Лабораторная работа № 5

Использование XML с базами данных

Общие сведения

XML (eXtensible Markup Language) – это расширяемый язык разметки, рекомендованный Консорциумом Всемирной паутины (World Wide Web Consortium, W3C). Спецификация XML фактически представляет собой свод общих синтаксических правил для кодирования структурированных и слабоструктурированных данных. Хотя XML – текстовый формат, но визуально структура XML-документа может быть представлена как дерево элементов, описываемых тегами. Ниже в качестве примера представлен XML-документ, описывающий сведения об одной книге, написанной несколькими авторами и имеющей несколько вариантов цен.

```
<?xml version="1.0" encoding="utf-8"?>
<books>
  <book>
    <title>Professional XML Application</title>
    <ISBN>1-8001-152-5</ISBN>
    <authors>
      <author_name>Jun Duckett</author_name>
      <author_name>Peter Jones</author_name>
      <author_name>Stephen Mohr</author_name>
      <author_name>John Kauffman</author_name>
      <author_name>Michael Kay</author_name>
      <author_name>Guy Murphy</author_name>
    </authors>
    <description>XML is possibly the hottest topic to have hit the Internet since HTML, if you
want to learn what it is all about then this is the book for you.</description>
    <price US="$49.99"/>
    <price Ca="$79.95"/>
    <price UK="$49.95"/>
  </book>
</books>
```

Большинство производителей СУБД включают поддержку XML в свои продукты. Формы поддержки XML различаются, но все их можно условно разделить на пять следующих категорий.

1. **Вывод в формате XML**. Данные одной или более строк результата запроса можно представить в виде XML-документа. Это означает, что в ответ на SQL-запрос СУБД вместо обычного набора строк и столбцов генерирует XML-документ.
2. **Ввод в формате XML**. XML-документ может содержать данные, предназначенные для вставки в одну или более новых строк таблицы базы данных, или же в нем могут содержаться данные, предназначенные для обновления строки таблицы, либо данные, идентифицирующие удаляемую строку.
3. **Обмен данными в формате XML**. XML представляет собой средство для обмена данными между разными СУБД или серверами баз данных. Данные исходной базы данных преобразуются в XML-документ и направляются в принимающую базу данных, где они вновь преобразуются в формат базы данных. Такой способ обмена данными полезен для перемещения данных между реляционными базами данных и другими приложениями.
4. **Хранение данных в формате XML**. Реляционные базы данных могут принимать XML-документ как часть символьной строки или данных большого символьного объекта. На самом низком уровне поддержки XML весь XML-документ является содержимым одного столбца одной строки базы данных. При усиленной поддержке XML, по сравнению с этим элементарным уровнем, СУБД может позволять явно объявлять

столбцы как относящиеся к типу данных XML.

5. **Интеграция данных XML.** СУБД может выполнить синтаксический анализ XML-документа, разделить его на составляющие, и сохранить отдельные элементы в отдельных столбцах. После этого для поиска данных в полученной таблице может использоваться обычный язык SQL. В ответ на запрос СУБД может снова собрать XML-документ из хранящихся в таблице составляющих элементов.

Спецификация XML определяет два типа документов XML:

- **Правильно оформленные (well-formed) документы**. Правильно оформленный документ соблюдают правила синтаксиса XML.
- **Допустимые (valid) документы.** Допустимый документ правильно оформлен и соответствует некоторой схеме.

Существует множество программ для XML, известных как синтаксические анализаторы, некоторые из которых способны проверять документ на допустимость. В этом случае они называются проверяющими анализаторами (validating parser).

Существует два типа XML-анализаторов, поддерживающих два способа для обработки XML-документов:

- **Document Object Model (DOM).** DOM-анализаторы преобразуют XML-документ в иерархическую древовидную структуру. После этого при помощи API DOM программа может перемещаться по дереву вверх и вниз, следуя иерархической структуре документа. Интерфейс API DOM облегчает программистам доступ к структуре документа и его элементам. Изначально различные браузеры имели собственные DOM, несовместимые с остальными. Специалисты W3C классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM.
- **Simple API for XML (SAX).** SAX-анализаторы преобразуют XML-документ в последовательность обратных вызовов программы, которые информируют программу о каждой встреченной анализатором части документа. В ответ программа может выполнять определенные действия, например, реагировать на начало каждого раздела документа или на конкретный атрибут. Интерфейс API SAX предлагает использующим его программам более последовательный стиль обработки документа, лучше соответствующий программной структуре приложений, управляемых событиями.

Любой синтаксический анализатор XML проверит правильность оформления документа и, кроме того, сверит XML-документ со схемой. DOM-анализатор удобен в тех случаях, когда размер XML-документа относительно мал; этот анализатор генерирует в оперативной памяти древообразное представление документа, занимающее вдвое больше места, чем исходный документ. Анализатор типа SAX позволяет обрабатывать большие документы небольшими фрагментами. Но поскольку документ не находится в памяти целиком, программе приходится выполнять по нему несколько проходов, если она обрабатывает его фрагменты не по порядку.

Одной из самых сильных сторон реляционной модели является поддержка типов данных и структур данных, реализованная в определениях столбцов. Кроме того системный каталог реляционной базы данных содержит метаданные. При помощи запросов к системному каталогу можно узнать структуру базы данных, включая информацию о типах данных ее столбцов, наборе столбцов таблиц и отношениях между таблицами. Что касается XML-документов, то они, напротив, сами по себе содержат очень мало метаданных. Они отражают иерархическую структуру данных, но единственными содержащимися в них реальными данными о структуре являются имена элементов и атрибутов. XML-документ может строго соответствовать стандартам и все же иметь довольно необычную структуру. Например, ничто не мешает такому XML-документу содержать именованный элемент с текстовыми данными в одном экземпляре и вложенными элементами в другом, или же содержать именованный атрибут с целочисленным значением для одного элемента и дату для другого. Очевидно, что документ такой структуры, несмотря на строгое соответствие стандарту, не представляет данные, которые легко переносятся в базу данных и из нее. Поэтому при использовании XML для хранения обрабатываемых данных необходима более основательная поддержка типов данных и их жесткой структуры.

Имеется несколько стандартов и продуктов, решающих эту задачу различным образом:

- **Document Type Definition (DTD)**. Часть исходной спецификации XML, называемая «определение типа документа» (Document Type Definition, DTD), описывает способ определения и ограничения структуры документа. Синтаксические анализаторы XML могут анализировать содержимое XML-документа в контексте DTD и определять, является ли этот документ допустимым (соответствует ли он ограничениям DTD). В настоящее время в XML-технологии идёт отказ от использования DTD.
- **XML-Data.** Является одной из первых попыток исправить некоторые недостатки DTD. Она так и не получила одобрения W3C, но многие предложенные в ней идеи были реализованы в спецификации XML Schema.
- **Схема XML (XML Schema).** XML Schema является языком описания структуры XML-документа и содержит определения правил, которым должен подчиняться XML-документ. Для проверки XML-документа на соответствие схеме читающая программа может создать модель данных документа, которая включает: a) словарь (названия элементов и атрибутов); b) модель содержания (отношения между элементами и атрибутами и их структура); c) типы данных.
- **Стандарты промышленных групп**. Существует ряд отраслевых стандартов, определяющих типы XML-документов для использования в отдельных индустриальных сферах. Например, финансовые организации разрабатывают стандарты для описания финансовых объектов и маркетинговых данных. Производственные предприятия разрабатывают стандарты для описания своих документов - заказов, подтверждений принятия заказов и т. п. Все эти отраслевые стандарты обычно основываются на таких универсальных стандартах, как DTD и XML Schema.

