عنوان: **چرا XML و چرا پشتیبانی توکار از آن در SQL Server**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۴ ۱۴:۲۵
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, SQL Server, xml

مقدمه

فیلدهای XML از سال 2005 به امکانات توکار SQL Server اضافه شده‌اند و بسیاری از مزایای دنیای NoSQL را درون SQL Server رابطه‌ای مهیا می‌سازند. برای مثال با تعریف یک فیلد به صورت XML، می‌توان از هر ردیف به ردیفی دیگر، اطلاعات متفاوتی را ذخیره کرد؛ به این ترتیب امکان کار با یک فیلد که می‌تواند اطلاعات یک شیء را قبول کند و در حقیقت امکان تعریف اسکیمای پویا و متغیر را در کنار امکانات یک بانک اطلاعاتی رابطه‌ای که از اسکیمای ثابت پشتیبانی می‌کند، میسر می‌شود.
همچنین SQL Server در این حالت قابلیتی را ارائه می‌دهد که در بسیاری از بانک‌های اطلاعاتی NoSQL میسر نیست. در اینجا در صورت نیاز و لزوم می‌توان اسکیمای کاملا مشخصی را به یک فیلد XML نیز انتساب داد؛ هر چند این مورد اختیاری است و می‌توان یک un typed XML را نیز بکار برد. به علاوه امکانات کوئری گرفتن توکار از این اطلاعات را به کمک XPath ترکیب شده با T-SQL، نیز فراموش نکنید.
بنابراین اگر یکی از اهداف اصلی گرایش شما به سمت دنیای NoSQL، استفاده از امکان تعریف اطلاعاتی با اسکیمای متغیر و پویا است، فیلدهای نوع XML اس کیوال سرور را مدنظر داشته باشید.
یک مثال عملی: فناوری Azure Dev Fabric's Table Storage (نسخه Developer ویندوز Azure که روی ویندوزهای معمولی اجرا می‌شود؛ یک شبیه ساز خانگی) به کمک SQL Server و فیلدهای XML آن طراحی شده است.

چرا XML و چرا پشتیبانی توکار از آن در SQL Server

یک سند XML معمولا بیشتر از یک قطعه داده را در خود نگهداری می‌کند و نوع داده‌ی پیچیده محسوب می‌شود؛ برخلاف داده‌هایی مانند int یا varchar که نوع‌هایی ساده بوده و تنها یک قطعه از اطلاعات خاصی را در خود نگهداری می‌کنند. بنابراین شاید این سؤال مطرح شود که چرا از این نوع داده پیچیده در SQL Server پشتیبانی شده‌است؟
- از سال‌های نسبتا دور، از XML برای انتقال داده‌ها بین سیستم‌ها و سکوهای کاری مختلف استفاده شده‌است.
- استفاده‌ی گسترده‌ای در برنامه‌های تجاری دارد.
- بسیاری از فناوری‌های موجود از آن پشتیبانی می‌کنند.
برای مثال اگر با فناوری‌های مایکروسافتی کار کرده باشید، به طور قطع حداقل در یک یا چند قسمت از آن‌ها، مستقیما از XML استفاده شده‌است.
بنابراین با توجه به اهمیت و گستردگی استفاده از آن، بهتر است پشتیبانی توکاری نیز از آن داخل موتور یک بانک اطلاعاتی، پیاده سازی شده باشد. این مساله سهولت تهیه پشتیبان‌های خودکار، بازیابی آن‌ها و امنیت یکپارچه با SQL Server را به همراه خواهد داشت؛ به همراه تمام زیرساخت‌های مهیای در SQL Server.

روش‌های مختلف ذخیره سازی XML در بانک‌های اطلاعاتی رابطه‌ای

الف) ذخیره سازی متنی

این روش نیاز به نگارش خاصی از SQL Server یا بانک اطلاعاتی الزاما خاصی نداشته و با تمام بانک‌های اطلاعاتی رابطه‌ای سازگار است؛ مثلا از فیلدهای varchar برای ذخیره سازی آن استفاده شود. مشکلی که این روش به همراه خواهد داشت، از دست دادن ارزش یک سند XML و برخورد متنی با آن است. زیرا در این حالت برای تعیین اعتبار آن یا کوئری گرفتن از آن‌ها نیاز است اطلاعات را از بانک اطلاعاتی خارج کرده و در لایه‌ای دیگر از برنامه، کار جستجو پردازش آن‌ها را انجام داد.

ب) تجزیه XML به چندین جدول رابطه‌ای

برای مثال یک سند XML را درنظر بگیرید که دارای اطلاعات شخص و خریدهای او است. می‌توان این سند را به چندین فیلد در چندین جدول مختلف رابطه‌ای تجزیه کرد و سپس با روش‌های متداول کار با بانک‌های اطلاعاتی رابطه‌ای از آن‌ها استفاده نمود.

ج) ذخیره سازی آن‌ها توسط فیلدهای خاص XML
در این حالت با استفاده از فیلدهای ویژه XML می‌توان از فناوری‌های مرتبط با XML تمام و کمال استفاده کرد. برای مثال تهیه کوئری‌های پیچیده داخل همان بانک اطلاعاتی بدون نیاز به تجزیه سند به چندین جدول و یا خارج کردن آن‌ها از بانک اطلاعاتی و جستجوی بر روی آن‌ها در لایه‌ای دیگر از برنامه.

موارد کاربرد XML در SQL Server

کاربردهای مناسب

- اطلاعات، سلسله مراتبی و تو در تو هستند. XQuery و XPath در این موارد بسیار خوب عمل می‌کند.
- ساختار قسمتی از اطلاعات ثابت است و قسمتی از آن خیر. برای نمونه، یک برنامه‌ی فرم ساز را درنظر بگیرید که هر فرم آن هر چند دارای یک سری خواص ثابت مانند نام، گروه و امثال آن است، اما هر کدام دارای فیلدهای تشکیل دهنده متفاوتی نیز می‌باشد. به این ترتیب با استفاده از یک فیلد XML، دیگری نیازی به نگران بودن در مورد نحوه مدیریت اسکیمای متغیر مورد نیاز، نخواهد بود.
نمونه‌ی دیگر آن ذخیره سازی خواص متغیر اشیاء است. هر شیء دارای یک سری خواص ثابت است اما خواص توصیف کننده‌ی آن‌ها از هر رکورد به رکوردی دیگر متفاوت است.

کاربردهای نامناسب

- کل اطلاعات را داخل فیلد XML قرار دادن. هدف از فیلدهای XML قرار دادن یک دیتابیس داخل یک سلول نیست.
- ساختار تعریف شده کاملا مشخص بوده و به این زودی‌ها هم قرار نیست تغییر کند. در این حالت استفاده از قابلیت‌های رابطه‌ای متداول SQL Server مناسب‌تر است.
- قرار دادن اطلاعات باینری بسیار حجیم در سلول‌های XML ایی.

تاریخچه‌ی پشتیبانی از XML در نگارش‌های مختلف SQL Server

الف) SQL Server 2000
در SQL Server 2000 روش (ب) توضیح داده شده در قسمت قبل، پشتیبانی می‌شود. در آن برای تجزیه یک سند XML به معادل رابطه‌ای آن، از تابعی به نام OpenXML استفاده می‌شود و برای تبدیل این اطلاعات به XML از روش for XML ... Select می‌توان کمک گرفت. همچنین تاحدودی مباحث XPath Queries نیز در آن گنجانده شده‌است.

ب) SQL Server 2005
در نگارش 2005 آن، برای اولین بار نوع داده‌ای ویژه XML معرفی گشت به همراه امکان تعریف اسکیمای XML و اعتبارسنجی آن و پشتیبانی از XQuery برای جستجوی سریع بر روی داده‌های XML داخل همان بانک اطلاعاتی، بدون نیاز به استخراج اطلاعات XML و پردازش مجزای آن‌ها در لایه‌ای دیگر از برنامه.

ج) SQL Server 2008 به بعد
در اینجا فاز نگهداری این نوع داده خاص شروع شده و بیشتر شامل یک سری بهبودهای کوچک در کارآیی و نحوه‌ی استفاده از آن‌ها می‌شود.

استفاده از XML با کمک SQLCLR

از SQL Server 2005 به بعد، امکان استفاده از کلیه‌ی امکانات موجود در فضای نام System.Xml دات نت، در SQL Server نیز به کمک SQL CLR مهیا شده‌است. همچنین از SQL Server 2008 به بعد، امکانات فضای نام System.Xml.Linq و مباحث LINQ to XML نیز توسط SQL CLR پشتیبانی می‌شوند.
البته این امکانات در SQL Server 2005 نیز قابل استفاده هستند، اما اسمبلی شما unsafe تلقی می‌شود. پس از آزمایشات و

بررسی کافی، فضای نام مرتبط با LINQ to XML و امکانات آن، به عنوان اسمبلی‌هایی امن و قابل استفاده در SQL Server 2008 به بعد، معرفی شده‌اند.

مزایای وجود فیلد ویژه XML در SQL Server

پس از اینکه فیلدهای XML به صورت یک نوع داده بومی بانک اطلاعاتی SQL Server معرفی شدند، مزایای ذیل بلافاصله در اختیار برنامه نویس‌ها قرار گرفت:
- امکان تعریف آن‌ها به صورت یک ستون جدولی خاصی
- استفاده از آن‌ها به عنوان یک پارامتر رویه‌های ذخیره شده
- امکان تعریف خروجی توابع scalar سفارشی تعریف شده به صورت XML
- امکان تعریف متغیرهای T-SQL از نوع XML

برای مثال در اینجا نحوه‌ی تعریف یک جدول جدید دارای فیلدی از نوع XML را مشاهده می‌کنید:

```
CREATE TABLE xml_tab
(
  id INT,
  xml_col  XML
)
```

- پشتیبانی از فناوری‌های XML ایی مانند اعتبارسنجی اسکیما و نوشتن کوئری‌های پیشرفته با XQuery و XPath.
- امکان تعریف ایندکس‌های XML ایی اضافه شده‌است.

چه نوع XML ایی را می‌توان در فیلدهای XML ذخیره کرد؟

فیلدهای XML امکان ذخیره سازی داده‌های XML خوش فرم را مطابق استاندارد یک XML، دارند. حداکثر اندازه قابل ذخیره سازی در یک فیلد XML دو گیگابایت است.
البته امکانات مهیای در SQL Server در بسیاری از موارد فراتر از استاندارد یک XML هستند. به این معنا که در فیلدهای XML می‌توان Documents و یا Fragments را ذخیره سازی کرد. یک سند XML یا Document حاوی تنها یک ریشه اصلی است؛ اما یک Fragment می‌تواند بیش از یک ریشه اصلی را در خود ذخیره کند. یک مثال:

```
DECLARE @xml_tab TABLE (xml_col XML)
-- document
INSERT @xml_tab VALUES ('<person/>')
-- fragment
INSERT @xml_tab VALUES ('<person/><person/>')
SELECT * FROM @xml_tab
```

مدل داده‌ای XML در SQL Server بر مبنای استانداردهای XQuery و XPath طراحی شده‌است و این استانداردها Fragments را به عنوان یک قطعه داده XML معتبر، قابل پردازش می‌دانند؛ علاوه بر آن مقادیر null و خالی را نیز معتبر می‌دانند. برای مثال عبارات ذیل معتبر هستند:

```
DECLARE @xml_tab TABLE (xml_col XML)
-- text only
INSERT @xml_tab VALUES ('data data data .....')
-- empty string
INSERT @xml_tab VALUES ('')
-- null value
INSERT @xml_tab VALUES (null)
SELECT * FROM @xml_tab
```

همچنین امکان ذخیره سازی یک متن خالی بدون فرمت نیز در اینجا مجاز است. بنابراین به کمک T-SQL می‌توان برای مثال نوع داده varchar و varchar max را به XML تبدیل کرد و برعکس. امکان تبدیل Text و NText (منسوخ شده) نیز به XML وجود دارد ولی

در این حالت خاص، عکس آن، پشتیبانی نمی‌شود.
به علاوه باید دقت داشت که در SQL Server نوع داده‌ای XML برای ذخیره سازی داده‌ها بکار گرفته می‌شود. به این معنا که در اینجا پیشوندهای فضاهای نام XML بی‌معنا هستند.

```
DECLARE @xml_tab TABLE (xml_col XML)
INSERT @xml_tab VALUES ('<doc/>')
INSERT @xml_tab VALUES ('<doc xmlns="http://www.doctors.com"/>')
-- این سه سطر در عمل یکی هستند
INSERT @xml_tab VALUES ('<doc xmlns="http://www.documents.com"/>')
INSERT @xml_tab VALUES ('<dd:doc xmlns:dd="http://www.documents.com"/>')
INSERT @xml_tab VALUES ('<rr:doc xmlns:rr="http://www.documents.com"/>')
SELECT * FROM @xml_tab
```

در این مثال، سه insert آخر در عمل یکی درنظر گرفته می‌شوند.

ذخیره سازی داده‌های XML Encoding

SQL Server امکان ذخیره سازی اطلاعات متنی را به فرمت UFT8، اسکی و غیره، دارد. اما جهت پردازش فیلدهای XML و ذخیره سازی آن‌ها از Collation پیش فرض بانک اطلاعاتی کمک خواهد گرفت. البته ذخیره سازی نهایی آن همیشه با فرمت UCS2 است (یونیکد دو بایتی).

```
DECLARE @xml_tab TABLE  (id INT, xml_col XML)

INSERT INTO @xml_tab
VALUES
  (
5,
N'<?xml version="1.0" encoding="utf-8"?>
<doc1>
  <row name="vahid"></row>
</doc1>
')
```

برای نمونه به مثال فوق دقت کنید. اگر آن‌را اجرا کنید، برنامه با خطای ذیل متوقف خواهد شد:

```
XML parsing: line 1, character 38, unable to switch the encoding
```

علت اینجا است که با قرار دادن N در ابتدای رشته XML ایی در حال ذخیره سازی، آن‌را به صورت یونیکد دوبایتی معرفی کرده‌ایم اما encoding سند در حال ذخیره سازی utf-8 تعریف شده‌است و این‌دو با هم سازگاری ندارند.
برای حل این مشکل باید N ابتدای رشته را حذف کرد. روش دوم، معرفی و استفاده از utf-16 است بجای utf-8 در ویژگی encoding.
همچنین در این حالت اگر encoding را utf-16 معرفی کنیم و ابتدای رشته در حال ذخیره سازی N قرار نگیرد، باز با خطای unable to switch the encoding مواجه خواهیم شد.

نحوه‌ی ذخیره سازی اطلاعات XML ایی در SQL Server

SQL Server فرمت اطلاعات XML وارد شده را حفظ نمی‌کند. برای مثال اگر قطعه کد زیر را اجرا کنید

```
DECLARE @xml_tab TABLE  (id INT, xml_col XML)

INSERT INTO @xml_tab
VALUES
  (
5,
'<?xml version="1.0" encoding="utf-8"?><doc1><row name="vahid"></row></doc1>'
  )

SELECT * FROM @xml_tab
```

خروجی Select انجام شده به صورت زیر است:

```
<doc1>
  <row name="vahid" />
</doc1>
```

اطلاعات و داده نهایی، بدون تغییری از آن قابل استخرج است. اما اصطلاحا lexical integrity آن حفظ نشده و نمی‌شود. بنابراین در اینجا ذکر سطر xml version ضروری نیست و یا برای مثال اگر ویژگی‌ها را توسط " و یا ' مقدار دهی کنید، همیشه توسط " ذخیره خواهد شد.

ذخیره سازی داده‌هایی حاوی کاراکترهای غیرمجاز XML

اطلاعات دنیای واقعی همیشه به همراه اطلاعات تک کلمه‌ای ساده نیست. ممکن است نیاز شود انواع و اقسام حروف و تگ‌ها نیز در این بین به عنوان داده ذخیره شوند. روش حل استاندارد آن بدون نیاز به دستکاری اطلاعات ورودی، استفاده از CDATA است:

```
DECLARE @xml_tab TABLE  (id INT, xml_col XML)

INSERT INTO @xml_tab
VALUES
  (
5,
'<person><![CDATA[ 3 > 2 ]]></person>'
  )

SELECT * FROM @xml_tab
```

در این حالت خروجی select اطلاعات ذخیره شده به صورت زیر خواهد بود:

```
<person> 3 &gt; 2 </person>
```

به صورت خودکار قسمت CDATA پردازش شده و اصطلاحا حروف غیرمجاز XML ایی به صورت خودکار escape شده‌اند.

محدودیت‌های فیلدهای XML

- امکان مقایسه مستقیم را ندارند؛ بجز مقایسه با نال. البته می‌توان XML را تبدیل به مثلا varchar کرد و سپس این داده رشته‌ای را مقایسه نمود. برای مقایسه با null توابع isnull و coalesce نیز قابل بکارگیری هستند.
- order by و group by بر روی این فیلدها پشتیبانی نمی‌شود.
- به عنوان ستون کلید قابل تعریف نیست.
- به صورت منحصربفرد و unique نیز قابل علامتگذاری و تعریف نیست.
- فیلدهای XML نمی‌توانند دارای collate باشند.

عنوان: **پشتیبانی از XML Schema در SQL Server**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۵ ۲:۱۱
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, SQL Server, xml

XML Schema چیست؟

XML Schema معرف ساختار، نوع داده‌ها و المان‌های یک سند XML است. البته باید درنظر داشت که تعریف XML Schema کاملا اختیاری است و اگر تعریف شود مزیت اعتبارسنجی داده‌های در حال ذخیره سازی در بانک اطلاعاتی را به صورت خودکار به همراه خواهد داشت. در این حالت به نوع داده‌ای XML دارای اسکیما، typed XML و به نوع بدون اسکیما، untyped XML گفته می‌شود.
به یک نوع XML، چندین اسکیمای مختلف را می‌توان نسبت داد و به آن XML schema collection نیز می‌گویند.

