ایندکس چیست؟

* ایندکس لیستی از داده ها است که بر اساس موضوعی خاصی مرتب شده باشند.
* کاربران نمی توانند index ها را ببینند، آنها تنها برای سریعتر کردن جستجوها و query ها کاربرد دارند.

دلایل استفاده ایندکس

* افزایش سرعت جستجو در اطلاعات جدول
* بررسی سریع عدم وجود مقدار تکراری درPK / UK   در عملیات درج
* افزایش کارایی عملیات جوین ( حذف Lookup - حافظه مورد نیاز  - Hash Joinکاهش Memory contention )
* افزایش سرعت عملیات گروهبندی (Group By )، مرتب سازی (Sort)، توابع ستونی
* افزایش کارایی عملیات حذف و بروزرسانی (Foreign Key)
* افزایش سرعت جستجوی مقدار خاص و یا رنجی از رکوردها (Clustered)
* کاهش مدت زمان Locking  جدول در عملیات ( همزمانیCheckDB )
* و ...

هزینه استفاده از Index

جدول پایگاه داده که دارای یک یا چند Non-Clustered Index است نیاز به حافظه و فضای دیسک سخت بیشتری دارد و به دلیل وجود این نوع Index ها عملیات ورود و تغییرات داده ها در جدول زمان بیشتر خواهد برد. مدیریت داده ها و تراکنشها در جداولی که داده های آنها در حال تغییر است زمانبر خواهد بود. پس در نتیجه وجود Clustered Index هزینه بیشتری نسبت به Non-Clustered Index دارد.

بررسی انواع ایندکس‌ها

* Clustered Index
* Non-Clustered Index
* XML Index
* Spatial Index
* Column Stored Index

نکته:

* اكر جدول شما PK‌داشته باشه به طور خودكار يك Clustered Index‌براي كنترل PK‌ايجاد می شود.
* اگر جدول تون ايندكس نداشته باشه بدترين چيز ممكن اتفاق مي افتاد SQL براي پيدا كردن داده هاي / ركوردهاي مورد نظر شما مجبور كل جدول را Scan (پويش كردن /گشتن براي پيدا كردن داده) كنه كه زمان و IO و... زيادي با توجه حجم داده شما خواهد برد.
* برای هر جدول میتونیم 249 تا ایندکس از این مدل داشته باشیم.

یک دیتابیس عملیاتی با جداول بدون ایندکس، تنها زمانی که تعداد کمی رکورد در جداول وجود داشته باشد، سرعت قابل قبولی در اجرای کوئری ها خواهد داشت اما با افزایش تدریجی رکوردها و کاربران، عملاً با شکست مواجه خواهیم شد. ممکن است شما هم این مورد را تجربه کرده باشید که در زمان توسعه برنامه، کلیه عملیات با سرعت خوبی اجرا می شوند اما بعد از تحویل به مشتری و گذشت یک مدت زمان، با شکایات کندی سرعت از سوی مشتری مواجه شده اید.

در مباحث Performance Tuning خواهیم دید که ایندکس ها قلب و روح بهینه سازی محسوب می شوند. شما به ندرت، کتاب، آموزش و یا دوره را پیدا می کنید که فاقد بحث **ایندکس** باشد. در واقع پس از فراگیری مباحث مقدماتی و تئوریک، از زمان شروع به انجام پروژه و توسعه برنامه، با **ایندکس** سر و کار پیدا خواهید کرد تا فرآیند پشتیبانی و زمانی که برنامه در حال سرویس دهی باشد یعنی از زمانی که جداول را ایجاد می کنید یا نیاز به استفاده از یک تکنولوژی (Partitioning- Full Text Search) دارید تا زمان رفع مشکل کندی سرعت و ... همه و همه مستلزم آشنایی عمیق با **ایندکس ا**ست.

