**Использование XML совместно с SQL**

**XML и современные базы данных**

Алексей Ширшов

**Введение**

Этот материал посвящен тем нововведениям, которые появились в SQL и технологиях доступа к базам данных благодаря XML. Статья описывает базовые механизмы и возможности использования XML в MS SQL Server и ADO. Статья не претендует на фундаментальные исследования в этой области, так как XML и SQL Server вещи по природе своей необъятные. Кроме того, уровень поддержки XML в SQL Server постоянно увеличивается, и за всеми изменениями чрезвычайно трудно уследить. Например, после выхода SQLXML 3.0, стало возможным использовать SQL Server в качестве сервера Web-служб. К сожалению, эта тема в статье не освещается, но в будущем, возможно, найдется время и для нее.

Своим появлением статья обязана тому беспорядку, который был в голове у автора по данному вопросу.

**Поддержка XML в Microsoft SQL Server 2000**

Microsoft SQL Server 2000 содержит встроенные средства для работы с XML. Результирующую выборку можно представлять в формате XML с помощью ключевых слов for xml оператора select, а также сделать запрос из документа XML с помощью оператора OPENXML.

FOR XML

Этот оператор предназначен для представления результирующего набора строк в виде XML-документа. Рассмотрим его синтаксис:

|  |
| --- |
| [ FOR { BROWSE | XML { RAW | AUTO | EXPLICIT }  [ , XMLDATA ]  [ , ELEMENTS ]  [ , BINARY BASE64 ]  }  ] |

Назначение ключевого слова BROWSE не относится к теме нашей статьи.

FOR XML RAW – Каждая строка представляется в виде элемента <row/>. Название поля формирует название атрибута, а значение поля – значение атрибута.

FOR XML AUTO – Документ XML форматируется точно так же, как и при XML RAW, только название элемента, представляющего строку, заменяется на название таблицы.

FOR XML EXPLICIT – Самый сложный и гибкий вариант для создания XML-документов. В этом режиме можно формировать документы практически любой формы, однако для этого сам запрос должен быть написан по определенным правилам. Более подробно они рассматриваются ниже.

XMLDATA – Иногда бывает полезно получить не только сами данные, но и их схему. Схема данных также записывается в формате XML. Она определяет типы элементов и атрибутов, накладывает ограничения на их значения, и вообще представляет метаинформацию, позволяющую проверить документ на действительность (validity). Существует несколько разновидностей (форматов) схем данных. SQL Server использует XDR-схемы (XML Data Reduced). Подробную документацию по XDR можно найти в [1]. Ключевое слово XMLDATA может быть использовано для всех трех режимов формирования XML-документа (raw, auto и explicit).

ELEMENTS – Ключевое слово, использующееся только совместно с FOR XML AUTO. При его указании поля формируются как элементы: название поля соответствует названию элемента, а значение поля – значению элемента.

BINARY BASE64 – Определяет, как будут выведены двоичные данные (binary data).

|  |
| --- |
| ПРЕДУПРЕЖДЕНИЕ  SQL Server не позволяет использовать предикат GROUP BY совместно с FOR XML AUTO. |

Примеры

Для простоты и удобства будем использовать стандартную базу данных PUBS из поставки SQL Server 2000. Надо сказать, что Query Analyzer – не лучшее средство для просмотра XML-документов, т.к. результат он помещает в одну ячейку, как текстовое поле (или в одну строку, обрезая текст, при выводе результата в виде текста). Поэтому, если вы хотите испробовать все примеры сами, обратитесь к разделу IIS и XML-функции SQL Server.

Начнем с рассмотрения FOR XML RAW:

|  |
| --- |
| select au\_fname, au\_lname, address  from authors  where au\_fname like 'M%'  for xml raw |

Этот запрос возвращает имена всех авторов, начинающиеся с буквы M. Вот результаты в формате XML:

|  |
| --- |
| <row au\_fname="Marjorie" au\_lname="Green" address="309 63rd St. #411" />  <row au\_fname="Michael" au\_lname="O'Leary" address="22 Cleveland Av. #14" />  <row au\_fname="Meander" au\_lname="Smith" address="10 Mississippi Dr." />  <row au\_fname="Morningstar" au\_lname="Greene" address="22 Graybar House Rd." />  <row au\_fname="Michel" au\_lname="DeFrance" address="3 Balding Pl." /> |

Теперь заменим xml raw на xml auto:

|  |
| --- |
| <authors au\_fname="Marjorie" au\_lname="Green" address="309 63rd St. #411" />  <authors au\_fname="Michael" au\_lname="O'Leary" address="22 Cleveland Av. #14" />  <authors au\_fname="Meander" au\_lname="Smith" address="10 Mississippi Dr." />  <authors au\_fname="Morningstar" au\_lname="Greene" address="22 Graybar House Rd." />  <authors au\_fname="Michel" au\_lname="DeFrance" address="3 Balding Pl." /> |

Как видите, изменения невелики. Вместо названия элемента «row» подставляется имя таблицы. Теперь добавим к этому запросу ключевое слово ELEMENTS.

|  |
| --- |
| select au\_fname, au\_lname, address  from authors  where au\_fname like 'M%'  for xml auto, elements |

Вот результаты:

|  |
| --- |
| <authors>  <au\_fname>Marjorie</au\_fname>  <au\_lname>Green</au\_lname>  <address>309 63rd St. #411</address>  </authors>  <authors>  <au\_fname>Michael</au\_fname>  <au\_lname>O'Leary</au\_lname>  <address>22 Cleveland Av. #14</address>  </authors>  <authors>  <au\_fname>Meander</au\_fname>  <au\_lname>Smith</au\_lname>  <address>10 Mississippi Dr.</address>  </authors>  <authors>  <au\_fname>Morningstar</au\_fname>  <au\_lname>Greene</au\_lname>  <address>22 Graybar House Rd.</address>  </authors>  <authors>  <au\_fname>Michel</au\_fname>  <au\_lname>DeFrance</au\_lname>  <address>3 Balding Pl.</address>  </authors> |

Документ получился более громоздким: все поля представлены элементами.

С помощью ключевого слова XMLDATA можно получить документ со схемой данных.

|  |
| --- |
| select au\_fname, au\_lname, address  from authors  where au\_fname like 'M%'  for xml auto, xmldata |

Этот запрос вернет такой документ:

|  |
| --- |
| <Schema name="Schema1" xmlns="urn:schemas-microsoft-com:xml-data"  xmlns:dt="urn:schemas-microsoft-com:datatypes">  <ElementType name="authors" content="empty" model="closed">  <AttributeType name="au\_fname" dt:type="string" />  <AttributeType name="au\_lname" dt:type="string" />  <AttributeType name="address" dt:type="string" />  <attribute type="au\_fname" />  <attribute type="au\_lname" />  <attribute type="address" />  </ElementType>  </Schema>  <authors xmlns="x-schema:#Schema1"  au\_fname="Marjorie" au\_lname="Green" address="309 63rd St. #411" />  <authors xmlns="x-schema:#Schema1"  au\_fname="Michael" au\_lname="O'Leary" address="22 Cleveland Av. #14" />  <authors xmlns="x-schema:#Schema1"  au\_fname="Meander" au\_lname="Smith" address="10 Mississippi Dr." />  <authors xmlns="x-schema:#Schema1"  au\_fname="Morningstar" au\_lname="Greene" address="22 Graybar House Rd." />  <authors xmlns="x-schema:#Schema1"  au\_fname="Michel" au\_lname="DeFrance" address="3 Balding Pl." /> |

Как видите, теперь документу предшествует ХDR-схема, позволяющая проверить правильность его содержимого.

FOR XML EXPLICIT

В данном режиме можно формировать практически любые документы; структура результирующего XML-документа определяется непосредственно в самом запросе. Запрос может принимать очень сложный вид, ниже будут рассмотрены правила, по которым он составляется.

Первое поле всегда называется tag и представляет собой числовой идентификатор таблицы. Вы можете указывать для него любые числовые значения. В результирующем документе он не появляется, и нужен только, чтобы определить связи между таблицами для формирования иерархического документа. Второе поле называется parent и тоже может представлять любой числовой идентификатор. Он определяет родительскую таблицу для данной таблицы и в результирующем документе также не присутствует. Если родитель отсутствует, указывается 0 или NULL. Для иерархических выборок нужно использовать несколько запросов, объединенных с помощью оператора UNION ALL.

После двух обязательных полей tag и parent следуют поля, которые требуется выбрать из таблицы. Для них должен быть задан псевдоним, определяющий тип XML-узла, его название и другую информацию. Вот синтаксис этого псевдонима:

|  |
| --- |
| ElementName!TagNumber!AttributeName!Directive |

Рассмотрим назначение каждой из частей этого псевдонима:

ElementName – имя элемента, в котором будет находится элемент данного поля. Обычно здесь указывается имя таблицы.

TagNumber – идентификатор таблицы, в которой находится данное поле. Одному и тому же ElementName всегда должен соответствовать один TagNumber. Он также не отображается ни в каком виде в результирующем документе, однако нужен для определения иерархических связей между таблицами.

AttributeName – имя атрибута (или элемента), представляющего данное поле.

Directive – по существу, представляет собой тип узла. Может принимать следующие значения:

|  |  |
| --- | --- |
| element | поле представляется в виде элемента |
| xml | То же самое, что element, но не выполняет трансформации текста. Например, символ меньше (<) не превращается в ссылку &lt; |
| cdata | значение поля обрамляется соответствующим образом. Имя атрибута указывать совместно с этим ключевым словом нельзя. |
| hide | позволяет скрыть поле |
| id,idref,idrefs | позволяют делать ссылки на другие элементы. Эти атрибуты имеют значение, только если создается схема данных. |

Рассмотрим небольшой пример. В самом начале мы рассматривали запрос с использованием FOR XML RAW:

|  |
| --- |
| select au\_fname,au\_lname,address from authors where au\_fname like 'M%' for xml raw |

Перепишем его с использованием FOR XML EXPLICIT:

|  |
| --- |
| select 1 as tag,  0 as parent,  au\_fname as 'authors!1!fname',  au\_lname as 'authors!1!lname',  address as 'authors!1!address'  from authors where au\_fname like 'M%'  for xml explicit |

Результат будет точно таким же. Немного исправим запрос для демонстрации возможностей директивы FOR XML EXPLICIT.

|  |
| --- |
| select 1 as tag,  0 as parent,  au\_fname as 'authors!1!fname',  au\_lname as 'authors!1!lname!element',  address as 'authors!1!!cdata'  from authors where au\_fname like 'M%'  for xml explicit |

Вот результат:

|  |
| --- |
| <authors lname="Green">  <fname>Marjorie</fname>  <![CDATA[ 309 63rd St. #411 ]]>  </authors>  <authors lname="O'Leary">  <fname>Michael</fname>  <![CDATA[ 22 Cleveland Av. #14 ]]>  </authors>  <authors lname="Smith">  <fname>Meander</fname>  <![CDATA[ 10 Mississippi Dr. ]]>  </authors>  <authors lname="Greene">  <fname>Morningstar</fname>  <![CDATA[ 22 Graybar House Rd. ]]>  </authors>  <authors lname="DeFrance">  <fname>Michel</fname>  <![CDATA[ 3 Balding Pl. ]]>  </authors> |

Неплохо для одного запроса! Поскольку для поля au\_lname указан атрибут element, оно представлено в виде элемента. Адрес находится в секции CDATA.