XML schema collections پیش فرض و سیستمی

تعدادی XML Schema پیش فرض در SQL Server تعریف شده‌اند که به آن‌ها sys schema collections گفته می‌شود.

```
Prefix - Namespace
xml = http://www.w3.org/XML/1998/namespace
xs = http://www.w3.org/2001/XMLSchema
xsi = http://www.w3.org/2001/XMLSchema-instance
fn = http://www.w3.org/2004/07/xpath-functions
sqltypes = http://schemas.microsoft.com/sqlserver/2004/sqltypes
xdt = http://www.w3.org/2004/07/xpath-datatypes
(no prefix) = urn:schemas-microsoft-com:xml-sql
(no prefix) = http://schemas.microsoft.com/sqlserver/2004/SOAP
```

در اینجا پیشوندها و فضاهای نام sys schema collections را ملاحظه می‌کنید. از این اسکیماها برای تعاریف strongly typed امکانات موجود در SQL Server کمک گرفته شده‌است.
اگر علاقمند باشید تا این تعاریف را مشاهده کنید به مسیر Program Files\Microsoft SQL Server\version\Tools\Binn\schemas\sqlserver در جایی که SQL Server نصب شده‌است مراجعه نمائید. برای مثال در مسیر Tools\Binn\schemas\sqlserver\2006\11\events فایل events.xsd قابل مشاهده است و یا در مسیر Tools\Binn\schemas\sqlserver\2004\07 اسکیمای ابزارهای query processor و show plan قابل بررسی می‌باشد.
مهم‌ترین آن‌ها را در پوشه Tools\Binn\schemas\sqlserver\2004\sqltypes در فایل sqltypes.xsd می‌توانید ملاحظه کنید. اگر به محتوای آن دقت کنید، قسمتی از آن به شرح ذیل است:

```
<xsd:simpleType name="char">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="nchar">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="varchar">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="nvarchar">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="text">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="ntext">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
```

در اینجا نوع‌های توکار char تا ntext به xsd:string نگاشت شده‌اند و برای اعتبارسنجی datetime و نگاشت آن، از الگوی ذیل استفاده می‌شود؛ به همراه حداقل و حداکثر قابل تعریف:

```
<xsd:simpleType name="datetime">
    <xsd:restriction base="xsd:dateTime">
      <xsd:pattern value="((000[1-9])|(00[1-9][0-9])|(0[1-9][0-9]{2})|([1-9][0-9]{3}))-((0[1-
9])|(1[012]))-((0[1-9])|([12][0-9])|(3[01]))T(([01][0-9])|(2[0-3]))(:[0-5][0-9]){2}(\.[0-
9]{2}[037])?"/>
      <xsd:maxInclusive value="9999-12-31T23:59:59.997"/>
      <xsd:minInclusive value="1753-01-01T00:00:00.000"/>
    </xsd:restriction>
  </xsd:simpleType>
```

ادیتور SQL Server managment studio به خوبی، گشودن، ایجاد و یا ویرایش فایل‌هایی با پسوند xsd را پشتیبانی می‌کند.

تعریف XML Schema و استفاده از آن جهت تعریف یک strongly typed XML

XML Schema مورد استفاده در SQL Server حتما باید در بانک اطلاعاتی ذخیره شود و برای خواندن آن، برای مثال از فایل سیستم استفاده نخواهد شد.

```
CREATE XML SCHEMA COLLECTION invcol AS
'<xs:schema ... targetNamespace="urn:invoices">
...
</xs:schema>
'

CREATE TABLE Invoices(
id int IDENTITY PRIMARY KEY,
invoice XML(invcol)
)
```

در اینجا نحوه‌ی تعریف کلی یک XML Schema collection و سپس انتساب آن‌را به یک ستون XML ملاحظه می‌کنید. ستون invoice که از نوع XML تعریف شده، ارجاعی را به اسکیمای تعریف شده دارد.
در ادامه نحوه‌ی تعریف یک اسکیمای نمونه قابل مشاهده است:

```
CREATE XML SCHEMA COLLECTION geocol AS
'<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:geo"
           elementFormDefault="qualified"
           xmlns:tns="urn:geo">
  <xs:simpleType name="dim">
    <xs:restriction base="xs:int" />
  </xs:simpleType>
  <xs:complexType name="Point">
    <xs:sequence>
      <xs:element name="X" type="tns:dim"  minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="Y" type="tns:dim"  minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Point" type="tns:Point" />
</xs:schema>'
```

در این اسکیما، یک نوع ساده به نام dim تعریف شده‌است که محدودیت آن، ورود اعداد صحیح می‌باشد. همچنین امکان تعریف نوع‌های پیچیده نیز در اینجا وجود دارد. برای مثال نوع پیچیده Point دارای دو المان X و Y از نوع dim در ادامه تعریف شده‌است. المانی که نهایتا بر این اساس در XML ظاهر خواهد شد توسط xs:element تعریف شده‌است.
اکنون برای آزمایش اسکیمای تعریف شده، جدول geo_tab را به نحو ذیل تعریف می‌کنیم و سپس سعی در insert دو رکورد در آن خواهیم کرد:

```
declare @geo_tab table(
 id int identity primary key,
 point xml(content geocol)
)

insert into @geo_tab values('<Point xmlns="urn:geo"><X>10</X><Y>20</Y></Point>')
insert into @geo_tab values('<Point xmlns="urn:geo"><X>10</X><Y>test</Y></Point>')
```

در اینجا اگر دقت کنید، برای تعریف نام اسکیمای مورد استفاده، واژه content نیز ذکر شده‌است. Content مقدار پیش فرض است و در آن پذیرش XML Fragments یا محتوای XML ایی با بیش از یک Root element مجاز است. حالت دیگر آن document است که تنها یک Root element را می‌پذیرد.
در این مثال، insert اول با موفقیت انجام خواهد شد؛ اما insert دوم با خطای ذیل متوقف می‌شود:

```
XML Validation: Invalid simple type value: 'test'. Location: /*:Point[1]/*:Y[1]
```

همانطور که ملاحظه می‌کنید، چون در insert دوم، در المان عددی Y، مقدار test وارد شده‌است و تطابقی با اسکیمای تعریف شده ندارد، insert آن مجاز نخواهد بود.

یافتن محل ذخیره سازی اطلاعات اسکیما در SQL Server

اگر علاقمند باشید تا با محل ذخیره سازی اطلاعات اسکیما، نوع‌های تعریف شده و حتی محل استفاده از آن‌ها در بانک‌های اطلاعاتی مختلف موجود آشنا شوید و گزارشی از آن‌ها تهیه کنید، می‌توانید از کوئری‌های ذیل استفاده نمائید:

```
select * from sys.xml_schema_collections
select * from sys.xml_schema_namespaces
select * from sys.xml_schema_elements
select * from sys.xml_schema_attributes
select * from sys.xml_schema_types
select * from sys.column_xml_schema_collection_usages
select * from sys.parameter_xml_schema_collection_usages
```

باید دقت داشت زمانیکه یک schema در حال استفاده است (یک رکورد ثبت شده مقید به آن تعریف شده باشد)، امکان drop آن نخواهد بود. حتما باید اطلاعات و ستون مرتبط، ارجاعی را به schema نداشته باشند تا بتوان آن schema را حذف کرد.
محتوای اسکیمای ذخیره شده به شکل xsd تعریف شده، ذخیره سازی نمی‌شود. بلکه اطلاعات آن تجزیه شده و سپس در جداول سیستمی SQL Server ذخیره می‌گردند. هدف از اینکار، بالا بردن سرعت اعتبارسنجی XML typedها است.
بنابراین بدیهی است در این حالت اطلاعاتی مانند commnets موجود در xsd تهیه شده در بانک اطلاعاتی ذخیره نمی‌گردند.
برای بازیابی اطلاعات اسکیمای ذخیره شده می‌توان از متد xml_schema_namespace استفاده کرد:

```
declare @x xml
select @x = xml_schema_namespace(N'dbo', N'geocol')
print convert(varchar(max), @x)
```

برای تعریف و یا تغییر یک XML Schema نیاز به دسترسی مدیریتی یا dbo است (به صورت پیش فرض). همچنین برای استفاده از Schema تعریف شده، کاربر متصل به SQL Server باید دسترسی Execute و References نیز داشته باشد.

نحوه‌ی ویرایش یک schema collection موجود

چند نکته:
- امکان alter یک schema collection وجود دارد.
- می‌توان یک schema جدید را به collection موجود افزود.
- امکان افزودن (و نه تغییر) نوع‌های یک schema موجود، میسر است.
- امکان drop یک اسکیما از collection موجودی وجود ندارد. باید کل collection را drop کرد و سپس آن‌را تعریف نمود.
- جداولی با فیلدهای nvarchar را می‌توان به فیلدهای XML تبدیل کرد و برعکس.
- امکان تغییر یک فیلد XML به حالت untyped و برعکس وجود دارد.

فرض کنید که می‌خواهیم اسکیمای متناظر با یک ستون XML را تغییر دهیم. ابتدا باید آن ستون XML ایی را Alter کرده و قید اسکیمای آن‌را برداریم. سپس باید اسکیمای موجود را drop و مجددا ایجاد کرد. همانطور که پیشتر ذکر شد، اگر اسکیمایی در حال

استفاده باشد، قابل drop نیست. در ادامه مجددا باید ستون XML ایی را تغییر داده و اسکیمای آن‌را معرفی کرد.
روش دوم مدیریت این مساله، اجازه دادن به حضور بیش از یک اسکیما در مجموعه است. به عبارتی نگارش‌بندی اسکیما که به نحو ذیل قابل انجام است:

```
alter XML SCHEMA COLLECTION geocol add @x
```

در اینجا به collection موجود، یک اسکیمای جدید (برای مثال نگارش دوم اسکیمای فعلی) اضافه می‌شود. در این حالت geocol، هر دو نوع اسکیمای موجود را پشتیبانی خواهد کرد.

نحوه‌ی import یک فایل xsd و ذخیره آن به صورت اسکیما

اگر بخواهیم یک فایل xsd موجود را به عنوان xsd معرفی کنیم می‌توان از دستورات ذیل کمک گرفت:

```
declare @x xml
set @x = (select * from openrowset(bulk 'c:\path\file.xsd', single_blob) as x)
CREATE XML SCHEMA COLLECTION geocol2 AS @x
```

در اینجا به کمک openrowset فایل xsd موجود، در یک متغیر xml بارگذاری شده و سپس در دستور ایجاد یک اسکیما کالکشن جدید استفاده می‌شود.
از openrowset برای خواندن یک فایل xml موجود، جهت insert محتوای آن در بانک اطلاعاتی نیز می‌توان استفاده کرد.

محدودیت‌های XML Schema در SQL Server

تمام استاندارد XML Schema در SQL Server پشتیبانی نمی‌شود و همچنین این مورد از نگارشی به نگارشی دیگر نیز ممکن است تغییر یافته و بهبود یابد. برای مثال در SQL Server 2005 از xs:any پشتیبانی نمی‌شود اما در SQL Server 2008 این محدودیت برطرف شده‌است. همچنین مواردی مانند xs:keyref، xs:key، xs:notation، xs:redefine، xs:include و xs:unique در SQL Server پشتیبانی نمی‌شوند.

یک نکته‌ی تکمیلی

برنامه‌ای به نام xsd.exe به همراه Visual Studio ارائه می‌شود که قادر است به صورت خودکار از یک فایل XML موجود، XML Schema تولید کند. اطلاعات بیشتر

عنوان: **استفاده از XQuery - قسمت اول**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۵ ۲۲:۴۲
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, SQL Server, xml

XQuery زبانی است که در ترکیب با T-SQL، جهت کار با نوع داده‌ای XML در SQL Server مورد استفاده قرار می‌گیرد. XQuery یک زبان declarative است. عموما زبان‌های برنامه نویسی یا declarative هست و یا imperative. در زبان‌های imperative مانند سی‌شارپ، در هر بار، یک سطر به پردازشگر برای توضیح اعمالی که باید انجام شوند، معرفی خواهد شد. در زبان‌های declarative، توسط زبانی سطح بالا، به پردازشگر عنوان می‌کنیم که قرار است جواب چه چیزی باشد. در این حالت پردازشگر سعی می‌کند تا بهینه‌ترین روش را برای یافتن پاسخ بیابد. SQL و XQuery، هر دو جزو زبان‌های declarative هستند.
XQuery پیاده سازی شده در SQL Server با استانداردهای XQuery 1.0 و XPath 2.0 سازگار است. XQuery برای کار با نودهای مختلف یک سند XML، از XPath استفاده می‌کند. همچنین باید دقت داشت که این زبان به بزرگی و کوچکی حروف حساس است. در آن تمام واژه‌های کلیدی lowercase هستند و تمام متغیرها با علامت $ شروع می‌شوند.

ورودی و خروجی در XQuery

استاندارد XQuery از یک سری توابع ورودی مانند doc برای کار با یک سند و collection برای پردازش چندین سند کمک می‌گیرد. SQL Server از هیچکدام از این توابع پشتیبانی نمی‌کند. در اینجا از XQuery، به کمک متدهای نوع داده‌ای XML استفاده خواهد شد. این متدها شامل موارد ذیل هستند:
- query : یک xml را به عنوان ورودی گرفته و نهایتا یک خروجی XML دیگر را بر می‌گرداند.
- exist : خروجی bit دارد؛ true یا false.
- value : یک خروجی SQL Type را ارائه می‌دهد.
- nodes : خروجی جدولی دارد.
- modify : برای تغییر اطلاعات بکار می‌رود.

این موارد را در طی مثال‌هایی بررسی خواهیم کرد. بنابراین در ادامه نیاز است یک سند XML را که در طی مثال‌های این قسمت مورد استفاده قرار خواهد گرفت، به شرح ذیل مدنظر داشته باشیم:

```
DECLARE @data XML

SET @data =
'<people>
 <person>
  <name>
<givenName>name1</givenName>
<familyName>lname1</familyName>
  </name>
  <age>33</age>
  <height>short</height>
 </person>
 <person>
  <name>
<givenName>name2</givenName>
<familyName>lname2</familyName>
  </name>
  <age>40</age>
  <height>short</height>
 </person>
 <person>
  <name>
<givenName>name3</givenName>
<familyName>lname3</familyName>
  </name>
  <age>30</age>
  <height>medium</height>
 </person>
</people>'
```

در اینجا people در ریشه سند قرار گرفته و سپس سه شخص به مجموعه نودهای آن اضافه شده‌اند.

همانطور که در قسمت قبل نیز ذکر شد، اگر اطلاعات شما در یک فایل XML قرار دارند، نحوه‌ی خواندن آن به شکل یک فیلد XML با کمک openrowset مطابق دستورات زیر خواهد بود:

```
declare @data xml
set @data = (select * from openrowset(bulk 'c:\path\data.xml', single_blob) as x)
```

بررسی متد query

متد query یک XQuery متنی را دریافت کرده، آن‌را بر روی XML ورودی اجرا نموده و سپس یک خروجی XML دیگر را ارائه خواهد داد.
اگر به کتاب‌های استاندارد XQuery مراجعه کنید، به یک چنین کوئری‌هایی خواهید رسید:

```
for $p in doc("data.xml")/people/person
 where $p/age > 30
 return $p/name/givenName/text()
```

همانطور که عنوان شد، متد doc در SQL Server پیاده سازی نشده‌است. بجای آن حداقل از دو روشی که برای مقدار دهی متغیر data عنوان شد، می‌توان استفاده کرد. پس از آن معادل کوئری فوق در SQL Server به نحو ذیل توسط متد query نوشته می‌شود:

```
SELECT @data.query('
 for $p in /people/person
 where $p/age > 30
 return $p/name/givenName/text()
 ')
```

این کوئری givenName تمام اشخاص بالای 30 سال را از سند XML مطرح شده در ابتدای بحث، استخراج می‌کند. خروجی آن نیز یک XML است و اگر آن‌را در SQL Server managment studio اجرا کنید، یک خط آبی زیر نتیجه‌ی آن کشیده می‌شود که بیانگر لینکی است، به محتوای XML حاصل.

بررسی متد value

در ادامه متد value را بررسی خواهیم کرد. در اینجا قصد داریم مقدار سن اولین شخص را نمایش دهیم:

```
SELECT @data.value('/people/person/age', 'int')
```

پارامتر اول متد value یک XQuery است و پارامتر دوم آن، نوع داده‌ای که قرار است بازگشت داده شود. در اینجا اگر اطلاعاتی یافت نشود، نال بازگشت داده خواهد شد.
اگر کوئری فوق را اجرا کنیم با خطای ذیل مواجه خواهیم شد:

```
XQuery [value()]: 'value()' requires a singleton (or empty sequence), found operand of type
'xdt:untypedAtomic *'
```

در اینجا چون از XML Schema استفاده نشده، به untyped Atomic اشاره شده‌است و * پس از آن به zero to many اشاره دارد که برخلاف خروجی zero to one متد value است. این متد، صفر یا حداکثر یک مقدار را باید بازگشت دهد.
برای رفع این مشکل و اشاره به اولین شخص، می‌توان از روش ذیل استفاده کرد:

```
SELECT @data.value('(/people/person/age)[1]', 'int')
```

تولید schema برای سند XML بحث جاری

با استفاده از برنامه Infer.exe مایکروسافت به سادگی می‌توان برای یک سند XML، فایل Schema ایجاد کرد. این برنامه را از اینجا می‌توانید دریافت کنید. پس از آن، اگر فرض کنیم اطلاعات سند XML مثال فوق در فایلی به نام people.xml ذخیره شده‌است، می‌توان schema آن‌را توسط دستور ذیل تولید کرد:

```
Infer.exe people.xml -o schema.xsd
```

people.xml و people.xsd

که نهایتا چنین شکلی را خواهد داشت:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="people">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="person">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="givenName" type="xs:string" />
                    <xs:element name="familyName" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="age" type="xs:unsignedByte" />
              <xs:element name="height" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

البته این فایل تولید شده به صورت خودکار، نوع age را unsignedByte تشخیص داده است که در صورت نیاز می‌توان آن‌را به int تبدیل کرد. ولی در کل خروجی آن بسیار با کیفیت و نزدیک به واقعیت است.
این خروجی را که اکنون به صورت یک فایل xsd، در کنار فایل xml معرفی شده به آن می‌توان یافت، با استفاده از openrowset قابل بارگذاری است:

```
declare @schema xml
set @schema = (select * from openrowset(bulk 'c:\path\schema_1.xsd', single_blob) as x)
```

و یا حتی می‌توان یک متغیر از نوع XML را تعریف و سپس محتوای آن را به صورت رشته‌ای در همانجا مقدار دهی کرد.
سپس از این متغیر برای تعریف یک اسکیما کالکشن جدید استفاده خواهیم کرد:

```
CREATE XML SCHEMA COLLECTION poeple_xsd AS @schema
```

در ادامه می‌توان متغیر data را که جهت مقدار دهی سند XML در ابتدای بحث تعریف کردیم، به صورت strongly typed تعریف کنیم:

```
DECLARE @data XML(poeple_xsd)
SET @data = 'مانند قبل با همان محتوایی که در ابتدای بحث عنوان شد'
```

اینبار اگر کوئری ذیل را برای یافتن سن اولین شخص اجرا کنیم:

```
SELECT @data.value('/people/person[1]/age', 'int')
```

خطای واضح‌تری را دریافت خواهیم کرد:

```
XQuery [value()]: 'value()' requires a singleton (or empty sequence), found operand of type
'xs:unsignedByte *'
```

در اینجا xs:unsignedByte بجای xdt:untypedAtomic پیشین گزارش شده‌است.
مشکل کوئری نوشته در اینجا این است که زمانیکه نوع XML تعریف می‌شود، پیش فرض آن content است. یعنی در این حالت چندین root elemnt مجاز هستند. بنابراین person 1 درخواستی، می‌تواند چندین خروجی داشته باشد که در متد value مجاز نیست. این متد، پیش از اجرای کوئری، توسط parser تعیین اعتبار می‌شود و الزاما نیازی نیست تا حتما اجرا شده و سپس مشخص شود که چندین خروجی حاصل آن است.
اینبار تنها کاری که باید برای رفع مشکل گزارش شده انجام شود، تغییر content پیش فرض به document است:

```
DECLARE @data XML(document poeple_xsd)
```

تغییر دیگری نیاز نیست. حتی نیاز نیست از پرانتزها برای مشخص کردن اولین age استفاده کنیم. چون به کمک schema دقیقا مشخص شده‌است که این سند، چه ساختاری دارد و همانند مثال ابتدای بحث، دیگر یک xml untyped نیست.

sequences در XQuery

Sequences بسیار شبیه به آرایه‌ای از آیتم‌ها هستند و منظور مجموعه‌ای از نودها یا مقادیر آن‌ها است. برای مثال به ورودی کوئری‌های XQuery به شکل توالی از یک سند و به خروجی آن‌ها همانند توالی صفر تا چند نود نگاه کنید.

```
DECLARE @x XML
SET @x=''
SELECT @x.query(
'
1,2
(: 1,2 :)
')
```

در مثال فوق یک توالی اصطلاحا دو atomic value را ایجاد کرده‌ایم. این آیتم‌ها با کاما از یکدیگر جدا می‌شوند. همچنین x، پیش از بکارگیری مقدار دهی شده‌است تا null نباشد. عبارتی که بین (: :) قرار می‌گیرد، یک کامنت تفسیر خواهد شد.