Стандарты для метаданных и типов документов XML быстро развиваются и расширяются. На сайте консорциума W3C можно найти ссылки на различные стандарты, связанные с XML и информацию об их состоянии. Информация об отраслевых стандартах имеется по адресу http://www.xml.org на Web-узле, созданном организацией OASIS (Organization for the Advancernent of Structured Inforrnation Systerns). На этом узле представлен реестр XML-стандартов, классифицированный по отраслям.

SQL предоставляет разработчикам средства для поиска, преобразования и извлечения структурированных данных из баз данных. Естественно ожидать, что для поиска информации в XML-документах тоже должны иметься подобные возможности. Результатом первых попыток создания средств поиска и преобразования данных стали две спецификации:

- **XLST** (**Extensible Stylesheet Language Transformation**). XSLT предназначен для преобразования XML-документа. Преобразованием управляет таблица стилей, в которой указано, какие элементы входного XML-документа необходимо преобразовать и как они должны объединяться с другими элементами для получения выходного XML-документа. Одним из часто используемых применений XLST является преобразование одной общей версии Web-страницы в различные формы, предназначенные для вывода на экранах разного размера и разных типов.
- **XPath** (**XML Path Language**). При использовании XLST часто требуется выбрать отдельные элементы или группы элементов, которые должны быть преобразованы, или же пройти по иерархии элементов, чтобы выбрать данные из родительских и дочерних элементов. Язык XPath первоначально был частью языка XLST и использовался для выбора элементов и навигации по элементам. Однако вскоре оказалось, что он полезен и для других применений, и его спецификация была выделена отдельно.

Позднее проявились некоторые недостатки и ограничения использования языка XPath как полноценного языка запросов. Была сформирована рабочая группа, задачей которой стала разработка спецификации средства формирования запросов, носившего рабочее название XML Query или XQuery.

В основе языка SQL лежит модель представления данных в форме строк и столбцов, а в основе языка XQuery – древовидная иерархия узлов, представляющая XML-документ. В XQuery используется даже более детализированная структура, чем в XML-документах и схеме XML. С запросами к базам данных связаны следующие узлы XQuery:

- **Узел элемента**. Узлы этого типа представляют отдельные элементы.
- **Текстовый узел**. Узлы этого типа представляют содержимое элементов. Текстовый узел является дочерним по отношению к соответствующему узлу элемента.
- **Узел атрибута**. Узлы этого типа представляют атрибуты и значения атрибутов элемента. Узел атрибута является дочерним по отношению к соответствующему узлу элемента.
- **Узел документа**. Это специализированный узел элемента, представляющий верхний или *корневой* уровень документа.

Для навигации по дереву элементов и идентификации одного или более элементов для обработки в XQuery используется выражение пути. Выражение пути выполняет в XQuery ту же роль, что в SQL играют выражения запросов. Оно идентифицирует отдельный узел в дереве элементов XQuery, определяя последовательность шагов, которые нужно пройти по иерархии, чтобы достичь ее корневого уровня. Выражения пути XQuery бывают двух видов.

- **Корневое выражение пути**. Начинается от корня дерева элементов и спускается вниз по иерархии до целевого узла.
- **Относительное выражение пути**. Начинается с текущего узла дерева элементов (того узла, который обрабатывается в данный момент) и спускается или поднимается по иерархии до целевого узла.

Примечания

1. Для первого знакомства с различными технологиями XML рекомендуются школы консорциума W3C по адресу http://www.w3schools.com/.
2. Для более глубокого знакомства с технологиями XML рекомендуются стандарты W3C по адресу http://www.w3.org/standards/. Адреса этих стандартов:
    a. XML (http://www.w3.org/TR/REC-xml/)
    b. Namespaces in XML (http://www.w3.org/TR/REC-xml-names)
    c. XML Schema (http://www.w3.org/TR/xmlschema-0 и др.)
    d. XSL (http://www.w3.org/TR/xsl/)
    e. XSLT (http://www.w3.org/TR/xslt)
    f. XPath (http://www.w3.org/TR/xpath)
    g. XQuery (http://www.w3.org/XML/Query/)
    h. XML DOM (http://www.w3.org/DOM/DOMTR)
3. Справочник по XML-стандартам применительно к платформе .NET Framework находится по адресу https://msdn.microsoft.com/ru-ru/library/ms256177(v=vs.120).aspx.
4. Для работы с различными XML-документами рекомендуется использовать Liquid XML Studio 2015 по адресу http://www.liquid-technologies.com/xml-studio.aspx. Liquid XML Studio представляет собой среду разработки XML-документов, включающую множество редакторов и XML-инструментов. В состав Liquid XML Studio входят:
    a. графический редактор схем XSD
    b. редактор XML
    c. средство интеграции с Microsoft Visual Studio
    d. построитель выражений XPath
    e. редактор и отладчик XSLT
    f. редактор и отладчик XQuery и др.

Часть 1. Извлечение данных из базы данных SQL Server в формате XML

Запрос SELECT возвращает результаты в виде набора строк. При необходимости можно получать результаты SQL-запроса в формате XML. Для этого в запросе необходимо указать предложение FOR XML. Предложение FOR XML может использоваться в запросах верхнего уровня и во вложенных запросах. Предложение FOR XML верхнего уровня можно использовать только в инструкции SELECT. Во вложенных запросах предложение FOR

XML можно использовать в инструкциях INSERT, UPDATE и DELETE. Оно также может использоваться в инструкциях присваивания. В предложении FOR XML можно указать один из следующих режимов: RAW, AUTO, EXPLICIT, PATH.

Базовый синтаксис предложения FOR XML

```
[ FOR { BROWSE | <XML> } ]
<XML> ::=
XML
   {
    { RAW [ ('ElementName') ] | AUTO }
     [
       <CommonDirectives>
       [ , { XMLDATA | XMLSCHEMA [ ('TargetNameSpaceURI') ]} ]
       [ , ELEMENTS [ XSINIL | ABSENT ]
     ]
   | EXPLICIT
     [
       <CommonDirectives>
       [ , XMLDATA ]
     ]
   | PATH [ ('ElementName') ]
     [
       <CommonDirectives>
       [ , ELEMENTS [ XSINIL | ABSENT ] ]
     ]
   }

 <CommonDirectives> ::=
  [ , BINARY BASE64 ]
  [ , TYPE ]
  [ , ROOT [ ('RootName') ] ]
```

Семантика режимов RAW, AUTO, EXPLICIT

- В режиме RAW создается одиночный элемент <row> для каждой строки набора строк, возвращенного инструкцией SELECT. XML-иерархию можно создать с помощью написания вложенных запросов FOR XML.
- В режиме AUTO вложенность XML создается эвристически, в зависимости от метода определения инструкции SELECT. Управление формой создаваемой XML структуры минимально. Для создания XML-иерархии, расширяющей возможности XML-структуры, созданной эвристически в режиме AUTO, можно написать вложенные запросы FOR XML.
- Режим EXPLICIT предоставляет больше возможностей по управлению формой XML-структуры. В XML-структуре могут быть использованы смешанные атрибуты и элементы. Это требует особого формата для результирующего набора строк, создаваемого в результате выполнения запроса. Формат этого набора строк затем сопоставляется с формой XML-структуры. Преимущества режима EXPLICIT состоят в возможности использования смешанных атрибутов и элементов, в возможности создания упаковщиков и вложенных составных свойств, создания значений, разделенных пробелами (например, атрибут OrderID может содержать список значений идентификаторов порядка), и смешанного содержимого. Однако написание запросов в режиме EXPLICIT может быть очень обременительным. Можно использовать некоторые новые возможности предложения FOR XML, например написание вложенных запросов FOR XML в режиме RAW/AUTO/PATH и директивы TYPE вместо использования режима EXPLICIT для создания иерархий. Вложенные запросы FOR XML могут создавать любую XML структуру, которую можно создать с помощью режима EXPLICIT.