Теперь рассмотрим, как формировать иерархические документы. Возьмем такой запрос:

|  |
| --- |
| select pub\_name,city,fname,lname  from publishers as p  join employee as e on p.pub\_id = e.pub\_id  where pub\_name like 'Binnet%' or pub\_name like 'New Moon%'  order by pub\_name,city,fname,lname |

Он возвращает имена всех служащих издательств Binnet & Hardley и New Moon Books. Результат запроса (20 записей) я приводить не буду, отмечу только, что он содержит большое количество повторяющихся названий издательств, т.к. результат представлен в реляционной форме. Мы же хотим получить следующее:

|  |
| --- |
| <pubs PubName="Binnet & Hardley" City="Washington">  <employee First\_Name="Anabela" Last\_Name="Domingues" />  <employee First\_Name="Daniel" Last\_Name="Tonini" />  <employee First\_Name="Elizabeth" Last\_Name="Lincoln" />  <employee First\_Name="Helen" Last\_Name="Bennett" />  <employee First\_Name="Lesley" Last\_Name="Brown" />  <employee First\_Name="Martine" Last\_Name="Rance" />  <employee First\_Name="Paolo" Last\_Name="Accorti" />  <employee First\_Name="Paul" Last\_Name="Henriot" />  <employee First\_Name="Peter" Last\_Name="Franken" />  <employee First\_Name="Victoria" Last\_Name="Ashworth" />  </pubs>  <pubs PubName="New Moon Books" City="Boston">  <employee First\_Name="Gary" First\_Name="Thomas" />  <employee First\_Name="Howard" First\_Name="Snyder" />  <employee First\_Name ="Karin" First\_Name="Josephs" />  <employee First\_Name ="Laurence" Last\_Name="Lebihan" />  <employee First\_Name ="Martin" Last\_Name="Sommer" />  <employee First\_Name ="Mary" Last\_Name="Saveley" />  <employee First\_Name ="Matti" Last\_Name="Karttunen" />  <employee First\_Name ="Palle" Last\_Name="Ibsen" />  <employee First\_Name ="Roland" Last\_Name="Mendel" />  <employee First\_Name ="Timothy" Last\_Name="O'Rourke" />  </pubs> |

И как, спросите вы? Примерно так:

|  |
| --- |
| select 1 as tag, -- первый подзапрос  0 as parent,  pub\_name as 'pubs!1!PubName',  city as 'pubs!1!City',  NULL as 'employee!2!First\_Name',  NULL as 'employee!2!Last\_Name'  from publishers as pubs  where pub\_name like 'Binnet%' or pub\_name like 'New Moon%'  union all select 2 as tag, -- второй подзапрос  1 as parent,  pubs.pub\_name,  pubs.city,  fname,  lname  from employee as e, publishers as pubs  where (pub\_name like 'Binnet%' or pub\_name like 'New Moon%')  and pubs.pub\_id = e.pub\_id  order by 'pubs!1!PubName', 'pubs!1!City',  'employee!2!First\_Name', 'employee!2!Last\_Name'  for xml explicit |

Давайте рассмотрим все по порядку. Сначала выполняется первый подзапрос. Его результат приведен в таблице 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| tag | parent | pubs!1!PubName | pubs!1!City | employee!2!First\_Name | employee!2!Last\_Name |
| 1 | 0 | New Moon Books | Boston | NULL | NULL |
| 1 | 0 | Binnet & Hardley | Washington | NULL | NULL |

Таблица 1.

Затем второй (Таблица 2).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| tag | parent | pub\_name | city | fname | lname |
| 2 | 1 | Binnet & Hardley | Washington | Paolo | Accorti |
| 2 | 1 | Binnet & Hardley | Washington | Victoria | Ashworth |
| 2 | 1 | Binnet & Hardley | Washington | Helen | Bennett |
| 2 | 1 | Binnet & Hardley | Washington | Lesley | Brown |
| ... | ... | ... | ... | ... | ... |

Таблица 2.

Затем происходит сортировка, и на основе полей tag и parent SQL Server формирует иерархический XML документ.

|  |
| --- |
| ПРИМЕЧАНИЕ  Для отладки подобных запросов лучше не указывать оператор FOR XML EXPLICIT. Тогда данные будут представлены в обычной реляционной форме. |

На этом мы, пожалуй, закончим рассмотрение оператора FOR XML EXPLICIT – приведение примеров использования всех атрибутов заняло бы слишком много места.

OPENXML

Функция OPENXML является аналогом OPENROWSET, OPENDATASOURCE и OPENQUERY, которые позволяют выполнять запросы из удаленных источников. Вот ее синтаксис:

|  |
| --- |
| OPENXML(idoc int [in],rowpattern nvarchar[in],[flags byte[in]])  [WITH (SchemaDeclaration | TableName)] |

Аргументы:

idoc – хендл XML-документа, полученный при помощи хранимой процедуры sp\_xml\_preparedocument;

rowpattern – локализуемая группа XPath или, проще говоря, XPath-выражение;

flags – набор флагов, указывающих на то, как должны быть сопоставлены данные документа XML и реляционного набора строк;

ShemaDeclaration – определение полей реляционного набора строк в формате:

|  |
| --- |
| ColName ColType [ColPattern | MetaProperty] |

Где

ColName – имя поля.

ColType – тип поля. Допускаются все типы SQL Server.

ColPattern - локализуемая группа XPath для поля.

MetaProperty – метасвойство. Его мы рассматривать не будем.

XML-документ подготавливается с помощью хранимой процедуры sp\_xml\_preparedocument. Процедура использует анализатор MSXML для проверки документа на правильность и возвращает хендл документа. После завершения работы с OPENXML хендл нужно закрыть с помощью процедуры sp\_xml\_removedocument.

|  |
| --- |
| ПРИМЕЧАНИЕ  sp\_xml\_preparedocument подготавливает XML-документ, представляя его в виде объектной модели DOM (Document Object Model). Если вы работаете с большими документами, это может вызвать некоторые проблемы. |

Как видно из синтаксиса, вы можете не указывать флаги и определения полей для реляционного набора строк. В этом случае SQL Server создаст внутреннее представление XML-документа в так называемом "edge table"-формате. Он практически не читаем, однако при большом желании его можно использовать. Описание этого формата выходит за рамки данной статьи, но в качестве доказательства того, что с ним можно работать, приведу пример. Пусть у нас имеется такой XML-документ:

|  |
| --- |
| <?xml version="1.0" encoding="windows-1251" ?>  <rsdn>  <forums date="09.01.03">  <forum name="WinAPI" totalposts="16688"  description="Системное программирование">  <moderators/>  <top-poster>Alex Fedotov</top-poster>  </forum>  <forum name="COM" totalposts="10116"  description="Компонентные технологии">  <moderators/>  <top-poster>Vi2</top-poster>  </forum>  <forum name="Delphi" totalposts="5001"  description="Delphi и Builder">  <moderators>  <moderator name="Sinclair"/>  <moderator name="Hacker\_Delphi"/>  </moderators>  <top-poster>Sinclair</top-poster>  </forum>  <forum name="DB" totalposts="6606" description="Базы данных">  <moderators>  <moderator name="\_MarlboroMan\_"/>  </moderators>  <top-poster>Merle</top-poster>  </forum>  </forums>  </rsdn> |

Вот запрос, возвращающий общее количество сообщений для каждого форума:

|  |
| --- |
| exec sp\_xml\_preparedocument @hdoc out, @\_xmlbody  select [text] as totalposts  from openxml(@hdoc,'/rsdn/forums/forum') as f  join (select [id],localname \  from openxml(@hdoc,'/rsdn/forums/forum')  where localname = 'totalposts') as d on d.[id] = f.parentid  exec sp\_xml\_removedocument @hdoc |

Результатом его будет следующая таблица:

|  |
| --- |
| totalposts |
| 16688 |
| 10116 |
| 5001 |
| 6606 |

Не советую использовать подобный метод в рабочих проектах, и не только потому, что он неэффективен (как видно из примера, XML-документ сканируется дважды). Рассмотрим пример, выдающий тот же самый результат с использованием XPath.

|  |
| --- |
| exec sp\_xml\_preparedocument @hdoc out, @\_xmlbody  select \*  from openxml(@hdoc,'/rsdn/forums/forum')  with(totalposts varchar(100) 'attribute::totalposts')  exec sp\_xml\_removedocument @hdoc |

Здесь, чтобы ограничить реляционный набор строк, я воспользовался XPath-выражением.

Выражение attribute::totalposts означает, что для поля totalposts будет использоваться значение одноименного атрибута. Гораздо чаще в XPath-выражениях используется сокращенная запись:

«attribute::» можно заменить символом @;

«self::node()» можно заменить на точку (.);

«parent::node()» можно заменить на две точки (..).

Другие сокращения можно найти в спецификации XPath.

Давайте рассмотрим более сложный пример: выберем название форума, модератора и дату создания статистики для всех форумов, у которых больше 6000 сообщений.

|  |
| --- |
| exec sp\_xml\_preparedocument @hdoc out, @\_xmlbody  select  forum as 'Форум',  case when moders is null  then 'нет'  else moders  end as 'Модератор',  [date] as 'Дата создания'  from openxml(@hdoc,'/rsdn/forums/forum[attribute::totalposts > "6000"]') with  (  moders varchar(50) 'moderators/moderator/attribute::name',  forum varchar(50) 'attribute::name',  [date] varchar(50) 'parent::node()/attribute::date'  )  exec sp\_xml\_removedocument @hdoc |

Часть запроса, использующую XPath, можно переписать в сокращенной форме :

|  |
| --- |
| openxml(@hdoc,'/rsdn/forums/forum[@totalposts > "6000"]') with  (  moders varchar(50) 'moderators/moderator/@name',  forum varchar(50) '@name',  [date] varchar(50) '../@date'  ) |

Везде далее я буду пользоваться сокращенной записью.

