - результат четвертого шага: последовательность узлов-элементов description, перечисленных в порядке документа.

**(Q1) Перечислить описания всех товаров, предлагаемых к продаже Смитом.**

document("items.xml")/child::\*

/child::item [child::seller = "Smith"]

/child::description

На практике, выражения пути часто записываются с помощью сокращенного синтаксиса. Пожалуй, наиболее важным является то, что спецификатор оси может быть пропущен в том случае, когда используется ось child. Поскольку child является наиболее часто используемой осью, такое сокращение помогает сократить длину многих выражений пути. К примеру, Q1 можно сократить следующим образом:

document("items.xml")

/\*/item[seller = "Smith"]/description

Разделение двух шагов двойным, а не одинарным слэшем означает, что второй шаг может выполнять поиск в нескольких уровнях иерархии, используя для этого ось descendants, а не одноуровневую ось child. Так, Q2 выполняет поиск элементов description, которые являются потомками (необязательно прямыми) корневого узла данного документа. Результат Q2 - это последовательность узлов-элементов, которые могут, в принципе, быть найдены на различных уровнях иерархии узлов (хотя в нашем примере все узлы description находятся на одном и том же уровне).

**(Q2) Перечислить все элементы описания товаров, имеющиеся в документе items.xml.**

document()//description

В выражении пути одинарная точка (<.>) указывает на контекстный узел, а две последовательные точки (<..>) - на предка контекстного узла. Эти нотации представляют собой сокращенное указание осей self и axes соответственно. Имена, присутствующие в выражениях пути, как правило, интерпретируются как имена узлов-элементов, однако если имя имеет префикс <@>, оно интерпретируется как имя узла-атрибута. Это сокращение для шага, который выполняет поиск вдоль оси attribute. Эти аббревиатуры иллюстрируются в Q3, где 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 Q3 является единственный узел-атрибут.

**(Q3) Найти атрибут статуса для товара, который является предком данного описания товара.**

$description/../@status

**Предикаты.**

В XQuery предикат (predicate) - это заключенное в квадратные скобки выражение, которое используется для фильтрации последовательности значений. Предикаты часто применяются в шагах выражения пути. Например, в шаге item[seller = ] фраза seller = - это предикат, который применяется для выбора определенных узлов item и отбрасывания остальных. Будем называть объекты последовательности, фильтруемые с помощью предиката, объектами-кандидатами. Предикат вычисляется для каждого объекта-кандидата с использованием этого объекта-кандидата в качестве контекстного объекта для вычисления выражения предиката. Термин <контекстный объект> - это обобщение термина <контекстный узел>, и ему может соответствовать как узел, так и атомарное значение. В предикатном выражении одинарная точка (<.>) обозначает контекстный объект. Каждый объект-кандидат выбирается или отвергается в соответствии со следующими правилами.

Если в результате вычисления предикатного выражения получается булевское значение, то объект-кандидат выбирается в том случае, если значение предикатного выражение равно true. Этот тип предиката иллюстрируется в примере, где выбираются узлы item, имеющие узел-потомок reserve-price, чье значение больше 1000:

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. Такой тип предиката представлен в примере, где выбирается пятый узел item по оси child:

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 может применяться для проверки существования узла-потомка, удовлетворяющего некоторому условию. Это иллюстрирует пример, где выбираются узлы item, у которых имеется узел-потомок reserve-price, вне зависимости от его значения:

item [reserve-price]

Внутри предикатов часто используется несколько видов операций и функций.

Операции сравнения значений (value comparison operator): eq, ne, lt, le, gt, ge. Эти операции 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. Например, item[reserve-price gt 1000] выбирает узел item только в том случае, если он имеет в точности один узел-потомок reserve-price со значением, большим 1000.

Общие операции сравнения (general comparison operator): =, !=, >, >=, <, <=. Эти операции 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. Например, item[reserve-price = 1000] выбирает узел item, если у него имеется хотя бы один узел-потомок со значением, равным 1000.

Операции сравнения узлов (node comparison operator): is и isnot. Эти операторы определяют идентичность двух узлов. Например, $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 (order comparison operator). Эти операции сравнивают позиции двух узлов. Например, $node1 << $node2 принимает значенией true, если узел, связанный с $node1, в порядке документа встречается раньше, чем узел, связанный с $node2.

Логические операции (logical operator): and и or. Эти операции могут использоваться для объединения логических условий в предикате. Например, следующий предикат выбирает узлы item, имеющие ровно один элемент-потомок seller со значением , а также, по крайней мере, один элемент-потомок reserve-price с любым значением.

item [seller eq and reserve-price].

Отрицание (negation): not. Это скорее функция, а не операция. Она служит для инвертирования булевых величин.