- Режим PATH совместно с вложенным запросом FOR XML обеспечивает гибкость режима EXPLICIT более простым образом.

Семантика параметров XMLDATA, XMLSCHEMA, ELEMENTS, BINARY BASE64, TYPE, ROOT

- XMLDATA указывает, что будет возвращена встроенная XDR-схема. Эта схема присоединяется к документу в качестве встроенной схемы.

a. XMLSCHEMA возвращает встроенную XML-схему W3C (XSD). Также при определении этой директивы можно указать целевой URI пространства имен. При этом в схему возвращается указанное пространство имен.

b. ELEMENTS возвращает столбцы в виде вложенных элементов. В противном случае они сопоставляются с XML-атрибутами. Этот параметр поддерживается только режимами RAW, AUTO и PATH. Пи использовании этой директивы можно дополнительно указать ключевые слова XSINIL или ABSENT. Ключевое слово XSINIL указывает на то, что элемент имеет атрибут **xsi:nil**, установленный в значение True для столбцов со значением NULL. По умолчанию или при указании вместе с параметром ELEMENTS ключевого слова ABSENT, для значений NULL столбцы не создаются.

c. BINARY BASE64 означает, что любые двоичные данные, возвращенные запросом, будут представлены в формате base64. Чтобы получить двоичные данные при помощи режимов RAW и EXPLICIT, необходимо указать этот параметр. В режиме AUTO двоичные данные возвращаются по умолчанию.

d. TYPE указывает на то, что запрос возвращает результаты в виде типа **xml**.

e. ROOT [('*RootName*')] указывает, что к результирующему XML-документу будет добавлен один элемент верхнего уровня. Дополнительно можно указать имя корневого элемента, который необходимо сформировать. Значение по умолчанию - «root».

Пример 1.

```
USE Northwind
GO
SELECT TOP 1 EmployeeID, LastName, FirstName, Title, TitleOfCourtesy, BirthDate
FROM Employees
FOR XML AUTO, ELEMENTS
```

Результаты

```
<Employees>
  <EmployeeID>1</EmployeeID>
  <LastName>Davolio</LastName>
  <FirstName>Nancy</FirstName>
  <Title>Sales Representative</Title>
  <TitleOfCourtesy>Ms.</TitleOfCourtesy>
  <BirthDate>1948-12-08T00:00:00</BirthDate>
</Employees>
```

Пример 2.

```
USE Northwind
GO
SELECT TOP 1 EmployeeID, LastName, FirstName, Title, TitleOfCourtesy, BirthDate
FROM Employees
FOR XML RAW, XMLDATA
```

Результаты

```
<Schema name="Schema1" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-
com:datatypes">
  <ElementType name="row" content="empty" model="closed">
    <AttributeType name="EmployeeID" dt:type="i4" />
    <AttributeType name="LastName" dt:type="string" />
    <AttributeType name="FirstName" dt:type="string" />
    <AttributeType name="Title" dt:type="string" />
    <AttributeType name="TitleOfCourtesy" dt:type="string" />
```

```
    <AttributeType name="BirthDate" dt:type="dateTime" />
    <attribute type="EmployeeID" />
    <attribute type="LastName" />
    <attribute type="FirstName" />
    <attribute type="Title" />
    <attribute type="TitleOfCourtesy" />
    <attribute type="BirthDate" />
  </ElementType>
</Schema>
<row  xmlns="x-schema:#Schema1"  EmployeeID="1"  LastName="Davolio"  FirstName="Nancy"  Title="Sales
Representative" TitleOfCourtesy="Ms." BirthDate="1948-12-08T00:00:00" />
```

Пример 3.

```
USE Northwind
GO
SELECT TOP 1 *
FROM Customers
FOR XML RAW('Customer'), ELEMENTS
GO
```

Результаты

```
<Customer>
  <CustomerID>ALFKI</CustomerID>
  <CompanyName>Alfreds Futterkiste</CompanyName>
  <ContactName>Maria Anders</ContactName>
  <ContactTitle>Sales Representative</ContactTitle>
  <Address>Obere Str. 57</Address>
  <City>Berlin</City>
  <PostalCode>12209</PostalCode>
  <Country>Germany</Country>
  <Phone>030-0074321</Phone>
  <Fax>030-0076545</Fax>
</Customer>
```

Пример 4.

```
USE Northwind
GO
SELECT TOP 1 *
FROM Customers
WHERE Region is null
FOR XML PATH('Customer'), ELEMENTS XSINIL
GO
```

Результаты

```
<Customer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <CustomerID>ALFKI</CustomerID>
  <CompanyName>Alfreds Futterkiste</CompanyName>
  <ContactName>Maria Anders</ContactName>
  <ContactTitle>Sales Representative</ContactTitle>
  <Address>Obere Str. 57</Address>
  <City>Berlin</City>
  <Region xsi:nil="true" />
  <PostalCode>12209</PostalCode>
  <Country>Germany</Country>
  <Phone>030-0074321</Phone>
  <Fax>030-0076545</Fax>
  <CreditLimit xsi:nil="true" />
</Customer>
```

Задание 1

Из таблиц базы данных, созданной в ЛР № 1, извлечь данные с помощью конструкции FOR XML; проверить все режимы конструкции FOR XML. Если написание запросов режима EXPLICIT окажется трудоемким, то обратиться к разделу MSDN Library «Использование режима EXPLICIT совместно с предложением FOR XML» по адресу https://msdn.microsoft.com/ru-ru/library/ms189068(v=sql.120).aspx.

Часть 2. Загрузка и сохранение XML-документа в базе данных SQL Server

Для получения доступа к XML-данным так, как если бы это был реляционный набор строк, используется функция OPENXML. Записи в наборе строк могут быть сохранены в таблицах базы данных. OPENXML может использоваться в инструкциях SELECT и SELECT INTO в любых позициях, где в качестве источника могут присутствовать поставщики наборов строк, представления или функция OPENROWSET. Для понимания OPENXML необходимо иметь общее представление о запросах XPath

Чтобы писать запросы в отношении XML-документа с использованием OPENXML, необходимо сначала вызвать хранимую процедуру sp_xml_preparedocument. Таким образом производится синтаксический анализ XML-документа и возвращается дескриптор для проанализированного документа, готового к использованию. Проанализированный документ является представлением дерева объектной модели документа различных узлов в XML-документе. Дескриптор документа передается OPENXML. Затем OPENXML выдает представление документа в виде набора строк, основываясь на переданных ей аргументах. Наконец, внутреннее отображение XML-документа должно быть удалено из памяти посредством вызова системной хранимой процедуры sp_xml_removedocument.

