اطمینان از اینکه همه ی جدول ها یک Clustered Index تعریف شده دارند

دو نوع ذخیره سازی برای جدول ها در SQL Server وجود دارد، جدول های Heap و جدول های Clustered. جدول های Heap جدول هایی هستند که clustered index روی خود ندارند و جدول های Clustered جدول هایی هستند که دارای clustered index می باشند. دلایل کمی وجود دارند که علت توصیه شدن به اینکه جدول ها دارای clustered index باشند را توضیح می دهند.

توضیحات

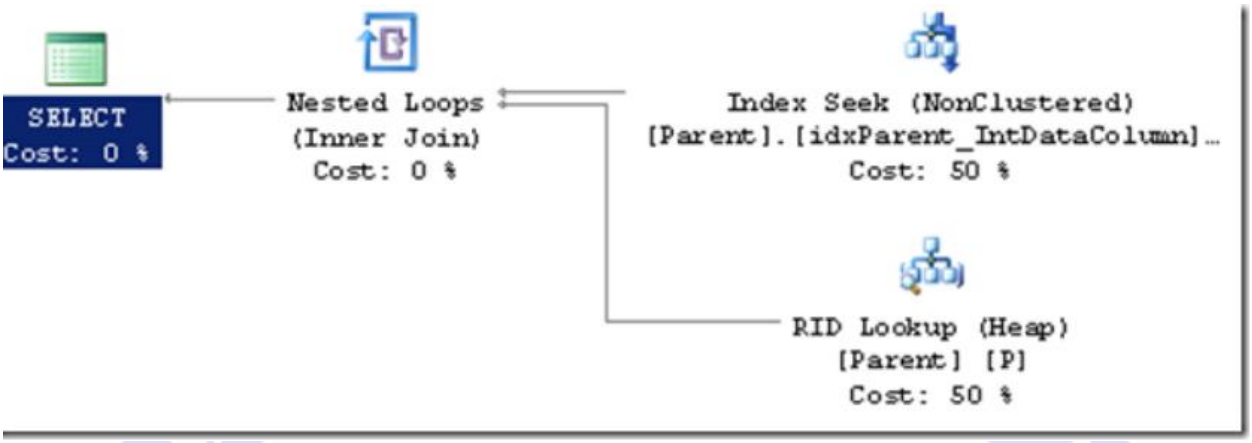
اولین مزیت داشتن یک clustered index روی ستون این است که وقتی query را براساس ستون ایندکس شده انجام می دهید، یک lookup ذخیره می کند، زیرا داده بخشی از ایندکس می باشد. به این علت است که توصیه می شود clustered index خود را روی ستون هایی ایجاد کنید که بیشترین استفاده را در عبارتهای WHERE دارند و نه لزوما روی ستون کلید اصلی اگر تا این حد برای دسترسی به جدول استفاده نمی شود. به عنوان مثال اگر یک جدول OrderDetail دارید که در آن کلید اصلی یک ماهیت ستون OrderDetailID باشد، اما اکثر queryها با استفاده از ستون OrderID به جدول دسترسی دارند، پس اگر clustered index روی ستون OrderID باشد بهتر است زیرا که این مسئله هنگامی که جدول ها در دسترس هستند، قفل های سطح ردیف بیشتری تولید میکند. اجازه بدهید نگاهی به یک query ساده در جدول اصلی داسته باشیم. ابتدا باید یک ایندکس روی یکی از ستون های ثانویه ایجاد کنیم و همچنین clustered index را از این جدول حذف کنیم. در اینجا وضعیت های SQL را مشاهده می کنید که هر دوی این عملکردها را انجام می دهد:

```
ALTER TABLE dbo.Parent DROP CONSTRAINT PK_Parent

ALTER TABLE dbo.Parent ADD CONSTRAINT
 PK_Parent PRIMARY KEY NONCLUSTERED (ParentID)

CREATE NONCLUSTERED INDEX idxParent_IntDataColumn
ON [dbo].[Parent] ([IntDataColumn])
```

و در اینجا query مربوط به جدول اصلی را مشاهده می کنید:

```
SELECT * FROM [dbo].[Parent] P
WHERE P.IntDataColumn=32433
```

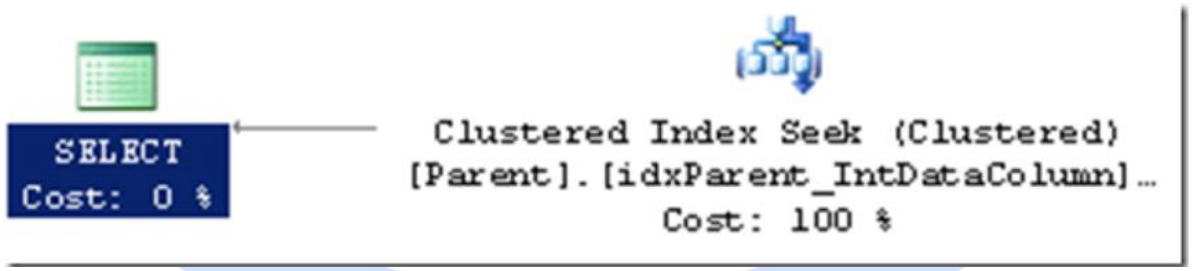
با نگاه کردن به explain plan برای این query متوجه می شویم که SQL Optimizer پس از یافتن رکورد در ایندکس باید یک lookup روی جدول Heap انجام دهد:



اکنون اجازه بدهید این ایندکس را روی InDataColumn به عنوان Clustered Index دوباره ایجاد کنیم. در اینجا وضعیت های SQL را مشاهده می کنید:

```
DROP INDEX Parent.idxParent_IntDataColumn

CREATE CLUSTERED INDEX idxParent_IntDataColumn
ON [dbo].[Parent] ([IntDataColumn])
```

با چک کردن explain plan متوجه می شویم که SQL Optimizer تنها باید یک جستجو برای ایندکس انجام دهد.