همچنین باید دقت داشت که این توالی خطی تفسیر می‌شود.

```
DECLARE @x XML
SET @x=''
SELECT @x.query(
'
for $x in (1,2,3)
for $y in (4,5)
return ($x,$y)
')
```

در اینجا یک جوین کارتزین نوشته شده است، که در آن یک x با یک y جوین خواهد شد. شاید تصور کنید که خروجی آن مجموعه‌ای است با سه عضو که هر عضو آن با دو عضو دیگر جوین می‌شود. اما اگر کوئری فوق را اجرا کنید، یک خروجی خطی را مشاهده خواهید کرد.

به علاوه در SQL Server امکان تعریف Heterogeneous sequences وجود ندارد؛ به عبارتی توالی بین مقادیر و نودها مجاز نیست.
برای مثال اگر کوئری زیر را اجرا کنید:

```
DECLARE @x XML
SET @x=''
```

```
SELECT @x.query(
'
1, <node/>
')
```

با خطای ذیل مواجه خواهید شد:

```
XQuery [query()]: Heterogeneous sequences are not allowed: found 'xs:integer' and
'element(node,xdt:untyped)'
```

عنوان: **استفاده از XQuery - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۷ ۰:۵
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, SQL Server, xml

در ادامه‌ی مباحث XQuery، سایر قابلیت‌های توکار SQL Server را برای کار با اسناد XML بررسی خواهیم کرد.

کوئری گرفتن از اسناد XML دارای فضای نام، توسط XQuery

در مثال زیر، تمام المان‌های سند XML، در فضای نام http://www.people.com تعریف شده‌اند.

```
DECLARE @doc XML
SET @doc ='
<p:people xmlns:p="http://www.people.com">
 <p:person name="Vahid" />
 <p:person name="Farid" />
</p:people>
'
SELECT @doc.query('/people/person')
```

اگر کوئری فوق را برای یافتن اشخاص اجرا کنیم، خروجی آن خالی خواهد بود (و یا یک empty sequence)؛ زیرا کوئری نوشته شده به دنبال اشخاصی است که در فضای نام خاصی تعریف نشده‌اند.
سعی دوم احتمالا روش ذیل خواهد بود

```
SELECT @doc.query('/p:people/p:person')
```

که به خطای زیر منتهی می‌شود:

```
XQuery [query()]: The name "p" does not denote a namespace.
```

برای حل این مشکل باید از مفهومی به نام prolog استفاده کرد. هر XQuery از دو قسمت prolog و body تشکیل می‌شود.
قسمت prolog می‌تواند شامل تعاریف فضاهای نام، متغیرها، متدها و غیره باشد و قسمت body، همان کوئری تهیه شده‌است. البته SQL Server از قسمت prolog استاندارد XQuery، فقط تعریف فضاهای نام آن‌را مطابق مثال ذیل پشتیبانی می‌کند:

```
SELECT @doc.query('
declare default element namespace "http://www.people.com";
/people/person
')
```

یک سند XML ممکن است با بیش از یک فضای نام تعریف شود. در این حالت خواهیم داشت:

```
SELECT @doc.query('
declare namespace aa="http://www.people.com";
/aa:people/aa:person
')
```

در اینجا در قسمت prolog، برای فضای نام تعریف شده در سند XML، یک پیشوند را تعریف کرده و سپس، استفاده از آن مجاز خواهد بود.
روش دیگر تعریف فضای نام، استفاده از WITH XMLNAMESPACES، پیش از تعریف کوئری است:

```
WITH XMLNAMESPACES(DEFAULT 'http://www.people.com')
SELECT @doc.query('/people/person')
```

البته باید دقت داشت، زمانیکه WITH XMLNAMESPACES تعریف می‌شود، عبارت T-SQL پیش از آن باید با یک سمی‌کالن خاتمه یابد؛ و گرنه یک خطای دستوری خواهید گرفت.
در اینجا نیز امکان کار با چندین فضای نام وجود دارد و برای این منظور تنها کافی است از تعریف Alias استفاده شود. فضاهای نام بعدی با یک کاما از هم مجزا خواهند شد.

```
WITH XMLNAMESPACES('http://www.people.com' AS aa)
SELECT @doc.query('/aa:people/aa:person')
```

عبارات XPath و FLOWR

XQuery از دو نوع عبارت XPath و FLOWR می‌تواند استفاده کند. XQuery همیشه از XPath برای انتخاب داده‌ها و نودها استفاده می‌کند. در اینجا هر نوع XPath سازگار با استاندارد 2 آن، یک XQuery نیز خواهد بود. برای انجام اعمالی بجز انتخاب داده‌ها، باید از عبارات FLOWR استفاده کرد؛ برای مثال برای ایجاد حلقه، مرتب سازی و یا ایجاد نودهای جدید.
در مثال زیر که data آن در قسمت قبل تعریف شد، دو کوئری نوشته شده یکی هستند:

```
SELECT @data.query('
 (: FLOWE :)
 for $p in /people/person
 where $p/age > 30
 return $p
 ')

SELECT @data.query('
(: XPath :)
/people/person[age>30]
')
```

اولین کوئری به روش FLOWR تهیه شده‌است و دومین کوئری از استاندارد XPath استفاده می‌کند. از دیدگاه SQL Server این دو یکی بوده و حتی Query Plan یکسانی نیز دارند.

XPath بسیار شبیه به مسیر دهی‌های یونیکسی است. بسیار فشرده بوده و همچنین مناسب است برای کار با ساختارهای تو در تو و سلسله مراتبی. مثال زیر را درنظر بگیرید:

```
/books/book[1]/title/chapter
```

در اینجا books، المان ریشه است. سپس به اولین کتاب این ریشه اشاره می‌شود. سپس به المان عنوان و مسیر نهایی، به فصل ختم می‌شود. البته همانطور که در قسمت‌های پیشین نیز ذکر شد، حالت content، پیش فرض بوده و یک فیلد XML می‌تواند دارای چندین ریشه باشد.

در XPath توسط قابلیتی به نام محور می‌توان به المان‌های قبلی یا بعدی دسترسی پیدا کرد. این محورهای پشتیبانی شده در SQL Server عبارتند از self (خود نود)، child (فرزند نود)، parent (والد نود)، decedent (فرزند فرزند فرزند ...)و attribute (دسترسی به ویژگی‌ها). محورهای استانداردی مانند preceding-sibling و following-sibling در SQL Server با عملگرهایی مانند >> و << پشتیبانی می‌شوند.

مثال‌هایی از نحوه‌ی استفاده از محورهای XPath

اینبار قصد داریم یک سند XML نسبتا پیچیده را بررسی کرده و اجزای مختلف آن‌را به کمک XPath بدست بیاوریم.

```
DECLARE @doc XML
SET @doc='
<Team name="Project 1" xmlns:a="urn:annotations">
  <Employee id="544" years="6.5">
```

```
    <Name>User 1</Name>
<Title>Architect</Title>
<Expertise>Games</Expertise>
<Expertise>Puzzles</Expertise>
<Employee id="101" years="7.1" a:assigned-to="C1">
 <Name>User 2</Name>
 <Title>Dev lead</Title>
 <Expertise>Video Games</Expertise>
 <Employee id="50" years="2.3" a:assigned-to="C2">
 <Name>User 3</Name>
 <Title>Developer</Title>
 <Expertise>Hardware</Expertise>
 <Expertise>Entertainment</Expertise>
</Employee>
</Employee>
  </Employee>
</Team>
'
```

در این سند، کارمند و کارمندانی را که باید به یک کارمند گزارش دهند، ملاحظه می‌کنید.
در XPath، محور پیش فرض، child است (اگر مانند کوئری زیر مورد خاصی ذکر نشود):

```
SELECT @doc.query('/Team/Employee/Name')
```

و اگر بخواهیم این محور را به صورت صریح ذکر کنیم، به نحو ذیل خواهد بود:

```
SELECT @doc.query('/Team/Employee/child::Name')
```

خروجی آن User1 است.

```
<Name>User 1</Name>
```

برای ذکر محور decedent-or-self می‌توان از // نیز استفاده کرد:

```
SELECT @doc.query('//Employee/Name')
```

با خروجی

```
<Name>User 1</Name>
<Name>User 2</Name>
<Name>User 3</Name>
```

در این حالت به تمام نودهای سند، در سطوح مختلف آن مراجعه شده و به دنبال نام کارمند خواهیم گشت.

برای کار با ویژگی‌ها و attributes از [] به همراه علامت @ استفاده می‌شود:

```
SELECT @doc.query('
declare namespace a = "urn:annotations";
//Employee[@a:assigned-to]/Name
')
```

در این کوئری، تمام کارمندانی که دارای ویژگی assigned-to واقع در فضای نام urn:annotations هستند، یافت خواهند شد. با خروجی:

```
<Name>User 2</Name>
<Name>User 3</Name>
```

معادل طولانی‌تر آن ذکر کامل محور attribute است بجای @

```
SELECT @doc.query('
declare namespace a = "urn:annotations";
//Employee[attribute::a:assigned-to]/Name
')
```

و برای یافتن کارمندانی که دارای ویژگی assigned-to نیستند، می‌توان از عملگر not استفاده کرد:

```
SELECT @doc.query('
declare namespace a = "urn:annotations";
//Employee[not(@a:assigned-to)]/Name
')
```

با خروجی

```
<Name>User 1</Name>
```

و اگر بخواهیم تعداد کارمندانی را که به user 1 مستقیما گزارش می‌دهند را بیابیم، می‌توان از count به نحو ذیل استفاده کرد:

```
SELECT @doc.query('count(//Employee[Name="User 1"]/Employee)')
```

در XPath برای یافتن والد از .. استفاده می‌شود:

```
SELECT @doc.query('//Employee[../Name="User 1"]')
```

برای مثال در کوئری فوق، کارمندانی که والد آن‌ها user 1 هستند، یافت می‌شوند.
استفاده از .. در SQL Server به دلایل کارآیی پایین توصیه نمی‌شود. بهتر است از همان روش قبلی کوئری تعداد کارمندانی که به user 1 مستقیما گزارش می‌دهند، استفاده شود.

عبارات FLOWR

FLOWR هسته‌ی XQuery را تشکیل داده و قابلیت توسعه XPath را دارد. FLOWR مخفف for، let، order by، where و retrun است.
از for برای تشکیل حلقه، از let برای انتساب، از where و order by برای فیلتر و مرتب سازی اطلاعات و از return برای بازگشت نتایج کمک گرفته می‌شود. FLOWR بسیار شبیه به ساختار SQL عمل می‌کند.
معادل عبارت SQL

```
Select p.name, p.job
from people as p
where p.age > 30
order by p.age
```

با عبارات FLOWR، به صورت زیر است:

```
for $p in /people/person
where $p.age > 30
order by $p.age[1]
return ($p/name, $p/job)
```

همانطور که مشاهده می‌کنید علت انتخاب FLOWR در اینجا عمدی بوده‌است؛ زیرا افرادی که SQL می‌دانند به سادگی می‌توانند شروع به کار با عبارات FLOWR کنند.
تنها تفاوت مهم، در اینجا است که در عبارات SQL، خروجی کار توسط select، در ابتدای کوئری ذکر می‌شود، اما در عبارات

FLOWR در انتهای آن‌ها.

از let برای انتساب مجموعه‌ای از نودها استفاده می‌شود:

```
let $p := /people/person
return $p
```

تفاوت آن با for در این است که در هر بار اجرای حلقه‌ی for، تنها با یک نود کار خواهد شد، اما در let با مجموعه‌ای از نودها سر و کار داریم. همچنین let از نگارش 2008 اس کیوال سرور به بعد قابل استفاده‌است.

یک نکته

اگر به order by دقت کنید، به اولین سن اشاره می‌کند. Order by در اینجا با تک مقدارها کار می‌کند و امکان کار با مجموعه‌ای از نودها را ندارد. به همین جهت باید طوری آن‌را تنظیم کرد که هربار فقط به یک مقدار اشاره کند.
هر زمانیکه به خطای requires a singleton برخوردید، یعنی دستورات مورد استفاده با یک سری از نودها کار نکرده و نیاز است دقیقا مشخص کنید، کدام مقدار مدنظر است.

مثال‌هایی از عبارات FLOWR

دو کوئری ذیل یک خروجی 1 2 3 را تولید می‌کنند

```
DECLARE @x XML = '';
SELECT @x.query('
for $i in (1,2,3)
return $i
');

SELECT @x.query('
let $i := (1,2,3)
return $i
');
```

در کوئری اول، هر بار که حلقه اجرا می‌شود، به یکی از اعضای توالی دسترسی خواهیم داشت. در کوئری دوم، یکبار توالی تعریف شده و کار با آن در یک مرحله صورت می‌گیرد.
در ادامه اگر سعی کنیم به این کوئری‌ها یک order by را اضافه کنیم، کوئری اول با موفقیت اجرا شده،

```
DECLARE @x XML = '';
SELECT @x.query('
for $i in (1,2,3)
order by $i descending
return $i
');

SELECT @x.query('
let $i := (1,2,3)
order by $i descending
return $i
');
```

اما کوئری دوم با خطای ذیل متوقف می‌شود:

```
XQuery [query()]: 'order by' requires a singleton (or empty sequence), found operand of type
'xs:integer +'
```

در خطا عنوان شده‌است که مطابق تعریف، order by با یک مجموعه از نودها، مانند حاصل let کار نمی‌کند و همانند حلقه for نیاز به singleton یا atomic values دارد.

ساخت المان‌های جدید XML توسط عبارات FLOWR

ابتدا همان سند XML قسمت قبل را درنظر بگیرید:

```
DECLARE @doc XML  ='
<people>
 <person>
  <name>
<givenName>name1</givenName>
<familyName>lname1</familyName>
  </name>
  <age>33</age>
  <height>short</height>
 </person>
 <person>
  <name>
<givenName>name2</givenName>
<familyName>lname2</familyName>
  </name>
  <age>40</age>
  <height>short</height>
 </person>
 <person>
  <name>
<givenName>name3</givenName>
<familyName>lname3</familyName>
  </name>
  <age>30</age>
  <height>medium</height>
 </person>
</people>
'
```

در ادامه قصد داریم، المان‌های اشخاص را صرفا بر اساس مقدار givenName آن‌ها بازگشت دهیم:

```
SELECT @doc.query('
for $p in /people/person
return <person>
{$p/name[1]/givenName[1]/text()}
</person>
');
```

در اینجا نحوه‌ی تولید پویای تگ‌های XML را توسط FLOWR مشاهده می‌کنید. عبارات داخل {} به صورت خودکار محاسبه و جایگزین می‌شوند و خروجی آن به شرح زیر است:

```
<person>name1</person>
<person>name2</person>
<person>name3</person>
```

سؤال: اگر به این خروجی بخواهیم یک root element اضافه کنیم، چه باید کرد؟ اگر المان root دلخواهی را در return قرار دهیم، به ازای هر آیتم یافت شده، یکبار تکرار می‌شود که مدنظر ما نیست.

```
SELECT @doc.query('
<root>
{
for $p in /people/person
return <person>
{$p/name[1]/givenName[1]/text()}
</person>
}
</root>
');
```

بله. در این حالت نیز می‌توان از همان روشی که در return استفاده کردیم، برای کل حلقه و return آن استفاده کنیم. المان root به صورت استاتیک محاسبه می‌شود و هر آنچه که داخل {} باشد، به صورت پویا. با این خروجی:

```
<root>
  <person>name1</person>
  <person>name2</person>
  <person>name3</person>
</root>
```

مفهوم quantification در FLOWR

همان سند Team name=Project 1 ابتدای بحث جاری را درنظر بگیرید.

```
SELECT @doc.query('some $emp in //Employee satisfies $emp/@years >5')
-- true

SELECT @doc.query('every $emp in //Employee satisfies $emp/@years >5')
-- false
```

به عبارات some و every در اینجا quantification گفته می‌شود. در کوئری اول، می‌خواهیم بررسی کنیم، آیا در بین کارمندان، بعضی از آن‌ها دارای ویژگی (با @ شروع شده) years بیشتر از 5 هستند. در کوئری دوم، عبارت «بعضی» به «هر» تغییر یافته است.

