

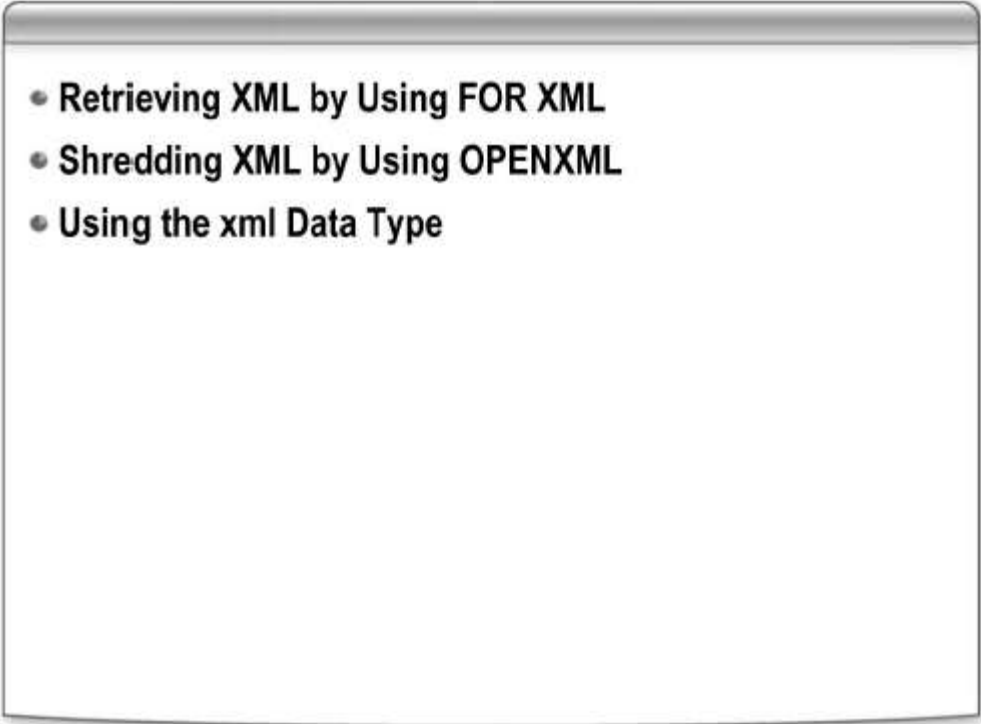
Работа с XML

Содержание:

[Урок 1: Получение XML при использовании FOR XML](#)

[Урок 2: Разбор XML при использовании OPENXML](#)

[Урок 3: Использование типа данных xml](#)

- 
- Retrieving XML by Using FOR XML
 - Shredding XML by Using OPENXML
 - Using the xml Data Type

Цели

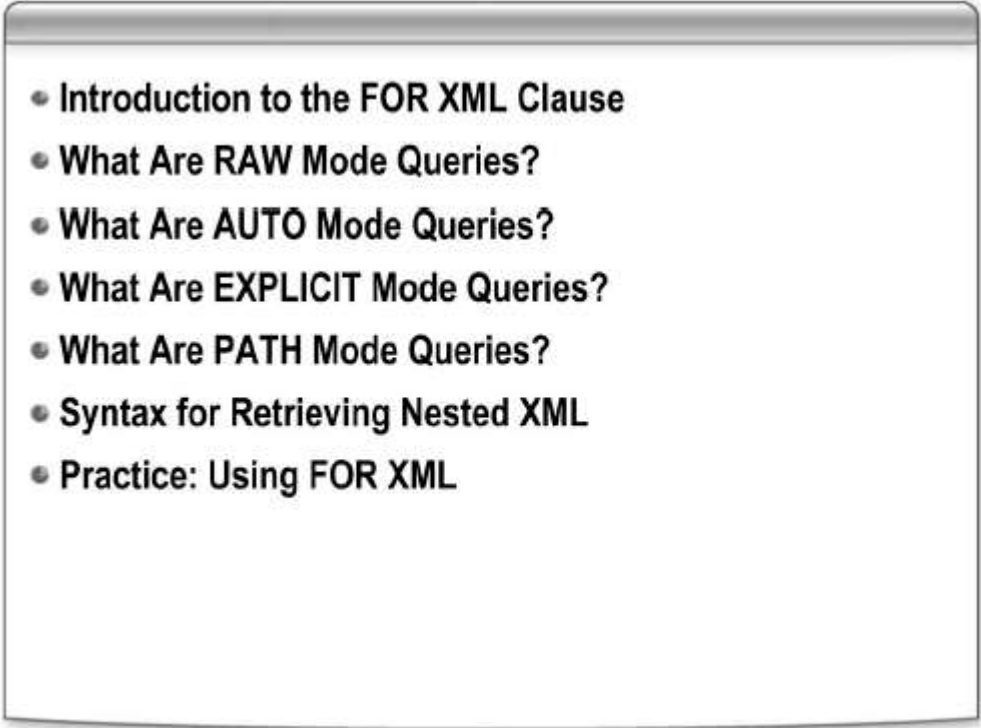
После завершения студенты смогут:

- Формировать XML-документы, используя фразу FOR XML.
- Выполнять разбор XML-документов, используя OPENXML.
- Использовать тип данных **xml**.

Введение

Возможность работы с Расширяемым Языком Разметки (XML) является общим требованием многих современных приложений. Часто разработчик приложений должен преобразовать данные между форматом XML и реляционным форматом, хранить XML и манипулировать XML-данными непосредственно в реляционной БД. Из этого модуля Вы узнаете, как преобразовать данные базы в формат XML, используя фразу FOR XML. Вы также узнаете, как преобразовать XML-документы для хранения в реляционных таблицах, используя функцию OPENXML. Наконец, Вы узнаете, как использовать тип данных **xml** Microsoft® SQL Server™ 2005, чтобы хранить XML-документы в базе данных в родном формате и как выполнить запросы к **xml** данным и их модификацию.

Урок 1: Формирование XML при использовании FOR XML

- 
- Introduction to the FOR XML Clause
 - What Are RAW Mode Queries?
 - What Are AUTO Mode Queries?
 - What Are EXPLICIT Mode Queries?
 - What Are PATH Mode Queries?
 - Syntax for Retrieving Nested XML
 - Practice: Using FOR XML

Цели урока

После завершения этого урока, студенты смогут:

- Описывать назначение фразы FOR XML.
- Описывать режим запросов RAW и объяснять их синтаксис.
- Описывать режим запросов AUTO и их синтаксис.
- Описывать режим запросов EXPLICIT и их синтаксис.
- Описывать режим запросов PATH и их синтаксис.
- Описывать синтаксис для получения вложенных XML-документов.

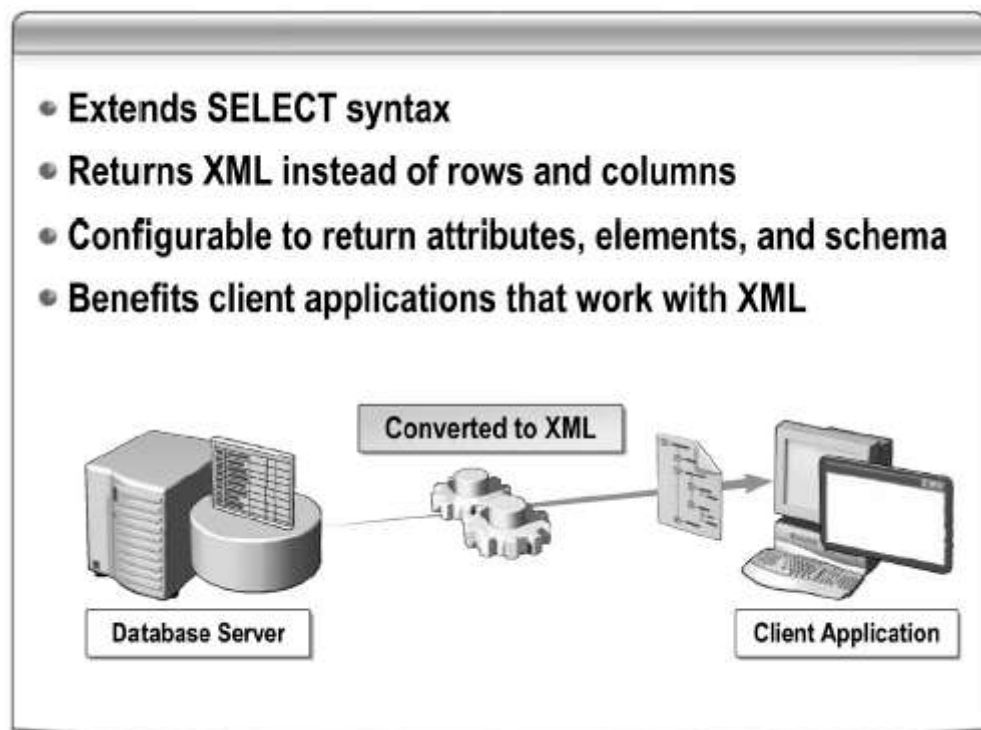
Введение

Фраза FOR XML является центральной для поиска данных XML в SQL Server 2005. Эта фраза является инструкцией для SQL Server, чтобы вернуть данные в формате XML, а не в виде набора строк. Разработчики приложений могут построить решения, которые выполняют

извлечение бизнес XML документы, таких как заказы, счета, или каталоги непосредственно из базы данных.

В этом уроке описывается, как использовать фразу FOR XML и ее различные опции, чтобы сформировать данные в формате XML.

Введение в FOR XML



Введение

Вы можете использовать фразу FOR XML в операторе SELECT языка Transact-SQL, чтобы получить данные в формате XML вместо строк и столбцов. Вы можете управлять форматом XML, определяя один из четырех режимов: RAW, AUTO, EXPLICIT или PATH. Кроме того, Вы можете определить различные опции для управления выводом.

Синтаксис фразы FOR XML

Фраза FOR XML прилагается к оператору SELECT, в соответствии со следующим синтаксисом.