Синтаксис функции OPENXML

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

Семантика параметров функции OPENXML

- *idoc* – дескриптор (описатель) внутреннего представления XML-документа. Внутреннее представление XML-документа создается при помощи вызова процедуры **sp_xml_preparedocument**.
- *rowpattern* – шаблон (выражение) XPath, используемый для идентификации узлов (в XML-документе, дескриптор которого передается в аргументе *idoc*), которые будут обработаны как строки.
- *flags* - указывает на сопоставление, которое должно использоваться между XML-данными и реляционным набором строк, а также на порядок заполнения переполненного столбца. Аргумент *flags* является необязательным входным параметром и может принимать одно из следующих значений: 0, 1, 2 и 8.
- *SchemaDeclaration* – схема набора строк, в которой указывают имена столбцов, их типы данных и сведения об их соответствии элементам XML-документа.
- *TableName* – название таблицы, которое может быть указано (вместо аргумента *SchemaDeclaration*), если таблица с необходимой схемой уже существует и не требует никакого шаблона столбцов.

Если отказываются от использования конструкции WITH, то в этом случае OPENXML возвращает набор строк в формате **краевой** таблицы. Краевые таблицы представляют собой структуру мелкогранулированного XML-документа (имена элементов/атрибутов, иерархия документа, пространства имен, и т. д.) в одной таблице.

Пример 1. Использование простой инструкции SELECT с OPENXML

```
DECLARE @idoc int
DECLARE @doc xml
SET @doc ='
<ROOT>
```

```
<Customer CustomerID="VINET" ContactName="Paul Henriot">
   <Order CustomerID="VINET" EmployeeID="5" OrderDate="1996-07-04T00:00:00">
      <OrderDetail OrderID="10248" ProductID="11" Quantity="12"/>
      <OrderDetail OrderID="10248" ProductID="42" Quantity="10"/>
   </Order>
</Customer>
<Customer CustomerID="LILAS" ContactName="Carlos Gonzlez">
   <Order CustomerID="LILAS" EmployeeID="3" OrderDate="1996-08-16T00:00:00">
      <OrderDetail OrderID="10283" ProductID="72" Quantity="3"/>
   </Order>
</Customer>
</ROOT>'
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
SELECT *
FROM OPENXML (@idoc, '/ROOT/Customer',1)
     WITH (CustomerID  varchar(10), ContactName varchar(20))
```

Результаты

```
CustomerID ContactName
---------- --------------------
VINET      Paul Henriot
LILAS      Carlos Gonzlez
```

Пример 2. Использование функции OPENROWSET для считывания данных из файла без их загрузки в целевую таблицу. Это позволяет использовать функцию OPENROWSET совместно с обычной инструкцией SELECT.

```
DECLARE @idoc int
DECLARE @doc xml
SELECT @doc = c FROM OPENROWSET(BULK 'D:\input.xml', SINGLE_BLOB) AS TEMP(c)
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
SELECT *
FROM OPENXML (@idoc, '/ROOT/Customer',1)
     WITH (CustomerID  varchar(10), ContactName varchar(20))
```

Результаты

```
CustomerID ContactName
---------- --------------------
VINET      Paul Henriot
LILAS      Carlos Gonzlez
```

Пример 3. Получение результатов в формате краевой таблицы

```
DECLARE @idoc int
DECLARE @doc xml
SELECT @doc = c FROM OPENROWSET(BULK 'D:\input.xml', SINGLE_BLOB) AS TEMP(c)
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
SELECT TOP 2 id, parented, nodetype, localname, prefix
FROM OPENXML (@idoc, '/ROOT/Customers')
EXEC sp_xml_removedocument @idoc
```

Результаты

```
id     nodetype            localname     prefix
------ -------------       ------------ -------------
2      1                   Customers    NULL
3      2                   CustomerID   NULL
```