عنوان: **متدهای توکار استفاده از نوع داده‌ای XML - قسمت اول**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۸ ۱۳:۱۰
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, SQL Server, xml

در دو قسمت قبل، XQuery را به عنوان یک زبان برنامه نویسی استاندارد مورد بررسی قرار دادیم. در ادامه قصد داریم ترکیب آن‌را با توابع ویژه توکار SQL Server جهت کار با نوع داده‌ای XML، مانند modify، exists و امثال آن، تکمیل نمائیم. اگر بخاطر داشته باشید، 5 متد توکار جهت کار با نوع داده‌ای XML در SQL Server پیش بینی شده‌اند:

- xml : query را به عنوان ورودی گرفته و نهایتا یک خروجی XML دیگر را بر می‌گرداند.
- exist : خروجی bit دارد؛ true یا false. ورودی آن یک XQuery است.
- value : یک خروجی SQL Type را ارائه می‌دهد.
- nodes : خروجی جدولی دارد.
- modify : برای تغییر اطلاعات بکار می‌رود.

استفاده از متد exist به عنوان جایگزین سبک وزن XML Schema

یکی از کاربردهای متد exist، تعریف قید بر روی یک ستون XML ایی جدول است. این روش، راه حل دوم و ساده‌ای است بجای استفاده از XML Schema برای ارزیابی و اعتبارسنجی کل سند. پیشنیاز اینکار، تعریف قید مدنظر توسط یک تابع جدید است:

```
CREATE FUNCTION dbo.checkPerson(@data XML)
RETURNS BIT WITH SCHEMABINDING AS
BEGIN
   RETURN @data.exist('/people/person')
END
GO

CREATE TABLE tblXML
(
id INT PRIMARY KEY,
doc XML CHECK(dbo.checkPerson(doc)=1)
)
GO
```

متد checkPerson به دنبال وجود نود people/person، در ریشه‌ی سند XML در حال ذخیره شدن می‌گردد. پس از تعریف این متد، نحوه‌ی استفاده از آن‌را توسط عبارت check در حین تعریف ستون doc ملاحظه می‌کنید.

اکنون برای آزمایش آن خواهیم داشت:

```
INSERT INTO tblXML (id,  doc) VALUES
(
 1, '<people><person name="Vahid"/></people>'
)

INSERT INTO tblXML (id,  doc) VALUES
(
 2, '<people><emp name="Vahid"/></people>'
)
```

Insert اول با موفقیت انجام خواهد شد. اما Insert دوم با خطای ذیل متوقف می‌شود:

```
The INSERT statement conflicted with the CHECK constraint "CK__tblXML__doc__060DEAE8".
The conflict occurred in database "testdb", table "dbo.tblXML", column 'doc'.
The statement has been terminated.
```

همچنین باید در نظر داشت که امکان ترکیب یک XML Schema و تابع اعمال قید نیز با هم وجود دارند. برای مثال از XML Schema برای تعیین اعتبار ساختار کلی سند در حال ذخیره سازی استفاده می‌شود و همچنین نیاز است تا منطق تجاری خاصی را توسط یک

تابع، پیاده سازی کرده و در این بین اعمال نمود.

استفاده از متد value برای دریافت اطلاعات

با کاربرد مقدماتی متد value در بازگشت یک مقدار scalar در قسمت‌های قبل آشنا شدیم. در ادامه مثال‌های کاربردی‌تر را بررسی خواهیم کرد.
ابتدا جدول زیر را با یک ستون XML در آن درنظر بگیرید:

```
CREATE TABLE xml_tab
(
 id INT IDENTITY PRIMARY KEY,
 xml_col  XML
)
```

سپس چند ردیف را به آن اضافه می‌کنیم:

```
INSERT INTO xml_tab
VALUES ('<people><person name="Vahid"/></people>')
INSERT INTO xml_tab
VALUES ('<people><person name="Farid"/></people>')
```

در ادامه می‌خواهیم id و نام اشخاص ذخیره شده در جدول را بازیابی کنیم:

```
SELECT
   id,
   xml_col.value('(/people/person/@name)[1]', 'varchar(50)') AS name
FROM
xml_tab
```

متد vlaue یک XPath را دریافت کرده، به همراه نوع آن و صفر یا یک نود را بازگشت خواهد داد. به همین جهت، با توجه به عدم تعریف اسکیما برای سند XML در حال ذخیره شدن، نیاز است اولین نود را صریحا مشخص کنیم.

یک نکته
اگر نیاز به خروجی از نوع XML است، بهتر است از متد query که در دو قسمت قبل بررسی شد، استفاده گردد. خروجی متد query همیشه یک untyped XML است یا نال. البته می‌توان خروجی آن‌را به یک typed XML دارای Schema نیز نسبت داد. در اینجا اعتبارسنجی در حین انتساب صورت خواهد گرفت.

استفاده از متد value برای تعریف قیود

از متد value همچنین می‌توان برای تعریف قیود پیشرفته نیز استفاده کرد. برای مثال فرض کنیم می‌خواهیم ویژگی Id سند XML در حال ذخیره شدن، حتما مساوی ستون Id جدول باشد. برای این منظور ابتدا نیاز است همانند قبل یک تابع جدید را ایجاد نمائیم:

```
CREATE FUNCTION getIdValue(@doc XML)
RETURNS int WITH SCHEMABINDING AS
BEGIN
  RETURN @doc.value('/*[1]/@Id', 'int')
END
```

این تابع یک int را باز می‌گرداند که حاصل مقدار ویژگی Id اولین نود ذیل ریشه است. اگر این نود، ویژگی Id نداشته باشد، null بر می‌گرداند.
سپس از این تابع در عبارت check برای مقایسه ویژگی Id سند XML در حال ذخیره شدن و id ردیف جاری استفاده می‌شود:

```
CREATE TABLE docs_tab
(
```

```
id INT PRIMARY KEY,
doc XML,
CONSTRAINT id_chk CHECK(dbo.getIdValue(doc)=id)
)
```

نحوه‌ی تعریف آن اینبار توسط عبارت CONSTRAINT است؛ زیرا در سطح جدول باید عمل کند (ارجاعی را به یک فیلد آن دارد) و نه در سطح یک فیلد؛ مانند مثال ابتدای بحث جاری.
در ادامه برای آزمایش آن خواهیم داشت:

```
INSERT INTO docs_tab (id,  doc) VALUES
(
 1, '<Invoice Id="1"/>'
)

INSERT INTO docs_tab (id,  doc) VALUES
(
 2, '<Invoice Id="1"/>'
)
```

Insert اول با توجه به یکی بودن مقدار ویژگی Id آن با id ردیف، با موفقیت ثبت می‌شود. ولی رکورد دوم خیر:

```
The INSERT statement conflicted with the CHECK constraint "id_chk".
The conflict occurred in database "testdb", table "dbo.docs_tab".
The statement has been terminated.
```

استفاده از متد value برای تعریف primary key

پیشتر عنوان شد که از فیلدهای XML نمی‌توان به عنوان کلید یک جدول استفاده کرد؛ چون امکان مقایسه‌ی محتوای کل آن‌ها وجود ندارد. اما با استفاده از متد value می‌توان مقدار دریافتی را به عنوان یک کلید اصلی محاسبه شده، ثبت کرد:

```
CREATE TABLE Invoices
(
 doc XML,
 id AS dbo.getIdValue(doc) PERSISTED PRIMARY KEY
)
```

Id در اینجا یک computed column است. همچنین باید به صورت PERSISTED علامتگذاری شود تا سپس به عنوان PRIMARY KEY قابل استفاده باشد.
برای آزمایش آن سعی می‌کنیم دو رکورد را که حاوی ویژگی id برابری هستند، ثبت کنیم:

```
INSERT INTO Invoices VALUES
(
 '<Invoice Id="1"/>'
)
INSERT INTO Invoices VALUES
(
 '<Invoice Id="1"/>'
)
```

مورد اول با موفقیت ثبت می‌شود. مورد دوم خیر:

```
Violation of PRIMARY KEY constraint 'PK__Invoices__3213E83F145C0A3F'.
Cannot insert duplicate key in object 'dbo.Invoices'. The duplicate key value is (1).
The statement has been terminated.
```

توابع دسترسی به مقدار داده‌ها در XQuery

تابع string ، data و text برای دسترسی به مقدار داده‌ها در XQuery پیش بینی شده‌اند.
اگر سعی کنیم مثال زیر را اجرا نمائیم:

```
DECLARE @doc XML
SET @doc = '<foo bar="baz" />'
SELECT @doc.query('/foo/@bar')
```

با خطای ذیل متوقف خواهیم شد:

```
XQuery [query()]: Attribute may not appear outside of an element
```

علت اینجا است که خروجی query از نوع XML است و ما در XPath نوشته شده درخواست بازگشت مقدار یک ویژگی را کرده‌ایم که نمی‌تواند به عنوان ریشه یک سند XML بازگشت داده شود. برای بازگشت مقدار ویژگی bar که baz است باید از متد data استفاده کرد:

```
DECLARE @doc XML
SET @doc = '<foo bar="baz" />'
SELECT @doc.query('data(/foo/@bar)')
```

متد data می‌تواند بیش از یک مقدار را در یک توالی بازگشت دهد:

```
DECLARE @x XML
SET @x = '<x>hello<y>world</y></x><x>again</x>'
SELECT @x.query('data(/*)')
```

در اینجا توسط متد data درخواست بازگشت کلیه root elementهای سند XML را کرده‌ایم. خروجی آن helloworld again خواهد بود.
اما اگر همین مثال را با متد string اجرا کنیم:

```
DECLARE @x XML
SET @x = '<x>hello<y>world</y></x><x>again</x>'
SELECT @x.query('string(/*)')
```

به خطای آشنای ذیل برخواهیم خورد:

```
XQuery [query()]: 'string()' requires a singleton (or empty sequence), found operand of type
'element(*,xdt:untyped) *'
```

در اینجا چون تابع string باید بیش از یک نود را پردازش کند، خطایی را صادر کرده‌است. برای رفع آن باید دقیقا مشخص کنیم که برای مثال تنها اولین عضو توالی را بازگشت بده:

```
SELECT @x.query('string(/*[1])')
```

خروجی آن helloworld است.
برای دریافت تمام کلمات توسط متد string می‌توان از اسلش کمک گرفت:

```
SELECT @x.query('string(/)')
```

با خروجی helloworldagain که تنها یک string value محسوب می‌شود؛ برخلاف حالت استفاده از متد data که دو مقدار یک توالی را بازگشت داده است.

نمونه‌ی دیگر آن مثال زیر است:

```
DECLARE @x XML = '<age>12</age>'
SELECT @x.query('string(/age[1])')
```

در اینجا نیز باید حتما اولین المان، صراحتا مشخص شود. هرچند به نظر این سند untyped XML تنها یک المان دارد، اما XQuery ذکر شده پیش از اجرای آن، تعیین اعتبار می‌شود. برای عدم ذکر اولین آیتم (در صورت نیاز)، باید XML Schema سند مرتبط، تعریف و در حین تعریف و انتساب مقدار آن، مشخص گردد. همچنین در اینجا به مباحث content و document که در قسمت‌های پیشین نیز ذکر شد باید دقت داشت. حالت پیش فرض content است و می‌تواند بیش از یک root element داشته باشد.

متد text اندکی متفاوت عمل می‌کند. برای بررسی آن، ابتدا یک schema collection جدید را تعریف می‌کنیم که داری تک المانی رشته‌ای است به نام Root.

```
CREATE XML SCHEMA COLLECTION root_el AS
'<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  targetNamespace="urn:geo">
      <xs:element name="Root" type="xs:string" />
</xs:schema>
'
GO
```

در ادامه اگر متد text را بر روی یک untyped XML که SChema آن مشخص نشده‌است، فراخوانی کنیم:

```
DECLARE @xmlDoc XML
SET @xmlDoc = '<g:Root xmlns:g="urn:geo">datadata...</g:Root>'
SELECT @xmlDoc.query('
declare namespace g="urn:geo";
/g:Root/text()
')
```

مقدار datadata... این المان Root را بازگشت خواهد داد. اینبار اگر untyped XML را با تعریف schema آن تبدیل به typed XML کنیم:

```
DECLARE @xmlDoc XML(root_el)
SET @xmlDoc = '<g:Root xmlns:g="urn:geo">datadata...</g:Root>'
SELECT @xmlDoc.query('
declare namespace g="urn:geo";
/g:Root[1]/text()
')
```

به خطای ذیل برخواهیم خورد:

```
XQuery [query()]: 'text()' is not supported on simple typed or
'http://www.w3.org/2001/XMLSchema#anyType'
elements, found 'element(g{urn:geo}:Root,xs:string) *'.
```

زمانیکه از Schema استفاده می‌شود، دیگر نیازی به استفاده از متد text نیست. فقط کافی است متد text را حذف کرده و بجای آن از متد data استفاده کنیم:

```
DECLARE @xmlDoc XML(root_el)
SET @xmlDoc = '<g:Root xmlns:g="urn:geo">datadata...</g:Root>'
SELECT @xmlDoc.query('
declare namespace g="urn:geo";
data(/g:Root[1])
')
```

به علاوه، در خطا ذکر شده‌است که متد text را بر روی simple types نمی‌توان بکار برد. این محدودیت در مورد complex types

که نمونه‌ای از آن‌را در قسمت معرفی Schema با تعریف Point مشاهده کردید، وجود ندارد. اما متد data قابل استفاده بر روی complex types نیست. ولی می‌توان متد data و text را با هم ترکیب کرد؛ برای مثال

```
data(/age/text())
```

اگر complex node را untyped تعریف کنیم (schema را قید نکنیم)، استفاده از متد data در اینجا نیز وجود خواهد داشت.

عنوان: **متدهای توکار استفاده از نوع داده‌ای XML - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۷:۲۷ ۱۳۹۲/۱۱/۲۹
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, SQL Server, xml

امکان ترکیب داده‌های یک بانک اطلاعاتی رابطه‌ای و XML در SQL Server به کمک یک سری تابع کمکی خاص به نام‌های sql:variable و sql:column پیش بینی شده‌است. sql:variable امکان استفاده از یک متغیر T-SQL را داخل یک XQuery میسر می‌سازد و توسط sql:column می‌توان با یکی از ستون‌های ذکر شده در قسمت select، داخل XQuery کار کرد. در ادامه به مثال‌هایی در این مورد خواهیم پرداخت.

ابتدا جدول xmlTest را به همراه چند رکورد ثبت شده در آن، درنظر بگیرید:

```
CREATE TABLE xmlTest
(
 id INT IDENTITY PRIMARY KEY,
 doc XML
)
GO
INSERT xmlTest VALUES('<Person name="Vahid" />')
INSERT xmlTest VALUES('<Person name="Farid" />')
INSERT xmlTest VALUES('<Person name="Mehdi" /><Person name="Hamid" />')
GO
```

استفاده از متد sql:column

در ادامه می‌خواهیم مقدار ویژگی name رکوردی را که نام آن Vahid است، به همراه id آن ردیف، توسط یک XQuery بازگشت دهیم:

```
SELECT doc.query('
for $p in //Person
where $p/@name="Vahid"
return <li>{data($p/@name)} has id = {sql:column("xmlTest.id")}</li>
')
FROM xmlTest
```

یک sql:column حتما نیاز به یک نام ستون دو قسمتی دارد. قسمت اول آن نام جدول است و قسمت دوم، نام ستون مورد نظر. در مورد متد data در قسمت قبل بیشتر بحث شد و از آن برای استخراج داده‌ی یک ویژگی در اینجا استفاده شده‌است. عبارات داخل {} نیز پویا بوده و به همراه سایر قسمت‌های ثابت return، ابتدا محاسبه و سپس بازگشت داده می‌شود.
اگر این کوئری را اجرا کنید، ردیف اول آن مساوی عبارت زیر خواهد بود

```
<li>Vahid has id = 1</li>
```

به همراه دو ردیف خالی دیگر در ادامه. این ردیف‌های خالی به علت وجود دو رکورد دیگری است که با شرط where یاد شده تطابق ندارند.
یک روش برای حذف این ردیف‌های خالی استفاده از متد exist است به شکل زیر:

```
SELECT doc.query('
for $p in //Person
where $p/@name="Vahid"
return <li>{data($p/@name)} has id = {sql:column("xmlTest.id")}</li>
')
FROM xmlTest
WHERE doc.exist('
for $p in //Person
where $p/@name="Vahid"
return <li>{data($p/@name)} has id = {sql:column("xmlTest.id")}</li>
')=1
```

در اینجا فقط ردیفی انتخاب خواهد شد که نام ویژگی آن Vahid است.
روش دوم استفاده از یک derived table و بازگشت ردیف‌های غیرخالی است:

```
SELECT * FROM
(
 (SELECT doc.query('
 for $p in //Person
 where $p/@name="Vahid"
 return <li>{data($p/@name)} has id = {sql:column("xmlTest.id")}</li>
 ') AS col1
 FROM xmlTest)
) A
WHERE CONVERT(VARCHAR(8000), col1)<>''
```

استفاده از متد sql:variable

```
DECLARE @number INT = 1
SELECT doc.query('
for $p in //Person
where $p/@name="Vahid"
return <li>{data($p/@name)} has number = {sql:variable("@number")}</li>
')
FROM xmlTest
```

در این مثال نحوه‌ی بکارگیری یک متغیر T-SQL را داخل یک XQuery توسط متد sql:variable ملاحظه می‌کنید.

استفاده از For XML برای دریافت یکباره‌ی تمام ردیف‌های XML

اگر کوئری معمولی ذیل را اجرا کنیم:

```
SELECT doc.query('/Person') FROM xmlTest
```

سه ردیف خروجی را مطابق سه رکوردی که ثبت کردیم، بازگشت می‌دهد.
اما اگر بخواهیم این سه ردیف را با هم ترکیب کرده و تبدیل به یک نتیجه‌ی واحد کنیم، می‌توان از For XML به نحو ذیل استفاده کرد:

```
DECLARE @doc XML
SET @doc = (SELECT * FROM xmlTest FOR XML AUTO, ELEMENTS)
SELECT @doc.query('/xmlTest/doc/Person')
```

بررسی متد xml.nodes

متد xml.nodes اندکی متفاوت است نسبت به تمام متدهایی که تاکنون بررسی کردیم. کار آن تجزیه‌ی محتوای XML ایی به ستون‌ها و سطرها می‌باشد. بسیار شبیه است به متد OpenXML اما کارآیی بهتری دارد.

```
DECLARE @doc XML ='
<people>
  <person><name>Vahid</name></person>
  <person><name id="2">Farid</name></person>
  <person><name>Mehdi</name></person>
  <person><name>Hooshang</name><name id="1">Hooshi</name></person>
  <person></person>
</people>
'
```

در اینجا یک سند XML را درنظر بگیرید که از چندین نود شخص تشکیل شده‌است. اغلب آن‌ها دارای یک name هستند. چهارمین نود، دو نام دارد و آخری بدون نام است.
در ادامه قصد داریم این اطلاعات را تبدیل به ردیف‌هایی کنیم که هر ردیف حاوی یک نام است. اولین سعی احتمالا استفاده از متد value خواهد بود:

```
SELECT @doc.value('/people/person/name', 'varchar(50)')
```

این روش کار نمی‌کند زیرا متد value، بیش از یک مقدار را نمی‌تواند بازگشت دهد. البته می‌توان از متد value به نحو زیر استفاده کرد:

```
SELECT @doc.value('(/people/person/name)[1]', 'varchar(50)')
```

اما حاصل آن دقیقا چیزی نیست که دنبالش هستیم؛ ما دقیقا نیاز به تمام نام‌ها داریم و نه تنها یکی از آن‌ها را.
سعی بعدی استفاده از متد query است:

```
SELECT @doc.query('/people/person/name')
```

در این حالت تمام نام‌ها را بدست می‌آوریم:

```
<name>Vahid</name>
<name id="2">Farid</name>
<name>Mehdi</name>
<name>Hooshang</name>
<name id="1">Hooshi</name>
```

اما این حاصل دو مشکل را به همراه دارد:
الف) خروجی آن XML است.
ب) تمام این‌ها در طی یک ردیف و یک ستون بازگشت داده می‌شوند.