Чтобы разобраться с флагами OPENXML, рассмотрим слегка модифицированный пример из MSDN:

|  |
| --- |
| DECLARE @idoc int  DECLARE @doc varchar(1000)  SET @doc ='  <root>  <Customer cid= "C1" city="Issaquah">  <name>Janine</name>  <Order oid="O1" date="1/20/1996" amount="3.5" />  <Order oid="O2" date="4/30/1997" amount="13.4">  Customer was very satisfied  </Order>  </Customer>  <Customer cid="C2" city="Oelde" >  <name>Ursula</name>  <Order oid="O4" date="1/20/1996" amount="10000">Happy Customer.</Order>  <Order oid="O3" date="7/14/1999" amount="100"  note="Wrap it blue white red">  Sad Customer.  <Urgency>Important</Urgency>  </Order>  </Customer>  </root>'  -- Создание внутреннего представления XML-документа.  EXEC sp\_xml\_preparedocument @idoc OUTPUT, @doc  SELECT \*  FROM OPENXML (@idoc, '/root/Customer', 2) WITH  (  cid char(5) '@cid',  [name] varchar(20),  oid char(5) 'Order/@oid',  amount float 'Order/@amount',  comment varchar(100) 'Order/text()'  )  -- Очистка  EXEC sp\_xml\_removedocument @idoc |

Результат будет следующим:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| cid | name | oid | amount | comment |
| C1 | Janine | O1 | 3.5 | Customer was very satisfied |
| C2 | Ursula | O4 | 10000.0 | Happy Customer. |

Отметим некоторые особенности:

В качестве режима отображения XML-данных на поля реляционной таблицы использовалось значение 2 (element-centric mapping). Это означает, что по умолчанию имена колонок получаемой реляционной таблицы будут соответствовать именам вложенных XML-элементов. Кроме этого, возможно использование значений 0, 1 и 8. 0 используется по умолчанию и означает использование attribute-centric mapping. 1, как ни странно, означает то же самое. Флаги 1 и 2 можно комбинировать по "или", т.е. если подставить 3, сначала будет произведена попытка найти атрибут с именем, соответствующим имени колонки, а затем (если атрибут не найден) будет произведен поиск элемента с соответствующим именем (иначе будет возвращен NULL). Благодаря тому, что в качестве флага было указано значение 2, для поля cid пришлось явно указать XPath-запрос, указывающий, что на эту колонку отображается атрибут cid. Для поля name не потребовалось непосредственного указания XPath-выражения. Если бы в качестве флага использовалось значение 1 (использование отображения атрибутов), то картина изменилась бы на противоположную: т.е. для cid не нужно бы было ничего указывать, а для name пришлось бы написать шаблон (т.е. просто выражение ‘name’).

Особенность применения XPath-выражений при отображении данных состоит в том, что возможна выборка данных, расположенных практически в любой части XML-документа (относительно текущей ветки). Так, можно обратиться к подветкам текущей ветки, родительским веткам, и даже получить данные на основе выполнения некоторого условия. Если бы вместо «comment varchar(100) 'Order/text()'» было написано «comment varchar(100) 'Order'», то колонка comment первой строки содержала бы пустую строку. Она бралась бы из первого заказа (O1). Но так как текста в этом элементе нет, функция text() возвратит для него false, что приведет к поиску текста в следующем по порядку элементе Order (заказе O2). Таким образом, в сформированной записи будет находиться информация из первого заказа и комментарий из второго. Прикладного смысла это действие не имеет, но замечательно демонстрирует гибкость техники отображения.

На этом мы с вами закончим рассмотрение конструкции OPENXML. Более подробную информацию можно получить в MSDN. Спецификацию XPath можно найти в [2].

IIS и XML функции SQL Server

Чтобы выполнять запросы к SQL Server через HTTP, необходимо настроить соответствующим образом интернет-сервер. Делается это с помощью мастера "Configure SQL XML Support in IIS". Я не буду описывать полностью его работу, при необходимости можете обратиться к [3]. Кроме этого, настроить виртуальный каталог можно программно с помощью объекта VDirMgr. Он находится в файле sqlvdr3.dll. Для использования класса из VB нужно добавить ссылку на библиотеку типов Microsoft SQL Virtual Directory Control 1.0 Type Library.

Мастер "Configure SQL XML Support in IIS" создает виртуальный каталог, для обработки запросов к которому назначается специальное isapi-расширение: sqlisapi.dll. Эта библиотека, используя провайдер SQL OLEDB, связывается с SQL Server для отправки запросов и получения результатов. Результирующие выборки, представленные уже в формате XML, передаются обратно вызывающей стороне по HTTP. С помощью мастера вы можете указать:

Учетную запись SQL Server или Windows, под которой будут выполняться все запросы;

Компьютер, на котором расположен SQL Server и базу данных;

Подкаталоги данного виртуального каталога для хранения различных типов файлов (шаблонов, схем). Подкаталоги могут быть трех предопределенных типов: schema, template и dbobject. В подкаталоге schema хранятся XDR или XSD схемы данных, которые можно непосредственно исполнять в URL-запросе. В подкаталоге с типом template хранятся шаблоны, исполнение которых разрешено через URL-запросы. dbobject – псевдокаталог, его мы рассматривать не будем.

|  |
| --- |
| ПРИМЕЧАНИЕ  В SQLXML 3.0 появился еще один тип подкаталогов – soap. Его рассмотрение также выходит за рамки данной статьи. |

Отдельно остановимся на настройках каталога. Вы можете:

Позволить или запретить указывать SQL-инструкции непосредственно в URL. Отмечу, что в качестве инструкций можно использовать не только запросы, но и любые другие операторы. Советую устанавливать эту опцию только для отладки, а в нормальном режиме работы выключать.

Позволить или запретить исполнение запросов, хранящихся в специальных шаблонах. Подробнее о шаблонах будет сказано далее.

Позволить или запретить использование запросов XPath;

Позволить или запретить использовать метод POST.

URL-запросы

Рассмотрим синтаксис URL-запроса к SQL Server:

|  |
| --- |
| http://iisserver/vroot?sql=sqlinstruction[&param=value[&param=value]...n] |

Здесь:

iisserver – имя интернет-сервера;

vroot – имя виртуального каталога;

sqlinstruction – любая SQL-инструкция;

param – имя параметра. Это не параметр SQL-инструкции или хранимой процедуры, это параметр шаблона или один из следующих предопределенных параметров: contenttype, outputencoding, root и xsl;

value – значение параметра.

Итак, предположим, вы сконфигурировали виртуальный каталог для использования базы данных PUBS и назвали его server\_pubs. Положим, ваш компьютер называется server. Попробуем написать первый URL-запрос:

|  |
| --- |
| http://server/server\_pubs/?sql=select au\_fname, au\_lname, address from authors where au\_fname like 'M%' for xml raw |

Но не все так просто. Ответ будет таким:

|  |
| --- |
| XML document must have a top level element. |

Первый блин, как всегда, комом! Дело в том, что XML-документ, формируемый SQL Server’ом, не имеет главного корневого элемента, без которого документ не может считаться правильно оформленным. Для указания корневого элемента нужно добавить параметр root.

|  |
| --- |
| http://server/server\_pubs/?sql=select au\_fname,au\_lname,address from authors where au\_fname like 'M%' for xml raw&root=my\_root |

В ответ будет выдано:

|  |
| --- |
| Incorrect syntax near 'M'. |

Что ж, опытные пользователи, наверное, сразу бы приметили знак процента в запросе. Он является зарезервированным символом в имени URL, его код равен 25. Учитывая это, перепишем запрос так:

|  |
| --- |
| http://server/server\_pubs/?sql=select au\_fname,au\_lname,address from authors where au\_fname like 'M%25' for xml raw&root=my\_root |

Ура! Получилось. Результат будет примерно таким же, как в самом первом примере этой статьи.

На случай, если вам нужно получить результаты в виде обычного HTML, можно создать шаблон преобразования на языке XSL и указать еще один параметр в URL – xsl. В качестве значения параметра нужно указать путь относительно выбранной вами виртуальной директории.

Составим шаблон трансформации:

|  |
| --- |
| <?xml version="1.0" ?>  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  version="1.0">    <xsl:template match = "\*">  <xsl:apply-templates />  </xsl:template>  <xsl:template match = "row">  <li>  <table><tr>  <td><xsl:value-of select = "attribute::au\_fname" /></td>  <td><xsl:value-of select = "attribute::au\_lname" />.</td>  <td>Address: <xsl:value-of select = "attribute::address" /></td>  </tr></table>  </li>  </xsl:template>  <xsl:template match = "/">  <html>  <body>  <ul>  <xsl:apply-templates select = "my\_root" />  </ul>  </body>  </html>  </xsl:template>  </xsl:stylesheet> |

Более подробную информацию о XSL можно найти в [4].

В ответ на следующий URL-запрос вы получите преобразованный XML-документ.

|  |
| --- |
| http://server/server\_pubs/?sql=select au\_fname,au\_lname,address from authors where au\_fname like 'M%25' for xml raw&root = my\_root&xsl = xsl\_for\_query.xsl |

На самом деле работа с URL-запросами интересна только первые несколько минут. Действительно, очень неудобно возиться со строкой адреса и бесконечными символами процента в ней, к тому же URL-запросы не очень хороши в смысле безопасности. Альтернативой им являются шаблоны.

Шаблоны

Шаблоны в контексте этой статьи являются обычными XML-документами, составленными по определенным правилам. Основным их содержанием является SQL-запрос или вызов хранимой процедуры. Шаблоны также используются для запросов XPath, однако их я коснусь чуть позже. Параметры шаблона задаются в URL-строке. Если они там не указаны, берутся значения по умолчанию из соответствующих тегов <param>.

Шаблоны хранятся на сервере, поэтому в смысле безопасности их использование намного предпочтительнее, чем URL-запросов. Чтобы ISAPI-расширение, которое их обрабатывает, поняло, что вы вызываете шаблон, его нужно поместить в свой виртуальный каталог. Обычно он называется template. Настроить его можно в упоминавшемся ранее мастере "Configure SQL XML Support in IIS" на закладке Virtual Names.

Структура шаблона выглядит так:

|  |
| --- |
| <?xml version="1.0" ?>  <your\_root xmlns:sql="urn:schemas-microsoft-com:xml-sql"  sql:xsl="xsl file name">  <xql:header>  <xql:param name="your\_param\_name"> param\_value </sql:param>  <xql:param name="your\_param\_name"> param\_value </sql:param>...n  </xql:header>  <sql:query>  любое SQL-выражение  </sql:query>  </your\_root> |

Для форматирования результатов исполнения шаблона может быть использована XSL-трансформация. Для этого необходимо задать атрибут xsl, значение которого есть относительный или полный путь до файла, содержащего шаблон трансформации. Атрибут xsl необязателен, как и раздел header.