FOR 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' ) ] ]

```

Обычно используемые режимы и опции фразы FOR XML описываются в следующей таблице:

Режим/опция	Описание
режим RAW	Преобразовывает каждую строку в XML элемент с автоматически сгенерированным тегом. При использовании этого режима можно произвольно определить имя для тега элемента.
режим AUTO	Возвращает результат запроса в виде простого вложенного XML-дерева. Каждая таблица фразы FROM, для которой во фразе SELECT перечислено не менее одного столбца, представляется в виде элемента XML. Столбцы, перечисленные во фразе SELECT, отображаются в соответствующие атрибуты элемента.
режим EXPLICIT	Собственный формат, определенный в запросе, определяет формат результирующих данных XML.
режим PATH	Обеспечивает наиболее простой режим для сочетания элементов и атрибутов и введения дополнительных вложений для представления сложных свойств.
опция ELEMENTS	Возвращает столбцы как подэлементы, а не как атрибуты для режимов RAW, AUTO и PATH.
опция BINARY	Возвращает двоичные данные полей, такие как image , в кодировке

BASE64	base64.
опция ROOT	Добавляет корневой элемент к получающемуся XML-документу. Когда запрос возвращает множество строк в виде XML, то по умолчанию этот XML-документ не имеет корневого элемента. Данные XML без корневого элемента называются <i>фрагментом</i> . Однако если нужно получить полный, правильно построенный документ XML с корневым элементом, Вы должны определить эту опцию. При этом можно произвольно определить название для корневого элемента.
опция TYPE	Возвращает результаты запроса как тип данных xml .
опция XMLDATA	Возвращает XML-данные в виде схемы XDR.
опция XMLSCHEMA	Возвращает XML-схему XSD Консорциума Всемирной Паутины (W3C).

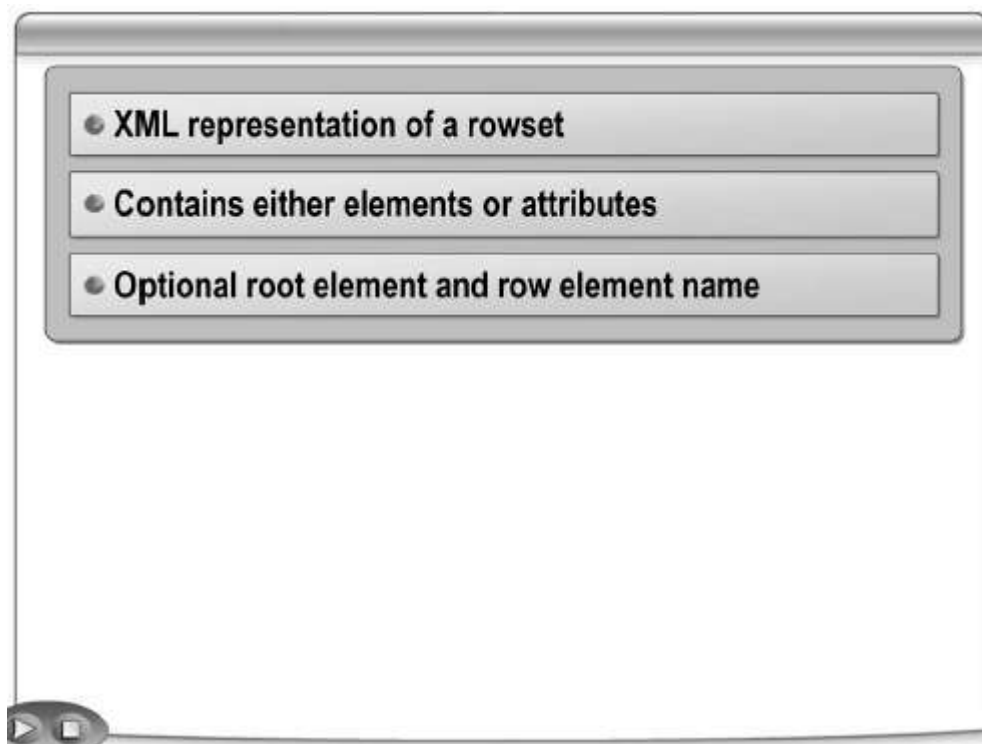
Примеры использования FOR XML

Существует множество ситуаций, в которых необходимо получить данные в формате XML вместо набора строк. Например, как описано в следующих сценариях доступа к данным:

- **Получение данных для публикации на Web-сайте.** Получая данные в формате XML, Вы можете применить стилевые таблицы XSLT для передачи данных в HTML-формате. Вы можете также применить различные стилевые таблицы к тем же самым данным XML, чтобы сгенерировать альтернативные форматы представления, чтобы поддержать различные устройства клиента, не переписывая логики доступа к данным.
- **Получение данных для обмена с торговым партнером.** XML - естественный формат данных, которые Вы хотите послать торговому партнеру. Получая коммерческую информацию в XML формате, Вы можете легко объединить свои системы с внешними организациями, и не имеет значения, какие технологии данных они используют внутренне.

Дополнительная информация. Дополнительную информацию об XML вообще см. в разделе “Understanding XML” на Web-сайте MSDN®. Дополнительную информацию о фразе FOR XML см. в разделе “Constructing XML Using FOR XML” в SQL Server Books Online.

Что такое запросы в режиме RAW?



Введение

Вы используете запросы в режиме RAW, чтобы получить XML представление набора строк. Приложения могут обработать XML в естественном формате или применить стилевые таблицы XSLT, чтобы преобразовать XML в необходимый деловой формат документа или представление в интерфейсе пользователя.

Рассмотрите следующие особенности режима RAW:

- элемент представляет каждую строку в результирующем наборе, который возвращает запрос.
- атрибут с именем столбца или псевдонимом, используемым в запросе, представляет каждый столбец в результирующем наборе, если не указана опция ELEMENTS, когда каждый столбец соответствует подэлементу элемента строки.
- запросы в режиме RAW могут включать агрегированные столбцы и фразу GROUP BY.

Получение данных в сгенерированных элементах-строках

В следующем примере показывается, как Вы можете получить XML-фрагмент, содержащий данные заказа при использовании фразы FOR XML в режиме RAW.

```
SELECT Cust.CustomerID CustID, CustomerType, SalesOrderID
FROM Sales.Customer Cust JOIN Sales.SalesOrderHeader [Order]
    ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML RAW
```

Этот запрос выводит XML-фрагмент в формате, который содержит сгенерированные элементы <raw>, показанные в следующем примере.

```
<row CustID="1" CustomerType="S" SalesOrderID="43860"/>
<row CustID="1" CustomerType="S" SalesOrderID="44501"/>
```

Заметьте, что столбец **CustomerID** использует псевдоним **CustID**, который определяет имя атрибута.

Получение данных в виде элементов

В следующем примере показывается, как можно получить те же данные в виде элементов вместо атрибутов, определяя опцию ELEMENTS.

```
SELECT Cust.CustomerID CustID, CustomerType, SalesOrderID
FROM Sales.Customer Cust JOIN Sales.SalesOrderHeader [Order]
    ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML RAW, ELEMENTS
```

Этот запрос выводит XML-фрагмент в формате, показанном в следующем примере.

```

<row>
  <CustID>1</CustID>
  <CustomerType>S</CustomerType>
  <SalesOrderID>43860</SalesOrderID>
</row>
<row>
  <CustID>1</CustID>
  <CustomerType>S</CustomerType>
  <SalesOrderID>44501</SalesOrderID>
</row>

```

Получение данных при использовании корневого элемента и настроенного имени элемента строки

В следующем примере показывается, как Вы можете получить те же самые данные, используя корневой элемент, определенный с помощью опции ROOT и изменяя имя элемента строки при помощи необязательного аргумента опции RAW.

```

SELECT Cust.CustomerID CustID, CustomerType, SalesOrderID
FROM Sales.Customer Cust JOIN Sales.SalesOrderHeader [Order]
  ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML RAW('Order'), ROOT('Orders')

```

Этот запрос предоставляет правильно построенный XML документ в формате, показанном в следующем примере.

<Orders>

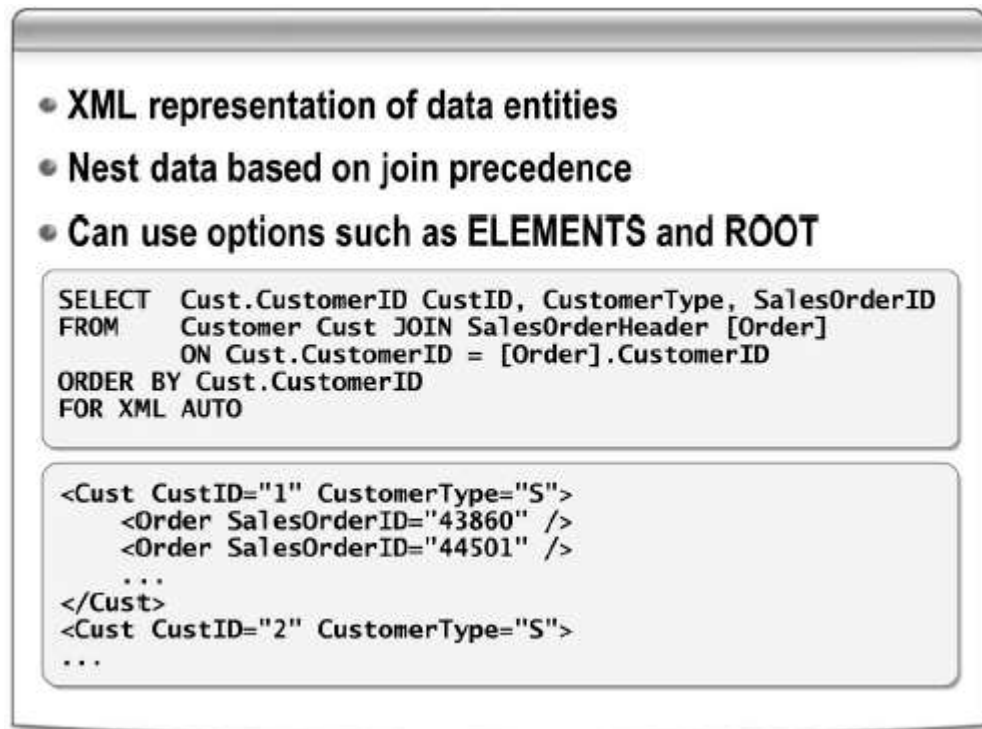
<Order CustID="1" CustomerType="S" SalesOrderID="43860"/>

<Order CustID="1" CustomerType="S" SalesOrderID="44501"/>

</Orders>

Заметьте, что, если бы была также определена опция ELEMENTS, то получающиеся строки содержали бы элементы вместо атрибутов.

Что такое запросы в режиме AUTO?



- XML representation of data entities
- Nest data based on join precedence
- Can use options such as **ELEMENTS** and **ROOT**

```
SELECT Cust.CustomerID CustID, CustomerType, SalesOrderID
FROM   Customer Cust JOIN SalesOrderHeader [Order]
      ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO
```

```
<Cust CustID="1" CustomerType="S">
  <Order SalesOrderID="43860" />
  <Order SalesOrderID="44501" />
  ...
</Cust>
<Cust CustID="2" CustomerType="S">
  ...
```

Введение

Запросы в режиме AUTO производят XML представления данных о сущностях. Опция AUTO имеет следующие особенности:

- Каждая строка, возвращаемая в соответствии с запросом, представляется элементом XML с тем же самым именем как у таблицы, из которой она была получена (или с псевдонимом, используемым в запросе).
- Каждое соединение JOIN в запросе преобразуется во вложенный элемент XML, уменьшая дублирование данных в получающемся фрагменте XML. Порядок операторов JOIN влияет на порядок вложенных элементов.
- Чтобы гарантировать, что дочерние элементы сопоставлены правильно с их родителем, используйте фразу ORDER BY, чтобы вернуть данные в правильном иерархическом порядке.
- Каждый столбец в результирующем наборе представлен атрибутом, если не определена опция ELEMENTS, когда каждый столбец представлен дочерним элементом.

- Агрегированные столбцы и фраза GROUP BY не поддерживаются в запросах в режиме AUTO (хотя можно использовать запросы в режиме AUTO, чтобы получить агрегированные данные из представления, которое использует фразу GROUP BY).

Получение вложенных данных при использовании режима AUTO

В следующем примере показывается, как можно использовать запросы в режиме AUTO, чтобы вернуть XML фрагмент, содержащий список заказов.

```
SELECT Cust.CustomerID CustID, CustomerType, SalesOrderID
FROM Sales.Customer Cust JOIN Sales.SalesOrderHeader [Order]
      ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO
```

Этот запрос производит фрагмент XML в формате, показанном в следующем примере.

```
<Cust CustID="1" CustomerType="S">
  <Order SalesOrderID="43860"/>
  <Order SalesOrderID="44501"/>
</Cust>
```

Заметьте, что столбец **CustomerID** и таблицы **Customer** и **SalesOrderHeader** используют псевдонимы для определения имен атрибута и элементов.

Получение данных в виде элементов

Следующий пример показывает, как Вы можете получить те же самые данные в виде элементов, а не в виде атрибутов, определяя опцию ELEMENTS.

```

SELECT Cust.CustomerID CustID, CustomerType, SalesOrderID
FROM Sales.Customer Cust JOIN Sales.SalesOrderHeader [Order]
ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO, ELEMENTS

```

Этот запрос производит фрагмент XML в формате, показанном в следующем примере.