کجا ایندکس بگذاریم :

اساسا ایندکس ها بروی فیلد هایی باید تنظیم شوند که بیشتر مورد جستجو قرار میگیرند. در مثال کلاس درس چون میدانیم که احتمالا جستجو برحسب نام و یا نام خانوادگی دانش آموزان خواهد بود می توانیم بروی این ستون ها ایندکس بگذاریم. اگر جستجو برحسب کد منحصر به فردی باشد که اس کیو ال سرور به هر شاگرد میدهد از انجاییکه این فیلد کلید اصلی بوده و دارای کلاسترد ایندکس می باشد. مسلما سریعترین جستجو را خواهیم داشت. در صورتی که ایندکس گذاری بروی فیلدی همانند (Age) سن دانش آموزان چون بروی آن جستجویی نداریم مفد بنظر نمی رسد و چه بسا باعث کندی بانک اطلاعاتی نیز خواهد شد. چون هنگام درج یک رگورد جدید موتور پایگاه داده بایستی ایندکس هاسی تنظیم شده بروی ستون ها را بروز رسانی Update کند هر چه تعداد بیشتری ایندکس داشته باشیم فرایند بروز رسانی ایندکس ها کند تر خواهد شد پس درج رکورد جدید و یا بروز رسانی اطلاعات فیلد ها کند تر خواهد شد.

بنابراین ایندکس گذاری بایستی در حد نیاز و متعادل انجام شود. نبود ان باعث کندی جستجو شده و زیادی آن نیز باعث کندی فرایند های درج و بروز رسانی رکورد ها میشود .

**Clustered Index**: اين نوع ايندكس ها مانند ديكشنري هستند يعني اينكه داده ها مورد نظر شما را مانند يك ديكشنري مرتب مي كنند. يك جدول هم مي تواند فقط يك نوع از اين ايندكس ها داشته باشد كه به طور معمولا خيلي ها از آن در هنگام ايجاد PK‌ استفاده مي كنند كه ممكن در بعضي مواقع بهنيه نباشد.

**Non-Clustered Index:** اين نوع ايندكس ها مانند ايندكس انتهاي كتاب هستند اگ به انتهاي كتاب مراجعه كنيد مشاهده مي كنيد كه برخي از كلمات در انتهاي كتاب نوشته شده و جلوي آنها رجوع به يك شماره صفحه خاص ذكر شده است.

**Unique**میتوان مشخص کرد که ایندکس Unique باشد یا خیر. در صورت Unique بودن ایندکس، در حقیقتUnique Constraint هم در جدول اِعمال می‌شود.

**ساخت یک Index**

|  |
| --- |
| Create Clustered Index Table\_2\_name ON Table\_2 (name) |

**مثال Index با Unique**

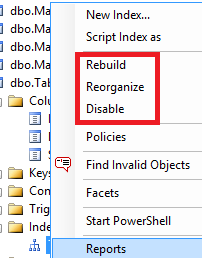
|  |
| --- |
| Create Unique NonClustered Index Table\_2\_name ON Table\_2 (name ASC) |
| Create Unique Clustered Index Table\_2\_name ON Table\_2 (name DESC) |

**جهت نمایش Index های یک جدول**

|  |
| --- |
| select \* from sys.indexes WHERE object\_name(object\_id)='Table\_1' |

**عملیات‌های مربوط به Index**

اگر بر روی یک ایندکس کلیک راست بکنیم، منویی مانند شکل زیر ظاهر می‌شود:



**Rebuild**: این گزینه برای ساخت مجدد ایندکس و حذف فضاهای خالی و سازماندهی مجدد ایندکس به کار می‌رود. در صورت بالا بودن آمار تغییرات جدول، استفاده از این امکان هر چند روز توصیه می‌شود.

**Reorganize**: این آیتم، ایندکس را از نو نمی‌سازد ولی تا حد امکان فضاهای خالی را از بین برده و ایندکس را نسبتاً سازماندهی می‌کند. این روش نسبت به روش قبل از سرعت بالاتری برخوردار است.

