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.