و این خروجی نیز چیزی نیست که برای ما مفید باشد. ما به ازای هر شخص نیاز به یک ردیف جداگانه داریم. اینجا است که متد xml.nodes مفید واقع می‌شود:

```
SELECT
tab.col.value('text()[1]', 'varchar(50)') AS name,
tab.col.query('.'),
tab.col.query('..')
from @doc.nodes('/people/person/name') AS tab(col)
```

خروجی متد xml.nodes یک table valued function است؛ یک جدول را باز می‌گرداند که دقیقا حاوی یک ستون می‌باشد. به همین جهت Alias آن‌را با tab col مشخص کرده‌ایم. tab متناظر است با جدول بازگشت داده شده و col متناظر است با تک ستون این جدول حاصل. این نام‌ها در اینجا مهم نیستند؛ اما ذکر آن‌ها اجباری است.
هر ردیف حاصل از این جدول بازگشت داده شده، یک اشاره‌گر است. به همین جهت نمی‌توان آن‌ها را مستقیما نمایش داد. هر سطر آن، به نودی که با آن مطابق XQuery وارد شده تطابق داشته است، اشاره می‌کند. در اینجا مطابق کوئری نوشته شده، هر ردیف به یک نود name اشاره می‌کند. در ادامه برای استخراج اطلاعات آن می‌توان از متد text استفاده کرد.
اگر قصد داشتید، اطلاعات کامل نود ردیف جاری را مشاهده کنید می‌توان از

```
tab.col.query('.'),
```

استفاده کرد. دات در اینجا به معنای self است. دو دات (نقطه) پشت سرهم به معنای درخواست اطلاعات والد نود می‌باشد.
روش دیگر بدست آوردن مقدار یک نود را در کوئری ذیل مشاهده می‌کنید؛ value دات و data دات. خروجی value مقدار آن نود

است و خروجی data مقدار آن نود با فرمت XML.

```
SELECT
tab.col.value('.', 'varchar(50)') AS name,
tab.col.query('data(.)'),
tab.col.query('.'),
tab.col.query('..')
from @doc.nodes('/people/person/name') AS tab(col)
```

همچنین اگر بخواهیم اطلاعات تنها یک نود خاص را بدست بیاوریم، می‌توان مانند کوئری ذیل عمل کرد:

```
SELECT
tab.col.value('name[.="Farid"][1]', 'varchar(50)') AS name,
tab.col.value('name[.="Farid"][1]/@id', 'varchar(50)') AS id,
tab.col.query('.')
from @doc.nodes('/people/person[name="Farid"]') AS tab(col)
```

در مورد کار با جداول، بجای متغیرهای T-SQL نیز روال کار به همین نحو است:

```
DECLARE @tblXML TABLE (
 id INT IDENTITY PRIMARY KEY,
 doc XML
 )

INSERT @tblXML VALUES('<person name="Vahid" />')
INSERT @tblXML VALUES('<person name="Farid" />')
INSERT @tblXML VALUES('<person />')
INSERT @tblXML VALUES(NULL)

SELECT
id,
doc.value('(/person/@name)[1]', 'varchar(50)') AS name
FROM @tblXML
```

در اینجا یک جدول حاوی ستون XML ایی ایجاد شده‌است. سپس چهار ردیف در آن ثبت شده‌اند. در آخر مقدار ویژگی نام این ردیف‌ها بازگشت داده شده‌است.

نکته : استفاده‌ی وسیع SQL Server از XML برای پردازش کارهای درونی آن

بسیاری از ابزارهایی که در نگارش‌های جدید SQL Server اضافه شده‌اند و یا مورد استفاده قرار می‌گیرند، استفاده‌ی وسیعی از امکانات توکار XML آن دارند. مانند:
Showplan، گراف‌های dead lock، گزارش پروسه‌های بلاک شده، اطلاعات رخدادها، SSIS Jobs، رخدادهای Trace و ...

مثال اول: کدام کوئری‌ها در Plan cache، کارآیی پایینی داشته و table scan را انجام می‌دهند؟

```
CREATE PROCEDURE LookForPhysicalOps (@op VARCHAR(30))
AS
SELECT sql.text, qs.EXECUTION_COUNT, qs.*, p.*
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(sql_handle) sql
CROSS APPLY sys.dm_exec_query_plan(plan_handle) p
WHERE query_plan.exist('
declare default element namespace "http://schemas.microsoft.com/sqlserver/2004/07/showplan";
/ShowPlanXML/BatchSequence/Batch/Statements//RelOp/@PhysicalOp[. = sql:variable("@op")]
') = 1
GO

EXECUTE LookForPhysicalOps 'Table Scan'
EXECUTE LookForPhysicalOps 'Clustered Index Scan'
EXECUTE LookForPhysicalOps 'Hash Match'
```

اطلاعات Query Plan در SQL Server با فرمت XML ارائه می‌شود. در اینجا می‌خواهیم یک سری متغیر مانند Clustered Index Scan و امثال آن‌را از ویژگی PhysicalOp آن کوئری بگیریم. بنابراین از متد sql:variable کمک گرفته شده‌است.
اگر علاقمند هستید که اصل این اطلاعات را با فرمت XML مشاهده کنید، کوئری نوشته شده را تا پیش از where آن یکبار مستقلا اجرا کنید. ستون آخر آن query_plan نام دارد و حاوی اطلاعات XML ایی است.

مثال دوم: استخراج اپراتورهای رابطه‌ای (RelOp) از یک Query Plan ذخیره شده

```
WITH XMLNAMESPACES(DEFAULT N'http://schemas.microsoft.com/sqlserver/2004/07/showplan')
SELECT RelOp.op.value(N'../../@NodeId', N'int') AS ParentOperationID,
RelOp.op.value(N'@NodeId', N'int') AS OperationID,
RelOp.op.value(N'@PhysicalOp', N'varchar(50)') AS PhysicalOperator,
RelOp.op.value(N'@LogicalOp', N'varchar(50)') AS LogicalOperator,
RelOp.op.value(N'@EstimatedTotalSubtreeCost ', N'float') AS EstimatedCost,
RelOp.op.value(N'@EstimateIO', N'float') AS EstimatedIO,
RelOp.op.value(N'@EstimateCPU', N'float') AS EstimatedCPU,
RelOp.op.value(N'@EstimateRows', N'float') AS EstimatedRows,
cp.plan_handle AS PlanHandle,
st.TEXT AS QueryText,
qp.query_plan AS QueryPlan,
cp.cacheobjtype AS CacheObjectType,
cp.objtype AS ObjectType
FROM sys.dm_exec_cached_plans cp
CROSS APPLY sys.dm_exec_sql_text(cp.plan_handle) st
CROSS APPLY sys.dm_exec_query_plan(cp.plan_handle) qp
CROSS APPLY qp.query_plan.nodes(N'//RelOp') RelOp(op)
```

در اینجا کار کردن با WITH XMLNAMESPACES در حین استفاده از متد xml.nodes ساده‌تر است؛ بجای قرار دادن فضای نام در تمام کوئری‌های نوشته شده.

بررسی متد xml.modify

تا اینجا تمام کارهایی که صورت گرفت و نکاتی که بررسی شدند، به مباحث select اختصاص داشتند. اما insert، delete و یا update قسمتی از یک سند XML بررسی نشدند. برای این منظور باید از متد xml.modify استفاده کرد. از آن در عبارات update و یا set کمک گرفته شده و ورودی آن نباید نال باشد. در ادامه در طی مثال‌هایی این موارد را بررسی خواهیم کرد.
ابتدا فرض کنید که سند XML ما چنین شکلی را دارا است:

```
DECLARE @doc XML = '
<Invoice>
<InvoiceId>100</InvoiceId>
<CustomerName>Vahid</CustomerName>
<LineItems>
<LineItem>
<Sku>134</Sku>
<Quantity>10</Quantity>
<Description>Item 1</Description>
<UnitPrice>9.5</UnitPrice>
</LineItem>
<LineItem>
<Sku>150</Sku>
<Quantity>5</Quantity>
<Description>Item 2</Description>
<UnitPrice>1.5</UnitPrice>
</LineItem>
</LineItems>
</Invoice>
'
```

در ادامه قصد داریم یک نود جدید را پس از CustomerName اضافه کنیم.

```
SET @doc.modify('
insert <InvoiceInfo><InvoiceDate>2014-02-10</InvoiceDate></InvoiceInfo>
```

```
after /Invoice[1]/CustomerName[1]
')

SELECT @doc
```

اینکار را با استفاده از دستور insert، به نحو فوق می‌توان انجام داد. از عبارت Set و متغیر doc مقدار دهی شده، کار شروع شده و سپس نود جدیدی پس از (after) اولین نود CustomerName موجود insert می‌شود. Select بعدی نتیجه را نمایش خواهد داد.

```
<Invoice>
  <InvoiceId>100</InvoiceId>
  <CustomerName>Vahid</CustomerName>
  <InvoiceInfo>
        <InvoiceDate>2014-02-10</InvoiceDate>
  </InvoiceInfo>
  <LineItems>
...
```

در SQL Server 2008 به بعد، امکان استفاده از متغیرهای T-SQL نیز در اینجا مجاز شده‌است:

```
SET @x.modify('insert sql:variable("@x") into /doc[1]')
```

بنابراین اگر نیاز به تعریف متغیری در اینجا داشتید از جمع زدن رشته‌ها استفاده نکنید. حتما نیاز است متغیر تعریف شود و گرنه باخطای ذیل متوقف خواهید شد:

```
The argument 1 of the XML data type method "modify" must be a string literal.
```

افزودن ویژگی‌های جدید به یک سند XML توسط متد xml.modify

اگر بخواهیم یک ویژگی (attribute) جدید را به نود خاصی اضافه کنیم می‌توان به نحو ذیل عمل کرد:

```
SET @doc.modify('
insert attribute status{"backorder"}
into /Invoice[1]
')

SELECT @doc
```

که خروجی دو سطر ابتدایی آن پس از اضافه شدن ویژگی status با مقدار backorder به نحو ذیل است:

```
<Invoice status="backorder">
  <InvoiceId>100</InvoiceId>
....
```

حذف نودهای یک سند XML توسط متد xml.modify

اگر بخواهیم تمام LineItemها را حذف کنیم می‌توان نوشت:

```
SET @doc.modify('delete /Invoice/LineItems/LineItem')
SELECT @doc
```

با این خروجی:

```
<Invoice status="backorder">
```

```
  <InvoiceId>100</InvoiceId>
  <CustomerName>Vahid</CustomerName>
  <InvoiceInfo>
      <InvoiceDate>2014-02-10</InvoiceDate>
  </InvoiceInfo>
  <LineItems />
</Invoice>
```

به روز رسانی نودهای یک سند XML توسط متد xml.modify

اگر نیاز باشد تا مقدار یک نود را تغییر دهیم می‌توان از replace value of استفاده کرد:

```
SET @doc.modify('replace value of
  /Invoice[1]/CustomerName[1]/text()[1]
  with "Farid"
')
SELECT @doc
```

با خروجی ذیل که در آن نام اولین مشتری با مقدار Farid جایگزین شده است:

```
<Invoice status="backorder">
  <InvoiceId>100</InvoiceId>
  <CustomerName>Farid</CustomerName>
  <InvoiceInfo>
       <InvoiceDate>2014-02-10</InvoiceDate>
  </InvoiceInfo>
  <LineItems />
</Invoice>
```

replace value of فقط با یک نود کار می‌کند و همچنین، فقط مقدار آن نود را تغییر می‌دهد. به همین جهت از متد text استفاده شده‌است. اگر از text استفاده نشود با خطای ذیل متوقف خواهیم شد:

```
The target of 'replace value of' must be a non-metadata attribute or an element with simple typed
content.
```

به روز رسانی نودهای خالی توسط متد xml.modify

باید دقت داشت، نودهای خالی (بدون مقدار)، مانند LineItems پس از delete کلیه اعضای آن در مثال قبل، قابل replace نیستند و باید مقادیر جدید را در آن‌ها insert کرد. یک مثال:

```
DECLARE @tblTest AS TABLE (xmlField XML)

INSERT INTO @tblTest(xmlField)
VALUES
 (
'<Sample>
   <Node1>Value1</Node1>
   <Node2>Value2</Node2>
   <Node3/>
</Sample>'
)

DECLARE @newValue VARCHAR(50) = 'NewValue'

UPDATE @tblTest
SET xmlField.modify(
'insert text{sql:variable("@newValue")} into
  (/Sample/Node3)[1] [not(text())]'
)
```

```
SELECT xmlField.value('(/Sample/Node3)[1]','varchar(50)') FROM @tblTest
```

در این مثال اگر از replace value of برای مقدار دهی نود سوم استفاده می‌شد:

```
UPDATE @tblTest
SET xmlField.modify(
'replace value of (/Sample/Node3/text())[1]
  with sql:variable("@newValue")'
)
```

تغییری را پس از اعمال دستورات مشاهده نمی‌کردید؛ زیرا این المان ()text ایی را برای replace شدن ندارد.

عنوان: **بررسی کارآیی و ایندکس گذاری بر روی اسناد XML در SQL Server - قسمت اول**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۱ ۱:۲۹
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, SQL Server, xml

در ادامه‌ی مباحث پشتیبانی از XML در SQL Server، به کارآیی فیلدهای XML ایی و نحوه‌ی ایندکس گذاری بر روی آن‌ها خواهیم پرداخت. این مساله در تولید برنامه‌هایی سریع و مقیاس پذیر، بسیار حائز اهمیت است.
در SQL Server، کوئری‌های انجام شده بر روی فیلدهای XML، توسط همان پردازشگر کوئری‌های رابطه‌ای متداول آن، خوانده و اجرا خواهند شد و امکان تعریف یک XQuery خارج از یک عبارت SQL و یا T-SQL وجود ندارد. متدهای XQuery بسیار شبیه به system defined functions بوده و Query Plan یکپارچه‌ای را با سایر قسمت‌های رابطه‌ای یک عبارت SQL دارند.

مفهوم Node table

داده‌های XML ایی برای اینکه توسط SQL Server قابل استفاده باشند، به صورت درونی تبدیل به یک node table می‌شوند. به این معنا که نودهای یک سند XML، به یک جدول رابطه‌ای به صورت خودکار تجزیه می‌شوند. این جدول درونی در صورت بکارگیری XML Indexes در جدول سیستمی sys.internal_tables قابل مشاهده خواهد بود. SQL Server برای انجام اینکار از یک XmlReader خاص خودش استفاده می‌کند. در مورد XMLهای ایندکس نشده، این تجزیه در زمان اجرا صورت می‌گیرد؛ پس از اینکه Query Plan آن تشکیل شد.

بررسی Query Plan فیلدهای XML ایی

جهت فراهم کردن مقدمات آزمایش، ابتدا جدول xmlInvoice را با یک فیلد XML ایی untyped درنظر بگیرید:

```
CREATE TABLE xmlInvoice
(
 invoiceId INT IDENTITY PRIMARY KEY,
 invoice XML
)
```

سپس 6 ردیف را به آن اضافه می‌کنیم:

```
INSERT INTO xmlInvoice
VALUES('
<Invoice InvoiceId="1000" dept="hardware">
<CustomerName>Vahid</CustomerName>
<LineItems>
<LineItem><Description>Gear</Description><Price>9.5</Price></LineItem>
</LineItems>
</Invoice>
 ')

INSERT INTO xmlInvoice
VALUES('
<Invoice InvoiceId="1002" dept="garden">
<CustomerName>Mehdi</CustomerName>
<LineItems>
<LineItem><Description>Shovel</Description><Price>19.2</Price></LineItem>
</LineItems>
</Invoice>
 ')

INSERT INTO xmlInvoice
VALUES('
<Invoice InvoiceId="1003" dept="garden">
<CustomerName>Mohsen</CustomerName>
<LineItems>
<LineItem><Description>Trellis</Description><Price>8.5</Price></LineItem>
</LineItems>
</Invoice>
 ')

INSERT INTO xmlInvoice
```

```
VALUES('
<Invoice InvoiceId="1004" dept="hardware">
<CustomerName>Hamid</CustomerName>
<LineItems>
<LineItem><Description>Pen</Description><Price>1.5</Price></LineItem>
</LineItems>
</Invoice>
 ')

INSERT INTO xmlInvoice
VALUES('
<Invoice InvoiceId="1005" dept="IT">
<CustomerName>Ali</CustomerName>
<LineItems>
<LineItem><Description>Book</Description><Price>3.2</Price></LineItem>
</LineItems>
</Invoice>
 ')

INSERT INTO xmlInvoice
VALUES('
<Invoice InvoiceId="1006" dept="hardware">
<CustomerName>Reza</CustomerName>
<LineItems>
<LineItem><Description>M.Board</Description><Price>19.5</Price></LineItem>
</LineItems>
</Invoice>
 ')
```

همچنین برای مقایسه، دقیقا جدول مشابهی را اینبار با یک XML Schema مشخص ایجاد می‌کنیم.

```
CREATE XML SCHEMA COLLECTION invoice_xsd AS
 ' <xs:schema attributeFormDefault="unqualified"
 elementFormDefault="qualified"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Invoice">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CustomerName" type="xs:string" />
        <xs:element name="LineItems">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="LineItem">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Description" type="xs:string" />
                    <xs:element name="Price" type="xs:decimal" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="InvoiceId" type="xs:unsignedShort" use="required" />
      <xs:attribute name="dept" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>'

Go

CREATE TABLE xmlInvoice2
(
invoiceId INT IDENTITY PRIMARY KEY,
invoice XML(document invoice_xsd)
)
Go
```

سپس مجددا همان 6 رکورد قبلی را در این جدول جدید نیز insert خواهیم کرد.