Во всех приведенных примерах имена элементов и атрибутов были простыми идентификаторами. Однако в соответствии с рекомендацией XML Namespace, элементами и атрибутам позволяется иметь имена, состоящие из двух частей, где первая часть - префикс пространства имен, за которым следует двоеточие. Имя, имеющее префикс пространства имен, называется QName. Каждый префикс пространства имен должен быть связан с URI (универсальный идентификатор ресурсов), который уникальным образом определяет пространство имен. Это соглашение позволяет каждому приложению определять имена в своем собственном пространстве, не опасаясь коллизий с именами, определенными другими приложениями, что дает возможность однозначно ссылаться на имена, указываемые в различных приложениях. Если бы префикс auction был связан с URI пространства имен нашего приложения для проведения интерактивного аукциона, то шаг item [reserve-price > 1000] мог бы быть записан с помощью QName следующим образом:

auction:item [auction:reserve-price > 1000]

**Конструкторы элементов**.

Выражения пути - мощное средство, но им свойственно существенное ограничение: они способны выбирать только существующие узлы. В полном языке запросов необходимо наличие средства конструирования новых элементов и атрибутов, а также возможность указания их информационного наполнения и взаимосвязи. Это обеспечивается в XQuery с помощью вида выражения, называемого конструктором элементов (element constructor).

Простейший конструктор элементов создает элемент в полном соответствии с синтаксисом XML. Например, следующее выражение конструирует элемент с именем highbid, имеющий атрибут status и два элемента-потомка с именами itemno и bid-amount.

<highbid status = "pending">

<itemno>4871</itemno>

<bid-amount>250.00</bid-amount>

</highbid>

В этом примере значения элементов и атрибутов - константы. Однако во многих случаях необходимо создавать элемент или атрибут, значением которых является вычисляемое выражение. Выражение, заключенное в фигурные скобки, необходимо вычислить, а не трактовать как символьный текст. В конструкторе элемента это выражение вычисляется и заменяется своим значением. В следующем примере значения элементов и атрибутов вычисляются. Переменные $s, $i и $bids, используемые в этих выражениях, должны быть связаны с некоторыми выражениями.

<highbid status = "{$s}">

<itemno> {$i} </itemno>

<bid-amount>

{max($bids[itemno = $i]/bid-amount)}

</bid-amount>

</highbid>

В следующем примере конструктор элементов содержит выражение, заключенное в фигурные скобки, которое генерирует один атрибут и два подэлемента. Переменная $b должна быть связана с некоторым выражением.

<highbid>

{

$b/@status

$b/itemno

$b/bid-amount

}

</highbid>

Узел-элемент, созданный конструктором элемента, является новым узлом, обладающим собственной индивидуальностью. Если, как в приведенном примере, вновь созданный элемент имеет узлы-потомки и атрибуты, порожденные из существующих узлов, то новые узлы-потомки и атрибуты являются копиями узлов, из которых они были получены, но как узлы они индивидуальны.

В приведенных примерах конструкторов элементов, хотя содержимое элементов может быть вычисляемым, имя конструируемого элемента - известная константа. Однако иногда необходимо сконструировать элемент, имя которого, как и его содержимое, вычисляется. Для этого в XQuery определяется специальный вид конструктора, называемого вычисляемым конструктором элемента (computed element constructor). Он состоит из ключевого слова element, за которым следуют два выражения в фигурных скобках - первое вычисляет имя элемента, а второе - его содержимое.

Чтобы привести пример использования вычисляемого конструктора, предположим, что переменная $e связана с элементом, имеющим числовое значение. Нам нужно сконструировать новый элемент, имеющий то же имя, что и $e, и те же атрибуты, что у $e, но его значение должно быть вдвое больше значения $e. Этого можно добиться с помощью выражения, в котором функция data используется для получения числового значения исходного узла.

element

{name($e)}

{$e/@\*, data($e)\*2}

Подобно вычисляемому конструктору элемента, в XQuery обеспечивается вычисляемый конструктор атрибута (computed attribute constructor), состоит из ключевого слова attribute, за которым следуют два выражения в фигурных скобках - первое вычисляет имя атрибута, а второе - значение. Конструктор атрибута может использоваться везде, где допустим атрибут. Следующий конструктор атрибута на основе связанной переменной $p мог бы сгенерировать атрибут, который выглядит как father = или mother =.

attribute

{if $p/sex = "M" then "father" else "mother"}

{$p/name}

**Итерация и сортировка.**

Итерация - важная часть языка запросов. XQuery предлагает способ выполнять итерацию над последовательностью значений, по очереди связывая переменную с каждым значением и вычисляя выражения для каждого связывания переменной.

В наиболее простой форме итерация в XQuery задается оператором for, в котором указывается имя переменной и предоставляется последовательность значений, над которой переменная итерируется. Далее указывается оператор return, который содержит выражение, вычисляемое для каждого связывания переменной; см. ниже.

for $n in (2, 3) return $n + 1

Результатом этого итеративного выражения будет последовательность (3, 4).