**Disable**: برای غیر فعال کردن ایندکس به کار می‌رود.

**جهت بروز رسانی Index**

|  |
| --- |
| ALTER INDEX ALL ON Table\_1 REBUILD WITH (FILLFACTOR = 90) |
| ALTER INDEX Table\_1\_Name ON Table\_1 REBUILD WITH (FILLFACTOR = 90) |

**حذف Index**

|  |
| --- |
| DROP INDEX Table\_1\_Name ON Table\_1 |

**غیر On-Off کردن یک Index**

|  |
| --- |
| ALTER INDEX Table\_1\_Name ON Table\_1 DISABLE |
| ALTER INDEX Table\_1\_Name ON Table\_1 REBUILD |

**تفاوت بین قیود unique و primary key**

از آنجایی که قیود unique و primary key شباهت های بسیاری دارند. برای افراد بسیاری این سوال به وجود می آید که چه تفاوت هایی بین این دو وجود دارد. که در ادامه به آنها اشاره خواهد شد.

**ویژگی ها و شباهت ها**

هر دوی این کلید ها برای به اجرا در آوردن جامعیت داده ها (enforcing data integrity) به کار گرفته می شوند. در حقیقت هر دو، نوعی قید (constraint) می باشند، در واقع قیود مکانیزم استانداردی را برای SQL Server آماده می کنند تا جامعیت داده ها توسط آنها اعمال شود.

نوع ستون parent-key در جدول parent (در یک قید foreign key) حتما باید منحصر بفرد باشد. پس از هر دو کلید یکتا و اولیه می توانیم به عنوان یک parent-key یا referenced key استفاده کنیم.

هر دو برای جلوگیری از وارد شدن نمونه های تکراری (duplicate) مورد استفاده قرار می گیرند. یعنی هر دو برای اعمال uniqueness (یکتایی) ستون یا ستونهایی (در صورت ترکیبی بودن) بکار گرفته می شوند.

زمانی که یک قید کلید اولیه و یا یکتا ایجاد می کنیم یک ایندکس منحصر بفرد (unique index) توسط Database Engine ساخته می شود. از این ایندکس برای سرعت بخشیدن به query هایی که بر اساس این ستونها هستند مورد بهره بری قرار می گیرد.

اگر مقادیر تکراری در جدول درج شده باشند (در ستون یا ترکیب ستونهای مورد نظر) نمی توانیم این قیود را ایجاد کنیم.

هر دوی این قیود می توانند از ترکیب چند ستون ایجاد شوند.

**تفاوت ها**

کلید اصلی (primary key تلفظ می شود "پرایمری کی") اجازه نمی دهد که مقدار NULL جزئی از مقادیر کلید باشد، یعنی ما اجازه نداریم که ستون هایی که در PK شرکت دارند را Nullable در نظر بگیریم. ولی برعکس آن کلید یکتا به ما این امکان را می دهد که مقادیر NULL را برای ستون یا ستونهایی (در صورتی که قید composite/compund باشد) که در UK شرکت دارند درج کنیم. البته مقادیر NULL مثل تمام مقادیر دیگر تنها یکبار قابل درج شدند هستند. مثلا اگر ستون A کلید یکتا و allow null باشد نمی توانیم دو سطر با مقدار NULL در ستون A داشته باشیم.

چندین قید unique می توانیم برای یک جدول ایجاد کنیم (که آن وابسته به محدودیت تعداد شاخصی است که در یک جدول می توانیم ایجاد کنیم) ولی تنها یک قید primary key جدول می تواند داشته باشد.

زمانی که یک قید کلید اولیه ایجاد می کنید، یک unique clustered index روی ستون یا ستونها بطور اتوماتیک ایجاد می شود اگر یک clustered index قبلا ایجاد نشده باشد، بدلیل اینکه هر جدول تنها یک Clustered index می تواند داشته باشد. (هیچ اجباری به clustered index بودن primary key وجود ندارد، شما می توانید ستون یا ستونهای دیگری را به عنوان clustered index انتخاب کنید)

زمانی که یک قید کلید یکتا ایجاد می کنید بطور پیشفرض یک unique index برای اعمال جامعیت و سرعت بخشیدن به جستجو ایجاد می شود نوع این شاخص nonclustered است. بدلیل اینکه هر جدول تنها یک clustered index می تواند داشته باشد آن هم بطور پیشرفض به primary key اختصاص داده می شود.