Вот как будет выглядеть шаблон, основанный на изрядно уже поднадоевшем вам запросе.

|  |
| --- |
| <?xml version="1.0" ?>  <my\_root xmlns:sql="urn:schemas-microsoft-com:xml-sql"  sql:xsl="xsl\_for\_query.xsl">  <sql:query>  select au\_fname,au\_lname,address  from authors where au\_fname like 'M%' for xml raw  </sql:query>  </my\_root> |

В нем используется та же схема преобразования, что и в предыдущем примере. Теперь попробуйте его вызвать (предположим, вы его сохранили под именем first\_template.xml):

|  |
| --- |
| http://server/server\_pubs/template/first\_template.xml |

Результат получается довольно странным: IE представляет HTML-документ (получаемый при трансформации) в формате XML. Делает он это на вполне законных основаниях, и чтобы результат выдавался все-таки в формате HTML, нужно явно указать тип выходного потока. Это легко сделать с помощью параметра contenttype.

|  |
| --- |
| http://server/server\_pubs/template/first\_template.xml?contenttype=text/html |

Достоинства шаблонов очевидны:

Так как шаблон находится на сервере, вы полностью контролируете его содержимое;

Шаблоны намного проще в использовании;

Тело шаблона и используемая схема преобразования скрыты от пользователя;

Шаблоны можно динамически изменять или создавать, что придает интернет-серверу дополнительную гибкость;

С помощью шаблонов можно выполнять запросы XPath, но об этом уже в следующем разделе.

В шаблонах можно использовать следующие атрибуты (рассмотрены наиболее используемые):

client-side-xml – Булева переменная, принимающая значение 0 или 1. Если указывается 1, то при выборке используется форматирование XML-документа на клиенте. Т.е. SQL Server выполняет обычный запрос, передает рекордсет клиенту, и уже там производится формирование документа. Более подробно клиентские курсоры будут рассмотрены в разделе ADO и XML.

|  |
| --- |
| ПРИМЕЧАНИЕ  В данном случае клиентом является компьютер, откуда поступает запрос к SQL Server-y, то есть машина, где расположен SQLXMLOLEDB-провайдер. В случае использования ADO – это машина клиента. В случае использования шаблонов – сервер IIS. |

nullvalue – позволяет задавать строку, которая в URL-запросе и запросе XPath будет означать NULL.

is-xml – атрибут параметра, принимающий значение 0 и 1, используется в разделе header. По умолчанию он равен 1. Это означает, что значение параметра интерпретируется как фрагмент xml, поэтому, например, &lt; не заменяется. Если задано значение 0, параметр интерпретируется как обычный текст.

Назначение других атрибутов можно найти в MSDN.

Запросы XPath

XPath не рассчитан на работу с реляционными данными. Чтобы использовать XPath-запросы для выборки реляционных данных, необходимо создать схему данных XDR или XSD. XDR была разработана несколько лет назад при активном участии Microsoft, т.к. в то время необходимость в схемах данных была, а, по существу, самих схем не было. С появлением XSD популярность и актуальность применения XDR начали падать.

|  |
| --- |
| ПРИМЕЧАНИЕ  Спецификацию XSD можно найти в [5]. |

Схема данных выполняет две важные функции: задает структуру будущего XML-документа и определяет, какие поля и таблицы должны использоваться при выполнении запроса XPath. Такие схемы называются аннотированными схемами запросов, а атрибуты, связывающие объекты базы данных с XML-узлами – аннотациями. До выхода в свет SQLXML 2.0 можно было использовать только аннотированные схемы на основе SDR [6]. Однако сейчас лучше использовать аннотированные схемы на основе спецификации XSD [7]. Некоторую информацию по преобразованию схем из XDR в более новый формат XSD можно найти в [8].

Вот синтаксис шаблонов с использованием запросов XPath:

|  |
| --- |
| <your\_root xmlns:sql="urn:schemas-microsoft-com:xml-sql">  <sql:header>  <xql:param name="your\_param\_name"> param\_value </sql:param>  <xql:param name="your\_param\_name"> param\_value </sql:param>...n  </sql:header>  <sql:xpath-query mapping-schema="your\_schema.xml">  XPath query  </sql:xpath-query>  </your\_root> |

В этом примере аннотированная схема должна находится в файле your\_schema.xml. Как видно из синтаксиса, возможно создание параметризированных запросов XPath. Параметр в запросе обозначается начальным символом $.

Рассмотрим пример аннотированной схемы XDR, который будет использоваться для запросов XPath. В результирующем документе будут присутствовать имена, фамилии и адреса всех авторов:

|  |
| --- |
| <?xml version="1.0" ?>  <Schema xmlns="urn:schemas-microsoft-com:xml-data"  xmlns:sql="urn:schemas-microsoft-com:xml-sql"  xmlns:dt="urn:schemas-microsoft-com:datatypes">  <ElementType name="address" />  <ElementType name="Authors" sql:relation="Authors">  <AttributeType name="au\_fname" dt:type="string" />  <AttributeType name="au\_lname" dt:type="string" />  <attribute type="au\_fname"/>  <attribute type="au\_lname" />  <element type="address" sql:field="address"/>  </ElementType>  </Schema> |

Здесь используется аннотация relation для того, чтобы указать, с какой таблицей будет связан элемент Authors. Дочерние элементы наследуют связь с таблицей, указанной для родительского ElementType. Связывание полей таблицы или представления (view) можно выполнять явно, с использованием аннотации field. В данном примере для элементов AttributeType этого делать не нужно, т.к. отображения на соответствующие поля выполняются автоматически. Однако для дочерних элементов ElementType, которые по умолчанию связываются с таблицами, такая аннотация может быть необходима. Наиболее часто используемые аннотации приведены далее.

Теперь можно перейти к самому шаблону. Предположим, аннотированную схему вы сохранили под именем MySchema.xml.

|  |
| --- |
| ПРИМЕЧАНИЕ  IIS различает тип XML-документа только на основе каталога, где он находится. Даже если мы не настроили специальным образом IIS на исполнение схем, их лучше хранить в одном месте. Я рекомендую хранить схемы и шаблоны в разных виртуальных каталогах. Например, template для шаблонов, schema – для схем. |

Вот так выглядит шаблон, выбирающий имена, фамилии и адреса всех авторов:

|  |
| --- |
| <my\_root xmlns:sql="urn:schemas-microsoft-com:xml-sql">  <sql:xpath-query mapping-schema="../Schema/MySchema.xml">  /Authors  </sql:xpath-query>  </my\_root> |

Так как схемы XDR постепенно вытесняются схемами XSD, перепишем пример с использованием XSD.

|  |
| --- |
| ПРЕДУПРЕЖДЕНИЕ  Я полтора дня потерял, когда первый раз пытался выполнить запрос XPath на XSD-схеме. ISAPI-расширение упорно выдавало ошибку «XPath: unable to find /authors in the schema». В конце концов, после непродолжительных консультаций с одним из участников форума сайта www.sql.ru, проблема была решена. Суть ее в следующем: при создании виртуального каталога я использовал оснастку mmc SQL IIS Admin.MSC, которая входит в стандартный комплект MS SQL Server’а и ничего не знает о новых возможностях SQLXML 3.0. Новая оснастка лежит в %Program Files%\SQLXML 3.0 и называется sqlisad3.msc. Ее можно запустить из меню Start->Programs->SQLXML 3.0->Configure IIS Support. Всегда пользуйтесь только ею. |

Вот список основных отличий XDR от XSD[9], к которому я очень часто обращаюсь:

|  |  |
| --- | --- |
| XDR | XSD |
| Schema | schema |
| ElementType | element |
| AttributeType | attribute |
| attribute | none |

С учетом этого схема будет выглядеть так:

|  |
| --- |
| <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  xmlns:sql="urn:schemas-microsoft-com:mapping-schema">  <xsd:element name="authors" sql:relation="authors">  <xsd:complexType>  <xsd:attribute name="au\_fname" sql:field="au\_fname" />  <xsd:attribute name="au\_lname" sql:field="au\_lname" />  <xsd:attribute name="address" sql:field="address" />  </xsd:complexType>  </xsd:element>  </xsd:schema> |

Здесь явно указаны аннотации, позволяющие связать XML-элементы с таблицей authors и соответствующими полями. В данном случае все они не обязательны, т. к. SQLXML может вывести связи из названий узлов. Вот пример, где аннотации действительно необходимы. Для разнообразия адрес и фамилия вынесены в отдельные элементы:

|  |
| --- |
| <?xml version="1.0" encoding="windows-1251" ?>  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  xmlns:sql="urn:schemas-microsoft-com:mapping-schema">  <xsd:element name="Авторы" sql:relation="authors">  <xsd:complexType>  <xsd:sequence>  <xsd:element name="Фамилия" sql:field="au\_lname"/>  <xsd:element name="Адрес" sql:field="address" />  </xsd:sequence>  <xsd:attribute name="Имя" sql:field="au\_fname"/>  </xsd:complexType>  </xsd:element>  </xsd:schema> |

Если схема находится в виртуальном каталоге, тип которого schema, вы можете выполнять XPath-запросы, непосредственно указывая их в URL. Результирующий документ может не иметь корневого элемента, поэтому не забывайте указывать параметр root.

|  |
| --- |
| http://server/server\_pubs/schema/xsd\_map\_schema.xml/Авторы?root=root |

Вот другие аннотации, часто используемые в схемах:

relationship – элемент, обозначающий связь с соответствующими друг другу первичными и внешними ключами. Используется для построения иерархических XML-документов.

is-constant – указывает, что соответствующий элемент не связывается с таблицей или колонкой базы данных.

map-field – булева переменная, принимающая значение 0 или 1. Значение 0 указывает, что соответствующий элемент не будет присутствовать в результирующем документе. С выходом SQLXML 3.0 аннотация была переименована в mapped.

limit-field и limit-value – предназначены для указания поля и его значения, по которым будет фильтроваться запрос к базе данных. Результат использования этих аннотаций аналогичен ограничению результирующего набора строк с помощью оператора WHERE.

use-cdata – указывает, что соответствующий XML-узел будет представлен в секции CDATA результирующего документа. Узел должен быть связан с полем таблицы или представления и не может применяться совместно с url-encode или ID, IDREF, IDREFS, NMTOKEN и NMTOKENS.

hide – булева переменная, принимающая значение 0 или 1. Указывает на то, что соответствующий XML-узел будет скрыт (значение 0) в результирующем XML-документе, однако может быть использован в запросе XPath. Не путайте её с аннотацией mapped. Различие в том, что при помощи mapped XML-узел совсем исключается из документа, так что не может быть использован в запросе XPath.