Пример 4. Загрузка и сохранение XML-документа в таблице базы данных

Создание тестовой базы данных Accounting

```
/*
** Внимание! Данный сценарий получен автоматически в среде SQL Server Management Studio,
** так что не принимайте его как образец того, как необходимо форматировать инструкции T-SQL
** при написании собственного сценария
*/
USE master
GO
EXEC msdb.dbo.sp_delete_database_backuphistory @database_name = N'Accounting'
GO
IF EXISTS (SELECT * FROM sys.databases WHERE name = 'Accounting')
       DROP DATABASE [Accounting]
GO
CREATE DATABASE Accounting
GO
USE Accounting
GO
/****** Создаем таблицу [dbo].[Employees] - Сотрудники ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Employees](
       [EmployeeID] [int] IDENTITY(1,1) NOT NULL,
       [FirstName] [varchar](25) NOT NULL,
       [MiddleInitial] [char](1) NULL,
       [LastName] [varchar](25) NOT NULL,
       [Title] [varchar](25) NOT NULL,
       [SSN] [varchar](11) NOT NULL,
       [Salary] [money] NOT NULL,
       [PriorSalary] [money] NOT NULL,
       [LastRaise]  AS ([Salary]-[PriorSalary]),
       [HireDate] [date] NOT NULL,
       [TerminationDate] [date] NULL,
       [ManagerEmpID] [int] NOT NULL,
       [Department] [varchar](25) NOT NULL,
 CONSTRAINT [PK_EmployeeID] PRIMARY KEY CLUSTERED
(
       [EmployeeID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY],
 CONSTRAINT [AK_EmployeeSSN] UNIQUE NONCLUSTERED
(
       [SSN] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Создаем таблицу [dbo].[Customers] - Заказчики ******/
SET ANSI_PADDING OFF
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Customers](
       [CustomerNo] [int] IDENTITY(1,1) NOT NULL,
       [CustomerName] [varchar](30) NOT NULL,
       [Address1] [varchar](30) NOT NULL CONSTRAINT [CN_CustomerAddress]  DEFAULT ('UNKNOWN'),
```

```
	[Address2] [varchar](30) NOT NULL,
	[City] [varchar](20) NOT NULL,
	[State] [char](2) NOT NULL,
	[Zip] [varchar](10) NOT NULL,
	[Contact] [varchar](25) NOT NULL,
	[Phone] [char](15) NOT NULL,
	[FedIDNo] [varchar](9) NOT NULL,
	[DateInSystem] [datetime] NOT NULL CONSTRAINT [CN_CustomerDefaultDateInSystem]  DEFAULT
(getdate()),
PRIMARY KEY CLUSTERED
(
	[CustomerNo] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
-- Наполняем таблицу Customers двумя строками
SET ANSI_PADDING OFF
INSERT INTO Customers
VALUES
   ('Billy Bob''s Shoes',
   '123 Main St.',
   ' ',
   'Vancouver',
   'WA',
   '98685',
   'Billy Bob',
   '(360) 555-1234',
   '931234567',
   GETDATE()
   )
INSERT INTO Customers
   (CustomerName,
    Address1,
    Address2,
    City,
    State,
    Zip,
    Contact,
    Phone,
    FedIDNo,
    DateInSystem)
VALUES
   ('MyCust',
    '123 Anywhere',
    '',
    'Reno',
    'NV',
    80808,
    'Joe Bob',
    '555-1212',
    '931234567',
    GETDATE ())
GO
/****** Создаем таблицу [dbo].[Shipper] - Грузоотправители ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Shippers](
	[ShipperID] [int] IDENTITY(1,1) NOT NULL,
	[ShipperName] [varchar](30) NOT NULL,
	[DateInSystem] [datetime] NOT NULL DEFAULT (getdate()),
```

```
PRIMARY KEY CLUSTERED
(
	[ShipperID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Создаем таблицу [dbo].[Orders] - Заказы ******/
SET ANSI_PADDING OFF
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Orders](
	[OrderID] [int] IDENTITY(1,1) NOT NULL,
	[CustomerNo] [int] NOT NULL,
	[OrderDate] [date] NOT NULL,
	[EmployeeID] [int] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[OrderID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Создаем таблицу [dbo].[OrderDetails] - Детализация заказов ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[OrderDetails](
	[OrderID] [int] NOT NULL,
	[PartNo] [varchar](10) NOT NULL,
	[Description] [varchar](25) NOT NULL,
	[UnitPrice] [money] NOT NULL,
	[Qty] [int] NOT NULL,
 CONSTRAINT [PKOrderDetails] PRIMARY KEY CLUSTERED
(
	[OrderID] ASC,
	[PartNo] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
ALTER TABLE [dbo].[Customers]  WITH NOCHECK ADD  CONSTRAINT [CN_CustomerDateInSystem] CHECK
(([DateInSystem]<=getdate()))
GO
ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [CN_CustomerDateInSystem]
GO
ALTER TABLE [dbo].[Customers]  WITH NOCHECK ADD  CONSTRAINT [CN_CustomerPhoneNo] CHECK  (([Phone]
like '([0-9][0-9][0-9]) [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]'))
GO
ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [CN_CustomerPhoneNo]
GO
ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT [FK_EmployeeHasManager] FOREIGN
KEY([ManagerEmpID])
REFERENCES [dbo].[Employees] ([EmployeeID])
GO
ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [FK_EmployeeHasManager]
GO
```

```
ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [FKOrderContainsDetails] FOREIGN
KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [FKOrderContainsDetails]
GO
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD FOREIGN KEY([CustomerNo])
REFERENCES [dbo].[Customers] ([CustomerNo])
GO
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [FK_EmployeeCreatesOrder] FOREIGN
KEY([EmployeeID])
REFERENCES [dbo].[Employees] ([EmployeeID])
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_EmployeeCreatesOrder]
GO
-- Наполняем таблицу Employees шестью строками
SET IDENTITY_INSERT Employees ON;
INSERT INTO Employees
   (
   EmployeeID,
   FirstName,
   LastName,
   Title,
   SSN,
   Salary,
   PriorSalary,
   HireDate,
   ManagerEmpID,
   Department,
   TerminationDate
   )
VALUES
   (
   1,
   'Peter',
   'Principle',
   'Partner',
   '456-78-9012',
   150000,
   130000,
   '1990-01-01',
   1,
   'Executive',
   NULL
   ),
   (
   2,
   'Billy Bob',
   'Boson',
   'Head Cook & Bottle Washer',
   '123-45-6789',
   100000,
   80000,
   '1990-01-01',
   1,
   'Cooking and Bottling',
   NULL
   ),
   (
   3,
   'Joe',
   'Dokey',
   'Minion',
   '234-56-7890',
```

```
   50000,
   40000,
   '2000-01-01',
   2,
   'Downstairs',
   NULL
   ),
   (
   4,
   'Steve',
   'Smith',
   'Trader',
   '345-67-8901',
   60000,
   60000,
   '2006-04-01',
   1,
   'Currency Trading',
   '2007-01-31'
   ),
   (
   5,
   'Howard',
   'Kilroy',
   'Trader',
   '567-89-0123',
   65000,
   60000,
   '2007-02-01',
   1,
   'Currency Trading',
   NULL
   ),
   (
   6,
   'Mary',
   'Contrary',
   'Minion',
   '678-90-1234',
   65000,
   60000,
   '1998-06-15',
   2,
   'Currency Trading',
   '2008-01-01'
   );
SET IDENTITY_INSERT Employees OFF;
-- Добавляем в таблицу Customers третью строку, отличающуюся от первой в колонках CustomerNo и --
DateInSystem
INSERT INTO Customers
VALUES
   ('Billy Bob''s Shoes',
   '123 Main St.',
   ' ',
   'Vancouver',
   'WA',
   '98685',
   'Billy Bob',
   '(360) 555-1234',
   '931234567',
   GETDATE()
   )
-- Добавляем в таблицу Shippers одну строку
INSERT INTO Shippers (ShipperName)
VALUES ('United Parcel Service')
```

```
-- Таблицы Order и OrdersDetails остаются пустыми
```

Использование инструкции INSERT с OPENXML

```
USE Accounting;
GO

-- Объявляем переменные
DECLARE @idoc      int; -- Дескриптор документа внутреннего представления XML-документа
DECLARE @xmldoc    nvarchar(4000); -- XML-документ
-- Определяем XML-документ
SET @xmldoc = '
<ROOT>
<Shipper ShipperID="100" CompanyName="Billy Bob&apos;s Great Shipping"/>
<Shipper ShipperID="101" CompanyName="Fred&apos;s Freight"/>
</ROOT>
';
-- При помощи вызова процедуры sp_xml_preparedocument создаем внутреннее представление
-- XML-документа @xmldoc, на которое будем ссылаться в дальнейшем с помощью дескриптора @idoc
EXEC sp_xml_preparedocument @idoc OUTPUT, @xmldoc;
-- Получаем полную информацию из таблицы Shippers перед выполнение инструкции INSERT
SELECT * FROM Shippers;
-- ShipperID является столбцом IDENTITY, поэтому присваиваем свойству IDENTITY_INSERT значение --
ON, что позволяет вставлять явные значения в столбец идентификаторов таблицы
SET IDENTITY_INSERT Shippers ON
-- Смотрим на XML-данные в табличном формате
SELECT * FROM OPENXML (@idoc, '/ROOT/Shipper', 0) WITH (
    ShipperID           int,
    CompanyName         nvarchar(40));
-- Выполняем инструкцию INSERT в таблицу Shippers
INSERT INTO Shippers (ShipperID, ShipperName)
SELECT * FROM OPENXML (@idoc, '/ROOT/Shipper', 0) WITH (
    ShipperID           int,
    CompanyName         nvarchar(40));
-- Присваиваем свойству IDENTITY_INSERT первоначальное значение OFF
SET IDENTITY_INSERT Shippers OFF;
-- Смотрим на таблицу Shippers после выполнения инструкции INSERT
SELECT * FROM Shippers;
-- Удаляем XML-документ из памяти
EXEC sp_xml_removedocument @idoc;
```

Протокол выполнения сценария

```
ShipperID   ShipperName                    DateInSystem
----------- ------------------------------ -----------------------
          1 United Parcel Service          2012-10-14 21:30:14.687

ShipperID   CompanyName
----------- ------------------------------
        100 Billy Bob's Great Shipping
        101 Fred's Freight

ShipperID   ShipperName                    DateInSystem
----------- ------------------------------ -----------------------
          1 United Parcel Service          2012-10-14 21:30:14.687
        100 Billy Bob's Great Shipping     2012-10-14 21:39:33.020
        101 Fred's Freight                 2012-10-14 21:39:33.020
```

Задание 2

С помощью функции OPENXML и OPENROWSET выполнить загрузку и сохранение XML-документа в таблице базы данных, созданной в ЛР № 1.