**نگهداری ایندکس‌ها در اس‌کیوال سرور**

پس از مدتی که از شروع به کار یک سیستم می‌گذرد، همانطور که تعریف ایندکس‌های مفید سرعت جستجوها را بالا می‌برد، ایجاد fragmentation در آن‌ها نیز تاثیر منفی در کارآیی خواهد داشت. به همین منظور نیاز است هر از چندگاهی بررسی شود میزان fragmentation ایندکس‌ها چقدر است. اگر این میزان بیش از 30 درصد بود توصیه شده است که از دستور  [DBCC INDEXDEFRAG](http://msdn.microsoft.com/en-us/library/aa258286%28SQL.80%29.aspx)استفاده شود یا بازسازی مجدد ( [rebuild](http://www.mssqltips.com/tip.asp?tip=1367) ) ایندکس‌ها صورت گیرد.

یکی دیگر از امکانات dmv های اس کیوال سرورهای 2005 به بعد، ارائه آمار میزان fragmentation ایندکس‌ها است که کوئری آن به صورت زیر می‌تواند باشد:

|  |
| --- |
| SELECT OBJECT\_NAME(DMV.object\_id) AS TABLE\_NAME,  SI.name AS INDEX\_NAME,  avg\_fragmentation\_in\_percent AS FRAGMENT\_PERCENT,  DMV.record\_count  FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(), NULL, NULL, NULL, 'SAMPLED') AS DMV  LEFT OUTER JOIN sys.indexes AS SI  ON DMV.object\_id = SI.object\_id  AND DMV.index\_id = SI.index\_id  WHERE avg\_fragmentation\_in\_percent > 10  AND index\_type\_desc IN ( 'CLUSTERED INDEX', 'NONCLUSTERED INDEX' )  AND DMV.record\_count >= 2000  ORDER BY TABLE\_NAME DESC; |

باید در نظر داشت که اجرای این کوئری بر روی یک دیتابیس حجیم زمان‌بر بوده و احتمالا عملکرد سیستم را تحت تاثیر قرار می‌دهد. بنابراین استفاده از آن در خارج از ساعات کاری باید مد نظر باشد. بازسازی ایندکس‌ها نیز به همین صورت است.

برای بازسازی تمامی ایندکس‌های یک دیتابیس مفروض می‌توان از کوئری زیر استفاده کرد:

|  |
| --- |
| DECLARE @TableName VARCHAR(255);  DECLARE @sql NVARCHAR(500);  DECLARE @fillfactor INT;  SET @fillfactor = 80;  DECLARE TableCursor CURSOR FOR  SELECT '[' + OBJECT\_SCHEMA\_NAME([object\_id]) + '].[' + name + ']' AS TableName FROM sys.tables;  OPEN TableCursor;  FETCH NEXT FROM TableCursor INTO @TableName;  WHILE @@FETCH\_STATUS = 0  BEGIN  SET @sql = 'ALTER INDEX ALL ON ' + @TableName + ' REBUILD WITH (FILLFACTOR = ' + CONVERT(VARCHAR(3), @fillfactor) + ')';  EXEC (@sql);  FETCH NEXT FROM TableCursor INTO @TableName;  END;  CLOSE TableCursor;  DEALLOCATE TableCursor; |
| Exec sp\_msforeachtable 'SET QUOTED\_IDENTIFIER ON; ALTER INDEX ALL ON ? REBUILD'  QUOTED\_IDENTIFIER وقتی ON باشه، شما میتوانید از کلمات کلیدیه خود SQL Server بعنوان نام Objectها داخل "" استفاده کنید. و وقتی OFF باشد، میتوانید stringها رو داخل "" قرار بدهید |

**Non-Clustered Indexes**

تقریبا در تمام دیتابیس‌ها به راه‌های دیگری برای دسترسی به داده‌های جداول نیاز خواهد شد که لزوما این داده‌ها براساس ترتیب هنگام ذخیره سازی، مرتب نیستند. در چنین شرایطی ایندکس‌های غیر خوشه‌ای بر سر کار خواهند آمد.