Рассмотрим пример, демонстрирующий работу некоторых из перечисленных аннотаций. В разделе FOR XML EXPLICIT был составлен иерархический список издательств и служащих, которые в них работают. Попробуем получить такой же XML-документ с помощью аннотированной схемы:

|  |
| --- |
| <?xml version="1.0" ?>  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  xmlns:sql="urn:schemas-microsoft-com:mapping-schema">  <xsd:annotation>  <xsd:appinfo>  <sql:relationship name="PubsEmployees"  parent="publishers" parent-key="pub\_id"  child="employee" child-key="pub\_id" />  </xsd:appinfo>  </xsd:annotation>  <xsd:element name="pubs" sql:relation="publishers">  <xsd:complexType>  <xsd:sequence>  <xsd:element name="employee"  sql:relation="employee"  sql:relationship="PubsEmployees" >  <xsd:complexType>  <xsd:attribute name="First\_Name" sql:field="fname"  type="xsd:string" />  <xsd:attribute name="Last\_Name" sql:field="lname"  type="xsd:string" />  <xsd:attribute name="minit" sql:field="minit"  type="xsd:string" sql:hide="1" />  </xsd:complexType>  </xsd:element>  </xsd:sequence>  <xsd:attribute name="PubName" sql:field="pub\_name" type="xsd:string"/>  <xsd:attribute name="City" sql:field="city" type="xsd:string"  sql:limit-field="pub\_name" sql:limit-value="Binnet &amp; Hardley" />  </xsd:complexType>  </xsd:element>  </xsd:schema> |

Здесь использован атрибут hide для удаления XML-узла из результирующего документа, но сохранить возможность его использования в XPath-запросах. Атрибуты limit-field и limit-value ограничивают количество издательств одним – «Binnet & Hardley». Раздел xsd:annotation вынесен, однако его можно было использовать и внутри узла <xsd:element name="employee"…>. В таком случае имя элемента relationship можно было не указывать.

Примерно так можно использовать эту схему:

|  |
| --- |
| http://server/server\_pubs/schema/schema1.xml/pubs/employee[@minit=""]?root=root |

Схемы можно непосредственно указывать в шаблоне. Например:

|  |
| --- |
| <?xml version="1.0" encoding="windows-1251" ?>  <my\_root xmlns:sql="urn:schemas-microsoft-com:xml-sql">  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  xmlns:ms="urn:schemas-microsoft-com:mapping-schema"  id="InLineSchema1" sql:is-mapping-schema="1">  <xsd:element name="Авторы" ms:relation="authors">  <xsd:complexType>  <xsd:attribute name="Имя" ms:field="au\_fname"/>  <xsd:attribute name="Фамилия" ms:field="au\_lname"/>  <xsd:attribute name="Адрес" ms:field="address"/>  </xsd:complexType>  </xsd:element>  </xsd:schema>  <sql:xpath-query mapping-schema="#InLineSchema1">  /Авторы  </sql:xpath-query>  </my\_root> |

Для того чтобы запрос XPath использовал схему, а также для сокрытия ее в результирующем документе XML, указан атрибут is-mapping-schema. Он может принимать значения 0 или 1. Кроме этого, необходимо явно сослаться на используемую схему, так как их в шаблоне может быть несколько. Это делается путем добавления атрибута id в схему и атрибута mapping-schema – в раздел самого запроса.

Создание аннотированных схем не совсем тривиальная задача, требующая к тому же знания большого количества тонкостей. К счастью, в Microsoft разработали специальный инструмент для автоматической генерации схем – XML View Mapper. Его можно бесплатно скачать по адресу http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/443/msdncompositedoc.xml . У него хороший графический интерфейс и достаточно неплохая документация; думаю, вы с ним разберетесь быстро. Единственный его недостаток – отсутствие возможности сохранять схемы в формате XSD. Надеюсь, в будущем эта возможность появится.



К сожалению, SQL Server 2000 лишь частично поддерживает спецификацию XPath и возможности использования XPath-запросов к базе данных. Ниже приведена сводка поддерживаемых и не поддерживаемых возможностей.

Поддерживаемая функциональность XPath:

|  |  |
| --- | --- |
| Тип | Значения |
| Оси (axis) | attribute, child, parent, and self |
| Операторы сравнения | =, !=, <, <=, >, >= |
| Арифметические операторы | +, -, \*, div |
| Функции явного преобразования | number(), string(), Boolean() |
| Булевы операторы | And, or |
| Булевы функции | true(), false(), not() |

Не поддерживаемая функциональность:

|  |  |
| --- | --- |
| Тип | Значения |
| Оси | ancestor, ancestor-or-self, descendant, descendant-or-self (//), following, following-sibling, namespace, preceding, preceding-sibling |
| Арифметические операторы | Mod |
| Строковые функции | string(), concat(), starts-with(), contains(), substring-before(), substring-after(), substring(), string-length(), normalize(), translate() |
| Булевы функции | lang() |
| Числовые функции | sum(), floor(), ceiling(), round() |
| Оператор объединения | | |

Замечания по разделу

Вы можете использовать неограниченное количество простых запросов (query) и запросов XPath (XPath-query) в одном шаблоне, а также использовать их вперемешку.

В качестве простых запросов (query) могут выступать любые SQL-инструкции. Шаблоны могут быть использованы для изменения данных, хотя это и не лучшее решение. Другие методы изменения данных рассматриваются в разделе Апдейтаграммы и XML Bulk Load.

|  |  |
| --- | --- |
| Пространство имен | Назначение |
| urn:schemas-microsoft-com:xml-sql | Шаблоны и Аннотации XDR |
| urn:schemas-microsoft-com:xml-data | Схемы XDR |
| http://www.w3.org/2001/XMLSchema | Схемы XSD |
| urn:schemas-microsoft-com:mapping-schema | Аннотации XSD |

ADO и XML

Исторически самой первой и самой известной возможностью работы с XML-документами в ADO было сохранение объекта Recordset в формате XML. До этого вы могли сохранять Recordset’ы только в бинарном формате adPersistADTG. Он использовался для передачи наборов строк посредством RDS (Remote Data Services). Работу с обоими форматами поддерживает OLE DB Persistence Provider. Кроме сохранения, можно также загружать (восстанавливать) объект Recordset из файлов. Сохранение и последующая загрузка рекордсета из файла в формате XML дали возможность использования XML-документов в качестве баз данных.

OLE DB Persistence Provider жестко задает формат результирующего XML-документа: для описания структуры и типов узлов всегда используется XDR, данные всегда помещаются в секцию data, а строки представляются элементом row. Названия и значения полей – соответствующие названия и значения атрибутов элемента row. Нет никакой возможности изменить этот формат, если он по каким-либо причинам вас не устраивает. Можно, конечно, написать шаблон трансформации на XSLT, но это уже дополнительные сложности.

С выходом ADO 2.5 появилась возможность сохранять рекордсет в IStream. Трудно переоценить все достоинства этого нововведения: рекордсет теперь можно было сохранять в объект DOMDocument, трансформировать XML-документ с помощью метода transformNode, добавлять свои элементы и атрибуты, и многое другое. Кроме этого, вы могли сохранять рекордсет в поток Response объектной модели ASP, причем как в формате adPersistADTG, так и в формате adPersistXML. В новой ADO 2.5 появился собственный объект Stream (естественно, поддерживающий интерфейс IStream). С его помощью вы могли сохранять и загружать данные из файла на диске в бинарном формате (LoadFromFile и SaveToFile), загружать и сохранять данные в виде текста (ReadText и WriteText) и выполнять другие не реляционные операции. Но довольно истории, давайте перейдем к примерам.

Возможности ADO 2.5

Сохранение и загрузка из файла в формате XML

Не мудрствуя лукаво, возьму запрос из самого первого примера этой статьи. Вот полный исходный текст vbs-скрипта:

|  |
| --- |
| Const adopenStatic = 3  Const adLockReadOnly = 1  Const adCmdText = 1  Const adPersistXML = 1  Dim rs  Set rs = CreateObject("ADODB.Recordset")  rs.Open "select au\_fname,au\_lname,address from authors where au\_fname like 'M%'", \_  "Provider=sqloledb;Data Source=server;Initial Catalog=pubs;" & \_  "User Id=user;Password=password;", adopenStatic, adLockReadOnly, adCmdText  rs.Save "c:\myrs.xml", adPersistXML |

Следующий пример демонстрирует загрузку XML-документа в объект Recordset:

|  |
| --- |
| Const adopenStatic = 3  Const adLockReadOnly = 1  Const adCmdFile = 256  Dim rs  Set rs = CreateObject("ADODB.Recordset")  rs.Open "c:\myrs.xml", "Provider=MSPersist;", adopenStatic, adLockReadOnly, adCmdFile |

Трансформация с помощью DOMDocument

В этом примере создается ASP-страница, при обращении к которой из базы будет выбран Recordset и сохранен в объект DOMDocument. Далее к документу будет применен шаблон трансформации, и результирующий HTML будет передан клиенту. Результат будет точно таким же, как в примере с трансформацией в разделе «URL-запросы». Код ASP:

|  |
| --- |
| <%  ' Define some constant for ADO.  Const adopenStatic = 3  Const adLockReadOnly = 1  Const adCmdText = 1  Const adPersistXML = 1  ' Creating objects  Dim rs,dom,stylesheet  Set dom = Server.CreateObject("MSXML2.DOMDocument")  Set stylesheet = Server.CreateObject("MSXML2.DOMDocument")  Set rs = Server.CreateObject("ADODB.Recordset")  ' Open recordset  rs.Open "select au\_fname,au\_lname,address from authors where au\_fname like 'M%'", \_  "Provider=sqloledb;Data Source=server;Initial Catalog=pubs;" & \_  "User Id=user;Password=password;", adopenStatic, adLockReadOnly, adCmdText  ' Save recordset to DOMDocument  rs.Save dom,adPersistXML  ' Loading stylesheet  stylesheet.async = false  stylesheet.load "C:\Inetpub\wwwroot\server\_pubs\format\_for\_ado.xsl"  ' Perform transformation  Response.Write dom.transformNode(stylesheet)  ' Cleanup  Set dom = nothing  Set stylesheet = nothing  Set rs = nothing  %> |

Шаблон трансформации практически не изменился:

|  |
| --- |
| <?xml version="1.0" ?>  <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl" version="1.0">    <xsl:template match = "\*">  <xsl:apply-templates />  </xsl:template>  <xsl:template match = "z:row">  <li>  <table><tr>  <td><xsl:value-of select = "@au\_fname" /></td>  <td><xsl:value-of select = "@au\_lname" />.</td>  <td>Address: <xsl:value-of select = "@address" /></td>  </tr></table>  </li>  </xsl:template>  <xsl:template match = "/">  <html>  <body>  <ul>  <xsl:apply-templates select = "xml/rs:data" />  </ul>  </body>  </html>  </xsl:template>  </xsl:stylesheet> |

Выдача Recordset’а в формате XML непосредственно в поток Response

Рассмотрим совсем легкий пример выдачи рекордсета в объект Response модели ASP.