```

<Cust>
  <CustID>1</CustID>
  <CustomerType>S</CustomerType>
  <Order>
    <SalesOrderID>43860</SalesOrderID>
  </Order>
  <Order>
    <SalesOrderID>44501</SalesOrderID>
  </Order>
</Cust>
...

```

Так же, как и в режиме RAW, Вы можете использовать опцию ROOT, как показано в следующем примере.

```

SELECT Cust.CustomerID CustID, CustomerType, SalesOrderID
FROM Sales.Customer Cust JOIN Sales.SalesOrderHeader [Order]
ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO, ELEMENTS, ROOT('Orders')

```

Этот запрос представляет документ XML в формате, показанном в следующем примере.

```
<Orders>
  <Cust>
    <CustID>1</CustID>
    <CustomerType>S</CustomerType>
    <Order>
      <SalesOrderID>43860</SalesOrderID>
    </Order>
    <Order>
      <SalesOrderID>44501</SalesOrderID>
    </Order>
  </Cust>
  ...
</Orders>
```

Что такое запросы в режиме EXPLICIT?

The screenshot shows a SQL Server Enterprise Manager window with a query editor and a results pane. The query editor contains a SELECT statement for XML EXPLICIT mode. The results pane shows a tabular representation of the XML output, with columns for Tag, Parent, Invoice!1!InvoiceNo, and Invoice!1!Date!Element. Below the table, the XML output is displayed, showing the structure of the XML document.

• **Tabular representations of XML documents**

Tag	Parent	Invoice!1!InvoiceNo	Invoice!1!Date!Element
1	NULL	43659	2001-07-01T00:00:00
1	NULL	43660	2001-07-02T00:00:00

• **Allow complete control of XML format**

```
SELECT 1 AS Tag, NULL AS Parent,
       SalesOrderID AS [Invoice!1!InvoiceNo],
       OrderDate AS [Invoice!1!Date!Element]
FROM    SalesOrderHeader
FOR XML EXPLICIT
```

```
<Invoice InvoiceNo="43659">
  <Date>2001-07-01T00:00:00</Date>
</Invoice>
<Invoice InvoiceNo="43660">...
```

Введение

Иногда деловые документы, которые Вы должны обменяться с торговыми партнерами, требуют формат XML, который не может быть получен при использовании режимов RAW или AUTO. Когда данные таблицы соответствуют элементу XML, таблица может быть представлена как

- значение элемента.
- атрибут.
- дочерний элемент.

Универсальные таблицы

Ключом к пониманию формирования настраиваемых XML документов является понятие универсальной таблицы. Универсальная таблица - табличное представление XML документа. Каждая строка в универсальной таблице представляет данные, которые будут представлены в виде элемента в получающемся документе XML.

Первые два столбца универсальной таблицы определяют в результирующем XML документе иерархическое положение элемента, который содержит данные строки. Эти столбцы:

Tag (Тег). Числовое значение, которое уникально идентифицирует тег для элемента, который содержит данные в этой строке.

Parent (Родитель). Числовое значение, которое идентифицирует непосредственный родительский тег для этого элемента.

Каждый различный XML тег в получающемся документе, который соответствует таблице или представлению в базе данных должен быть представлен различным значением **Tag** в универсальной таблице. Значение **Parent** определяет иерархическое положение тега в получающемся документе. Теги верхнего уровня фрагмента XML (и поэтому не имеющие непосредственного родительского элемента) имеют значение Parent, равное NULL. Например, документ XML, описанный ранее, содержит элемент **Invoice (счет)** без родителя и элемент **LineItem**, который является дочерним элементом элемента Invoice. В универсальной таблице теги для этих двух элементов назначены в таблицах **Tag**, чтобы идентифицировать их, а столбец **Parent** используется, чтобы определить, как вложены элементы.

Определение столбцов универсальной таблицы

Остальные столбцы универсальной таблицы содержат данные, которые будут представлены в документе. Название столбца определяет, будут ли данные представлены как значение элемента, атрибут или дочерний элемент. Столбцы данных в универсальной таблице имеют название с четырьмя частями в следующем формате.

ElementName! TagNumber! AttributeName! Directive

Следующая таблица описывает части названия столбца.

Часть имени	Описание
<i>ElementName</i>	название элемента, который содержит данные в этой строке.
<i>TagNumber</i>	уникальное число, которое идентифицирует признак (как определено в столбце Tag). Тот же самый <i>ElementName</i> должен использоваться последовательно с данным <i>TagNumber</i> .
<i>AttributeName</i>	(Необязательная часть) название атрибута или дочернего элемента, который представляет данные в этом столбце. Если этот столбец не указан, то данные представляются как значение элемента.
<i>Directive</i>	(Необязательная часть) Дополнительные инструкции форматирования, которые представляют данные как дочерний элемент или другой определенный формат XML.

Например, следующая универсальная таблица определяет информацию о счетах в XML формате.

Tag	Parent	Invoice!1!InvoiceNo!	1!Data!Element
1	NULL	43659	2001-07-01T00:00:00
1	NULL	43660	2001-07-01T00:00:00

XML, представленный этой таблицей, может выглядеть как в следующем примере.

```
<Invoice InvoiceNo="43659">
    <Date>2001-07-01T00:00:00</Date>
</Invoice>
<Invoice InvoiceNo="43660">
    <Date>2001-07-01T00:00:00</Date>
</Invoice>
```

Создание запроса для построения универсальной таблицы

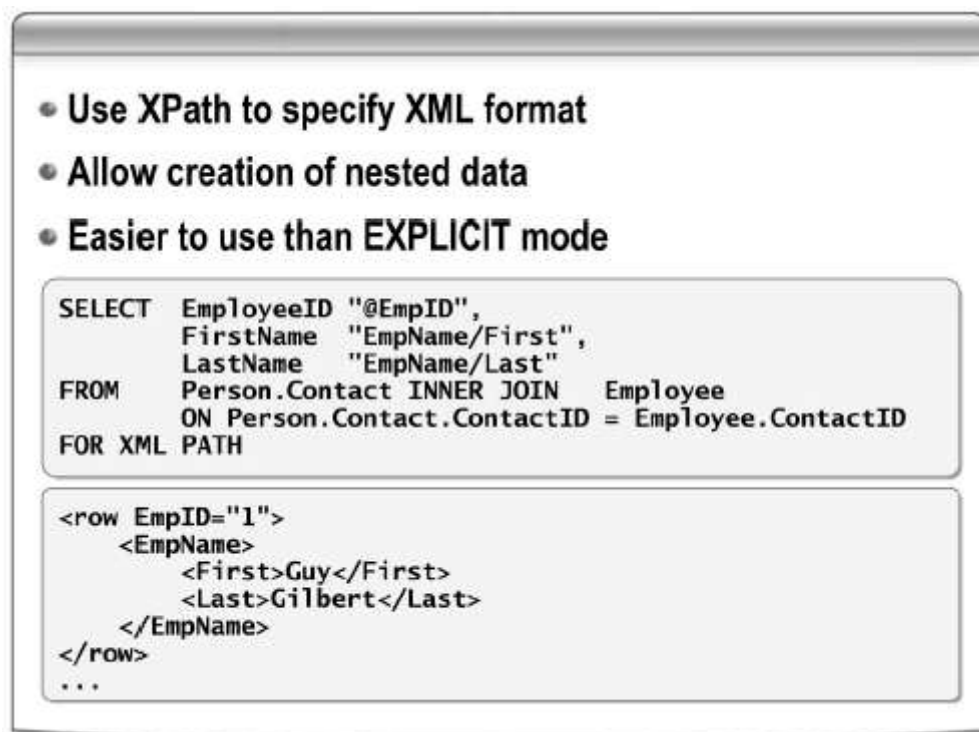
После того, как Вы определили универсальную таблицу, необходимую для получения желаемого XML-документа, Вы можете построить запрос Transact-SQL, необходимый, чтобы сгенерировать таблицу с использованием псевдонимов для именования столбцов. Вы можете назначить значения для столбцов **Tag** и **Parent** явно, как показано в следующем примере.

```
SELECT 1 AS Tag,
       NULL AS Parent,
       Sales.SalesOrderID AS [Invoice!1!InvoiceNo],
       OrderDate AS [Invoice!1!Date!Element]
FROM SalesOrderHeader
FOR XML EXPLICIT
```

Этот пример создает XML, показанный ранее в этой теме, комбинируя атрибуты и элементы для столбцов **SalesOrderID** для **OrderDate**.

Дополнительная информация. Дополнительную информацию о режиме EXPLICIT универсальных таблицах см. “Using EXPLICIT Mode” в SQL Server Books Online.

Что такое запросы в режиме PATH?



Введение

Режим PATH позволяет получить настроенный XML-документ при использовании синтаксиса XPath, чтобы значения соответствовали атрибутам, элементам, подэлементам, текстовым узлам и значениям данных. Эта способность соотнести столбцы таблиц с узлами XML-документа позволяют получить сложный XML без сложности запроса в режиме EXPLICIT.

Рассмотрите следующие особенности синтаксиса XPath:

- XML узлы в дереве выражаются как пути, отделенные символами /.
- Атрибуты обозначаются символами @ в начале имени атрибута.
- Для обозначения относительных путей можно использовать одну точку (.), чтобы представить текущий узел и две точки (..), чтобы представить родителя текущего узла.

Получение данных при использовании режима PATH

Следующий пример показывает, как Вы можете использовать запрос в режиме PATH, чтобы вернуть XML фрагмент, содержащий список служащих.

```

SELECT EmployeeID "@EmpID",
       FirstName "EmpName/First",
       LastName "EmpName/Last"
FROM Person.Contact INNER JOIN
       Employee ON Person.Contact.ContactID = Employee.ContactID
FOR XML PATH

```

Этот запрос позволяет получить фрагмент XML в формате, показанном в следующем примере.

```

<row EmpID="1">
  <EmpName>
    <First>Guy</First>
    <Last>Gilbert</Last>
  </EmpName>
</row>
<row EmpID="2">
  <EmpName>
    <First>Kevin</First>
    <Last>Browne</Last>
  </EmpName>
</row>

```

Заметьте, что столбец **EmployeeID** соответствует атрибуту **EmpID** со знаком @. Столбцы **FirstName** и **LastName** соответствуют подэлементам элемента **EmpName**.

Изменение имени элемента row

Следующий пример показывает, как Вы можете использовать дополнительный аргумент *ElementName* режима PATH, чтобы изменить название элемента строки, принятого по умолчанию.

