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 XML Readerها 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
```

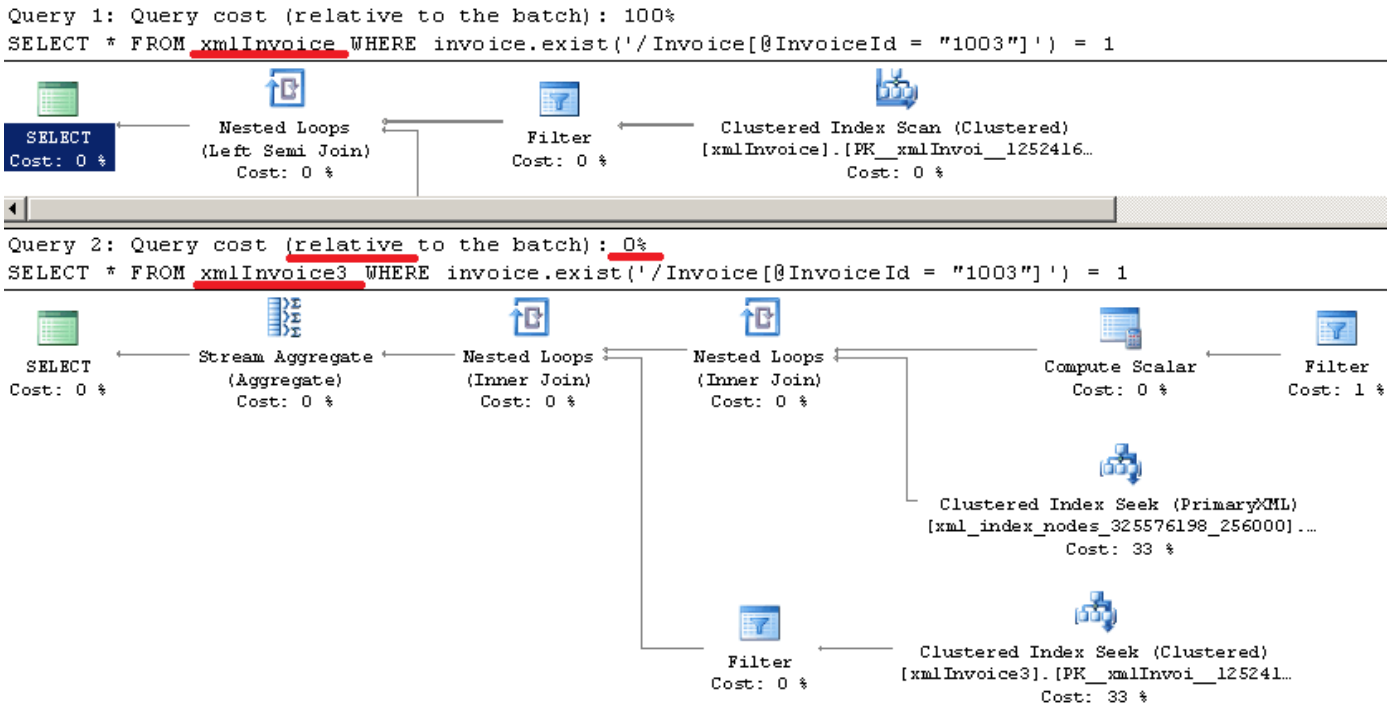
تاثیر primary XML indexها بر روی سرعت اجرای کوئری‌ها

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.