|  |
| --- |
| <%  ' Should specify this  Response.ContentType = "text/xml"  ' Define some constant for ADO.  Const adopenStatic = 3  Const adLockReadOnly = 1  Const adCmdText = 1  Const adPersistXML = 1  Dim rs  Set rs = Server.CreateObject("ADODB.Recordset")  ' Open recordset  rs.Open "select au\_fname,au\_lname,address from authors where au\_fname like 'M%'", \_  "Provider=sqloledb;Data Source=server;Initial Catalog=pubs;" & \_  "User Id=user;Password=password;", adopenStatic, adLockReadOnly, adCmdText  ' Save recordset to Response stream  rs.Save Response,adPersistXML  Set rs = nothing  %> |

Этот чрезвычайно простой пример демонстрирует богатые возможности манипулирования результирующим набором строк на клиенте. Рассмотрим использование объекта RDS.DataControl.

|  |
| --- |
| ПРИМЕЧАНИЕ  Возможно, эффективнее для RDS использовать формат adPersistADTG, однако XML для этого также прекрасно подходит. |

Пример взят из MSDN и слегка модифицирован:

|  |
| --- |
| <HTML>  <HEAD><TITLE>Пример ADO Recordset Persistence</TITLE></HEAD>  <BODY>  <OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33" ID="RDC1"  <PARAM NAME="URL" VALUE="http://yourserver/XMLPersist/XMLResponse.asp">  </OBJECT>  <TABLE DATASRC="#RDC1">  <TR>  <TD><SPAN DATAFLD="au\_fname"></SPAN></TD>  <TD><SPAN DATAFLD="au\_lname"></SPAN></TD>  <TD><SPAN DATAFLD="address"></SPAN></TD>  </TR>  </TABLE>  </BODY>  </HTML> |

А вот как создать на клиенте точную копию отправленного рекордсета (код на vbs):

|  |
| --- |
| Dim rs  Set rs = CreateObject("ADODB.Recordset")  rs.Open "http://server/server\_dir/sql2xml.asp" |

Возможности ADO 2.6

В этой версии библиотеки появилась возможность выполнять с помощью объекта Command не только SQL-запросы, но и XML-шаблоны и запросы XPath. Для этого было введено новое свойство Dialect. Далее приведены все известные на сегодняшний момент значения этого свойства [10].

|  |  |
| --- | --- |
| Тип команды | Значение в ADO |
| Запрос Transact-SQL | {C8B522D7-5CF3-11CE-ADE5-00AA0044773D} |
| Запрос XPath | {EC2A4293-E898-11D2-B1B7-00C04F680C56} |
| Запрос в XML-шаблоне | {5D531CB2-E6Ed-11D2-B252-00C04F681B71} |
| Поведение провайдера по умолчанию | {C8B521FB-5CF3-11CE-ADE5-00AA0044773D} |

Так как результат выполнения объекта Command теперь не всегда может иметь реляционный характер, его нельзя помещать в объект Recordset. Название нового стандартного свойства Output Stream говорит само за себя: результат выполнения объекта Command может быть сохранен в потоке (любом объекте, поддерживающим интерфейс IStream). Кроме этого, поскольку запрос (свойство CommandText) теперь может представлять собой запрос XPath, нужно как-то задать аннотированную схему для него. Это делается с помощью стандартного свойства Mapping Schema. Пора переходить к примерам.

Использование Command для формирования XML-документа на сервере

В этом примере показано, как сформировать XML-документ на сервере и отправить его клиенту с использованием ADO:

|  |
| --- |
| <%  ' Определяем константы ADO.  Const adopenStatic = 3  Const adLockReadOnly = 1  Const adCmdText = 1  Const adPersistXML = 1  Const adExecuteStream = &H400  ' создание объектов  Dim cmd,conn  Set cmd = Server.CreateObject("ADODB.Command")  Set conn = Server.CreateObject("ADODB.Connection")  conn.Provider = "sqloledb"  conn.Open "Data Source=server;Initial catalog=pubs;", "user", "password"  Set cmd.ActiveConnection = conn  cmd.CommandType = adCmdText  cmd.CommandText = "select au\_fname, au\_lname, address " \_  & "from authors where au\_fname like 'M%' for xml auto"  cmd.Properties("Output Stream") = Response  cmd.Properties("xml root") = "root"  cmd.Execute , , adExecuteStream  Set cmd = nothing  Set conn = nothing  %> |

Здесь используется встроенная инструкция FOR XML AUTO для формирования XML-документа на SQL Server’е. В результирующем документе отсутствует корневой элемент, и для его указания используется еще одно стандартное свойство – xml root. В качестве выходного потока был использован объект Response. Результат обращения к этой ASP-страничке вы, наверное, уже давно выучили наизусть: это будет XML-документ с именами, фамилиями и адресами авторов.

Выполнение шаблона

Возьмем шаблон из первого примера раздела «Шаблоны» (код на ASP):

|  |
| --- |
| <%  ' Нужно указать это:  Response.ContentType = "text/xml"  ' Определяем константы ADO.  Const adCmdText = 1  Const adExecuteStream = &H400  ' Создаем объекты  Dim cmd,conn  Set cmd = Server.CreateObject("ADODB.Command")  Set conn = Server.CreateObject("ADODB.Connection")  conn.Provider = "sqloledb"  conn.Open "Data Source=server;Initial catalog=pubs;", "user","password"  ' Шаблон  Dim s  s = "<?xml version='1.0' ?>" & \_  "<my\_root xmlns:sql='urn:schemas-microsoft-com:xml-sql'>" & \_  "<sql:query>" & \_  "select au\_fname,au\_lname,address from authors " & \_  "where au\_fname like 'M%' for xml auto" & \_  "</sql:query>" & \_  "</my\_root>"  Set cmd.ActiveConnection = conn  cmd.CommandType = adCmdText  cmd.CommandText = s  ' Диалект шаблона  cmd.Dialect = "{5D531CB2-E6Ed-11D2-B252-00C04F681B71}"  cmd.Properties("Output Stream") = Response  cmd.Execute , , adExecuteStream  Set cmd = nothing  Set conn = nothing  %> |

Выполнение запроса XPath

С помощью ADO также можно выполнять запросы XPath, правда, пока только на аннотированных XDR-схемах. Вот скрипт на vbs, выполняющий такой запрос:

|  |
| --- |
| ' Определяем константы ADO.  Const adCmdText = 1  Const adExecuteStream = &H400  ' Создание объектов  Dim cmd,conn,cmdStream  Dim ie  Set ie = CreateObject("InternetExplorer.Application")  ie.Navigate "about:blank"  Set cmd = CreateObject("ADODB.Command")  Set conn = CreateObject("ADODB.Connection")  Set cmdStream = CreateObject("ADODB.Stream")  conn.Provider = "sqloledb"  conn.Open "Data Source=server;Initial catalog=pubs;", "user", "password"  cmdStream.Open  Set cmd.ActiveConnection = conn  cmd.CommandType = adCmdText  ' XPath-запрос  cmd.CommandText = "Авторы"  ' Диалект XPath  cmd.Dialect = "{EC2A4293-E898-11D2-B1B7-00C04F680C56}"  cmd.Properties("Base Path") = "C:\Inetpub\wwwroot\server\_pubs"  cmd.Properties("Output Stream") = cmdStream  ' Аннотированная схема  cmd.Properties("Mapping Schema") = "schema\myschema.xml"  ' Шаблон трансформации  cmd.Properties("XSL") = "\template\first\_select.xsl"  cmd.Properties("xml root") = "my\_root"  cmd.Execute , , adExecuteStream  Dim str  str = cmdStream.ReadText  do  loop while ie.Busy  ie.Document.writeln CStr(str)  ie.Document.close  ie.visible = 1 |

Здесь я использовал новую аннотированную схему и шаблон трансформации.

Аннотированная схема:

|  |
| --- |
| <?xml version="1.0" encoding="windows-1251" ?>  <Schema xmlns="urn:schemas-microsoft-com:xml-data"  xmlns:sql="urn:schemas-microsoft-com:xml-sql"  xmlns:dt="urn:schemas-microsoft-com:datatypes">    <ElementType name="Авторы" sql:relation="authors">  <AttributeType name="Имя" dt:type="string" />  <AttributeType name="Фамилия" dt:type="string" />  <AttributeType name="Адрес" dt:type="string" />  <attribute type="Имя" sql:field="au\_fname" />  <attribute type="Фамилия" sql:field="au\_lname" />  <attribute type="Адрес" sql:field="address" />  </ElementType>  </Schema> |

Шаблон трансформации:

|  |
| --- |
| <?xml version="1.0" encoding="windows-1251" ?>  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">    <xsl:template match = "\*">  <xsl:apply-templates />  </xsl:template>  <xsl:template match = "/">  <html>  <body>  <table border="1" style="table-layout:fixed" width="600">  <tr bgcolor="teal">  <th><font color="white">Имя</font></th>  <th><font color="white">Фамилия</font></th>  <th><font color="white">Адрес</font></th>  </tr>  <xsl:for-each select="my\_root/Авторы">  <tr>  <td><font color="teal"><xsl:value-of select="@Имя"/></font></td>  <td><font color="teal"><xsl:value-of select="@Фамилия"/></font></td>  <td><font color="teal"><xsl:value-of select="@Адрес"/></font></td>  </tr>  </xsl:for-each>  </table>  </body>  </html>  </xsl:template>  </xsl:stylesheet> |

XML-документ на стороне клиента

Предположим, ваш SQL Server работает в очень напряженном режиме, и вы не хотите нагружать его еще больше, заставляя формировать XML-документы. Можно сделать так: создать обычный рекордсет и загрузить его в формате XML в объект DOMDocument или какой-либо другой. Решение на первый взгляд здравое, однако, поработав немного с таким документом, вы поймете, что это не то. Во-первых, структура такого XML-документа задается жестко без возможности изменения. Во-вторых, схема сохраняется в формате XDR, а он уже стар и может не удовлетворять вашим потребностям. К тому же возникают проблемы с выполнением шаблонов и XPath-запросами. Для решения этих (и многих других) проблем был создан новый сервисный провайдер – SQLXMLOLEDB. Как и другие сервисные провайдеры (например, MSDataShape) он предназначен только для преобразования результирующего набора строк в нужный формат. Так как SQLXMLOLEDB ничего не знает о форматах протоколов обмена данными с SQL-серверами, он использует для этого соответствующий провайдер. К сожалению, провайдер SQLXMLOLEDB пока умеет работать совместно только с SQLOLEDB-провайдером для SQL Server, однако в будущем (по крайней мере, я очень надеюсь) он сможет работать с другими провайдерами, например, MSDAORA (провайдер для Oracle). Если такая поддержка будет встроена, у нас будет возможность получать XML-документы от источников, которые не поддерживают XML напрямую.