```

SELECT EmployeeID "@EmpID",
       FirstName "EmpName/First",
       LastName "EmpName/Last"
FROM Person.Contact INNER JOIN
       Employee ON Person.Contact.ContactID = Employee.ContactID
FOR XML PATH('Employee')

```

Этот запрос позволяет получить фрагмент XML в формате, показанном в следующем примере.

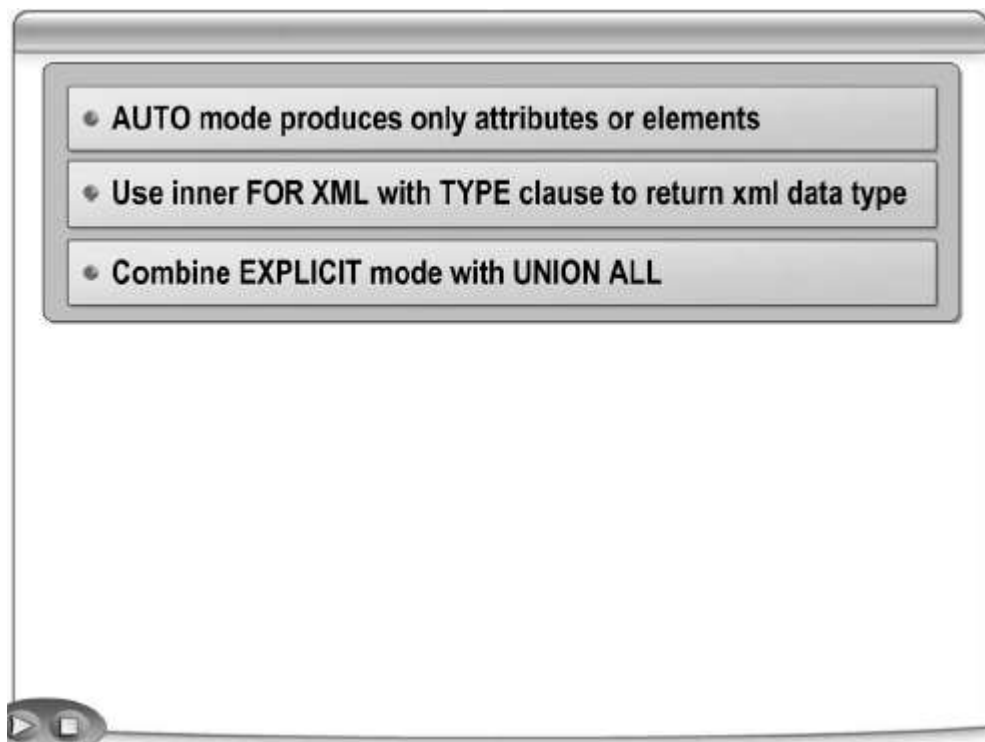
```

<Employee EmpID="1">
  <EmpName>
    <First>Guy</First>
    <Last>Gilbert</Last>
  </EmpName>
</Employee>
<Employee EmpID="2">
  <EmpName>
    <First>Kevin</First>
    <Last>Browne</Last>
  </EmpName>
</Employee>

```

Заметьте, что столбец **EmployeeID** соответствует атрибуту **EmpID** (**@EmpID**), а столбцы **FirstName** и **LastName** соответствуют подэлементам элемента **EmpName** (**EmpName/First** и **EmpName/Last**).

Синтаксис для получения вложенных XML-документов



Введение

Вложение XML позволяет Вам представить отношение родитель/потомок как иерархию XML – например, клиенты и их заказы, или заказы и их позиции. Есть несколько способов вложения элементов XML при использовании фразы FOR XML, включая:

- Определение соединений в запросах режима AUTO
- Определение опции TYPE в подзапросах, чтобы получить значения типа xml
- Объединение множества универсальных таблиц при использовании UNION ALL в запросах в режиме EXPLICIT

Использование режима AUTO для получения вложенных XML

Когда Вы определяете JOIN в запросе в режиме AUTO, SQL Server вкладывает получающиеся элементы, которые соответствуют таблицам в том же порядке, в котором они появляются во фразе SELECT.

Режим AUTO позволяет определить, соответствуют ли столбцы элементам или атрибутам во XML фрагменте при использовании опции ELEMENTS. Однако, Вы можете определить

только одну опцию для всего фрагмента, обеспечивая ограниченный контроль над форматом вывода.

В следующем примере все столбцы выведены как атрибуты, потому что не определена опция ELEMENTS.

```
SELECT Cust.CustomerID CustID, CustomerType, SalesOrderID
FROM Sales.Customer Cust JOIN Sales.SalesOrderHeader [Order]
      ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO
```

Этот запрос позволяет получить фрагмент XML в формате, показанном в следующем примере.

```
<Cust CustID="1" CustomerType="S">
  <Order SalesOrderID="43860"/>
  <Order SalesOrderID="44501"/>
</Cust>
```

Добавление опции ELEMENTS к предыдущему оператору Transact-SQL позволило бы вывести следующий документ.

```
<Cust>
  <CustID>1</CustID>
  <CustomerType>S</CustomerType>
  <Order>
    <SalesOrderID>43860</SalesOrderID>
  </Order>
  <Order>
    <SalesOrderID>44501</SalesOrderID>
  </Order>
```


</Cust>

...

Использование TYPE для получения типа данных xml в подзапросе

SQL Server 2005 включает тип данных **xml**. Определение директивы TYPE в запросе FOR XML возвращает результаты как **xml** значение вместо строки типа **varchar**. Наиболее существенное воздействие этого - способность вложения запросов FOR XML, чтобы получить многоуровневый XML-документ в запросах в режимах AUTO и RAW. Следующий пример показывает, как использовать директиву TYPE, чтобы вкладывать запросы FOR XML.

```
SELECT Name CategoryName,  
       (SELECT Name SubCategoryName  
        FROM Production.ProductSubCategory SubCategory  
        WHERE SubCategory.ProductCategoryID=Category.ProductCategoryID  
        FOR XML AUTO, TYPE, ELEMENTS)  
FROM Production.ProductCategory Category  
FOR XML AUTO
```

Результат предыдущего запроса показывается в этом примере.

```
<Category CategoryName="Accessories">  
  <SubCategory>  
    <SubCategoryName>Bike Racks</SubCategoryName>  
  </SubCategory>  
  <SubCategory>  
    <SubCategoryName>Bike Stands</SubCategoryName>  
  </SubCategory>  
</Category>  
...
```

Иначе, если бы внешний запрос не содержал фразы FOR XML AUTO, в выводе появился бы столбец с подкатегориями, содержащий xml данные.

CategoryName	
Accessories	<pre><SubCategory> <SubCategoryName> Bike Racks </SubCategoryName> </SubCategory> <SubCategory> <SubCategoryName> Bike Stands </SubCategoryName> </SubCategory></pre>

Столбец с **xml** данными не имеет имени, потому что запрос не определяет псевдоним для внутреннего запроса SELECT.

Дополнительная информация. Дополнительную информацию об использовании директивы TYPE см. в разделе “TYPE Directive in FOR XML Queries” в SQL Server Books Online.

Вложение таблиц при использовании режима EXPLICIT

Чтобы использовать режим EXPLICIT для получения XML документа, содержащего множество тегов, Вы должны написать индивидуальные запросы для каждого тега и затем слить их при использовании оператора UNION ALL.

Использование оператора UNION ALL в запросах режима EXPLICIT имеют следующие особенности:

- Чтобы объединение успешно выполнилось, каждый запрос должен возвращать согласованное множество столбцов. Можно назначать Null-значения столбцам, не используемым в текущем запросе.
- Необходимо использовать фразу ORDER BY, чтобы представить результаты объединения в правильной XML-иерархии.

Например, XML документ для счета мог бы выводиться в следующем формате.

```

<Invoice InvoiceNo="43659">
    <Date>2001-07-01T00:00:00</Date>
    <LineItem ProductID="709">Bike Socks, M</LineItem>
    <LineItem ProductID="711">Helmet, Blue</LineItem>
</Invoice>

```

Поскольку этот формат содержит два элемента XML (**Invoice** и **LineItem**), которые соответствуют таблицам или представлениям, необходимо использовать два запроса, чтобы получить данные.

Первый запрос должен произвести элемент **Invoice**, который содержит атрибут **InvoiceNo** и дочерний элемент **Date**. Поскольку запрос будет объединен с другим запросом с использованием оператора UNION ALL, необходимо также определить столбцы (как null) для **ProductID** и **Name**, которые возвращаются в другом запросе.

Второй запрос должен получить подэлементы **LineItem** внутри элемента **Invoice**. Эти подэлементы содержат **ProductID** (как атрибут **LineItem**) и **Name** (как значение **LineItem**). Необходимо также получить **OrderID**, чтобы соединить пункты заказа с их заказами.

Следующий пример показывает, как нужно объединить два запроса при использовании оператора UNION, чтобы получить элементы **Invoice** и **LineItem**, чтобы получить предыдущий XML-документ.

```

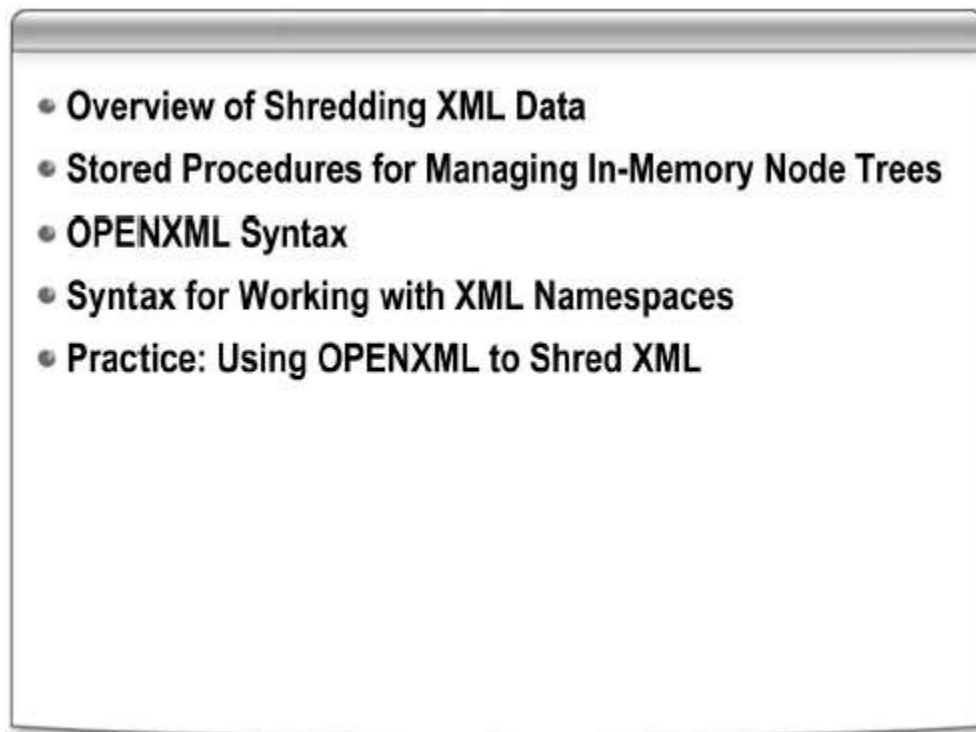
SELECT 1 AS Tag,
        NULL AS Parent,
        SalesOrderID AS [Invoice!1!InvoiceNo],
        OrderDate AS [Invoice!1!Date!Element],
        NULL AS [LineItem!2!ProductID],
        NULL AS [LineItem!2]
FROM Sales.SalesOrderHeader
UNION ALL
SELECT 2 AS Tag, 21
        1 AS Parent,
        OrderDetail.SalesOrderID,

```