در این جدول دوم، حالت پیش فرض content قبلی، به document تغییر کرده‌است. با توجه به اینکه می‌دانیم اسناد ما چه فرمتی دارند و بیش از یک root element نخواهیم داشت، انتخاب document سبب خواهد شد تا Query Plan بهتری حاصل شود.

در ادامه برای مشاهده‌ی بهتر نتایج، کش Query Plan و اطلاعات آماری جدول xmlInvoice را حذف و به روز می‌کنیم:

```
UPDATE STATISTICS xmlInvoice
DBCC FREEPROCCACHE
```

به علاوه در management studio بهتر است از منوی Query، گزینه‌ی Include actual execution plan را نیز انتخاب کنید (یا فشردن دکمه‌های Ctrl+M) تا پس از اجرای کوئری، بتوان Query Plan نهایی را نیز مشاهده نمود. برای خواندن یک Query Plan عموما از بالا به پایین و از راست به چپ باید عمل کرد. در آن نهایتا باید به عدد estimated subtree cost کوئری، دقت داشت.

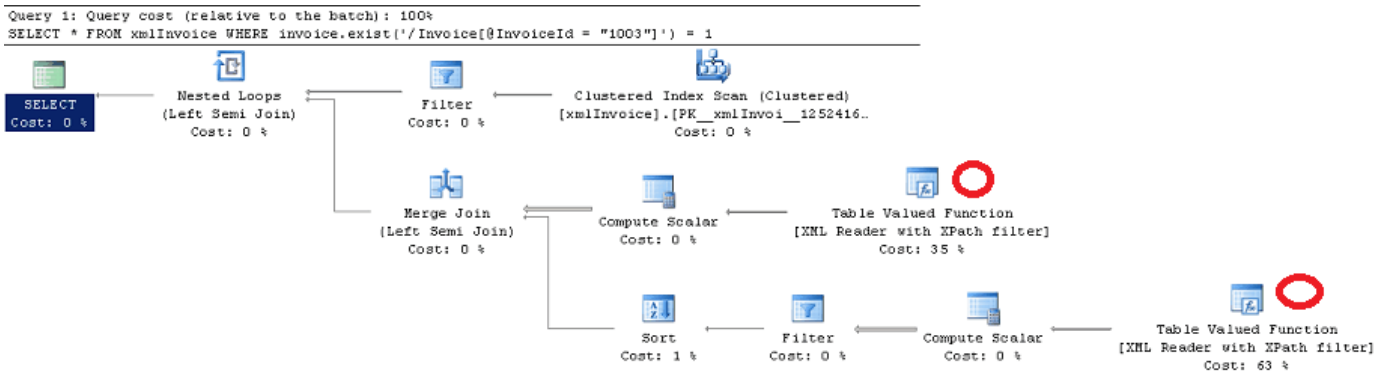
کوئری‌هایی را که در این قسمت بررسی خواهیم کرد، در ادامه ملاحظه می‌کنید. بار اول این کوئری‌ها را بر روی xmlInvoice و بار دوم، بر روی نگارش دوم دارای اسکیمای آن اجرا خواهیم کرد:

```
-- query 1
SELECT * FROM xmlInvoice
WHERE invoice.exist('/Invoice[@InvoiceId = "1003"]') = 1

-- query 2
SELECT * FROM xmlInvoice
WHERE invoice.exist('/Invoice/@InvoiceId[. = "1003"]') = 1

-- query 3
SELECT * FROM xmlInvoice
WHERE invoice.exist('/Invoice[1]/@InvoiceId[. = "1003"]') = 1

-- query 4
SELECT * FROM xmlInvoice
WHERE invoice.exist('(/Invoice/@InvoiceId)[1][. = "1003"]') = 1

-- query 5
SELECT * FROM xmlInvoice
WHERE invoice.exist('/Invoice[CustomerName = "Vahid"]') = 1

-- query 6
SELECT * FROM xmlInvoice
WHERE invoice.exist('/Invoice/CustomerName [.= "Vahid"]') = 1

-- query 7
SELECT * FROM xmlInvoice
WHERE invoice.exist('/Invoice/LineItems/LineItem[Description = "Trellis"]') = 1

-- query 8
SELECT * FROM xmlInvoice
WHERE invoice.exist('/Invoice/LineItems/LineItem/Description [.= "Trellis"]') = 1

-- query 9
SELECT * FROM xmlInvoice
WHERE invoice.exist('
for $x in /Invoice/@InvoiceId
where $x = 1003
return $x
') = 1

-- query 10
SELECT * FROM xmlInvoice
WHERE invoice.value('(/Invoice/@InvoiceId)[1]', 'VARCHAR(10)') = '1003'


-- یکبار هم با جدول شماره 2 که اسکیما دارد تمام این موارد تکرار شود

UPDATE STATISTICS xmlInvoice
DBCC FREEPROCCACHE

GO
```

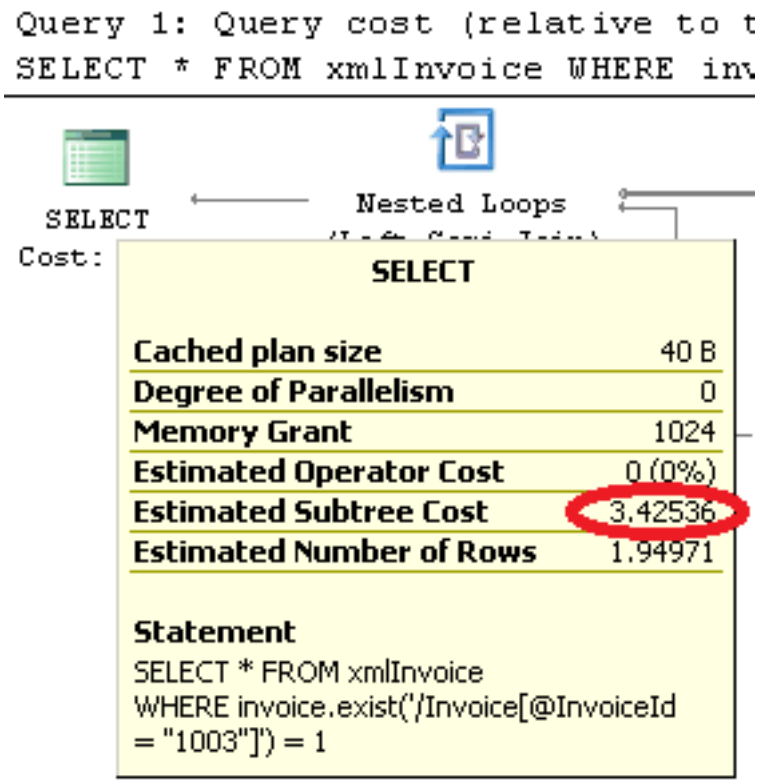
کوئری 1

همانطور که عنوان شد، از منوی Query گزینه‌ی Include actual execution plan را نیز انتخاب کنید (یا فشردن دکمه‌های Ctrl+M) تا پس از اجرای کوئری، بتوان Query Plan نهایی را نیز مشاهده کرد.
در کوئری 1، با استفاده از متد exist به دنبال رکوردهایی هستیم که دارای ویژگی InvoiceId مساوی 1003 هستند. پس از

اجرای کوئری، تصویر Query Plan آن به شکل زیر خواهد بود:



برای خواندن این تصویر، از بالا به پایین و چپ به راست باید عمل شود. هزینه‌ی انجام کوئری را نیز با نگه داشتن کرسر ماوس بر روی select نهایی سمت چپ تصویر می‌توان مشاهده کرد. البته باید درنظر داشت که این اعداد از دیدگاه Query Processor مفهوم پیدا می‌کنند. پردازشگر کوئری، بر اساس اطلاعاتی که در اختیار دارد، سعی می‌کند بهترین روش پردازش کوئری دریافتی را پیدا کند. برای اندازه گیری کارآیی، باید اندازه گیری زمان اجرای کوئری، مستقلا انجام شود.



در این کوئری، مطابق تصویر اول، ابتدا قسمت SQL آن (چپ بالای تصویر) پردازش می‌شود و سپس قسمت XML آن. قسمت XQuery این عبارت در دو قسمت سمت چپ، پایین تصویر مشخص شده‌اند. Table valued functionها جاهایی هستند که node table ابتدای بحث جاری در آن‌ها ساخته می‌شوند. در اینجا دو مرحله‌ی تولید Table valued functionها مشاهده می‌شود. اگر به جمع درصدهای آن‌ها دقت کنید، هزینه‌ی این دو قسمت، 98 درصد کل Query plan است.

سؤال: چرا دو مرحله‌ی تولید Table valued functionها در اینجا قابل مشاهده است؟ یک مرحله‌ی آن مربوط است به انتخاب نود Invoice و مرحله‌ی دوم مربوط است به فیلتر داخل [] ذکر شد برای یافتن ویژگی‌های مساوی 1003.

در اینجا و در کوئری‌های بعدی، هر Query Plan ایی که تعداد مراحل تولید Table valued function کمتری داشته باشد، بهینه‌تر است.

کوئری 5

اگر کوئری پلن شماره 5 را بررسی کنیم، به 3 مرحله تولید Table valued functionها خواهیم رسید. یک XML Reader برای خارج از [] (اصطلاحا به آن predicate گفته می‌شود) و دو مورد برای داخل [] تشکیل شده‌است؛ یکی برای انتخاب نود متنی و دیگری برای تساوی.

کوئری 7

اگر کوئری پلن شماره 7 را بررسی کنیم، به 3 مرحله تولید Table valued functionها خواهیم رسید که بسیار شبیه است به مورد 5. بنابراین در اینجا عمق بررسی و سلسله مراتب اهمیتی ندارد.

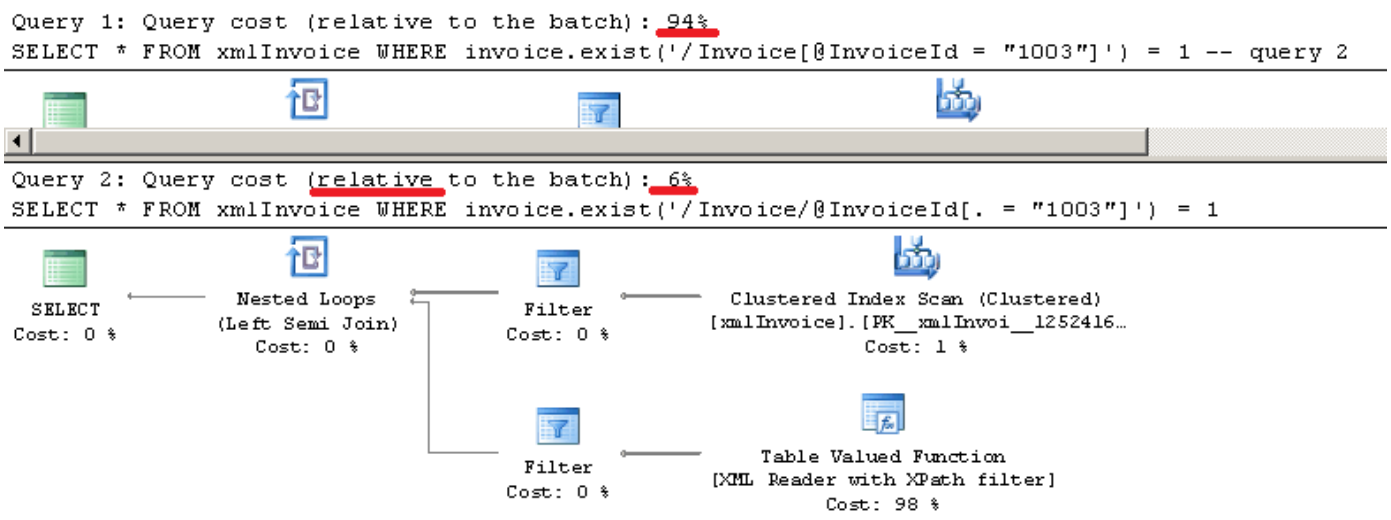
کوئری 9

کوئری 9 دقیقا معادل است با کوئری 1 نوشته شده؛ با این تفاوت که از روش FLOWR استفاده کرده‌است. نکته‌ی جالب آن، وجود تنها یک XML reader در Query plan آن است که باید آن‌را بخاطر داشت.

کوئری 2
کوئری 3
کوئری 4
کوئری 6
کوئری 8

اگر به این 5 کوئری یاد شده دقت کنید، از یک دات به معنای self استفاده کرده‌اند (یعنی پردازش بیشتری را انجام نده و از همین نود جاری برای پردازش نهایی استفاده کن). با توجه به بکارگیری متد exist، معنای کوئری‌های یک و دو، یکی‌است. اما در کوئری شماره 2، تنها یک XML Reader در Query plan نهایی وجود دارد (همانند عبارت FLOWR کوئری شماره 9).

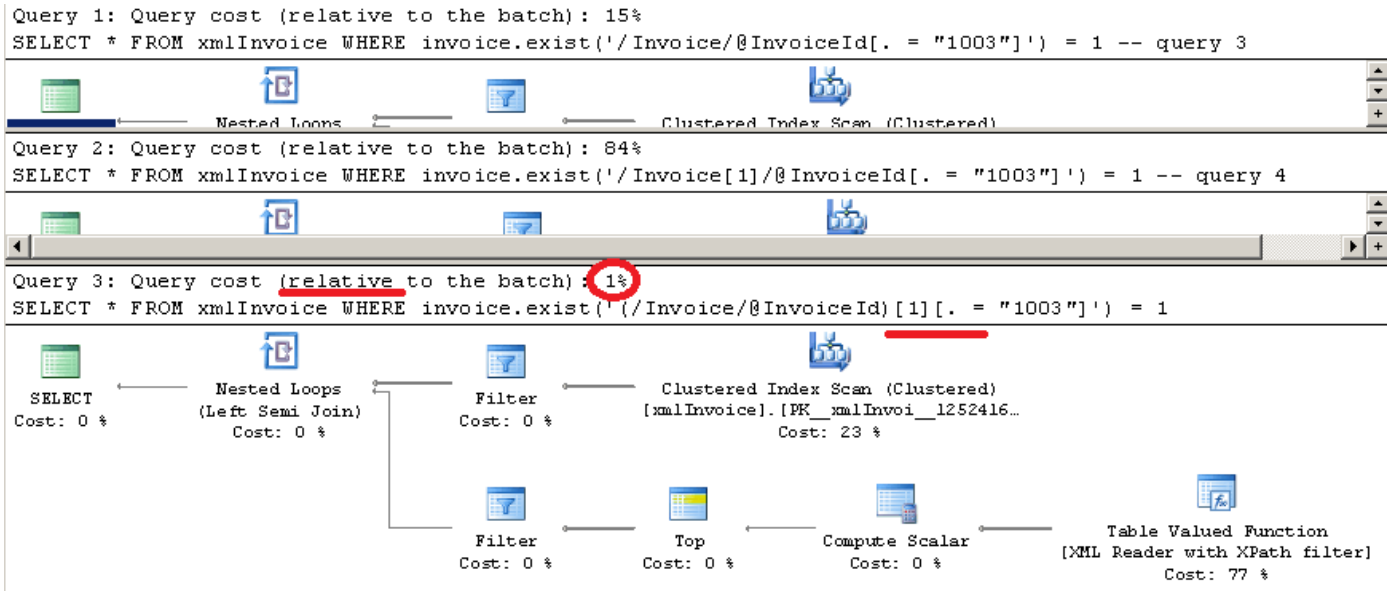
یک نکته: اگر می‌خواهید بدانید بین کوئری‌های 1 و 2 کدامیک بهتر عمل می‌کنند، از بین تمام کوئری‌های موجود، دو کوئری یاد شده را انتخاب کرده و سپس با فرض روش بودن نمایش Query plan، هر دو کوئری را با هم اجرا کنید.



در این حالت، کوئری پلن‌های هر دو کوئری را با هم یکجا می‌توان مشاهده کرد؛ به علاوه‌ی هزینه‌ی نسبی آن‌ها را در کل عملیات صورت گرفته. در حالت استفاده از دات و وجود تنها یک XML Reader، این هزینه تنها 6 درصد است، در مقابل هزینه‌ی 94 درصدی کوئری شماره یک.
بنابراین از دیدگاه پردازشگر کوئری‌های SQL Server، کوئری شماره 2، بسیار بهتر است از کوئری شماره 1.

در کوئری‌های 3 و 4، شماره نود مدنظر را دقیقا مشخص کرده‌ایم. این مورد در حالت سوم تفاوت محسوسی را از لحاظ کارآیی ایجاد نمی‌کند و حتی کارآیی را به علت اضافه کردن یک XML Reader دیگر برای پردازش عدد نود وارد شده، کاهش می‌دهد. اما کوئری 4 که عدد اولین نود را خارج از پرانتز قرار داده‌است، تنها در کل یک XML Reader را به همراه خواهد داشت.

سؤال: بین کوئری‌های 2، 3 و 4 کدامیک بهینه‌تر است؟



بله. اگر هر سه کوئری را با هم انتخاب کرده و اجرا کنیم، می‌توان در قسمت کوئری پلن‌ها، هزینه‌ی هر کدام را نسبت به کل مشاهده کرد. در این حالت کوئری 4 بهتر است از کوئری 2 و تنها یک درصد هزینه‌ی کل را تشکیل می‌دهد.

کوئری 10

کوئری 10 اندکی متفاوت است نسبت به کوئری‌های دیگر. در اینجا بجای متد exist از متد value استفاده شده‌است. یعنی ابتدا صریحا مقدار ویژگی InvoiceId استخراج شده و با 1003 مقایسه می‌شود.
اگر کوئری پلن آن‌را با کوئری 4 که بهترین کوئری سری exist است مقایسه کنیم، کوئری 10، هزینه‌ی 70 درصدی کل عملیات را به خود اختصاص خواهد داد، در مقابل 30 درصد هزینه‌ی کوئری 4. بنابراین در این موارد، استفاده از متد exist بسیار بهینه‌تر است از متد value.