В режиме форматирования XML-документа на клиенте можно использовать в запросе предикат GROUP BY и агрегатные функции, недоступные в режиме формирования XML-документа на сервере (в режиме FOR XML AUTO). Как и любой другой сервисный провайдер, SQLXMLOLEDB предоставляет свой синтаксис запросов. К счастью, он не так сложен, как, например, синтаксис провайдера MSDataShape и ограничивается, по существу, следующим:

|  |  |
| --- | --- |
| Оператор | Назначение |
| FOR XML RAW | Аналогичен серверному FOR XML RAW |
| FOR XML NESTED | Аналогичен серверному FOR XML AUTO |
| FOR XML EXPLICIT | Аналогичен серверному FOR XML EXPLICIT |

Архитектура формирования документов на клиентской стороне изображена на рисунке 1.

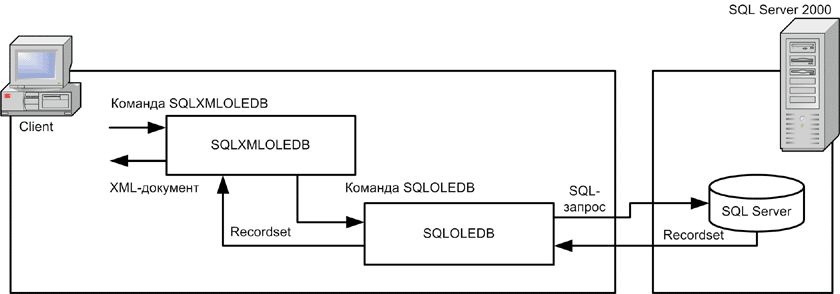


Рисунок 1.

|  |  |  |
| --- | --- | --- |
| Возможность | Формирование на клиенте | Формирование на сервере |
| Использование предиката GROUP BY и агрегатных функций совместно с FOR XML AUTO (NESTED) | Да | Нет |
| Возможность получения одновременно нескольких рекордсетов | Нет | Да |
| Задание имен элементов результирующего документа с помощью псевдонимов таблиц | Нет | Да |

Другие отличия вы можете найти в MSDN. Давайте перейдем к примерам.

Шаблон с агрегатными функциями

Ниже приведен пример использования FOR XML NESTED и атрибута client-side-xml. Шаблон возвращает документ с названиями городов и количеством авторов в каждом городе. Атрибут client-side-xml устанавливается в 1, что говорит об использовании клиентского форматирования.

|  |
| --- |
| <?xml version="1.0" ?>  <my\_root xmlns:sql="urn:schemas-microsoft-com:xml-sql">  <sql:query client-side-xml="1">  select count(\*) as number\_of\_authors,city  from authors  group by city  order by number\_of\_authors desc  for xml nested  </sql:query>  </my\_root> |

Формирование XML-документа на клиенте с помощью ADO

У объекта Command есть свойство ClientSideXML, которое позволяет формировать XML-документ на клиенте. Рассмотрим пример:

|  |
| --- |
| <%  Const adCmdText = 1  Const adExecuteStream = &H400  Response.ContentType = "text/xml"  ' Creating the objects  Dim conn,cmd  Set conn = Server.CreateObject("ADODB.Connection")  Set cmd = Server.CreateObject("ADODB.Command")  conn.Provider = "sqlxmloledb"  conn.Open "Data Source=server;Initial catalog=pubs;Data Provider=sqloledb", \_  "user","password"  Dim s  s = "select count(\*) as number\_of\_authors,city" & \_  "from authors" & \_  "group by city" & \_  "order by number\_of\_authors desc" & \_  "for xml nested"  Set cmd.ActiveConnection = c  cmd.Properties("ClientSideXML") = "True"  cmd.CommandType = adCmdText  cmd.CommandText = s  cmd.Properties("Output Stream") = Response  cmd.Properties("xml root") = "root"  cmd.Execute , , adExecuteStream  Set cmd = nothing  Set conn = nothing  %> |

В качестве основного провайдера задан провайдер SQLXMLOLEDB. Провайдер SQL Server’а задается с помощью строки Data Provider=sqloledb. Чтобы указать, что используется клиентское форматирование, свойство ClientSideXML установлено в true. Следует заметить, что это свойство (т.е. клиентское форматирование) можно использовать только для типа adCmdText. При вызове хранимых процедур его устанавливать нельзя.

Апдейтаграммы и XML Bulk Load

В этом разделе мы рассмотрим способы изменения данных (удаление, добавление и правка) с помощью XML-документов. Какие достоинства у этого метода перед использованием SQL-инструкций непосредственно в шаблоне? Во-первых, апдейтаграммы намного понятнее простым пользователям или неосведомленным программистам. Хотя синтаксис написания апдейтаграмм не так уж и прост, он подчиняется обычным правилам XML и, если вы совсем не разбираетесь в SQL, апдейтаграммы – лучший выбор. Кроме этого, вы просто не имеете другой столь же эффективной альтернативы, как XML Bulk Load для загрузки больших, очень больших объемов XML-данных. Но обо всем по порядку.

Апдейтаграммы

Апдейтаграммы – это особый тип шаблонов, в котором вместо запроса на выборку данных хранится информация, необходимая для изменения данных. Синтаксис их таков:

|  |
| --- |
| <your\_root xmlns:updg="urn:schemas-microsoft-com:xml-updategram"  [mapping-schema="your\_schema.xml"]>  <updg:header>  <updg:param name="param1\_name" />  <updg:param name="param2\_name" />  </updg:header>  <updg:sync>  <updg:before>  xml-узлы  </updg:before>  <updg:after [updg:returnid="your\_id"]>  xml-узлы  </updg:after>  </updg:sync>  </your\_root> |

Как видно из синтаксиса, апдейтаграмма может иметь параметры. Они, как обычно, задаются в URL и, как обычно, могут опускаться. Раздел header не является обязательным, но на практике без него обойтись трудно. Раздел before, по существу, определяет запись, над которой будет производиться действие. Он обязателен для изменения или удаления данных. В случае вставки его можно опустить. Раздел after определяет, как будет выглядеть найденная запись после выполнения апдейтаграммы. В случае удаления его указывать не нужно. Раздел sync определяет границы транзакции и буквально трансформируется в команды begin trans и commit trans. В случае неудачного выполнения шаблона транзакция откатывается с помощью rollback trans.

|  |
| --- |
| ПРИМЕЧАНИЕ  Все аспекты работы апдейтаграмм хорошо видны в профайлере. Если у вас возникают вопросы, рекомендую сразу же им воспользоваться. |

В каком случае может возникнуть ошибка? Причин много:

Раздел before не может однозначно определить запись, над которой будет производиться действие. Например, если вы захотите удалить все записи, у которых поле age равно 33, то в ответ получите ошибку Ambiguous delete, unique identifier required. Другая ошибка возникает, когда такую запись невозможно найти совсем. В этом случае генерируется сообщение Empty delete, no deletable rows found для операции удаления, или Empty update, no updatable rows found для операции обновления.

Откат транзакции в триггере.

Ошибка ограничения (constraint violation).

Для того чтобы SQL Server понял, какие таблицы и поля ему изменять, вы можете использовать аннотированные схемы XDR или XSD. Если схема не используется, по умолчанию названия элементов соответствуют названиям таблиц, названия и значения атрибутов – названию и значению полей соответственно.

Добавление данных

Для вставки данных используется раздел after. Раздел before опускается. Если вы хотите получить на выходе значение колонки identity, укажите атрибут returned (и атрибут at-identity, см. далее). Для элемента, обозначающего таблицу, вы можете дополнительно указать три атрибута:

id – будет рассмотрен в разделе Обновление данных;

at-identity – позволяет получить в результирующий XML-документ значение вставленного поля для колонки identity. Например, пусть имеется такая таблица

|  |
| --- |
| create table test1(\_id int identity,fld1 int) |

Вот шаблон, который вставляет в нее данные:

|  |
| --- |
| <ROOT xmlns:updg="urn:schemas-microsoft-com:xml-updategram">  <updg:sync>  <updg:after updg:returnid="ident\_field">  <test1 fld1="23" updg:at-identity="ident\_field"/>  </updg:after>  </updg:sync>  </ROOT> |

Результатом выполнения команды «http://server/server\_pubs/template/upd1.xml» будет следующий документ:

|  |
| --- |
| <ROOT xmlns:updg="urn:schemas-microsoft-com:xml-updategram">  <returnid><ident\_field>1</ident\_field></returnid>  </ROOT> |

guid – предназначен для генерации глобально уникального идентификатора (GUID).

Давайте рассмотрим более сложный пример с использованием схемы. Вот схема:

|  |
| --- |
| <?xml version="1.0" encoding="windows-1251" ?>  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  xmlns:sql="urn:schemas-microsoft-com:mapping-schema">  <xsd:element name="Авторы" sql:relation="authors">  <xsd:complexType>  <xsd:attribute name="Имя" type="xsd:string"  sql:field="au\_fname" />  <xsd:attribute name="Фамилия" type="xsd:string"  sql:field="au\_lname" />  <xsd:attribute name="Идентификатор" type="xsd:string"  sql:field="au\_id" />  <xsd:attribute name="Контракт" type="xsd:integer"  sql:field="contract" default="1"/>  </xsd:complexType>  </xsd:element>  </xsd:schema> |

Здесь мы явно использовали аннотации, так как названия XML-узлов составлены на русском. После выполнения следующего шаблона:

|  |
| --- |
| <?xml version="1.0" encoding="windows-1251" ?>  <ROOT xmlns:updg="urn:schemas-microsoft-com:xml-updategram">  <updg:sync mapping-schema="upd\_schema.xml">  <updg:after>  <Авторы Имя="alex" Фамилия="shirshov" Идентификатор="123-15-3452"/>  </updg:after>  </updg:sync>  </ROOT> |

мои имя и фамилия (да простят мне читатели нескромность) окажутся в таблице authors.

Обновление данных

Для обновления данных нужно использовать раздел before для определения обновляемых данных, и раздел after для определения конечных значений данных.