در ادامه الگوهای مختلف ایندکس گذاری مرتبط با ایندکس‌های غیر خوشه‌ای را بررسی کرده و برای هر کدام از آنها مثالی را بررسی خواهیم کرد. خواهیم دید هر ایندکسی که از جانب ما ایجاد می‌شود، نمیتوان مطمئن شد که توسط Sql Server  مورد استفاده قرار می‌گیرد!

این الگو‌ها در تعیین زمان و مکان ساخت ایندکس‌های غیر خوشه‌ای، به ما کمک خواهند کرد که به شرح زیر می‌باشند:

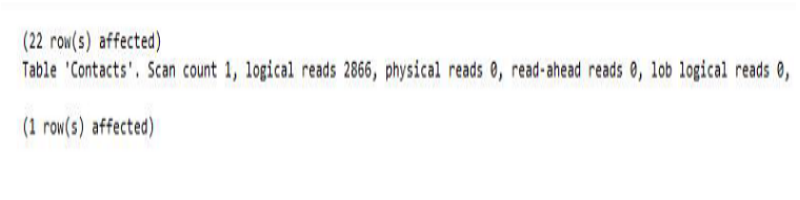
* Search Columns
* Index Intersection
* Multiple Columns
* Covering Indexes
* Included Columns
* Filterd Indexes
* Foreign Keys

**Search Columns**

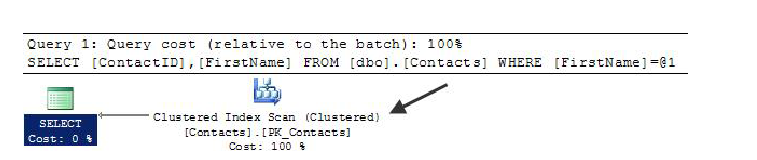
یکی از الگوهای اولیه‌، ساخت ایندکس‌های غیر خوشه‌ای براساس الگوهای جستجوی تعریف شده یا مورد انتظار می‌باشد. این الگو با اینکه خیلی شناخته شده است ولی گاهی اوقات به راحتی از کنار آن گذشته و از آن چشم پوشی می‌کنیم.

برای مثال اگر قرار است در جدول Contacts جستجویی براساس نام آنها داشته باشید، بهتر است یک ایندکس غیر خوشه‌ای بر روی فیلد نام ایجاد کنید. هدف اصلی از این الگو، کاهش هزینه‌ی Scan کردن دوباره‌ی ایندکس خوشه دار و انتقال این عملیات به ایندکس غیر خوشه داری که مسیر دسترسی مستقیم به دیتا را مهیا می‌کند. به مثال زیر توجه بفرمایید:

|  |
| --- |
| SET STATISTICS IO ON;  SELECT \* FROM dbo.Contacts WHERE FirstName = 'Catherine' AND LastName = 'Cox';  SET STATISTICS IO OFF; |

[](https://www.dotnettips.info/file/image?name=10-996ab9b52a974b3eadeb1d238f309dec.png)

22 رکورد را واکشی کرده است؛ ولی با خواندن 2866 page ! که این تعداد، تمام صفحات موجود در جدول می‌باشد. بنابراین واکشی این تعداد رکورد از کل رکورد‌های موجود در جدول (19000) نیاز به چک کردن همه‌ی صفحات را خواهد داشت که واقعا روش بهینه‌ای نمی‌باشد.

[](https://www.dotnettips.info/file/image?name=11-c8bb90979ff742f0898970b0017b8dff.png)

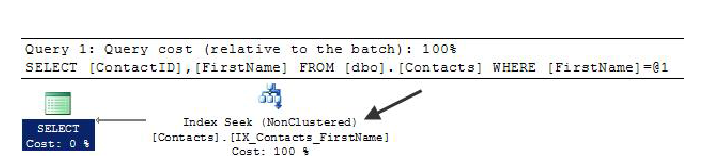
همانطور که در تصویر پلن کوئری بالا هم مشخص است، کل ایندکس خوشه دار ما Scan شده است که هزینه‌ی بالایی خواهد داشت.