استفاده از Schema collection و تاثیر آن بر کارآیی

تمام مراحلی را که در اینجا ملاحظه کردید، صرفا با تغییر نام xmlInvoice به xmlInvoice2، تکرار کنید. xmlInvoice2 دارای ساختاری مشخص است، به همراه ذکر صریح document حین تعریف ستون XML ایی آن.
تمام پاسخ‌هایی را که دریافت خواهید کرد با حالت بدون Schema collection یکی است.
برای مقایسه بهتر، یکبار نیز سعی کنید کوئری 1 جدول xmlInvoice را با کوئری 1 جدول xmlInvoice2 با هم در طی یک اجرا مقایسه کنید، تا بهتر بتوان Query plan نسبی آن‌ها را بررسی کرد.
پس از این بررسی و مقایسه، به این نتیجه خواهید رسید که تفاوت محسوسی در اینجا و بین این دو حالت، قابل ملاحظه نیست. در SQL Server از Schema collection بیشتر برای اعتبارسنجی ورودی‌ها استفاده می‌شود تا بهبود کارآیی کوئری‌ها.

بنابراین به صورت خلاصه
- متد exist را به value ترجیح دهید.
- اصطلاحا ordinal (همان مشخص کردن نود 1 در اینجا) را در آخر قرار دهید (نه در بین نودها).
- مراحل اجرایی را با معرفی دات (استفاده از نود جاری) تا حد ممکن کاهش دهید.

و ... **کوئری 4** در این سری، بهترین کارآیی را ارائه می‌دهد.

عنوان: **بررسی کارآیی و ایندکس گذاری بر روی اسناد XML در SQL Server - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۲ ۱۱:۰
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, SQL Server, xml

تا اینجا ملاحظه کردید که XQuery ایندکس نشده چگونه بر روی Query Plan تاثیر دارد. در ادامه، مباحث ایندکس گذاری بر روی اسناد XML ایی را مرور خواهیم کرد.

ایندکس‌های XML ایی

ایندکس‌های XML ایی، ایندکس‌های خاصی هستند که بر روی ستون‌هایی از نوع XML تعریف می‌شوند. هدف از تعریف آن‌ها، بهینه سازی اعمال مبتنی بر XQuery، بر روی داده‌های این نوع ستون‌ها است. چهار نوع XML Index قابل تعریف هستند؛ اما primary xml index باید ابتدا ایجاد شود. در این حالت جدولی که دارای ستون XML ایی است نیز باید دارای یک clustered index باشد. هدف از primary XML indexها، ارائه‌ی تخمین‌های بهتری است به بهینه ساز کوئری‌ها در SQL Server.

جزئیات primary XML indexها

زمانیکه یک primary xml index را ایجاد می‌کنیم، node table یاد شده در قسمت قبل را ، بر روی سخت دیسک ذخیره خواهیم کرد (بجای هربار محاسبه در زمان اجرا). متادیتای این اطلاعات ذخیره شده را در جداول سیستمی sys.indexes و sys.columns می‌توان مشاهده کرد. باید دقت داشت که تهیه‌ی این ایندکس‌ها، فضای قابل توجهی را از سخت دیسک به خود اختصاص خواهند داد؛ چیزی حدود 2 تا 5 برابر حجم اطلاعات اولیه. بدیهی است تهیه‌ی این ایندکس‌ها که نتیجه‌ی تجزیه‌ی اطلاعات XML ایی است، بر روی سرعت insert تاثیر خواهند گذاشت. Node table دارای ستون‌هایی مانند نام تگ، آدرس تگ، نوع داده آن، مسیر و امثال آن است.
زمانیکه یک Primary XML Index تعریف می‌شود، اگر به Query Plan حاصل دقت کنید، دیگر خبری از XMLهای مانند قبل نخواهد بود. در اینجا Clustered index seek قابل مشاهده‌است.

ایجاد primary XML indexها

همان مثال قسمت قبل را که دو جدول از آن به نام‌های xmlInvoice و xmlInvoice2 ایجاد کردیم، درنظر بگیرید. اینبار یک xmlInvoice3 را با همان ساختار و همان 6 رکوردی که معرفی شدند، ایجاد می‌کنیم. بنابراین برای آزمایش جاری، در مثال قبل ، هرجایی xmlInvoice مشاهده می‌کنید، آن‌را به xmlInvoice3 تغییر داده و مجددا جدول مربوطه و داده‌های آن‌را ایجاد کنید.
اکنون برای ایجاد primary XML index بر روی ستون invoice آن می‌توان نوشت:

```
CREATE PRIMARY XML INDEX invoice_idx ON xmlInvoice3(invoice)
 SELECT * FROM sys.internal_tables
```

کوئری دومی که بر روی sys.internal_tables انجام شده، محل ذخیره سازی این ایندکس را نمایش می‌دهد که دارای نامی مانند xml_index_nodes_325576198_256000 خواهد بود. دو عدد پس از آن table object id و column object id هستند.
در ادامه علاقمند هستیم که بدانیم داخل آن چه چیزی ذخیره شده‌است:

```
SELECT * FROM sys.xml_index_nodes_325576198_256000
```

اگر این کوئری را اجرا کنید احتمالا به خطای Invalid object name برخواهید خورد. علت اینجا است که برای مشاهده‌ی اطلاعات جداول داخلی مانند این، نیاز است حین اتصال به SQL Server، در قسمت server name نوشت (local):admin و حالت authentication نیز باید بر روی Windows authentication باشد. به آن اصطلاحا Dedicated administrator connection نیز می‌گویند. برای این منظور حتما نیاز است از طریق منوی Database Engine Query -> New -> File شروع کنید در غیراینصورت پیام Dedicated administrator connections are not supported را دریافت خواهید کرد.

اگر به این جدول دقت کنید، 6 ردیف اطلاعات XML ایی، به حدود 100 ردیف اطلاعات ایندکس شده، تبدیل گردیده‌است. با استفاده از دستور ذیل می‌توان حجم ایندکس تهیه شده را نیز مشاهده کرد:

```
sp_spaceused 'xmlInvoice3'
```

در صورت نیاز برای حذف ایندکس ایجاد شده می‌توان به نحو ذیل عمل کرد:

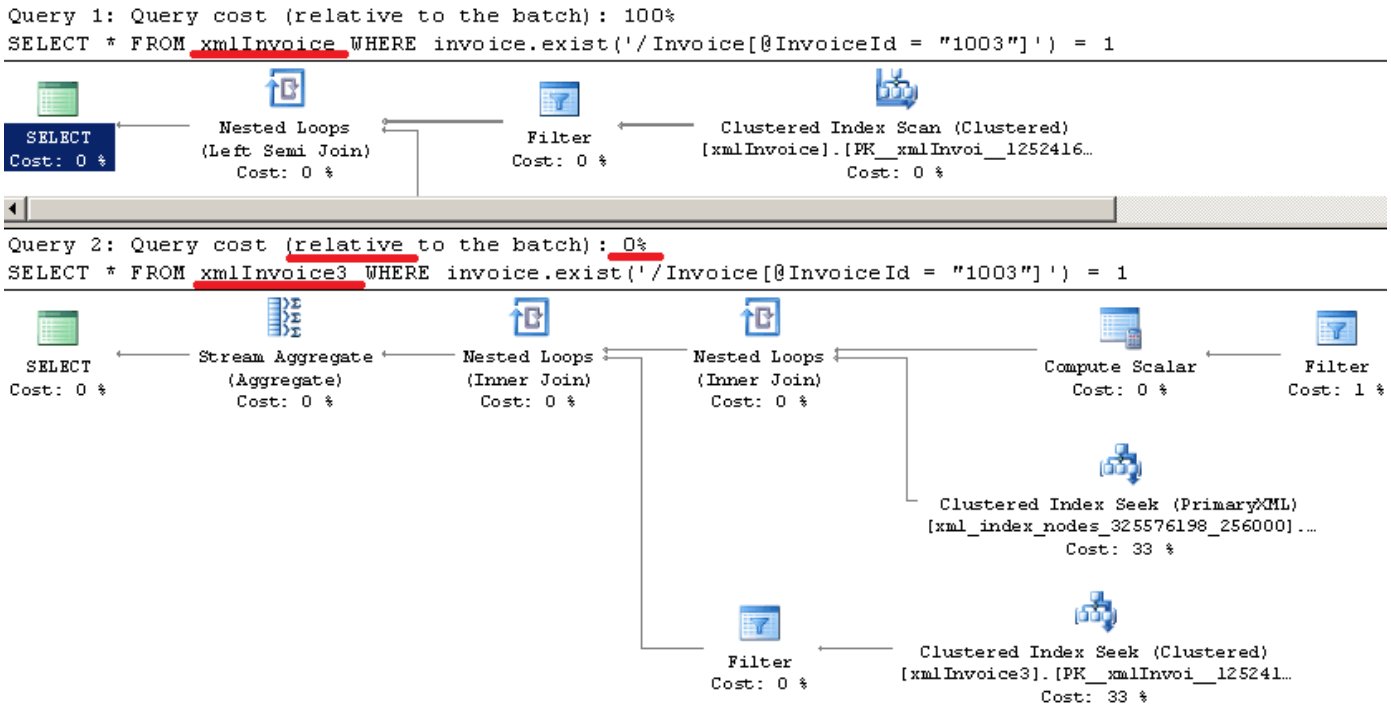
```
--DROP INDEX invoice_idx ON xmlInvoice3
```

تاثیر primary XML indexها بر روی سرعت اجرای کوئری‌ها

همان 10 کوئری قسمت قبل را درنظر بگیرید. اینبار برای مقایسه می‌توان به نحو ذیل عمل کرد:

```
SELECT * FROM xmlInvoice
WHERE invoice.exist('/Invoice[@InvoiceId = "1003"]') = 1

SELECT * FROM xmlInvoice3
WHERE invoice.exist('/Invoice[@InvoiceId = "1003"]') = 1
```

دو کوئری یکی هستند اما اولی بر روی xmlInvoice اجرا می‌شود و دومی بر روی xmlInvoice3. هر دو کوئری را انتخاب کرده و با استفاده از منوی Query، گزینه‌ی Include actual execution plan را نیز انتخاب کنید (یا فشردن دکمه‌های Ctrl+M) تا پس از اجرای کوئری، بتوان Query Plan نهایی را نیز مشاهده نمود.



چند نکته در این تصویر حائز اهمیت است:

- Query plan کوئری انجام شده بر روی جدول دارای primary XML index، مانند قسمت قبل، حاوی XML Readerها نیست.
- هزینه‌ی انجام کوئری بر روی جدول دارای XML ایندکس نسبت به حالت بدون ایندکس، تقریبا نزدیک به صفر است. (بهبود کارآیی فوق العاده)

اگر کوئری‌های دیگر را نیز با هم مقایسه کنید، تقریبا به نتیجه‌ی کمتر از یک سوم تا یک چهارم حالت بدون ایندکس خواهید رسید. همچنین اگر برای حالت دارای Schema collection نیز ایندکس ایجاد کنید، اینبار کوئری پلن آن اندکی (چند درصد) بهبود خواهد یافت ولی نه آنچنان.

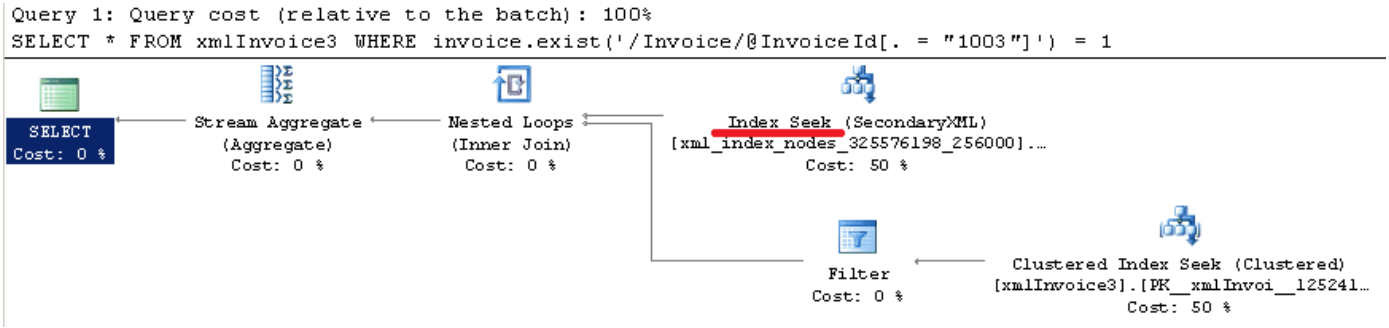
ایندکس‌های XMLایی ثانویه یا secondary XML indexes

سه نوع ایندکس XML ایی ثانویه نیز قابل تعریف هستند:
- VALUE : کار آن بهینه سازی کوئری‌های content و wildcard است.
- PATH : بهینه سازی انتخاب‌های مبتنی بر XPath را انجام می‌دهد.
- Property: برای بهینه سازی انتخاب خواص و ویژگی‌ها بکار می‌رود.

این ایندکس‌ها یک سری non-clustered indexes بر روی node tables هستند. برای ایجاد سه نوع ایندکس یاد شده به نحو ذیل می‌توان عمل کرد:

```
CREATE XML INDEX invoice_path_idx ON xmlInvoice3(invoice)
 USING XML INDEX invoice_idx FOR PATH
```

در اینجا یک path index جدید ایجاد شده‌است. ایندکس‌های ثانویه نیاز به ذکر ایندکس اولیه نیز دارند.
پس از ایجاد ایندکس ثانویه بر روی مسیرها، اگر اینبار کوئری دوم را اجرا کنیم، به Query Plan ذیل خواهیم رسید:

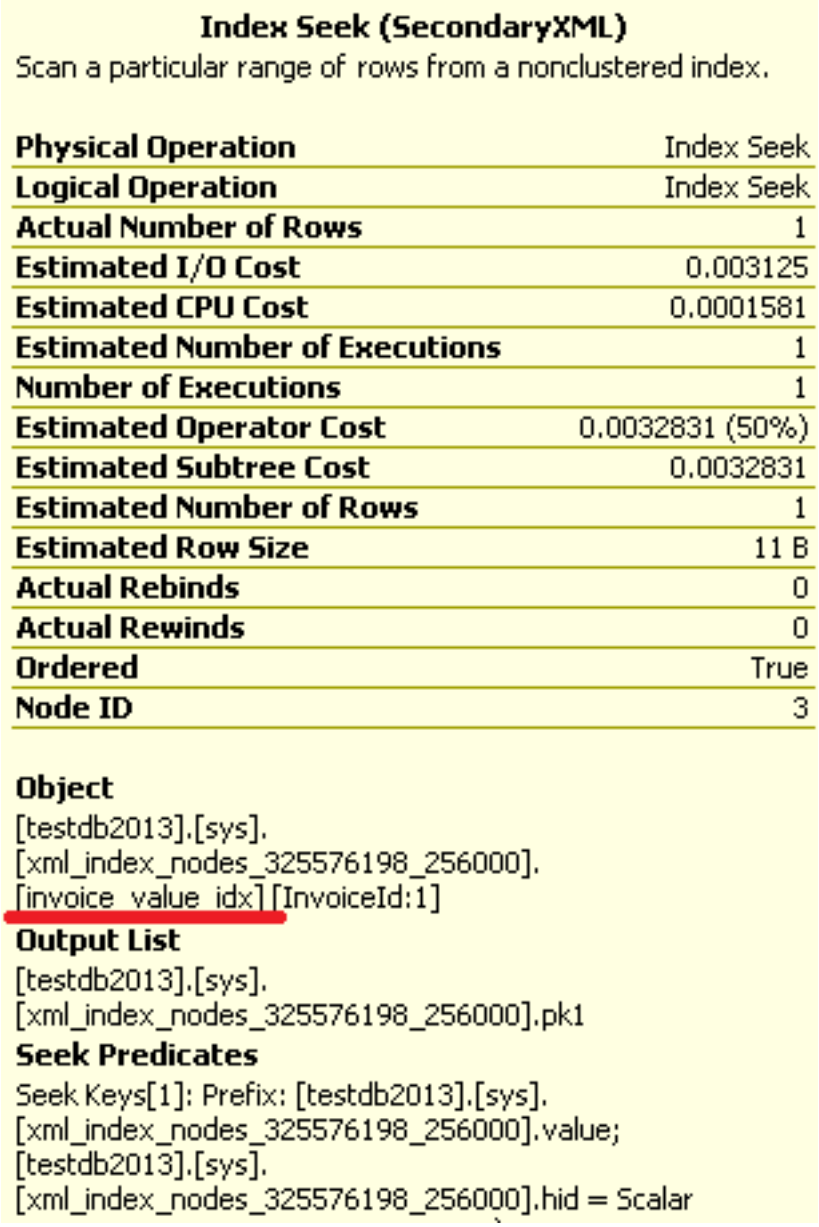


همانطور که مشاهده می‌کنید، نسبت به حالت primary index، وضعیت clustered index seek به index seek تغییر کرده‌است و همچنین دقیقا مشخص است که از کدام ایندکس استفاده شده‌است.
در ادامه دو نوع ایندکس دیگر را نیز ایجاد می‌کنیم:

```
CREATE XML INDEX invoice_value_idx ON xmlInvoice3(invoice)
 USING XML INDEX invoice_idx FOR VALUE

 CREATE XML INDEX invoice_prop_idx ON xmlInvoice3(invoice)
 USING XML INDEX invoice_idx FOR PROPERTY
```

سؤال: اکنون پس از تعریف 4 ایندکس یاد شده، کوئری دوم از کدام ایندکس استفاده خواهد کرد؟

در اینجا مجددا کوئری دوم را اجرا کرده و به قسمت Query Plan آن دقت خواهیم کرد:



Index Seek (SecondaryXML)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Number of Rows	1
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001581
Estimated Number of Executions	1
Number of Executions	1
Estimated Operator Cost	0.0032831 (50%)
Estimated Subtree Cost	0.0032831
Estimated Number of Rows	1
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	3

Object

[testdb2013].[sys].
[xml_index_nodes_325576198_256000].
[invoice_value_idx] [InvoiceId:1]

Output List

[testdb2013].[sys].
[xml_index_nodes_325576198_256000].pk1

Seek Predicates

Seek Keys[1]: Prefix: [testdb2013].[sys].
[xml_index_nodes_325576198_256000].value;
[testdb2013].[sys].
[xml_index_nodes_325576198_256000].hid = Scalar

برای مشاهده دقیق نام ایندکس مورد استفاده، کرسر ماوس را بر روی index seek قرار می‌دهیم. در اینجا اگر به قسمت object گزارش ارائه شده دقت کنیم، نام invoice_value_idx یا همان value index ایجاد شده، قابل مشاهده‌است؛ به این معنا که در کوئری دوم، اهمیت مقادیر بیشتر است از اهمیت مسیرها.