Давайте разберем пример:

|  |
| --- |
| <?xml version="1.0" encoding="windows-1251" ?>  <ROOT xmlns:updg="urn:schemas-microsoft-com:xml-updategram">  <updg:sync mapping-schema="upd\_schema.xml">  <updg:before>  <Авторы Имя="alex" Фамилия="shirshov" Идентификатор="123-15-3452"  updg:id="for\_update"/>  </updg:before>  <updg:after>  <Авторы Имя="Алексей" Фамилия="Ширшов" Идентификатор="123-15-3452"  updg:id="for\_update"/>  <Авторы Имя="alex" Фамилия="shirshov" Идентификатор="123-15-3453"  updg:id="for\_insert"/>  </updg:after>  </updg:sync>  </ROOT> |

Работает он так: сначала находится запись, однозначно идентифицируемая в разделе before (ее мы вставили в предыдущем примере). Этой записи, с помощью атрибута id, присваивается идентификатор. Зачем он нужен? Так как в разделе after мы указали две записи, то возникает неоднозначность: одну из них нужно использовать для вставки, а другую для обновления. Чтобы эту неоднозначность разрешить, в разделе after присутствует ссылка на найденную в разделе before запись посредством атрибута id. В результате выполнения шаблона будут исправлены имя и фамилия в записи с идентификатором 123-15-3452, а также добавлена новая запись.

Удаление

Это совсем тривиальная задача, главное помнить, что в разделе before запись должна однозначно определяться.

|  |
| --- |
| <?xml version="1.0" encoding="windows-1251" ?>  <ROOT xmlns:updg="urn:schemas-microsoft-com:xml-updategram">  <updg:sync mapping-schema="upd\_schema.xml">  <updg:before>  <Авторы Имя="alex" Фамилия="shirshov" Идентификатор="123-15-3453"/>  </updg:before>  </updg:sync>  </ROOT> |

В этом примере будет удалена запись с идентификатором 123-15-3453.

|  |
| --- |
| ПРИМЕЧАНИЕ  Атрибуты «Имя» и «Фамилия» приведены здесь для того, чтобы вам проще было понять, какая строка удаляется. Так как идентификатор уникально определяет запись, их можно было не указывать. |

Более подробную информацию об апдейтаграммах с примерами можно найти в MSDN или в [10].

XML Bulk Load

Предположим, у вас имеется XML-документ, содержимое которого нужно «залить» в базу. Методов решения проблемы, как всегда, очень много, и вся сложность состоит в выборе наиболее подходящего. Вы можете загрузить документ в DOMDocument и в цикле, выбирая значения элементов и атрибутов, производить добавление данных с помощью инструкции SQL. Если вы не знакомы с объектной моделью DOM, то можете написать свой парсер. Такое лобовое решение обычно принимают самые отважные и «крутые» программисты, которые не боятся трудностей написания нового парсера, изучения SQL и начальства. Что ж, лично я (хотя начальства не особо боюсь) к такой категории себя отнести не могу. Мне нужно решение, которое опирается на уже существующие возможности и технологии. Второе, что приходит в голову – передать XML-документ в хранимую процедуру и с помощью OPENXML «залить» данные в таблицу. Решение здравое и наиболее эффективное в большинстве случаев. Но что делать, если у вас имеется большой документ? Конечно, его можно все так же передавать в хранимую процедуру в параметре text или ntext и также разбирать с помощью OPENXML. Но все дело в том, что OPENXML использует DOM, а обработка больших документов таким способом имеет кучу недостатков. Мало того, что расходуется большое количество драгоценных системных ресурсов, это еще и медленно! Для больших документов идеальным вариантом является XML Bulk Load.

XML Bulk Load – это обычный COM-сервер, размещенный в DLL, и использующий для анализа XML-документа SAX (Simple API for XML). Благодаря этому он обрабатывает документ по частям, намного менее ресурсоемок и более быстр. Для обновления данных XML Bulk Load должен знать, какие XML-узлы соотносятся с полями в таблицах, и каков их тип. Для этого он использует все те же аннотированные схемы, которые могут быть написаны на XDR или XSD.

Семантика объекта XML Bulk Load не может показаться сложной – объект содержит всего один метод и несколько свойств. Рассмотрим наиболее используемые свойства:

ConnectionString – строка соединения с базой данных (формат SQLOLEDB).

ConnectionCommand – позволяет использовать уже существующий объект ADODB.Connection.

BulkLoad – если установлено в true, кроме генерации схемы (таблиц) происходит также закачка данных. Если установлен в false – создаются только таблицы (см. свойство SchemaGen). По умолчанию – true.

ErrorLogFile – позволяет указать файл, в который будут записываться сообщения об ошибках.

Transaction – если установлено в true, все операции XML Bulk Load выполняются в контексте одной транзакции. По умолчанию – false.

TempFilePath – директория, в которой будет создан файл лога транзакции. Настоятельно рекомендую устанавливать его самостоятельно, так как по-другому у меня просто не получалось. Свойство имеет значение, только если Transaction установлено в true. По умолчанию создает файл в папке %temp%.

SchemaGen – если установлено в true, создаются указанные в аннотированной схеме таблицы. Если таблицы уже существуют, используется свойство SGDropTables. По умолчанию – false.

SGDropTables – если установлено в true, существующие в базе таблицы удаляются перед закачкой. Для добавления данных к уже существующим, оставьте это свойство равным false (по умолчанию).

KeepIdentity – если установлено в true, то значения для поля типа identity выбираются из XML-документа, если false – SQL Server сам выполняет обновление данного поля. По умолчанию – false.

Другие свойства можно найти в документации, все они аналогичны настройкам утилиты bcp (bulk copy program – утилита командной строки, поставляемая с SQL Server, которая позволяет загружать/выгружать данные в/из текстового файла) и оператора BULK LOAD. Перейдем к примерам.

Создание новой таблицы и загрузка данных

В разделе "Апдейтаграммы", в примере "Добавление данных" мы создали простую таблицу из одного поля и добавили в нее значение. Вот как это можно сделать с помощью XML Bulk Load.

Аннотированная схема:

|  |
| --- |
| <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  xmlns:ms="urn:schemas-microsoft-com:mapping-schema">  <xsd:element name="test-table" ms:relation="test1">  <xsd:complexType>  <xsd:attribute name="identifier" ms:datatype="int" ms:field="\_id"/>  <xsd:attribute name="field1" ms:datatype="int" ms:field="fld1"/>  </xsd:complexType>  </xsd:element>  </xsd:schema> |

Она ссылается на таблицу test1 и две колонки типа int – \_id и fld1. В XML-документе соответствующие атрибуты будут называться identifier и field1.

Теперь скрипт:

|  |
| --- |
| Dim xbcp,adoStream  Set xbcp = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad.3.0")  Set adoStream = CreateObject ("ADODB.Stream")  adoStream.Open  adoStream.WriteText "<root><test-table field1='145' identifier='1' /></root>"  AdoStream.Position = 0  xbcp.ConnectionString = "Provider=sqloledb.1;Data Source=server;" & \_  "database=pubs;User ID=user;Password=password;"  xbcp.SchemaGen = true  xbcp.SGDropTables = true  xbcp.Transaction = true  xbcp.ErrorLogFile = "c:\error.log"  xbcp.TempFilePath = "c:\temp"  xbcp.Execute "xbcp\_schema.xml",adoStream |

Чтобы не создавать отдельно малюсенький XML-документ, в примере используется объект Stream из ADO. В качестве схемы stream использовать нельзя, а жаль.

|  |
| --- |
| ПРИМЕЧАНИЕ  Лично мне не удавалось запускать подобные скрипты на клиентской машине. Так как я не администратор домена, приходилось просить скопировать схему и скрипт на сервер и там уже запускать. Может, просто не хватало прав. |

Загрузка иерархических документов

Если у вас имеется иерархический документ, и вам необходимо его загрузить с сохранением родительско-дочерних связей, этот пример – для вас. В нем создается две таблицы: родительская test1 и дочерняя test2. Самое сложное при этом – написать правильную аннотированную схему.

|  |
| --- |
| <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  xmlns:ms="urn:schemas-microsoft-com:mapping-schema">  <xsd:element name="test-table1" ms:relation="test1">  <xsd:complexType>  <xsd:sequence>  <xsd:element name="test-table2" ms:relation="test2">  <xsd:annotation>  <xsd:appinfo>  <ms:relationship parent="test1"  parent-key="test1\_id"  child="test2"  child-key="test1\_id" />  </xsd:appinfo>  </xsd:annotation>  <xsd:complexType>  <xsd:attribute name="identifier" ms:datatype="int"  ms:field="test2\_id"/>  </xsd:complexType>  </xsd:element>  </xsd:sequence>  <xsd:attribute name="identifier" ms:datatype="int" ms:field="test1\_id"/>  <xsd:attribute name="field1" ms:datatype="int" ms:field="fld1"/>  </xsd:complexType>  </xsd:element>  </xsd:schema> |

Скрипт (написан на vbs):

|  |
| --- |
| Dim xbcp,adoStream  Set xbcp = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad.3.0")  Set adoStream = CreateObject ("ADODB.Stream")  adoStream.Open  adoStream.WriteText "<ROOT>" & \_  "<test-table1 identifier='10' field1='345'>" & \_  "<test-table2 identifier='1' />" & \_  "</test-table1>" & \_  "</ROOT>"  AdoStream.Position = 0  xbcp.ConnectionString = "Provider=sqloledb.1;Data Source=server;" & \_  "database=pubs;User ID=user;Password=password;"  xbcp.SchemaGen = true  xbcp.SGDropTables = true  xbcp.Transaction = true  xbcp.ErrorLogFile = "c:\error.log"  xbcp.TempFilePath = "c:\temp"  xbcp.Execute "xbcp\_schema.xml",adoStream |

После его выполнения будут созданы следующие таблицы.

|  |  |
| --- | --- |
| test1\_id | field1 |
| 10 | 345 |

Таблица test1.

|  |  |
| --- | --- |
| test1\_id | test2\_id |
| 10 | 1 |

Таблица test2.

**Заключение**

Ну вот, мы и добрались, наконец, до заключения. Хотя все рассмотренные методы работы с XML-документами опирались на технологии Microsoft, общая картина вам должна быть ясна: любой современный SQL-сервер уже не ограничивается работой только с реляционными данными. Например, в Oracle 9i также включена встроенная поддержка XML – тип таблиц XMLType. Вы можете делать SQL-запросы к документами XML и XML-запросы (запросы XPath) к реляционным таблицам, так же, как и в MS SQL Server[12]. В дальнейшем эти возможности будут улучшаться, например, уже сейчас мы можем сделать из SQL Server сервер Web-служб (Oracle 9i также поддерживает эту возможность). Остается только надеяться, что Microsoft в этом плане будет придерживаться стандартов, и не будет бежать впереди паровоза, как это было с XDR и схемами трансформации.

**Список литературы**

Описание XDR

Спецификация XPath

Настройка IIS для SQL Server

Спецификация XSL

Спецификация XSD

Использование SDR схем для запросов XPath

Использование XSD схем для запросов XPath

Преобразование XDR в XSD

XML для профессионалов, Дидье Мартин, Марк Бирбек и другие.

XML в MS SQL Server 2000 и технологиях доступа к данным, Алексей Шуленин.

Ограничения XML Bulk Load

Поддержка XML в Oracle 9i