حال با کد T-SQL زیر یک ایندکس غیر خوشه دار را بر روی فیلد FirstName ایجاد خواهیم کرد:

|  |
| --- |
| CREATE INDEX IX\_Contacts\_FirstName ON dbo.Contacts(FirstName); |

اگر دوباره کوئری قبلی را اجرا کنیم، به نتایج خیلی بهتری خواهیم رسید و تعداد صفحات خوانده شده به 2 کاهش یافته است!

Sql Server این بار به جای اسکن دوباره‌ی ایندکس خوشه دار، با استفاده از Index Seek و بهره بردن از ایندکس ایجاد شده‌ی توسط ما، یک پلن قابل قبول را برای ما ارائه داده است.

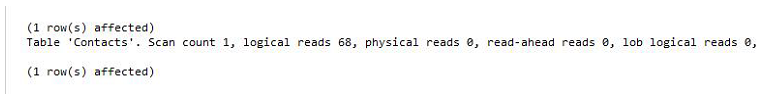
[](https://www.dotnettips.info/file/image?name=13-e2e7a26a029c4539b768acc041cc6b0a.png)

**Index Intersection**

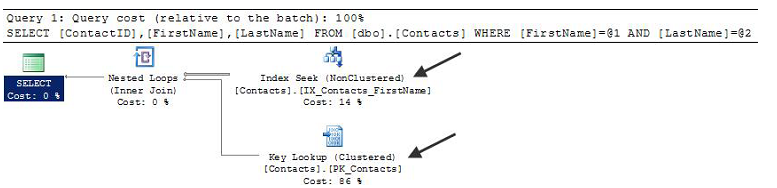
در برخی از سناریوها لازم است یکسری ستون دیگر هم علاوه بر ستونی که ایندکس را بر روی آن تعریف کرده‌ایم، در بخش شرط یا خروجی select استفاده شوند. یکی از راه‌حل‌ها، ایجاد یک ایندکس غیر خوشه‌ای که سایر ستون‌ها را نیز Include می‌کند، می‌باشد. با وجود ایندکس‌هایی که هر کدام از آنها می‌توانند برای ادا کردن بخشی از شروط، نقش ایفا کنند، Sql Server  هم با به کار بردن آنها می‌تواند رکوردهایی که در فصل مشترک حاصل از جسجتوی این ایندکس‌ها بدست آمده را به عنوان خروجی کوئری ما بازگشت دهد. این عملیات Index Intersection نام دارد. به مثال زیر توجه کنید:

|  |
| --- |
| SET STATISTICS IO ON;  SELECT ContactID,FirstName,LastName WHERE FirstName = 'Catherine' AND LastName = 'Cox';  SET STATISTICS IO OFF; |

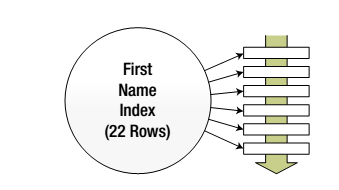
در کوئری بالا علاوه بر FirstName که یک ایندکس غیر خوشه دار را بر روی آن ایجاد کرده‌ایم، فیلد LastName را هم در بخش Select و شرط، مطرح کرده‌ایم. حالا اگر آن را اجرا کنیم، به آمار و پلن زیر دست خواهیم یافت:

[](https://www.dotnettips.info/file/image?name=21-4fee94f233494f96aa57fec4a29e0950.png)

بله تعداد Page‌های خوانده شده این بار به 68 افزایش یافته است که نسبت به حالت بدون LastName که 2 Page خوانده شده بود، زیاد است. همانطور که در پلن زیر مشخص است، به دلیل ایندکسی که برروی FirstName ایجاد کرده‌ایم، نمی‌تواند تمام داده‌های مورد نیاز کوئری را مهیا کند. عملیات Key Lookup و nested loop هم این بار اضافه شده‌اند. Sql Server همچنان استفاده از ایندکس موجود را در کنار Key Lookup از ایندکس خوشه دار، ارزان‌تر از اسکن ایندکس خوشه دار، تشخیص داده است.