```
        NULL,
        OrderDetail.ProductID,
        Product.Name
FROM Sales.SalesOrderDetail OrderDetail JOIN
        Sales.SalesOrderHeader OrderHeader
        ON OrderDetail.SalesOrderID= OrderHeader.SalesOrderID
JOIN Production.Product Product
        ON OrderDetail.ProductID = Product.ProductID
ORDER BY [Invoice!1!InvoiceNo], [LineItem!2!ProductID]
FOR XML EXPLICIT
```

[К содержанию](#)

Урок 2: Разбор документов XML с использованием синтаксиса OPENXML



Цели урока

После завершения этого урока, студенты смогут:

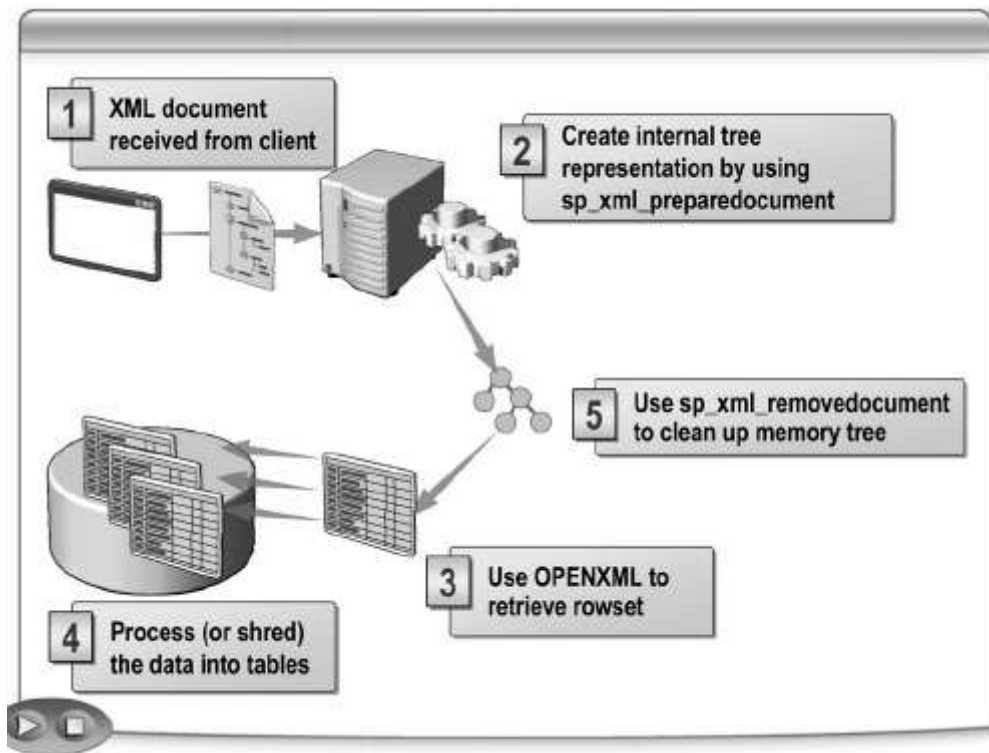
- Описать цель использования синтаксиса OPENXML.
- Описать хранимые процедуры для управления деревом узлов в памяти.
- Описать синтаксис OPENXML.
- Описать синтаксис для работы с пространством имен XML.

Введение

Набор строк содержит результат запроса в виде таблицы. В сценарии для обмена данными с торговым партнером, Вы, возможно, должны будете сгенерировать набор строк из документа XML. Например, розничный продавец мог бы послать заказы поставщику как документы XML. Поставщик должен тогда сгенерировать набор строк из XML-документа, чтобы вставить данные в одну или более таблиц в базе данных. Процесс преобразования данных XML в набор строк известен как "разбор" данных XML.

В этом уроке Вы узнаете, как разобрать XML-документ с использованием функции `OPENXML` и соответствующих хранимых процедур.

Краткий обзор разбора данных XML



Процесс разбора данных XML

Преобразование данных XML в набор строк включает следующие пять шагов:

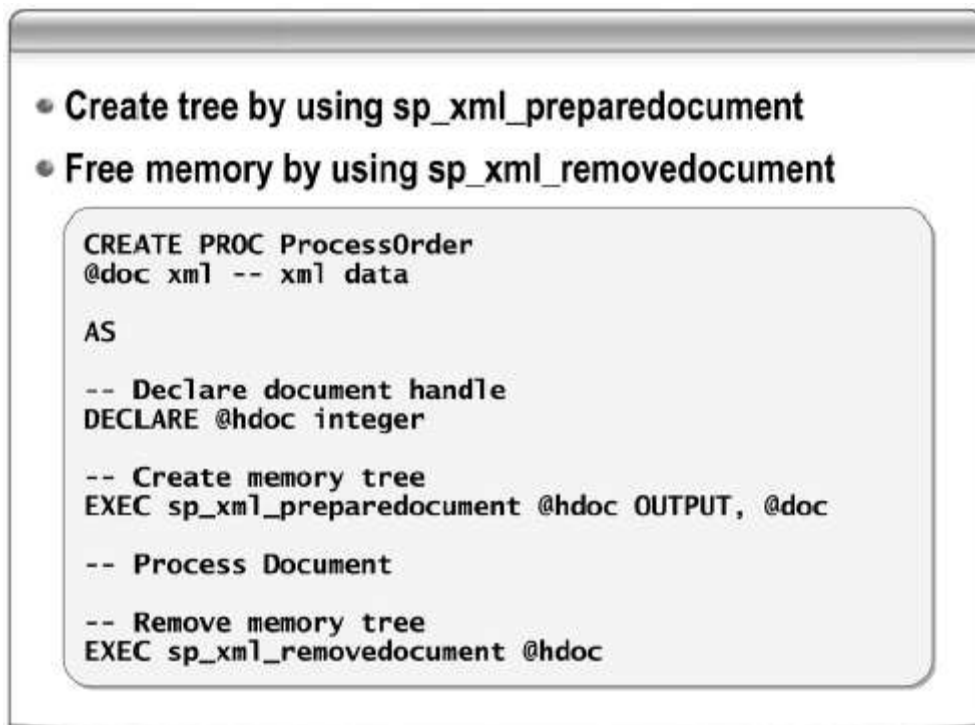
- 1. Получение XML-документа.** Когда приложение получает документ XML, возможна обработка документа при использовании кода Transact-SQL. Например, когда поставщик получает заказ XML от розничного продавца, поставщик регистрирует заказ в базе данных SQL Server. Чтобы обработать данные XML, обычно выполняется код Transact-SQL в форме хранимой процедуры, где строка XML передается как параметр.
- 2. Генерация внутреннего представления дерева.** Прежде, чем обработать документ, необходимо использовать хранимую процедуру `sp_xml_preparedocument`, чтобы разобрать документ XML и преобразовать его в памяти в древовидную структуру. Дерево концептуально подобно представлению документа XML согласно Data Object Document (DOM). Вы можете использовать только правильно построенный документ XML, чтобы сгенерировать внутреннее дерево.
- 3. Формирование набора строк из дерева.** Чтобы сгенерировать в памяти набор строк из данных в дереве, необходимо использовать функцию OPENXML.

Используйте синтаксис запросов XPath, чтобы определить узлы дерева, которые будут возвращаться в наборе строк.

4. Обработка данных набора строк. Используйте набор строк, созданный OPENXML для обработки данных любыми средствами, применяемыми для наборов строк. Вы можете выбирать, обновлять или удалять данные, используя команды Transact-SQL. Обычно OPENXML используется для вставки данных в постоянные таблицы базы данных. Например, заказ XML, полученный поставщиком, может содержать данные, которые должны быть вставлены в таблицы **SalesOrderHeader** и **SalesOrderDetail**.

5. Удаление внутреннего дерева, если оно больше не требуется. Поскольку древовидная структура хранится в памяти, используйте хранимую процедуру **sp_xml_removedocument** для освобождения памяти, когда дерево больше не требуется.

Хранимые процедуры для управления узлами деревьев в памяти



Введение

Прежде, чем Вы сможете обработать XML документ с использованием команды Transact-SQL, необходимо разобрать документ и преобразовать его в древовидную структуру в памяти.

Создание дерева с использованием `sp_xml_preparedocument`

Процедура `sp_xml_preparedocument` сохраняет разборы XML документ и производит внутреннее представление документа в виде дерева. Следующая таблица описывает параметры системной хранимой процедуры `sp_xml_preparedocument`.

Параметр	Описание
xmltext	Исходный XML документ, который нужно обработать. Этот параметр может принимать любой из следующих текстовых типов данных: nchar , varchar , nvarchar , текст, ntext , или xml .

hdoc	Дескриптор разобранного дерева XML.
xpath_namespaces	(Необязательный) пространство имен объявлений XML, которые используются в строке и столбце выражения XPath в операторах OPENXML.

Следующий пример показывает, как использовать системную хранимую процедуру **sp_xml_preparedocument**, для анализа XML-документа, переданного другой пользовательской хранимой процедуре.

```
CREATE PROC ProcessOrder @doc xml
AS
DECLARE @hdoc integer
EXEC sp_xml_preparedocument @hdoc OUTPUT, @doc
```

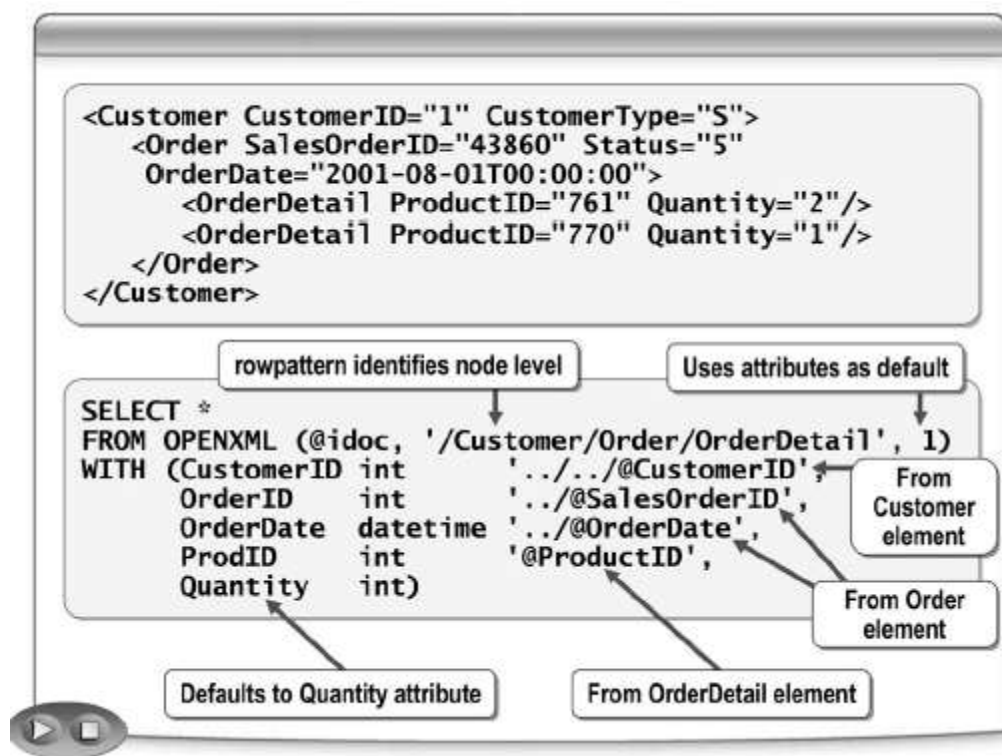
Удаление дерева с использованием **sp_xml_removedocument**

SQL Server хранит разобранные документы во внутреннем кэше. Чтобы избежать переполнения памяти, используйте системную хранимую процедуру **sp_xml_removedocument**, чтобы освободить дескриптор документа и уничтожить древовидную структуру, когда она больше не нужна. Необходимо вызывать **sp_xml_removedocument** в том же самом пакете запросов, что и процедуру **sp_xml_preparedocument**, которая сгенерировала дерево. Это необходимо потому, что параметр **hdoc**, используемый для указания на дерево, является локальной переменной, и если произойдет выход из области его видимости, то удалить это дерево из памяти станет невозможно.

Следующий пример показывает, как использовать системную хранимую процедуру **sp_xml_removedocument**:

```
EXEC sp_xml_removedocument @hdoc
```

Синтаксис OPENXML



Введение

После того, как Вы проанализировали XML документ при помощи хранимой процедуры **sp_xml_preparedocument** и получен дескриптор внутреннего дерева, можно сгенерировать набор строк из разобранного дерева.

Синтаксис OPENXML

Чтобы получить набор строк из дерева, нужно использовать функцию OPENXML. Затем можно уже писать команды Transact-SQL SELECT, UPDATE, или INSERT, которые изменяют базу данных. У функции OPENXML следующий синтаксис.

