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.

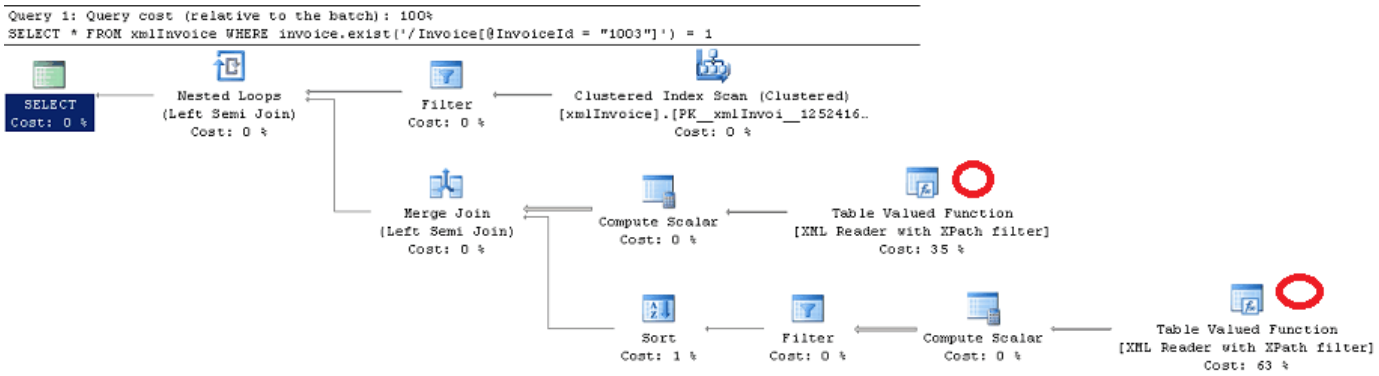
کوئری‌هایی را که در این قسمت بررسی خواهیم کرد، در ادامه ملاحظه می‌کنید. بار اول این کوئری‌ها را بر روی xmlInvoice و بار دوم، بر روی نگارش دوم دارای اسکیمای آن اجرا خواهیم کرد:

```
-- 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
```

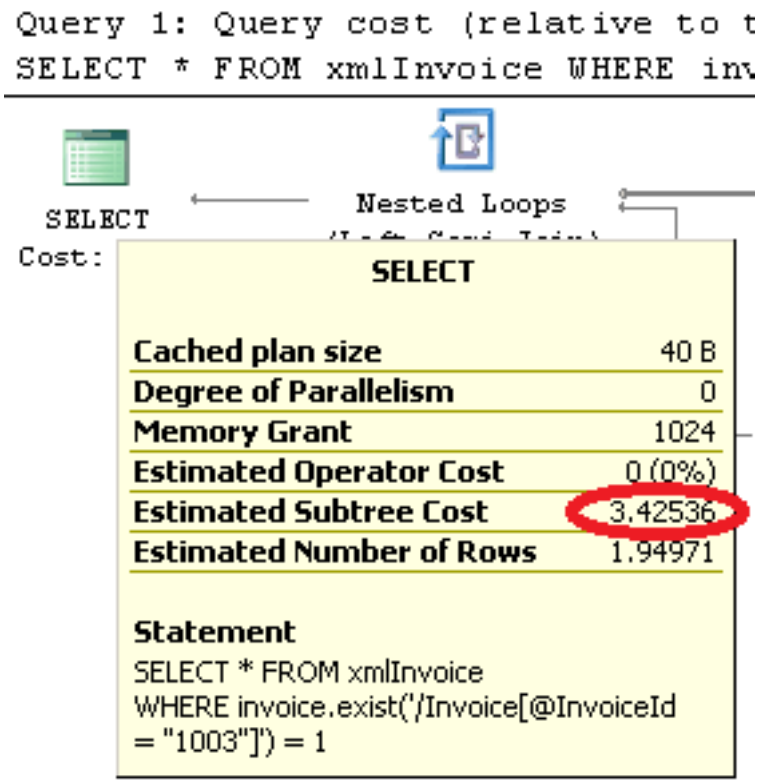
کوئری 1

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.