Часть 3. Создание DTD-описаний и XSD-схем

В настоящей части работы обсуждаются вопросы, связанные с созданием DTD-описаний и XSD-схем. В качестве примера будет рассматриваться XML-документ `Company.xml`, имеющий следующий вид:

```
<?xml version="1.0" encoding="utf-8"?>
<Company>
  <CompanyName>My Cool Startup Company</CompanyName>
  <RegDate>1997-07-02</RegDate>
  <Employee EmpNum="123" JobTitle="Director">
    <Name>Chris Peterson</Name>
    <Tel>222-7777-123</Tel>
    <Salary>79500.00</Salary>
  </Employee>
  <Employee EmpNum="456" JobTitle="Manager">
    <Name>Emily Williams</Name>
    <Tel>222-7777-456</Tel>
    <Salary>52750.00</Salary>
  </Employee>
  <Employee EmpNum="789" JobTitle="Programmer">
    <Name>Thomas Smith</Name>
    <Tel>222-7777-789</Tel>
    <DailyRate>425.00</DailyRate>
  </Employee>
</Company>
```

В этом документе отражены некоторые важные особенности XML, с которыми можно столкнуться при рассмотрении DTD и XSD, а именно:

- Элемент `<Company>` имеет дочерний элемент `<CompanyName>`, дочерний элемент `<RegDate>` и некоторое количество (от нуля и более) дочерних элементов `<Employee>`.
- Элемент `<CompanyName>` имеет строковое значение, а элемент `<RegDate>` значение типа даты.
- Элемент `<Employee>` имеет атрибут `EmpNum`, который может содержать произвольное строковое значение, и атрибут `JobTitle`, который может принимать одно из следующих значений: `"Director"`, `"Manager"` или `"Programmer"`.
- Элемент `<Employee` имеет три дочерних элемента: `<Name>` (строка), `<Tel>` (строка), а также либо `<Salary>`, либо `<DailyRate>` (оба - дробные числа).

Описание грамматики с помощью DTD

DTD-описания имеют особые взаимоотношения с XML, поскольку они включены в спецификацию XML. Несмотря на появление стандарта XSD, DTD-описания, вероятно, будут использоваться еще на протяжении определенного времени. DTD-описание для XML-документа, приведенного выше, будет иметь вид:

```
<!ELEMENT Company (CompanyName, RegDate, Employee*)>

<!ELEMENT CompanyName (#PCDATA)>
<!ELEMENT RegDate (#PCDATA)>
<!ELEMENT Employee (Name, Tel, (Salary | DailyRate))>

<!ATTLIST Employee
   EmpNum CDATA #REQUIRED
   JobTitle (Director | Manager | Programmer) #REQUIRED>
```

```
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Tel (#PCDATA)>
<!ELEMENT Salary (#PCDATA)>
<!ELEMENT DailyRate (#PCDATA)>
```

Объявления `<!ELEMENT>` определяют содержимое элемента, а объявления `<!ATTLIST>` - список атрибутов элемента. Ключевые слова `#PCDATA` и `CDATA` используются для обозначения текста, однако, они не позволяют описывать конкретные типы данных в DTD.

DTD-описания имеют ряд других ограничений:

- DTD не поддерживает пространств имен, которые, скорее всего, будут иметь все большее значение по мере того, как XML-разработчики будут все шире использовать их в будущем.
- Ограниченная расширяемость – например, в DTD отсутствует возможность описания типа элемента или атрибута через другие типы.

Более подробная информация о DTD-описаниях содержится в файле DTD.doc. Некоторые примеры использования DTD-описаний можно найти по адресам: http://www.oasis-open.org/cover/xml.html и http://www.dublincore.org/.

Описание грамматики с использованием XSD

Полная XSD-схема для XML-документа, приведенного выше, будет иметь вид:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Company" type="CompanyType"/>
  <xs:complexType name="CompanyType">
    <xs:sequence>
      <xs:element name="CompanyName" type="xs:string"/>
      <xs:element name="RegDate" type="xs:date"/>
      <xs:element name="Employee" type="EmployeeType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="EmployeeType">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Tel"  type="xs:string"/>
      <xs:choice>
        <xs:element name="Salary"    type="xs:double"/>
        <xs:element name="DailyRate" type="xs:double"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="EmpNum"   use="required" type="xs:string"/>
    <xs:attribute name="JobTitle" use="required" type="JobTitleType"/>
  </xs:complexType>
  <xs:simpleType name="JobTitleType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Director"/>
      <xs:enumeration value="Manager"/>
      <xs:enumeration value="Programmer"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

в XSD-схеме элементом, обозначающим документ, является <schema>. Для указания того, что данный документ представляет собой XSD-схему, необходимо задать следующий URI пространства имен:

http://www.w3.org/2001/XMLSchema. В соответствии с соглашением, в документах, являющихся XSD-схемами, используется префикс пространства имен xsd. Префикс xsd может не работать в .NET Framework. В этом случае следует использовать префикс xs, хотя пространство имен останется тем же самым - http://www.w3.org/2001/XMLSchema. За информацией о внесенных изменениях обращайтесь по адресу http://www.msdn.microsoft.com/.

Внутри XSD-схемы для объявления элементов и атрибутов применяются конструкции `<xs:element>` и `<xs:attribute>` соответственно. Например, конструкция `<xs:element name="Company" type="CompanyType"/>` определяет элемент с названием `<Company>` в данном экземпляре XML-документа.

Для описания содержимого элемента, имеющего сложную структуру, используется конструкция `<xs:complexType>`. Например, конструкция `<xs:complexType name="CompanyType">` описывает сложное содержимое элемента `<Company>`. Элемент `<Company>` обладает дочерним элементом `<CompanyName>`, за которым следует дочерний элемент `<RegDate>`, а за ним идет произвольное число элементов `<Employee>`.

Для описания новых простых типов применяется конструкция `<xs:simpleType>`.
Например, конструкция `<xs:simpleType name="JobTitleType">` позволяет создать новый тип `string`, который может принимать одно из следующих значений: `"Director"`, `"Manager"` или `"Programmer"`.

Создание XSD-схемы в Visual Studio

Чтобы создать XSD-схему в Visual Studio с применением существующего XML-документа необходимо:
1. В Visual Studio выполнить команду `Файл|Открыть|Файл`.
2. В диалоговом окне `Открыть файл` выбрать XML-документ и нажмите кнопку `Открыть`.
3. В Visual Studio выполнить команду `XML-код|Создать схему`. Откроется окно редактора с новым XSD-файлом.

Для демонстрационного файла `Company.xml` будет создана следующая схема XSD:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Company">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CompanyName" type="xs:string" />
        <xs:element name="RegDate" type="xs:date" />
        <xs:element maxOccurs="unbounded" name="Employee">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Name" type="xs:string" />
              <xs:element name="Tel" type="xs:string" />
              <xs:element minOccurs="0" name="DailyRate" type="xs:decimal" />
              <xs:element minOccurs="0" name="Salary" type="xs:decimal" />
            </xs:sequence>
            <xs:attribute name="EmpNum" type="xs:unsignedShort" use="required" />
            <xs:attribute name="JobTitle" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Создание XSD-схемы с использованием XSD Generator

Наряду с созданием XSD-схем посредством Visual Studio можно использовать для этой цели вызываемый из командной строки инструмент, который называется XSD Generator. Он является составной частью Visual Studio и предлагает некоторые дополнительные возможности для создания схем XSD с применением существующего исходного XML-документа.