```
OpenXML(idoc, rowpattern [, flags])
[WITH (SchemaDeclaration | TableName)]
```

<SchemaDeclaration> ::=

ColumnName ColumnType [colpattern],n

Следующая таблица описывает параметры функции OPENXML.

Параметр	Описание
<i>rowpattern</i>	XPath запрос, определяющий узлы, которые должны быть возвращены.
<i>idoc</i>	Дескриптор внутреннего представления дерева XML документа
<i>flags</i>	Необязательная маска бита <i>флагов</i> , которая определяет модель сопоставления: атрибутивную или элементную. <i>flags</i> принимает следующие значение: <ul style="list-style-type: none">■ 0 – использовать сопоставление по умолчанию (т.е. атрибутивную).■ 1 — получить значение атрибута (атрибутивная модель).■ 2 — получить значение элемента (элементная модель).■ 8 — получить значения и атрибута и элемента.
<i>SchemaDeclaration</i>	Объявление схемы набора строк для столбцов, которые будут возвращены, при помощи комбинации названий столбцов, типов данных и шаблонов.
<i>TableName</i>	Название существующей таблицы, схема которой должна использоваться для определения возвращаемых столбцов.

Использование функции OPENXML

Использовать функцию OPENXML можно везде, где используется поставщик набора строк, такой как таблица, представление, или функция OPENROWSET. OPENXML прежде всего используется в операторе SELECT, как показано в следующем примере.

```
SELECT *
```

```
FROM OPENXML (@idoc, '/Customer/Order/OrderDetail')
```

```
WITH (ProductID int, Quantity int)
```

Аргумент *rowpattern* устанавливает уровень узла для элемента **OrderDetail** и получает значения атрибутов, потому что не определено значение флага.

В предыдущем примере **@idoc** – дескриптор внутреннего представления дерева следующего XML документа заказов.

```
<Customer CustomerID="1" CustomerType="S">
  <Order SalesOrderID="43860" Status="5"
    OrderDate="2001-08-01T00:00:00">
    <OrderDetail ProductID="761" Quantity="2"/>
    <OrderDetail ProductID="770" Quantity="1"/>
  </Order>
</Customer>
```

Следующая таблица показывает набор строк, возвращенный предыдущим оператором OPENXML.

ProductID	Quantity
761	2
770	1

Использование объявления схемы

В ситуациях, когда Вы должны восстановить данные по всей иерархии, Вы можете использовать шаблон столбцов, содержащий XPath-шаблон. XPath-шаблон в параметре *colpattern* соотносится с XPath-шаблоном, определенным в параметре *rowpattern*. Вы можете использовать шаблоны столбцов, чтобы получить данные из элементов при использовании как атрибутивной, так и элементной моделей сопоставления.

Следующий пример показывает, как объявление схемы может указывать на узлы всюду по иерархии XML при использовании документа XML из предыдущего примера.

```
SELECT *
FROM OPENXML (@idoc, '/Customer/Order/OrderDetail', 1)
WITH (CustomerID int ' ../@CustomerID',
      OrderID int ' ../@SalesOrderID',
      OrderDate datetime ' ../@OrderDate',
```

ProdID int '@ProductID',
Quantity int)

В предыдущем примере значение *flags* установлено в 1 для того, чтобы признаки были восстановлены по умолчанию, если не определено иначе в параметре *colpattern*. Столбец **CustomerID** должен быть получен из элемента **Customer**, который расположен двумя уровнями выше текущего элемента **OrderDetail**. Это достигается при использовании комбинации *../* дважды для поднятия на два уровня. Столбец **Quantity** автоматически получается без указания шаблона столбца, потому что имя атрибута точно соответствует имени столбца, в отличие от **ProdID**.

Следующая таблица показывает набор строк, полученный предыдущим оператором OPENXML.

OrderID	CustomerID	OrderDate	ProdID	Quantity
43860	1	2001-08-01-00:00:00.000	761	2
43860	1	2001-08-01-00:00:00.000	770	1

Дополнительная информация Дополнительную информацию об использовании функции OPENXML см. в разделе “Quering XML Using OPENXML” в SQL Server Books Online.

Синтаксис для работы с пространством имен XML

- **sp_xml_preparedocument** accepts namespaces
- Use namespace prefix in all XPath expressions

```
<Customer xmlns="urn:AW_NS" xmlns:o="urn:AW_OrderNS"
  CustomerID="1" CustomerType="S">
  <o:Order SalesOrderID="43860" Status="5"
    OrderDate="2001-08-01T00:00:00">
    <o:OrderDetail ProductID="761" Quantity="2"/>
    <o:OrderDetail ProductID="770" Quantity="1"/>
  </o:Order>
</Customer>

EXEC sp_xml_preparedocument @idoc OUTPUT, @doc,
'<ROOT xmlns:rootNS="urn:AW_NS" xmlns:orderNS="urn:AW_OrderNS"/>'

SELECT * FROM OPENXML (@idoc,
'/rootNS:Customer/orderNS:Order/orderNS:OrderDetail')
WITH...
```

Введение

Много XML документов содержат пространства имен, чтобы помочь приложениям распознать элементы и избежать совпадений имен различных элементов. Хранимая процедура **sp_xml_preparedocument** поддерживает использование пространства имен при создании древовидной структуры для анализа.

Применение пространства имен к **sp_xml_preparedocument**

Хранимая процедура **sp_xml_preparedocument** принимает необязательный параметр, который определяет XML элемент с необходимыми объявлениями пространств имен, как показано в следующем примере, который включает два пространства имен.

```
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc,
'<ROOT xmlns:rootNS="urn:AW_NS" xmlns:orderNS="urn:AW_OrderNS" />'
```

Следующий пример показывает документ XML, который использует эти пространства имен.

```
<Customer xmlns="urn:AW_NS" xmlns:o="urn:AW_OrderNS"
  CustomerID="1" CustomerType="S">
  <o:Order SalesOrderID="43860" Status="5"
    OrderDate="2001-08-01T00:00:00">
    <o:OrderDetail ProductID="761" Quantity="2"/>
    <o:OrderDetail ProductID="770" Quantity="1"/>
  </o:Order>
</Customer>
```

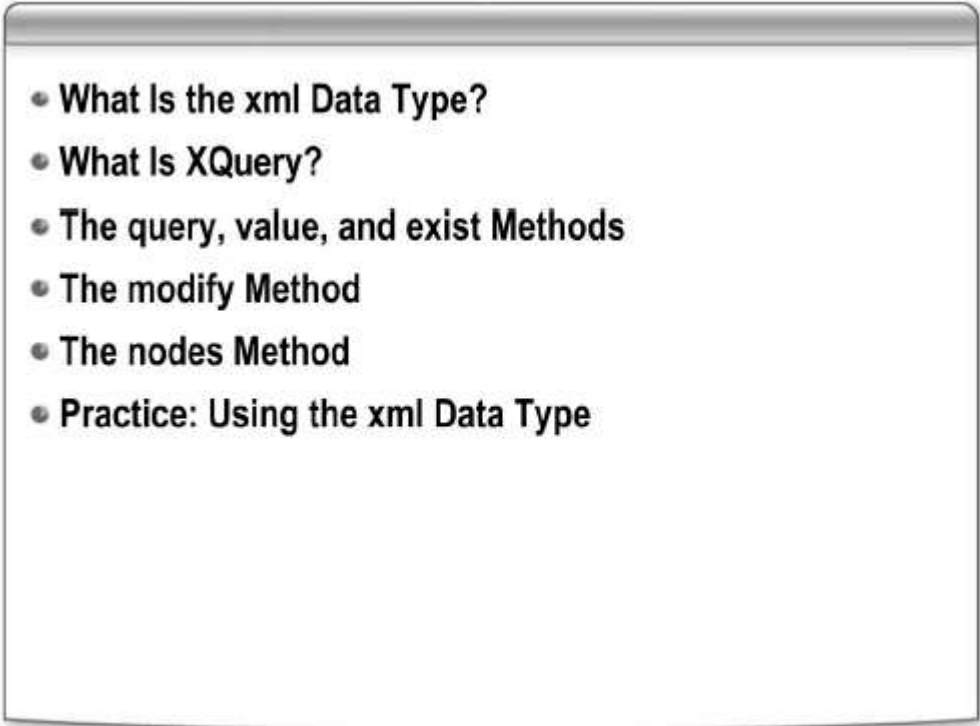
Использование пространств имен с XPath и OPENXML

Передавая пространства имен в хранимую процедуру **sp_xml_preparedocument**, Вы можете использовать информацию о пространстве имен в пределах выражения XPath, как показано в следующем примере.

```
SELECT *
FROM          OPENXML (@idoc,
                      '/rootNS:Customer/orderNS:Order/orderNS:OrderDetail')
WITH (OrderID      int           './@SalesOrderID',
      CustomerID    int           '../@CustomerID',
      OrderDate      datetime     './@OrderDate',
      ProdID         int          '@ProductID',
      Quantity       int)
```

Дополнительная информация. Дополнительную информацию о пространстве имен XML см. в разделе “Understanding XML Namespaces” на Вебсайте MSDN.

Урок 3: Использование типа данных xml

- 
- What Is the xml Data Type?
 - What Is XQuery?
 - The query, value, and exist Methods
 - The modify Method
 - The nodes Method
 - Practice: Using the xml Data Type

Цели урока

После завершения этого урока, студенты смогут:

- Определять тип данных **xml**.
- Определять запросы XQuery.
- Описывать методы **query**, **value**, и **exist** для запросов XML.
- Описывать метод **modify** для изменения XML.
- Описывать метод **nodes** для анализа XML.

Введение

SQL Server 2005 поддерживает тип данных **xml** для хранения XML документов и фрагментов в таблице, переменной или параметре. Тип данных **xml** имеет некоторые важные преимущества перед хранением данных в формате **text** или **varchar**, включая способность работать с Расширяемым Языком Запросов (XQuery) и модификацией или поиском определенных элементов или атрибутов внутри данных XML.

Из этого урока Вы узнаете о типе данных **xml** и о различных методах, связанных с ним. Вы также узнаете, как использовать XQuery, чтобы формировать запросы к данным XML.

Что такое тип данных xml?

- Native data type for XML
- Internal storage structure for XML InfoSet
- Use for tables, variables, or parameters
- Exposes methods to query and modify XML

```
-- usage within table definition
CREATE TABLE NewTable
( Col1 int primary key,
  Col2 xml )