[](https://www.dotnettips.info/file/image?name=22-2058a2cfa9e246c7a4d6af1d40eb828f.png)

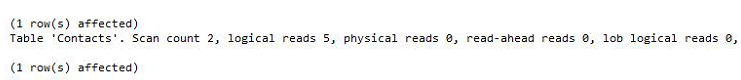
مشکل زمانی گریبان گیر ما خواهد شد که به ازای هر مطابقتی در ایندکس غیر خوشه دار، یک بار به ایندکس خوشه دار برای بررسی شرط بعدی و واکشی دیتا، رجوع خواهد شد. باید دقت کرد که Key Lookup  همیشه به عنوان مشکل مطرح نمی‌شود. ولی باعث افزایش غیرضروری هزینه‌های CPU و I/O برای کوئری خواهد شد.

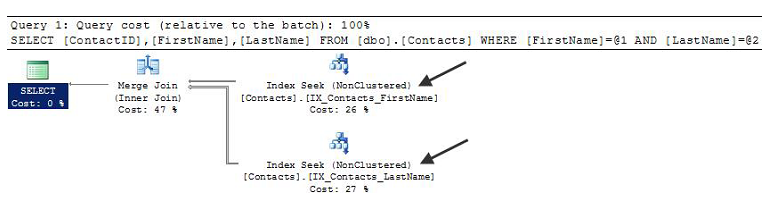
[](https://www.dotnettips.info/file/image?name=23-cad9f2773f674a109182bd2cbbd210a7.png)

برای استفاده از الگوی Index Intersection، یک ایندکس غیر خوشه دار برروی ستون  LastName ایجاد خواهیم کرد:

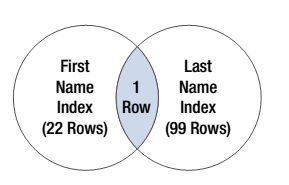
|  |
| --- |
| CREATE INDEX IX\_Contacts\_LastName ON dbo.Contacts(LastName); |

اگر این بار کوئری قبل را اجرا کنیم، به آمار و پلن زیر خواهیم رسید:

[](https://www.dotnettips.info/file/image?name=25-6bd87a8c224d4d0db5b74f8ed4f9e8b7.png)

[](https://www.dotnettips.info/file/image?name=26-5554093b25934cf69fa1db33c38ce31e.png)

بله تعداد Page‌های خوانده شده به 5 کاهش یافته و این بار به جای استفاده از Key Lookup، از دو index seek استفاده کرده است که هزینه‌ای کمتر را نسبت به حالت قبل خواهد داشت. به دلیل اینکه این دو ایندکس تمام دیتای لازم را می‌توانند مهیا کنند، دیگر نیازی به رجوع به ایندکس خوشه دار نخواهد بود. تصویر زیر در درک پلن بالا و این الگو می‌تواند مفید باشد:

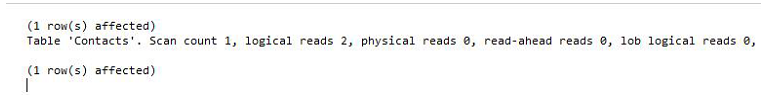
[](https://www.dotnettips.info/file/image?name=24-b833e10af84e4d6fb21f79be391cd5ea.png)

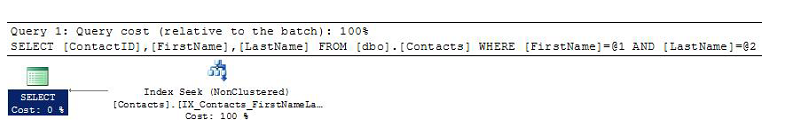
**Multiple Columns**

در دو الگوی قبل، بیشتر به ایجاد ایندکس‌، بر روی یک ستون متمرکز شده بودیم. اگر تعدادی از ستون‌ها در بخش شروط مربوط به کوئری مطرح شوند، بهتر است آنها را در قالب یک ایندکس نگهداری کنیم. برای نشان دادن تأثیر این مورد،  یک ایندکس غیر خوشه دار را بر روی دو ستون ایجاد می‌کنیم:

|  |
| --- |
| CREATE INDEX IX\_Contacts\_FirstNameLastName ON dbo.Contacts(FirstName, LastName);  SET STATISTICS IO ON;  SELECT \* FROM dbo.Contacts WHERE FirstName = 'Catherine' AND LastName = 'Cox';  SET STATISTICS IO OFF; |