Кроме того, XSD Generator может быть использован в качестве инструмента для решения двух дополнительных задач:

a. Для создания схемы XSD из типа или типов в файле сборки CLR.
b. Для создания классов CLR DataSet из файла схемы XSD.

Подробная информация об использовании инструмента XSD Generator находится в разделе «Инструмент определения схемы XML (Xsd.exe)» по адресу http://msdn.microsoft.com/ru-ru/library/x6c1kb0s.aspx

Чтобы создать XSD-схему из XML-документа откройте окно «Командная строка» и запустите инструмент XSD Generator, используя следующий синтаксис:

```
xsd file.xml [/outputdir:directory]
```

Эта команда создает схему XSD из `file.xml` и сохраняет ее в указанный каталог.

Если при запуске инструмента XSD Generator выдается сообщение об ошибке, убедитесь в том, что вы указали правильный путь. Например, на некоторой машине этот инструмент установлен в папке C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Bin

Инструмент XSD Generator генерирует XSD-схему с тем же именем, что и у XML-документа, но с расширением `.xsd`. Схема помещается в папку, указанную для выходного потока, либо в текущую папку, если параметр `/outputdir:directory` опущен.

Для демонстрационного файла `Company.xml` инструментом XSD Generator будет создана следующая схема XSD:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="Company">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CompanyName" type="xs:string" minOccurs="0" />
        <xs:element name="RegDate" type="xs:string" minOccurs="0" />
        <xs:element name="Employee" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Name" type="xs:string" minOccurs="0" msdata:Ordinal="0" />
              <xs:element name="Tel" type="xs:string" minOccurs="0" msdata:Ordinal="1" />
              <xs:element name="DailyRate" type="xs:string" minOccurs="0" msdata:Ordinal="2"
/>
              <xs:element name="Salary" type="xs:string" minOccurs="0" msdata:Ordinal="3" />
            </xs:sequence>
            <xs:attribute name="EmpNum" type="xs:string" />
            <xs:attribute name="JobTitle" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>
  <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="Company" />
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Иногда можно заметить, что XSD Generator присваивает тип данных "xs:string" всем элементам в схеме. Это происходит потому, что XML-документ не передает инструменту XSD Generator информацию, достаточную для гарантированного определения других типов данных. Схему можно модифицировать вручную так, чтобы в ней использовались корректные типы данных.

Задание 3

1. Создать DTD для XML-документа, набрав описание вручную с помощью какого-либо текстового редактора.
2. Создать XSD-схему из XML-документа в Visual Studio.
3. Создать XSD-схему из XML-документа с помощью инструмента XSD Generator.

Часть 4. Проверка допустимости XML-документа

.NET Framework позволяет проверить допустимость XML-документа программным путем с помощью DTD-описаний и XSD-схем. Перед выполнением проверки допустимости необходимо сообщить анализатору, какую схему или DTD необходимо применить.

Привязка XML-документа к DTD или схеме

При создании XML-документа можно включить в него статическую ссылку на DTD или XSD. Определение статической ссылки на DTD или схему имеет смысл в том случае, если заранее известно, какие DTD или схемы будут применяться, например, когда разрабатывается приложение для внутреннего использования и имеется доступ к соответствующей DTD или схеме. Однако существует также возможность динамического применения схемы с помощью объекта XmlSchemaCollection.

Привязка XML-документа к DTD

Для того чтобы привязать XML-документ к DTD, следует включить описание `<!DOCTYPE>` в начало документа. Ниже приводится простой пример XML-документа под именем `CompanySummaryWithDTD.xml`, который привязывается к DTD-файлу с именем `CompanySummary.dtd`:

```
<?xml version="1.0"?>
<!DOCTYPE CompanySummary SYSTEM "CompanySummary.dtd">
<CompanySummary CompanyName="My Cool Startup Company">
       <RegDate>1997-07-02</RegDate>
       <NumEmps>3</NumEmps>
</CompanySummary>
```

DTD для этого документа – `CompanySummary.dtd` – имеет следующий вид:

```
<!ELEMENT CompanySummary (RegDate, NumEmps)>
<!ATTLIST CompanySummary CompanyName CDATA #REQUIRED>
<!ELEMENT RegDate (#PCDATA)>
<!ELEMENT NumEmps (#PCDATA)>
```

Привязка XML-документа к XSD-схеме

В случае привязки XML-документа к XSD-схеме ситуация оказывается более сложной, поскольку каждый XML-документ может быть привязан сразу к нескольким XSD-схемам. Такое положение вещей складывается тогда, когда различные части XML-документа квалифицируются разными пространствами имен; в этом случае можно привязать отдельную XSD-схему для каждого пространства имен.

Синтаксис привязки XML-документа к XSD-схеме зависит от того, квалифицируется или нет содержимое документа каким-либо пространством имен.

Неквалифицируемое содержимое XML-документа

Если содержимое XML-документа не квалифицируется каким-либо пространством имен, то для привязки этого документа к XSD-схеме следует включить в него описания:

```
xmlns:xsi="http://www.w3.org/2001/SMLSchema-instance"
xsi:noNamespaceSchemaLocation="url-of-XSD-schema"
```

Например, следующий XML-документ под именем `CompanySummaryWithXSD.xml` привязывается к XSD-схеме с именем `CompanySummary.xsd`:

```
<?xml version="1.0"?>
<CompanySummary CompanyName="My Cool Startup Company"
            xmlns:xsi="http://www.w3.org/2001/SMLSchema-instance"
            xsi:noNamespaceSchemaLocation="file://CompanySummary.xsd">
       <RegDate>1997-07-02</RegDate>
       <NumEmps>3</NumEmps>
</CompanySummary>
```

Содержимое XML-документа, квалифицируемое пространством имен

Если содержимое XML-документа квалифицируется каким-либо пространством имен, то следует включить в XML-документ следующие объявления:

```
xmlns:xsi="http://www.w3.org/2001/SMLSchema-instance"
xsi:schemaLocation="namespace-url-of-XSD-schema"
```

Число объявлений `xsi:schemaLocation` может быть произвольным – по одному на каждое пространство имен, используемое в XML-документе. Кроме того, остается возможность использования объявления `xsi:noNamespaceSchemaLocation` в том случае, если XML-документ имеет неквалифицируемое содержимое.

