عنوان: **متدهای توکار استفاده از نوع داده‌ای XML - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۹ ۱۷:۲۷
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.