کوئری‌هایی مانند کوئری ذیل از property index استفاده می‌کنند:

```
SELECT * FROM xmlInvoice3
WHERE invoice.exist('/Invoice//CustomerName[text() = "Vahid"]') = 1
```

در اینجا با بکارگیری // به دنبال CustomerName در تمام قسمت‌های سند Invoice خواهیم گشت. البته کوئری پلن آن نسبتا پیچیده‌است و شامل primary index اسکن و clusterd index اسکن نیز می‌شود. برای بهبود قابل ملاحظه‌ی آن می‌توان به نحو ذیل از عملگر self استفاده کرد:

```
SELECT * FROM xmlInvoice3
WHERE invoice.exist('/Invoice//CustomerName[. = "Vahid"]') = 1
```

خلاصه نکات بهبود کارآیی برنامه‌های مبتنی بر فیلدهای XML

- در حین استفاده از XPath، ذکر محور parent یا استفاده از .. (دو دات)، سبب ایجاد مراحل اضافه‌ای در Query Plan می‌شوند. تا حد امکان از آن اجتناب کنید و یا از روش‌هایی مانند cross apply و xml.nodes برای مدیریت اینگونه موارد تو در تو استفاده نمائید.
- ordinals را به انتهای Path منتقل کنید (مانند ذکر [1] جهت مشخص سازی نودی خاص).
- از ذکر predicates در وسط یک Path اجتناب کنید.
- اگر اسناد شما fragment با چند root elements نیستند، بهتر است document بودن آن‌ها را در حین ایجاد ستون XML مشخص کنید.
- xml.value را به xml.query ترجیح دهید.
- عملیات casting در XQuery سنگین بوده و استفاده از ایندکس‌ها را غیرممکن می‌کند. در اینجا استفاده از اسکیما می‌تواند مفید باشد.
- نوشتن sub queryها بهتر هستند از چندین XQuery در یک عبارت SQL.
- در ترکیب اطلاعات رابطه‌ای و XML، استفاده از متدهای xml.exist و sql:column نسبت به xml.value جهت استخراج و مقایسه اطلاعات، بهتر هستند.
- اگر قصد تهیه خروجی XML از جدولی رابطه‌ای را دارید، روش select for xml کارآیی بهتری را نسبت به روش FLOWR دارد. روش FLOWR برای کار با اسناد XML موجود طراحی و بهینه شده‌است؛ اما روش select for xml در اصل برای کار با اطلاعات رابطه‌ای بهینه سازی گردیده‌است.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۴ ۱۳:۱۹

یک نکته‌ی تکمیلی

از SQL Server 2012 SP1 به بعد، ایندکس جدیدی به نام Selective XML Indexes به مجموعه‌ی ایندکس‌های قابل تعریف بر روی یک ستون XML ایی اضافه شده‌است و این مزایا را به همراه دارد:

- برخلاف ایندکس‌های اولیه و ثانویه بحث شده در مطلب جاری، کل محتوای سند را ایندکس نمی‌کنند. به همین جهت حجم کمتری را اشغال کرده و سرعت Insert و Update را کاهش نمی‌دهند.
- با استفاده از Selective XML Indexes تنها XPathهایی را که مشخص می‌کنید، ایندکس خواهند شد. بنابراین بر اساس کوئری‌های موجود، می‌توان ایندکس‌های بهتری را تعریف کرد.

این نوع ایندکس‌ها به صورت پیش فرض فعال نبوده و نیاز است از طریق رویه ذخیره شده سیستمی sp_db_selective_xml_index، فعال شوند.

```
sys.sp_db_selective_xml_index @dbname = 'dbname', @selective_xml_index = 'action: on|off|true|false'
```

و پس از آن برای تعریف یک ایندکس انتخابی خواهیم داشت:

```
create selective xml index index_name
on table_name(column_name)
for (<path>)
```

قسمت path آن برای مثال در عمل، چنین شکلی را می‌تواند داشته باشد:

```
for(
 pathColor = '/Item/Product/Color' as SQL nvarchar(20),
 pathSize = '/Item/Product/Size' as SQL int
)
```

عنوان: **استفاده از Full Text Search بر روی اسناد XML**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۳ ۲۳:۳۴
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, SQL Server, xml

امکان استفاده‌ی همزمان قابلیت Full Text Search و اسناد XML ایی نیز در SQL Server پیش بینی شده‌است. به این ترتیب می‌توان متون این اسناد را ایندکس و جستجو کرد. در این حالت تگ‌های XML ایی و ویژگی‌ها، به صورت خودکار حذف شده و در نظر گرفته نمی‌شوند. Syntax استفاده از Full text search در اینجا با سایر حالات و ستون‌های متداول رابطه‌ای SQL Server تفاوتی ندارد. به علاوه امکان ترکیب آن با یک XQuery نیز میسر است. در این حالت، Full text search، ابتدا انجام شده و سپس با استفاده از XQuery می‌توان بر روی این نتایج، نودها، مسیرها و ویژگی‌های خاصی را جستجو کرد.

نحوه‌ی استفاده از Full Text Search بر روی ستون‌های XML ایی

برای آزمایش، ابتدا یک جدول جدید را که حاوی ستونی XML ایی است، ایجاد کرده و سپس چند سند XML را که حاوی متونی نسبتا طولانی هستند، در آن ثبت می‌کنیم. ذکر CONSTRAINT در اینجا جهت دستور ایجاد ایندکس Full Text Search ضروری است.

```
CREATE TABLE ftsXML(
id INT IDENTITY PRIMARY KEY,
doc XML NULL
CONSTRAINT UQ_FTS_Id UNIQUE(id)
)
GO
INSERT ftsXML VALUES('
<book>
<title>Sample book title 1</title>
<author>Vahid</author>
<chapter ID="1">
<title>Chapter 1</title>
<content>
"The quick brown fox jumps over the lazy dog" is an English-language
pangram–a phrase that contains all of the letters of the English alphabet.
It has been used to test typewriters and computer keyboards, and in other
applications involving all of the letters in the English alphabet. Owing to its
brevity and coherence, it has become widely known.
</content>
</chapter>
<chapter ID="2">
<title>Chapter 2</title>
<content>
In publishing and graphic design, lorem ipsum is a placeholder text commonly used
to demonstrate the graphic elements of a document or visual presentation.
By replacing the distraction of meaningful content with filler text of scrambled
Latin it allows viewers to focus on graphical elements such as font, typography,
and layout.
</content>
</chapter>
</book>
')

INSERT ftsXML VALUES('
<book>
<title>Sample book title 2</title>
<author>Farid</author>
<chapter ID="1">
<title>Chapter 1</title>
<content>
The original passage began: Neque porro quisquam est qui dolorem ipsum quia dolor sit
amet consectetur adipisci velit
</content>
</chapter>
<chapter ID="2">
<title>Chapter 2</title>
<content>
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore
eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.
</content>
```

```
</chapter>
</book>
')
GO
```

سپس با استفاده از دستورات ذیل، Full text search را بر روی ستون doc جدول ایجاد شده، فعال می‌کنیم:

```
CREATE FULLTEXT CATALOG FT_CATALOG
GO
CREATE FULLTEXT INDEX ON ftsXML([doc])
KEY INDEX UQ_FTS_Id ON ([FT_CATALOG], FILEGROUP [PRIMARY])
GO
```

اکنون می‌توانیم با ترکیبی از امکانات Full Text Search و XQuery، از ستون doc، کوئری‌های پیشرفته و سریعی را تهیه کنیم.

راه اندازی سرویس Full Text Search

البته پیش از ادامه‌ی بحث به کنسول سرویس‌های ویندوز مراجعه کرده و مطمئن شوید که سرویس SQL Full-text Filter Daemon Launcher MSSQLSERVER در حال اجرا است. در غیراینصورت با خطای ذیل مواجه خواهید شد:

```
SQL Server encountered error 0x80070422 while communicating with full-text filter daemon host (FDHost)
process.
```

اگر این سرویس در حال اجرا است و باز هم خطای فوق ظاهر شد، مجددا به کنسول سرویس‌های ویندوز مراجعه کرد، در برگه‌ی خواص سرویس SQL Full-text Filter Daemon Launcher MSSQLSERVER، گزینه‌ی logon را یافته و آن‌را به local system account تغییر دهید و سپس سرویس را ری استارت کنید. پس از آن نیاز است دستور ذیل را نیز اجرا کنید:

```
sp_fulltext_service 'restart_all_fdhosts'
go
```

بعد از اینکار، بازسازی مجدد Full text search را فراموش نکنید. در این حالت در management studio، به بانک اطلاعاتی مورد نظر مراجعه کرده، نود Storage / Full Text Catalog را باز کنید. سپس بر روی FT_CATALOG ایجاد شده در ابتدای بحث کلیک راست کرده و از منوی ظاهر شده، گزینه‌ی Rebuild را انتخاب کنید. در غیراینصورت کوئری‌های ادامه‌ی بحث، خروجی خاصی را نمایش نخواهند داد.

استفاده از متد Contains

در ادامه، نحوه‌ی ترکیب امکانات Full text search و XQuery را ملاحظه می‌کنید:

```
-- استفاده از ایکس کوئری برای جستجو در نتایج حاصل
SELECT T.doc.value('(/book/title)[1]', 'varchar(100)') AS title
FROM
-- استفاده از اف تی اس برای جستجو
(SELECT * FROM ftsXML
WHERE CONTAINS(doc, '"Quick Brown Fox "')) AS T
```

ابتدا توسط متد Contains مرتبط به Full text search، ردیف‌های مورد نظر را یافته و سپس بر روی آن‌ها با استفاده از XQuery جستجوی دلخواهی را انجام می‌دهیم؛ از این جهت که Full text search تنها متون فیلدهای XML ایی را ایندکس می‌کند و نه تگ‌های آن‌ها را.
خروجی کوئری فوق، title 1 Sample book است.

Full text search امکانات پیشرفته‌تری را نیز ارائه می‌دهد. برای مثال در ردیف‌های ثبت شده داریم fox jumps، اما در متن

ورودی عبارت جستجو، jumped را وارد کرده و به دنبال نزدیک‌ترین رکورد به آن خواهیم گشت:

```
SELECT T.doc.value('(/book/title)[1]', 'varchar(100)') AS title
FROM
(SELECT * FROM ftsXML
WHERE CONTAINS(doc, 'FORMSOF (INFLECTIONAL ,"Quick Brown Fox jumped")')) AS T
```

و یا دو کلمه‌ی نزدیک به هم را می‌توان جستجو کرد:

```
SELECT T.doc.value('(/book/title)[1]', 'varchar(100)') AS title
FROM
(SELECT * FROM ftsXML
WHERE CONTAINS(doc, 'quick NEAR fox')) AS T
```

نکته‌ای در مورد متد Contains

هم Full text search و هم XQuery، هر دو دارای متدی به نام Contains هستند اما یکی نمی‌باشند.

```
SELECT doc.value('(/book/title)[1]', 'varchar(100)') AS title
FROM ftsXML
WHERE doc.exist('/book/chapter/content[contains(., "Quick Brown Fox")]') = 1
```

در اینجا نحوه‌ی استفاده از متد contains مرتبط با XQuery را مشاهده می‌کنید. اگر این کوئری را اجرا کنید، نتیجه‌ای را دریافت نخواهید کرد. زیرا در ردیف‌ها داریم quick brown fox و نه Quick Brown Fox (حروف ابتدای کلمات، بزرگ نیستند).
بنابراین متد contains مرتبط با XQuery یک جستجوی case sensitive را انجام می‌دهد.

عنوان: **بررسی Select For XML**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۲/۰۵ ۰:۱۰
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, SQL Server, xml

تعدادی افزونه‌ی T-SQL، از نگارش‌های پیشین SQL Server، جهت تولید خروجی XML از یک بانک اطلاعاتی رابطه‌ای، به همراه آن بوده‌اند که در این قسمت آن‌ها را بررسی خواهیم کرد.

پیشنیاز بحث

در ادامه، از بانک اطلاعاتی معروف northwind برای تهیه کوئری‌ها استفاده خواهیم کرد. بنابراین فرض بر این است که این بانک اطلاعاتی را پیشتر به وهله‌ی جاری SQL Server خود افزوده‌اید.

بررسی FOR XML RAW

از نگارش 2005 به بعد، Select for XML علاوه بر خروجی متنی XML، توانایی تولید خروجی از نوع XML را نیز یافته است. در ادامه 4 حالت مختلف خروجی آن‌را بررسی خواهیم کرد.

```
SELECT Customers.CustomerID, Orders.OrderID
FROM Customers, Orders
WHERE  Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerID
FOR XML RAW
```

خروجی For XML Raw کوئری فوق به نحو ذیل است:

```
<row CustomerID="ALFKI" OrderID="10643" />
<row CustomerID="ALFKI" OrderID="10692" />
```

Select for XML در اینجا به صورت خودکار، هر ردیف کوئری را تبدیل به یک المان row نموده و همچنین هر ستون کوئری را تبدیل به ویژگی‌های این المان (attributes) کرده‌است. همچنین باید دقت داشت که خروجی آن یک fragment است و دارای یک root element مشخص نیست.

برای تغییر حالت خروجی آن می‌توان از حالت ELEMENTS استفاده کرد:

```
SELECT Customers.CustomerID, Orders.OrderID
FROM Customers, Orders
WHERE  Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerID
FOR XML RAW, ELEMENTS
```

اینبار مقادیر هر ردیف خروجی، بجای ظاهر شدن در ویژگی‌ها، به صورت یک المان نمایش داده می‌شود:

```
<row>
  <CustomerID>ALFKI</CustomerID>
  <OrderID>10643</OrderID>
</row>
```

حالت پیشرفته‌تر FOR XML RAW را در ادامه ملاحظه می‌کنید:

```
SELECT Customers.CustomerID,
Orders.OrderID
FROM Customers,
Orders
WHERE  Customers.CustomerID = Orders.CustomerID
```

```
ORDER BY
Customers.CustomerID
FOR XML RAW('Customer'), ELEMENTS XSINIL, ROOT('Customers'), XMLSCHEMA('http://MyCustomers')
```

با استفاده از Root می‌توان Fragment حاصل را تبدیل به Document با یک Root element مشخص کرد. در قسمت Raw نیز می‌توان مقدار پیش فرض row را مقدار دهی کرد.

```
<Customers>
  <Customer xmlns="http://MyCustomers">
     <CustomerID>ALFKI</CustomerID>
     <OrderID>10643</OrderID>
  </Customer>
```

از XSINIL برای مشخص سازی المان‌های نال استفاده می‌شود. اگر XSINIL ذکر نشود، المان‌های نال در خروجی وجود نخواهند داشت.
ذکر XMLSCHEMA، سبب می‌شود تا SQL Server به صورت خودکار XML Schema را بر اساس اطلاعات ستون‌های رابطه‌ای مورد استفاده تولید کند.
این نکات را برای FOR XML AUTO نیز می‌توان بکار برد.

بررسی FOR XML AUTO

حالت دوم بکارگیری Select for XML به همراه عبارت Auto است:

```
SELECT Customers.CustomerID, Orders.OrderID
FROM Customers, Orders
WHERE  Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerID
FOR XML AUTO, ELEMENTS
```

با خروجی ذیل:

```
<Customers>
  <CustomerID>ALFKI</CustomerID>
  <Orders>
     <OrderID>10643</OrderID>
  </Orders>
  <Orders>
     <OrderID>10692</OrderID>
  </Orders>
</Customers>
```

در اینجا ابتدا شماره مشتری و سپس اطلاعات تمام خریدهای او ذکر می‌شوند.

بررسی For XML Explicit

اگر بخواهیم خروجی را تبدیل به ترکیبی از المان‌ها و ویژگی‌ها کنیم، می‌توان از For XML Explicit استفاده کرد:

```
SELECT 1 AS Tag,
NULL AS Parent,
Customers.CustomerID AS [Customers!1!CustomerID],
NULL AS [Order!2!OrderId]
FROM Customers
UNION ALL
SELECT 2,
1,
Customers.CustomerID,
Orders.OrderID
FROM Customers,
Orders
WHERE  Customers.CustomerID = Orders.CustomerID
```

```
ORDER BY
[Customers!1!CustomerID]
FOR XML EXPLICIT
```

با خروجی:

```
<Customers CustomerID="ALFKI">
  <Order OrderId="10643" />
  <Order OrderId="10692" />
  <Order OrderId="10702" />
  <Order OrderId="10835" />
  <Order OrderId="10952" />
  <Order OrderId="11011" />
</Customers>
```

برای استفاده از FOR XML EXPLICIT، باید به ازای هر سطح از سلسله مراتب مورد نظر، یک عبارت select را تهیه کرد که این‌ها نهایتا باید با هم UNION ALL شوند.
به علاوه دو ستون اضافی Tag و Parent نیز باید ذکر شوند. از این دو برای مشخص سازی سلسه مراتب استفاده می‌شوند.
!1! سبب تولید یک ویژگی در سطح اول می‌شود و !2! سبب تولید ویژگی دیگری در سطح دوم.

بررسی FOR XML PATH

همانطور که مشاهده می‌کنید، نوشتن FOR XML EXPLICIT نسبتا طولانی و پیچیده‌است. برای ساده سازی آن از نگارش 2005 به بعد، روش For XML Path معرفی شده‌است:

```
WITH XMLNAMESPACES('http://somens' AS au)
SELECT
  CustomerID AS [@au:CustomerID],
  CompanyName AS [Company/Name],
  ContactName AS [Contact/Name]
FROM Customers
 FOR XML PATH('Customer')
```

با خروجی:

```
<Customer xmlns:au="http://somens" au:CustomerID="ALFKI">
  <Company>
       <Name>Alfreds Futterkiste</Name>
  </Company>
  <Contact>
      <Name>Maria Anders</Name>
  </Contact>
</Customer>
```

در اینجا با استفاده از WITH XMLNAMESPACES یک فضای نام جدید را تعریف کرده و سپس نحوه‌ی استفاده از آن‌را توسط یک Alias مشاهده می‌کنید. در اینجا همچنین توسط Aliasها می‌توان یک مسیر مشخص را نیز تعریف کرد. رشته‌ای که در قسمت Path مشخص می‌شود، بیانگر نام المان‌های خروجی است.

یک نکته: اگر کوئری FOR XML PATH را اجرا کنید، نام ستون خروجی به صورت خودکار به XML_F5..6B تنظیم می‌شود. علت اینجا است که در حالت پیش فرض، نوع خروجی این افزونه، استریم است و نه XML. برای تبدیل آن به نوع XML باید یک Type را اضافه کرد:

```
FOR XML PATH('Customer'), Type
```

در این حالت خروجی FOR XML PATH قابل انتساب به یک متغیر T-SQL از نوع XML خواهد بود.