В операторе for можно указывать более одной переменной с последовательностью итерации для каждой из них. Такой оператор порождает кортежи связываний переменных, которые образуют декартово произведение итерационных последовательностей. Если не указано иное, кортежи связываний генерируются в порядке, сохраняющем порядок итерационных последовательностей, с использованием самой левой переменной как <самый внешний цикл>, а самую правую - как <самый внутренний цикл>. В примере оператор for содержит две переменные и две итерационные последовательности.

for $m in (2, 3), $n in (5, 10)

return <fact>{$m} times {$n} is

{$m \* $n} </fact>

В результате получается следующая последовательность из четырех элементов.

<fact>2 times 5 is 10 </fact>

<fact>2 times 10 is 20 </fact>

<fact>3 times 5 is 15 </fact>

<fact>3 times 10 is 30 </fact>

Операторы for и let - частные случаи более общего выражения, называемого FLWOR. В наболее общем виде выражение FLWOR может иметь несколько операторов for, несколько операторов let, необязательный оператор where, необязательный оператор order by, а также оператор return. Функция операторов for и let - связывание переменных. Каждый из них содержит одну или несколько переменных и выражение, присваиваемое каждой переменной. Результатом вычисления выражений являются последовательности, и выражения могут содержать ссылки на переменные, для которых связывание было выполнено в предыдущих операторах. Оператор for итерирует каждую переменную над ассоциированной с ней последовательностью, связывая переменную по очереди с каждым объектом последовательности, а как оператор let связывает каждую переменную сразу со всей ассоциированной последовательностью. Это различие иллюстрируется следующей парой операторов.

for $i in (1 to 3)

let $j := (1 to $i)

Эта пара операторов не является полным выражением FLWOR, поскольку в нем отсутствует условие return. Операторы for и let просто порождают последовательность кортежей связываний. Приведенный выше пример порождает следующую последовательность из трех пар связываний.

$i = 1, $j = 1

$i = 2, $j = (1,2)

$i = 3, $j = (1,2,3)

В общем случае, число кортежей связывания, порождаемых серией операторов for и let, равняется произведению мощностей выражений итерации в операторах for. Оператор let при отсутствии оператора for, конечно, порождает только один кортеж связывания.

Кортежи связывания, порожденные операторами for и let в FLWOR-выражении, фильтруются в соответствии с необязательным условием where. Оператор where содержит выражение, которое вычисляется для каждого кортежа связывания. Если значением выражения where являются булевское значение true или непустая последовательность (<проверка существования>), то кортеж связываний принимается; в противном случае он отвергается.

Затем в выражении FLWOR вычисляется оператор return по очереди и по одному разу для каждого оставшегося после проверки условия where кортежа связывания в соответствии с порядком, заданном в операторе order by. Результаты вычислений объединяется в последовательность, которая и является результатом выражения FLWOR.

Возможности FLWR иллюстрируются в запросе к базе данных аукциона Q4.

(Q4) 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

order by $i/itemno

return

<popular-item>

{

$i/itemno,

$i/description,

<bid-count> {count

($b)}</bid-count>

}

</popular-item>

Операторы for и let порождают пару связывания для каждого item в items.xml. В каждой паре связывания $i связан с товаром, а $b - с последовательностью, содержащей все ставки для этого товара. Оператор where оставляет только те связанные кортежи, в которых $b содержит более десяти ставок. Затем оператор return для каждого из этих связываний генерирует выходной элемент, содержащий номер товара, описание и число ставок.

По умолчанию, порядок выходной последовательности выражения FLWR соответствует порядку итерационных последовательностей. Перед любым выражением может стоять префиксная операция unordered, указывающая, что порядок результата не имеет значения. Такое указание повышает гибкость реализации, позволяя оптимизировать вычисление выражения.

Конечно, каждое выражение упорядочивания должно возвращать единственный результат, и эти результаты должны быть сравнимы с помощью оператора gt. В случае применения условия sortby пустая последовательность может считаться либо больше любого другого значения, либо меньше любого другого значения - как то определит пользователь.

Условие sortby часто полезно для переупорядочивания результатов выражения FLWR. Если необходимо отсортировать по убыванию bid-count элементы popular-item, сгенерированные в запросе Q4, то в конец Q4 можно добавить следующий оператор.

sortby bid-count descending

Важно понимать, что sortby не является частью выражения FLWR, а представляет собой отдельный вид выражений XQuery, который может использоваться для переупорядочивания любой последовательности, вне зависимости от того, сгенерирована она выражением FLWR или нет. Однако если после выражения FLWR стоит sortby, интеллектуальный оптимизатор поймет, что переупорядочивание выходных объектов снимает обычные ограничения на порядок кортежей связываний.

Q4 показывает, как выражение FLWR может походить на запрос с соединением в системе управления реляционной базой данных, а также на запрос с группировкой. Q4 похож на запрос с соединением, поскольку в нем коррелируются элементы, находящиеся в двух разных файлах - items.xml и bids.xml. Он также напоминает запрос с группировкой, поскольку ставки группируются по номеру товара, и вычисляется число ставок в каждой группе.