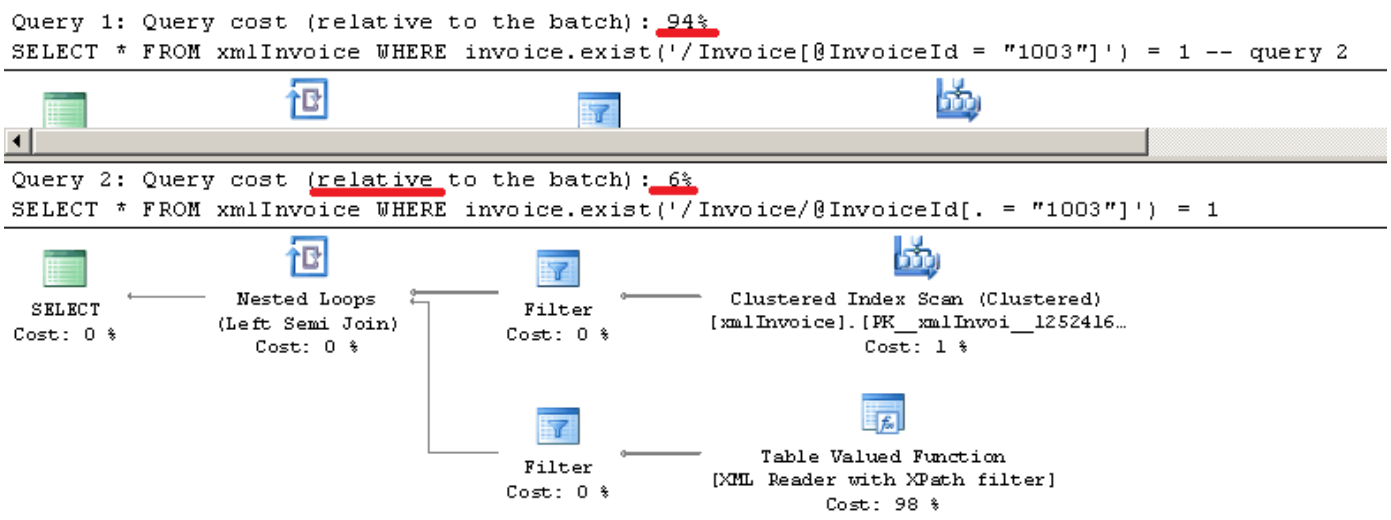
کوئری 9

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).

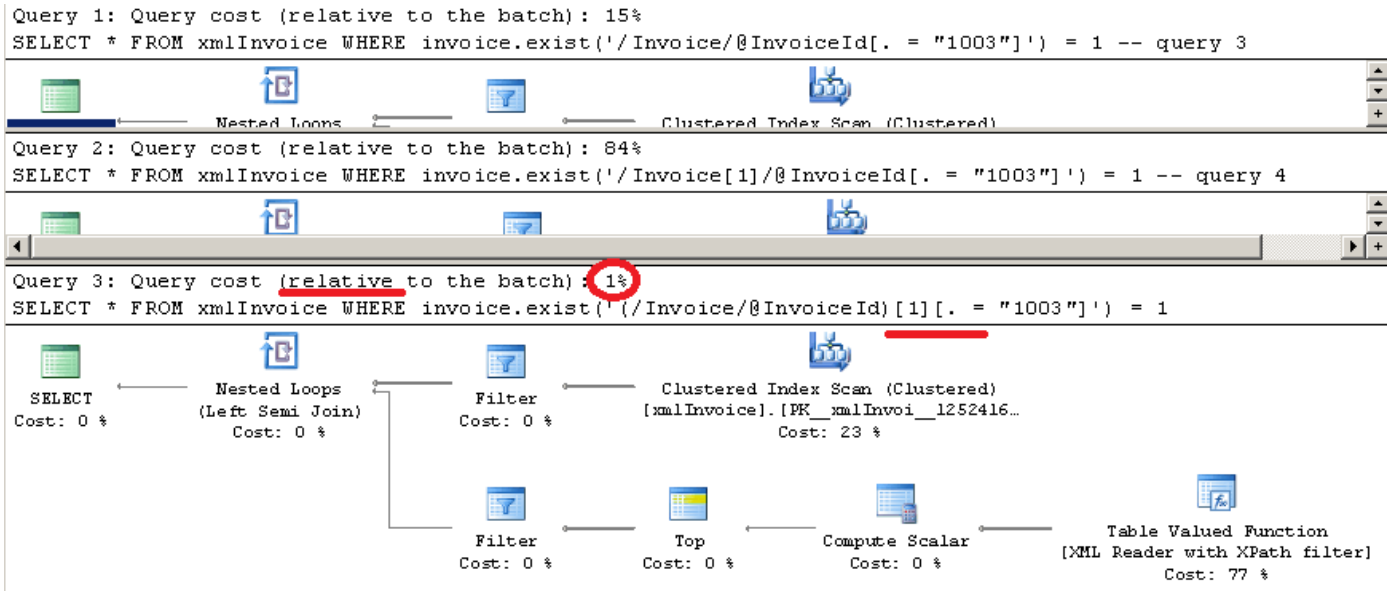
یک نکته: اگر می‌خواهید بدانید بین کوئری‌های 1 و 2 کدامیک بهتر عمل می‌کنند، از بین تمام کوئری‌های موجود، دو کوئری یاد شده را انتخاب کرده و سپس با فرض روش بودن نمایش Query plan، هر دو کوئری را با هم اجرا کنید.



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.