-- usage as local variable
declare @data xml

-- usage as parameter to stored procedure
CREATE PROCEDURE SaveData(@doc xml) AS ...
```

Введение

Тип данных **xml** является родным типом данных для хранения XML документов или фрагментов в столбце, локальной переменной или параметре до 2 гигабайт. Возможность хранения XML непосредственно в реляционной базе данных имеет много преимуществ для разработчиков приложений.

Преимущества хранения XML в родном формате

Преимущества хранения XML включают:

- И структурированные, и полуструктурированные данные хранятся в одном месте, облегчая управление ими.
- Вы можете определить содержимое переменной в реляционной модели.
- Вы можете выбрать самую подходящую модель данных для обеспечения специфических требований Вашего приложения, используя высоко оптимизированное хранилище данных и окружение запросов.

Функциональные возможности XML

Тип данных SQL Server 2005 **xml** хранит информационный набор документа XML в эффективном внутреннем формате. Данные можно рассматривать как оригинальный документ XML за исключением того, что не хранятся незначимые пробелы, порядок атрибутов, префиксы пространств имен и объявления XML. SQL Server 2005 обеспечивает следующие функциональные возможности для типа данных **xml**:

Индексирование XML. Столбцы, определенные как **xml**, могут быть проиндексированы при использовании индексов XML и полнотекстовые индексы. Это может значительно увеличить производительность запросов, которые получают данные XML.

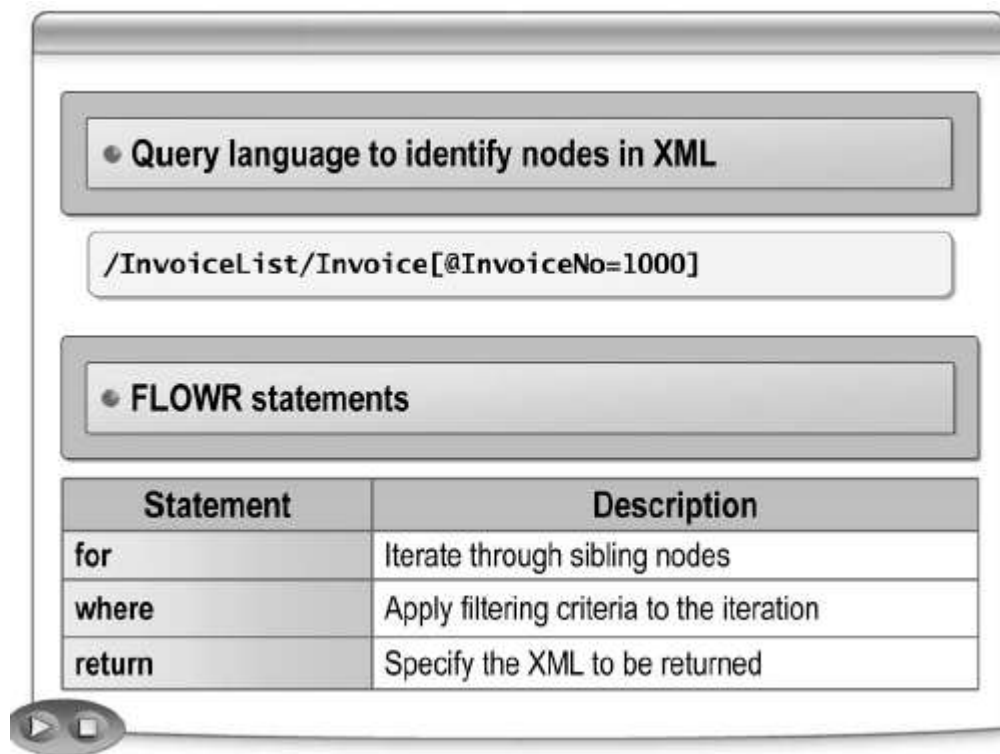
Методы получения данных на основе запросов XQuery. Тип данных **xml** поддерживает методы **query**, **value**, и **exist**. Они могут использоваться, чтобы извлечь данные из XML при использовании выражений XQuery.

Модификация данных на основе запросов XQuery. Тип данных **xml** поддерживает метод **modify**, который использует расширение спецификации XQuery, чтобы выполнить обновления данных XML.

Типизированный XML. Это XML-документ, который связан с некоторой схемой XML. Схема определяет элементы и атрибуты, которые разрешены в документе XML этого типа и определяет пространство имен для них. Когда используется тип данных **xml**, чтобы хранить типизированный XML, SQL Server проверяет соответствие XML-документа его схеме и оптимизирует внутреннее хранение данных, назначая соответствующие типы данных SQL Server тем данным, которые основаны на типах данных XML, определенных в схеме.

Дополнительная информация. Дополнительную информацию об XML InfoSet см. в “XML Information Set” спецификация W3C. Дополнительную информацию о типе данных **xml**, см. в “xml Data Type” в SQL Server Books Online.

Что такое XQuery?



Введение

XQuery используется, чтобы формировать запросы к данным XML. Синтаксис XQuery включает и расширяет выражения XPath 2.0 и позволяют выполнить сложные запросы к данным XML. Тип данных **xml** в SQL Server поддерживает методы, с помощью которых данные в **xml** могут быть получены или обновлены посредством выражения XQuery.

Поддержка XQuery в SQL Server 2005 основана на рабочем проекте W3C языковой спецификации XQuery 1.0 (доступного на Вебсайте W3C), и поэтому могут быть незначительные несоответствия с выпущенной спецификацией.

Синтаксис XQuery

Запрос XQuery состоит из двух главных секций: необязательная секция *Prolog*, в которой могут быть объявлены пространства имен и импортированы схемы; и секция *body (тело)*, в которой используются выражения XQuery, чтобы определить данные, которые будут получены. Выражением XQuery может быть простой путь, который описывает узлы XML, которые будут получены, или сложное выражение, которое генерирует результат в формате XML.

Путь XQuery основан на языке XPath и описывает местоположение узла в документе XML. Пути могут быть абсолютными (описание местоположения узла в дереве XML от корневого элемента) или относительными (описание местоположения узла относительно ранее идентифицированного узла). Примеры в следующей таблице показывают некоторые простые пути XQuery.

Примеры путей	Описание
/InvoiceList/Invoice	Все элементы Invoice непосредственно содержатся в корневом элементе InvoiceList
(/invoicelist/invoice) [2]	второй элемент Invoice в корневом элементе InvoiceList
(InvoiceList/Invoice/@InvoiceNo) [1]	Атрибут InvoiceNo первого элемента Invoice в корневом элементе InvoiceList
(InvoiceList/Invoice/Customer/text()) [1]	Текст первого элемента Customer в элементе Invoice в корневом элементе InvoiceList
/InvoiceList/Invoice [@InvoiceNo=1000]	Все элементы Invoice в элементе InvoiceList , которые имеют атрибут InvoiceNo с значением 1000

Операторы FLOWR

Языковая спецификация XQuery включает операторы **for**, **let**, **order by**, **where** и **return**, известные как операторы FLOWR (произносится "flower" - цветок). SQL Server 2005 поддерживает операторы **for**, **where**, и **return**, которые описаны в следующей таблице.

Оператор	Описание
for	Используется, чтобы повторить операцию через группу узлов на одном и том же уровне в XML документе.
where	Используется применять фильтры к узловым итерациям. XQuery включает функции, такие как COUNT , которые могут использоваться с WHERE.
return	Используется, чтобы определить, что XML возвратился изнутри повторения.

Следующий пример показывает выражение XQuery, которое включает ключевые слова **for**, **where**, и **return**.

```
for $i in /InvoiceList/Invoice
where count($i/Items/Item) > 1
return $i
```

Этот пример возвращает каждый элемент **Invoice**, который включает больше чем один элемент **Item** в его дочернем элементе **Items**.

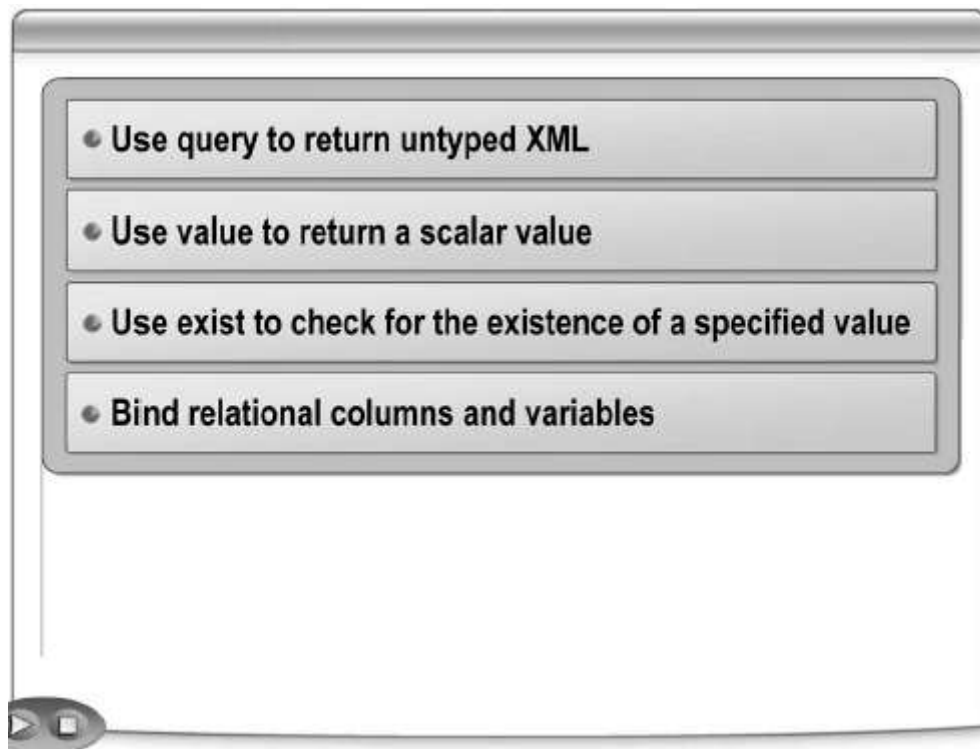
Работа с пространствами имен

Если запрашиваемый XML содержит пространство имен, то XQuery может включать объявление пространства имен в прологе запроса, используя следующий синтаксис.

```
xml.method('declare default element namespace "http://namespace";
method body')
```

Дополнительная информация Дополнительную информацию об использовании пространства имен с XQuery см. в разделе “XQuery Basics” в SQL Server Books Online.

Методы query, value, и exist



Введение

Тип данных SQL Server 2005 **xml** поддерживает четыре метода, которые могут использоваться, чтобы запрашивать или изменять данные XML. Эти методы вызываются при использовании синтаксиса *data_type.method_name*, знакомый большинству разработчиков. Понимание цели каждого из методов поможет создавать приложения, которые обрабатывают XML непосредственно в базе данных.

Метод query

Метод **query** используется, чтобы извлечь XML-данные из типа данных **xml**. XML-документ, полученный методом **query**, определяется по выражению XQuery, которое является параметром. Следующий пример показывает, как использовать метод query.

```
SELECT xmlCol.query('declare default element namespace
                    "http://schemas.adventure-works.com/InvoiceList";
                    <InvoiceNumbers>
```



```

    {
        for $i in /InvoiceList/Invoice
        return <InvoiceNo>
            {number($i/@InvoiceNo)}
        </InvoiceNo>
    }
</InvoiceNumbers>')

```

Метод value

Метод **value** используется, чтобы вернуть одно значение из документа XML. Чтобы использовать метод **value**, необходимо определить выражение XQuery, которое идентифицирует единственный узел в запрашиваемом XML-документе и тип данных Transact-SQL возвращаемого значения. Следующий пример показывает, как использовать метод **value**.

```

SELECT xmlCol.value('declare default element namespace
    "http://schemas.adventure-works.com/InvoiceList";
    /InvoiceList/Invoice/@InvoiceNo)[1]', 'int')

```

Метод exist

Метод **exist** используется для определения того, существует ли указанный узел в XML документе. Метод **exist** возвращает 1, если существует один или более экземпляров указанного узла в документе, и возвращает 0, если узел не существует. Следующий пример показывает, как использовать метод **exist**.

```

SELECT xmlCol.exist('declare default element namespace
    "http://schemas.adventure-works.com/InvoiceList";
    /InvoiceList/Invoice[@InvoiceNo=1000]')

```

Связывание реляционных столбцов и переменных

SQL Server 2005 поддерживает специфические для Microsoft расширения языка XQuery, позволяющие ссылаться на реляционные столбцы или переменные. Это называется *связыванием* реляционного столбца или переменной. Когда оператор SELECT, получающий данные из таблицы, включает **xml** метод для получения XML-документа из **xml** столбца, может использоваться функция **sql:column**, чтобы включать значения не **xml** столбца в XML-данные, как показано в следующем примере.