با نگاه کردن به SQL Profiler برای این query تایید می کنیم که در واقع داشتن clustered index به SQL Server اجازه می دهد تا query را با استفاده از منابع کمتری اجرا کند، به ویژه تعداد خواندن هایی که برای پردازش داده باید اجرا کند.

Table Type	CPU	Reads	Writes	Duration
Heap	0	7	0	6
Clustered	0	3	0	0

مزیت دوم داشتن clustered index روی جدول این است که راهی را برای سازماندهی مجدد داده ی جدول وقتی که چند بخشی (fragmented)، ارائه می دهد. اجازه بدهید روی جدول خود عملیات آپدیت اجرا کنیم که پس از آن چند بخشی می شود. همچنین جدول را با تنها کلید اولیه ی clustered در وضعیت اولیه ی خود قرار می دهیم تا مشاهده ی نتایج را آسانتر کنیم. در اینجا عبارات SQL برای اجرای این عملکردها را مشاهده می کنید.

```
DROP INDEX Parent.idxParent_IntDataColumn

ALTER TABLE dbo.Parent DROP CONSTRAINT PK_Parent

ALTER TABLE dbo.Parent ADD CONSTRAINT
 PK_Parent PRIMARY KEY CLUSTERED (ParentID)

DECLARE @x BIGINT
DECLARE @y BIGINT
SELECT @x=1
WHILE @x < 100000
BEGIN
  UPDATE [dbo].[Parent]
SET VarcharDataColumn='TESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTEST
TEST
TESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTES
TTEST
TESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTES
TTEST
TESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTES
TTEST
TESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTES
TTEST
TESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTES
TTEST
```

```
TESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTES
TTEST
TESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTES
TTEST
TESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTESTTES
T'+
CAST(@x AS VARCHAR)
   WHERE ParentID=@x
  SELECT @x=@x+1
END
```

ما می توانیم سطح چند بخشی جدول خود را با استفاده از query زیر دوباره چک کنیم.

```
SELECT index_level,avg_fragmentation_in_percent,fragment_count,avg_fragment_size_in_pages,page_
count
 FROM sys.dm_db_index_physical_stats(DB_ID(N'master'), OBJECT_ID(N'dbo.Parent'), NULL, NULL
, 'DETAILED')
```

از نتایج زیر مشاهده می کنیم که پس از اجرای آپدیت بالا، بخش هایی در جدول خود داریم.

index_level	avg_fragmentation_in_percent	fragment_count	avg_fragment_size_in_pages	page_count
0	14.3	3507	6.9	24394
1	5.3	111	1.0	112
2	0	1	1	1

اکنون اگر جدول ما clustered index نداشت، مجبور به ایجاد یک جدول موقت و داده های مرتبط به آن در این جدول خواهیم بود، سپس همه ی ایندکس ها را دوباره ایجاد می کنیم و بعد از آن جدول اصلی را حذف کرده و جدول موقت را تغییر نام می دهیم. همچنین قبل از انجام هرکدام از این عملکردها باید محدودیت های referential integrity (یکپارچگی ارجاعی) را غیرفعال کرده و وقتی کار انجام شد، آنها را دوباره اضافه کنیم. همه ی این عملکردها نیاز به خرابی برای نرم افزار (down time) دارند. از آنجایی که جدول ما دارای یک clustered index می باشد، ما می توانیم به سادگی این ایندکس را برای سازماندهی مجدد داده های جدول بازسازی کنیم. انجام بازسازی مجدد نیاز به زمان خرابی (down time) دارد، اما می توانیم از همه ی مراحل اضافه که به خاطر بارگذاری

مجدد لازم می شود، بگذریم. اگر توانایی آفلاین کردن برنامه برای انجام تعمیر و نگهداری را نداریم، هنگامی که جدول در دسترس باشد، SQL Sever توانایی انجام این عملکرد را به صورت آنلاین تامین می کند. در اینجا وضعیت SQL برای بازسازی آنلاین مشاهده می کنید. (نکته: به سادگی شرط WITH را حذف کرده و یا ON را به جای OFF جایگزین کنید تا یک بازسازی آفلاین منظم اجرا کنید.)

```
ALTER INDEX PK_Parent ON Parent REBUILD WITH (ONLINE=ON)
```

پس از اجرای عبارت بازسازی ایندکس، می توانیم مجددا با استفاده از sys.dm_db_index_physical_stats بخش شدن (fragmentation) را در جدول خود چک کنیم.

index_level	avg_fragmentation_in_percent	fragment_count	avg_fragment_size_in_pages	page_count
0	0.01	18	694.4	12500
1	0	4	7.5	30
2	0	1	1	1