В приводимом ниже XML-документе под именем `CompanySummaryWithXSDAndNS.xml` используется URI пространства имен `urn:MyUri` и осуществляется привязка к XSD-схеме с именем `CompanySummary.xsd` с целью описания содержимого для данного пространства имен:

```
<?xml version="1.0"?>
<CompanySummary CompanyName="My Cool Startup Company"
```

```
            xmlns="urn:MyUri"
            xmlns:xsi="http://www.w3.org/2001/SMLSchema-instance"
            xsi:schemaLocation="urn:MyUri file://CompanySummaryNS.xsd">
        <RegDate>1997-07-02</RegDate>
        <NumEmps>3</NumEmps>
</CompanySummary>
```

Следует также внести изменения в XSD-схему, чтобы задать пространство имен, используемое в данном экземпляре документа:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/SMLSchema"
            targetNamespace="namespace-urn:MyUri" elementFormDefault="qualified">
```

Выполнение проверки допустимости

.NET Framework прелагает класс `XmlValidatingReader` для выполнения проверки допустимости XML-документов. Класс `XmlValidatingReader` наследуется от класса `XmlReader`, который представляет собой быстрый курсор, движущийся по XML-документу в режиме “только вперед”.

В качестве примера предлагается написать программу, которая выполняет проверку допустимости XML-документа с использованием DTD и XSD-схемы. Начать надо с создания пустого консольного приложения, в которое затем будет последовательно включаться код, предназначенный для решения следующих задач:

- Создание объекта, выполняющего чтение с проверкой допустимости
- Задание необходимого типа проверки допустимости
- Задание метода для обработки событий, связанных с обнаруженными при выполнении проверки допустимости ошибками
- Чтение и проверка допустимости документа
- Реализация метода обработки событий, связанных с обнаруженными при выполнении проверки допустимости ошибками

Создание нового консольного приложения

Запустите Visual Studio и создайте новый проект для консольного приложения на C#. Используя операторы `using`, сделайте доступными в приложении следующие пространства имен:

```
using System;
using System.Xml;
using System.Xml.Schema;
```

Создание объекта, выполняющего чтение с проверкой допустимости

Для выполнения проверки допустимости необходимо создать объект `XmlValidatingReader`. Можно создать этот объект либо из уже существующего объекта `XmlReader`, либо из фрагмента XML, находящегося в `String` или `Stream`. Для получения более подробной информации обратитесь к разделу, относящемуся к конструктору `XmlValidatingReader`.

В приводимом примере демонстрируется создание объекта `XmlValidatingReader`, который проверяет допустимость XML-документа, расположенного на диске. Сначала создается объект `XmlTextReader` для чтения XML-документа с диска. Поскольку объект `XmlTextReader` не обладает возможностями по проверке допустимости, надо создать объект `XmlValidatingReader` и объединить его с объектом `XmlTextReader`. Добавьте этот код в начало метода `Main()`:

```
// Create a validating reader object
XmlTextReader tr = new XmlTextReader("CompanySummaryWithXSD.xml");
XmlValidatingReader vr = new XmlValidatingReader(tr);
```

Задание необходимого типа проверки допустимости

Объект `XmlValidatingReader` обеспечивает проверку допустимости с использованием DTD и XSD-схем. Для указания того, какой тип проверки допустимости требуется, следует присвоить свойству `ValidationType` одно из следующих значений:

- `ValidationType.DTD`. Используется DTD, указанное в XML-документе.
- `ValidationType.Schema`. Используется XSD-схема, указанная либо в XML-документе, либо в кэше схемы.
- `ValidationType.Auto`. Применяется наиболее подходящий тип проверки допустимости, который зависит от соответствующей информации, содержащейся в XML-документе.
- `ValidationType.None`. Проверка допустимости не производится.

Добавьте следующий код в метод `Main()`. Он указывает на необходимость проверки допустимости с использованием XSD-схемы:

```
// Specify the type of validation required
vr.ValidationType = ValidationType.Schema;
```

Задание метода для обработки событий, связанных с обнаруженными при выполнении проверки допустимости ошибками

Метод для обработки событий, связанных с обнаруженными при выполнении проверки допустимости ошибками, будет получать сообщения о возникших ошибках. Добавьте
следующий код в метод `Main()`. Этот код задает использование метода с именем `MyHandler`:

```
// Register a validation event handler method
vr.ValidationEventHandler += new ValidationEventHandler(MyHandler);
```

Чтение и проверка допустимости документа

`XmlValidatingReader` обладает методом `Read()`, который наследуется от `XmlReader` и позволяет читать содержимое XML-документа. Каждый раз, когда происходит обращение к методу `Read()`, он возвращает очередной узел XML-документа. По достижении конца документа метод `Read()` возвращает значение `false`. В этот момент необходимо закрыть `XmlValidatingReader`.

`XmlValidatingReader` сообщает обо всех ошибках, обнаруживаемых им при проверке допустимости XML-документа. В этом случае вызывается метод для обработки событий, которому передается информация об ошибках. Кроме того, при определенных условиях может возникнуть событие `XmlException` (например, при отсутствии XML-документа или схемы). Отсюда следует, что программа должна находиться внутри блока `try ... catch`, что позволит перехватить событие `XmlException` в случае его возникновения.

Добавьте следующий код в метод `Main()`. В нем используется метод `Read()` для считывания содержимого XML-документа. `XmlValidatingReader` производит проверку допустимости по мере считывания. Для иллюстрации возможностей `XmlValidatingReader` по обработке данных выводятся значения элемента `<NumEmps>`. Весь код заключен в блок `try ... catch`. Обратите внимание, что `XmlValidatingReader` закрывается в блоке `finally`; это приводит также к автоматическому закрытию находящегося на более низком уровне `XmlTextReader`:

```
// Read and validate the XML document
try
{
      while (vr.Read())
      {
            if (vr.NodeType == XmlNodeType.Element && vr.LocalName == "NumEmps")
            {
                  int num;
                  num = XmlConvert.ToInt32(vr.ReadElementString());
                  Console.WriteLine("Number of employees: " + num);
            }
      }
}
catch (XmlException ex)
{
      Console.WriteLine("XMLException occurred: " + ex.Message);
}
finally
{
      vr.Close();
}
```

Реализация метода обработки событий, связанных с обнаруженными при выполнении проверки допустимости ошибками

Ниже приводится простой метод обработки событий, который позволяет обрабатывать любые ошибки, обнаруживаемые по мере того, как `XmlValidatingReader` проходит по XML-документу. Добавьте следующий код после метода `Main()`:

```
// Validation event handler method
public static void MyHandler(object sender, ValidationEventArgs e)
{
      Console.WriteLine("Validation Error: " + e.Message);
}
```

Сборка и запуск приложения-примера

Чтобы собрать и запустить приложение-пример необходимо выполнить следующее:

- Выберите пункт `Build | Build` в Visual Studio .NET.
- Укажите `Debug | Start Without Debugging` для запуска приложения. Приложение будет анализировать и проверять допустимость XML-документа с именем `CompanySummaryWithXSD.xml`. Оно не обнаружит никаких ошибок в этом документе. (Обратите внимание: чтобы данная программа смогла найти файлы `CompanySummary.xsd` и `CompanySummaryWithXSD.xml`, они должны находиться в папке `bin`.)
- Внесите какие-либо ошибки в файл `CompanySummaryWithXSD.xml`. Например, удалите элемент `<NumEmps>`. Запустите приложение еще раз – оно обнаружит ошибку при выполнении проверки допустимости и выведет соответствующее сообщение. (Не забудьте убрать изменения, внесенные в файл `CompanySummaryWithXSD.xml`, с тем, чтобы он вновь стал допустимым.)

Полная версия приложения-примера находится в папке `ValidateCS`.

Задание 4

Написать консольное приложение на языке C#, которое выполняет проверку допустимости разработанного в текущей ЛР XML-документа, используя XSD-схему. Проведите эксперименты с XML-документом и убедитесь в том, что приложение действительно обнаруживает ошибки при проверке допустимости.