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.