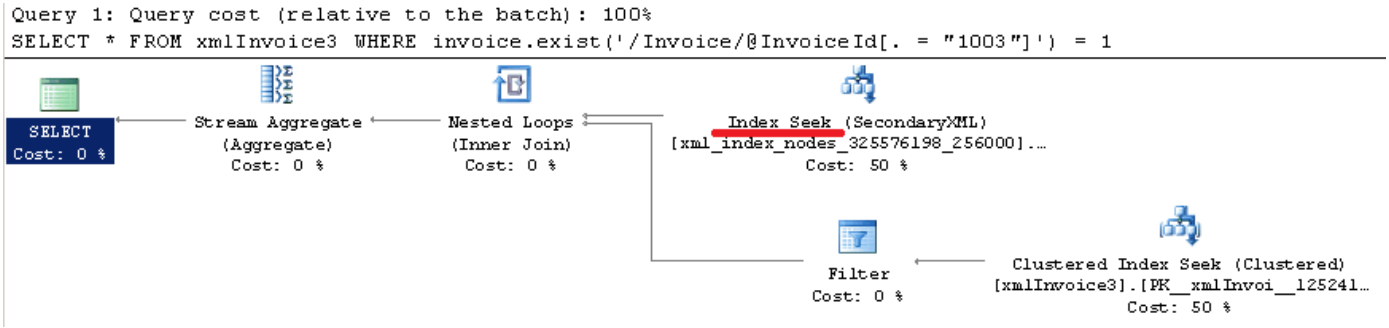
ایندکس‌های XMLایی ثانویه یا secondary XML indexes

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:



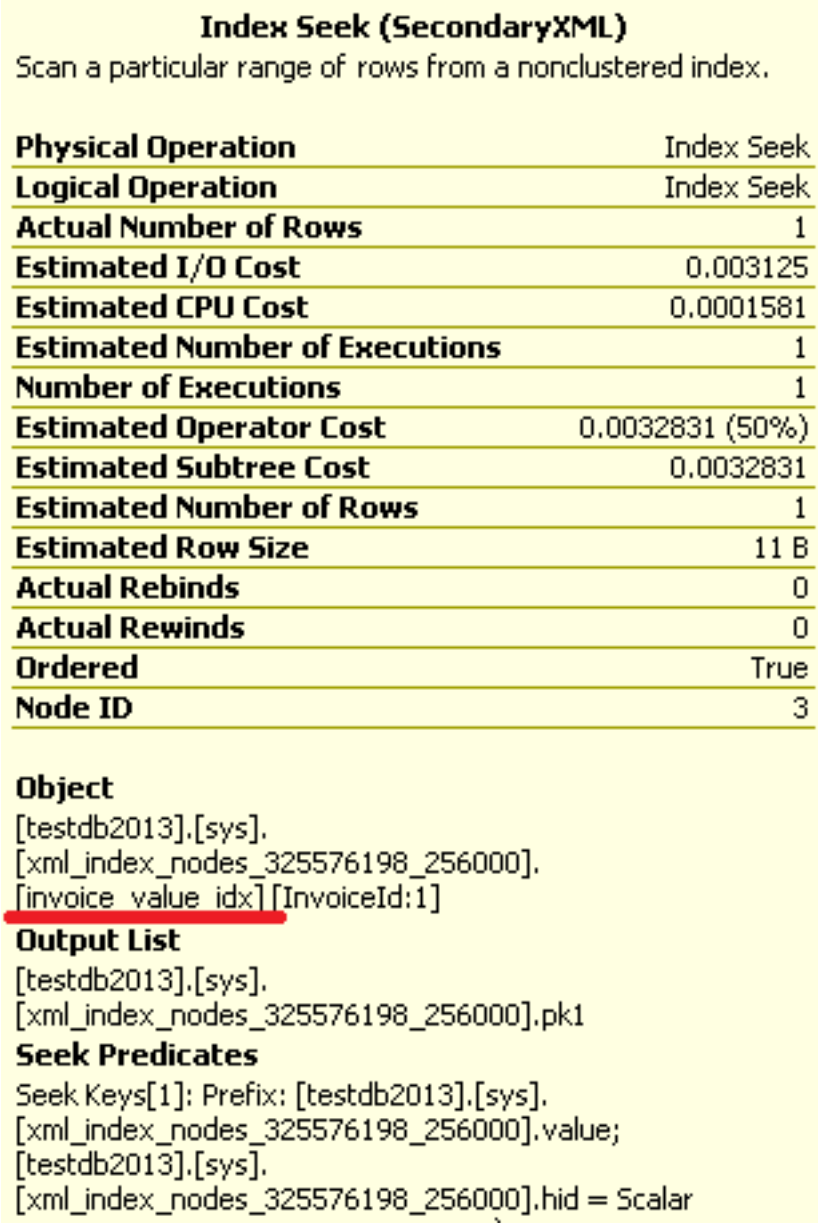
همانطور که مشاهده می‌کنید، نسبت به حالت primary index، وضعیت clustered index seek به index seek تغییر کرده‌است و همچنین دقیقا مشخص است که از کدام ایندکس استفاده شده‌است.
در ادامه دو نوع ایندکس دیگر را نیز ایجاد می‌کنیم:

```
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
)
```