```
SELECT StoreName, Invoices.query('declare default element namespace
                                "http://schemas.adventure-works.com/InvoiceList";
                                <Invoices>
                                <Store>{sql:column("StoreName")}</Store>
                                {
                                    for $i in /InvoiceList/Invoice
                                    return $i
                                }
                                </Invoices>') InvoicesWithStoreName
FROM Stores
```

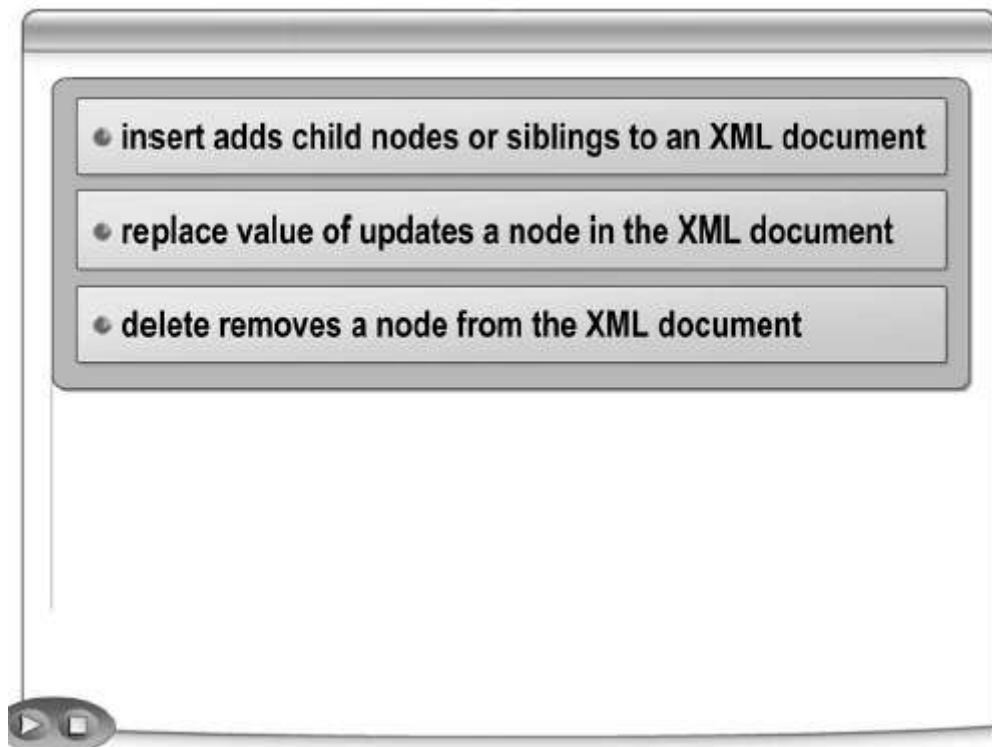
Точно так же может использоваться **sql:variable**, чтобы ссылаться на переменную в хранимой процедуре, как показано в следующем примере.

```
CREATE PROCEDURE GetInvoice(@store int, @invoiceNo int)
AS
SELECT Invoices.query('declare default element namespace
                        "http://schemas.adventure-works.com/InvoiceList";
                        <Invoices>
                        {
                            for $i in /InvoiceList/Invoice
                            where $i/@InvoiceNo = sql:variable("@invoiceNo")
```

```
        return $i
    }
    </Invoices>')
FROM #Stores
WHERE StoreID=@store
```

Дополнительная информация. Дополнительную информацию о методах типа данных **xml** см. в разделе “xml Data Type Methods” в SQL Server Books Online.

Метод modify



Введение

Вы можете использовать метод **modify**, чтобы обновить данные XML в типе данных **xml**. Метод **modify** использует три расширения для языковой спецификации XQuery: **insert**, **replace**, и **delete**. Эти расширения упоминаются как XML DML.

Оператор insert

Вы можете использовать оператор **insert**, чтобы добавить узлы к XML в столбце или переменной **xml**. У оператора **insert** следующий синтаксис.

```
insert Expression1 (  
    { as first | as last } into | after | before  
    Expression2 )
```

Параметры синтаксиса для ключевого слова **insert** описаны в следующей таблице.

Параметр	Описание
<i>Expression1</i>	узел, который будет вставлен. Это может быть литерал XML-например, <Item Product="5" Quantity="1"/> Это может также быть выражение element , чтобы вставить текстовый узел — например, element SalesPerson { "Alice" } Наконец, это может быть выражение attribute , чтобы вставить атрибут – например, attribute discount{"1.50"} . .
as first	Используется, чтобы вставить новый XML как первый элемент иерархии.
as last	Используется, чтобы вставить новый XML как последний элемент в иерархии.
into	Используется, чтобы вставить <i>Expression1</i> в <i>Expression2</i> .
after	Используется, чтобы вставить <i>Expression1</i> после <i>Expression2</i> .
before	Используется, чтобы вставить <i>Expression1</i> перед <i>Expression2</i> .
<i>Expression2</i>	Выражение XQuery, которое идентифицирует существующий узел в документе.

Следующий пример показывает, как использовать оператор **insert** в методе XQuery **modify**.

```
SET @xmlDoc.modify('declare default element namespace
                    "http://schemas.adventure-works.com/InvoiceList";
                    insert element salesperson { "Alice" } as first
                    into (/InvoiceList/Invoice)[1]')
```

Оператор **replace**

Вы можете использовать оператор **replace**, чтобы обновить значение **xml**. Оператор **replace** имеет следующий синтаксис.

```
replace value of Expression1 with Expression2
```

Параметры синтаксиса **replace** описаны в следующей таблице.

Параметр	Описание
<i>Expression1</i>	выражение XQuery, идентифицирующее узел, содержащий значение для замены
<i>Expression2</i>	новое значение узла.

Следующий пример показывает, как использовать оператор **replace** в методе **modify**.

```
SET xmlCol.modify('declare default element namespace
                  "http://schemas.adventure-works.com/InvoiceList";
                  replace value of
                      (/InvoiceList/Invoice/SalesPerson/text())[1]
                  with "Holly"')
```

Оператор delete

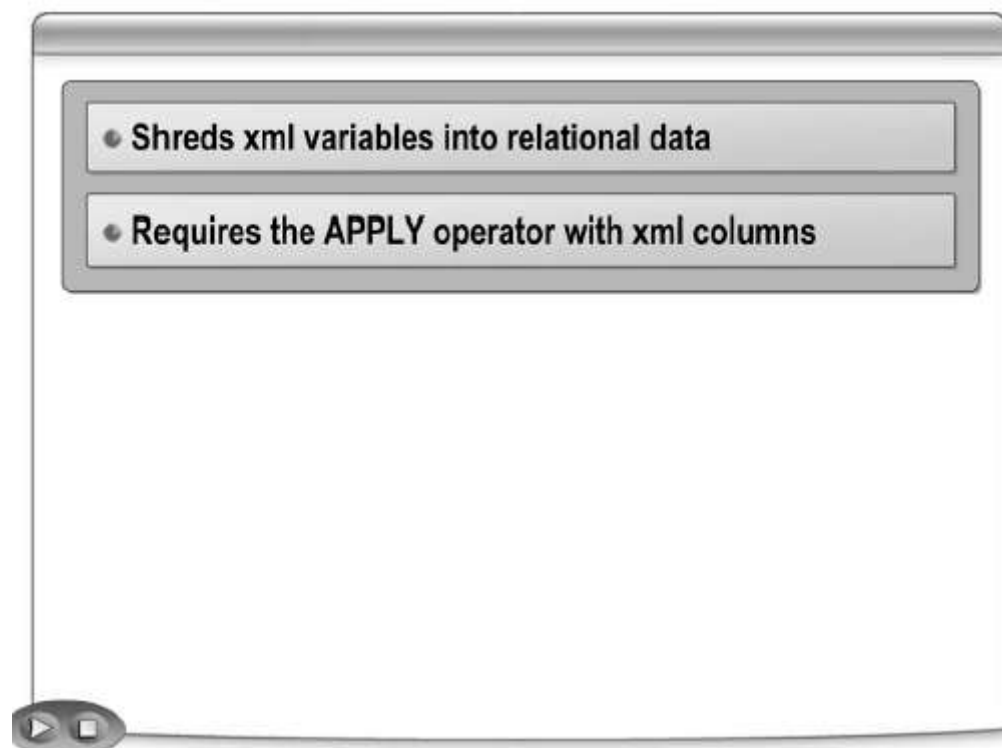
Вы можете использовать оператор **delete**, чтобы удалить узел из **xml**-значения. У оператора **delete** следующий синтаксис.

```
delete Expression
```

Параметр *Expression* - выражение XQuery, идентифицирующее узел, который будет удален. Следующий пример показывает, как использовать оператор **delete** в методе **modify**.

```
SET xmlCol.modify('declare default element namespace
                  "http://schemas.adventure-works.com/InvoiceList";
                  delete (/InvoiceList/Invoice/SalesPerson)[1]')
```

Метод nodes



Введение

Тип данных **xml** поддерживает метод **nodes**, который нужно использовать, чтобы сгенерировать реляционное представление данных XML. Метод **nodes** возвращает набор строк, в котором каждый узел, идентифицированный выражением XQuery, возвращается как контекстный узел, из которого могут извлечь данные последующие запросы.

Синтаксис метода nodes

У метода **nodes** следующий синтаксис.

`xmlvalue.nodes (XQuery) [AS] Table(Column)`

Параметры синтаксиса узлов описаны в следующей таблице.

Параметр	Описание
<i>Xmlvalue</i>	xml -переменная или столбец.
<i>XQuery</i>	Выражение XQuery, которое идентифицирует узлы, которые должен

	вернуть метод nodes .
<i>Table (Column)</i>	Имя таблицы и столбца, в который будут записаны результаты. Эта таблица может использоваться в последующих запросах, чтобы извлечь данные из контекстных узлов.

Как использовать метод **nodes** с **xml**-переменной

Чтобы извлечь из **xml**-переменной данные в реляционном формате, нужно использовать методы **query**, **value**, или **exist** с набором строк, возвращаемым методом **nodes**. Следующий пример показывает, как извлечь из **xml**-переменной данные заказа в реляционном формате.

```

DECLARE @xmlOrder xml

SET @xmlOrder = '<?xml version="1.0" ?>
    <Order OrderID="1000" OrderDate="2005-06-04">
        <LineItem ProductID="1" Price="2.99" Quantity="3" />
        <LineItem ProductID="2" Price="3.99" Quantity="1" />
    </Order>'

SELECT nCol.value('@ProductID', 'integer') ProductID,
       nCol.value('@Quantity', 'integer') Quantity
FROM @xmlOrder.nodes('/Order/LineItem') AS nTable(nCol)

```

Этот код возвращает следующие результаты.

ProductID	Quantity
1	3
2	1

Как использовать метод **nodes** с **xml**-столбцом

Вы можете использовать оператор **APPLY** с методом **nodes**, чтобы возвращать из **xml**-столбца данные в реляционном формате. Следующий пример показывает, как использовать метод **nodes**, чтобы извлечь данные из **xml**-столбца, который содержит документы заказа в том же формате, что и переменная в предыдущем примере.


```

DECLARE @xmlOrder xml

SELECT nCol.value('..@OrderID[1]', 'int') OrderID,
       nCol.value('..@OrderDate[1]', 'datetime') OrderDate,
       nCol.value('@ProductID[1]', 'int') ProductID,
       nCol.value('@Price[1]', 'money') Price,
       nCol.value('@Quantity[1]', 'int') Quantity
FROM Orders_X
CROSS APPLY OrderDoc.nodes('/Order/LineItem') AS nTable(nCol)

```

Этот код вернет результаты в виде следующей таблицы:

OrderID	OrderDate	ProductID	Price	Quantity
1000	2005-06-04-00:00:00.000	1	2.99	1
1000	2005-06-04-00:00:00.000	2	3.99	2
1001	2005-06-04-00:00:00.000	2	3.99	1
1002	2005-06-04-00:00:00.000	2	3.99	1