با اجرای کوئری بالا به آمار و پلن زیر خواهیم رسید:

[](https://www.dotnettips.info/file/image?name=31-ab07c880396b4870bfcd8c068e682594.png)

[](https://www.dotnettips.info/file/image?name=32-8d02301163ec4da1bc99976a2ed805e0.png)

باید توجه داشت هر زمان که نیاز است یکسری فیلد، در قسمت شرطی خیلی از کوئری‌ها تکرار شوند، ایجاد کردن یک ایندکس برروی آنها به صورت یکجا، ایده‌ی خوبی خواهد بود.

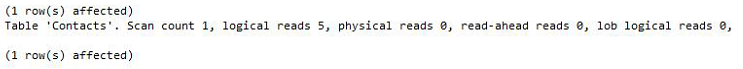
الگوی Multiple Columns هم به مانند الگوی Search Columns باید هنگام ایندکس گذاری دیتابیس در نظر گرفته شود و از اهمیت بالایی برخوردار است. باید توجه داشت اگر فیلدهایی که در قسمت شرطی کوئری مطرح می‌شوند، متغییر باشد، استفاده از الگوی Index Intersection مفید خواهد. ولی برای مواقعی که نیاز است یکسری فیلد به صورت یکجا در بخش شرطی کوئری مطرح شوند، الگوی Multiple Columns کارآیی بهتری خواهد داشت. از این دو الگوی مطرح شده که در تناقض باهم قرار دارند، می‌توان به نحوی استفاده برد تا هزینه‌ی کلی را کاهش داد.

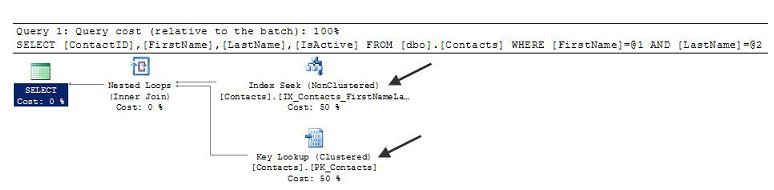
**Covering Index**

الگوی بعدی، ایندکس پوشش دهنده نام گرفته است. همانند نامی که دارد، هدف آن نگهداری یکسری ستون در ستون‌های ایندکس تولیدی که اتفاقا این ستون‌ها در قسمت شرطی کوئری قرار ندارند، ولی قرار است به عنوان خروجی Select برگردانده شوند، می‌باشد.  
این الگو به عنوان یک روش استاندارد ایندکس گذاری در Sql Server مطرح بوده است. البته در ادامه و با بروز شدن روش‌هایی که می‌توان ایندکس‌ها را ایجاد کرد، این الگو نسبت به قبل کمتر مفید است! از آن جهت که یک روش شناخته شده می‌باشد، در این قسمت این مورد را هم مطرح کردیم. به مثال زیر توجه کنید:

|  |
| --- |
| SET STATISTICS IO ON;  SELECT ContactID,FirstName,LastName,IsActive FROM dbo.Contacts WHERE FirstName = 'Catherine' AND LastName = 'Cox';  SET STATISTICS IO OFF; |

در کوئری بالا این بار قصد داریم خصوصیت IsActive را که در ایندکس IX\_Contacts\_FirstNameLastName نگهداری نمی‌شود و همچنین در قسمت شرطی هم مطرح نشده و نیازی به آن نبوده، هم واکشی کنیم. با توجه به نتایج بدست آمده که در آمار و پلن زیر مشخص است، باز هم تعداد Page‌های خوانده شده به 5 افزایش یافته و بار دیگر، Key Lookup و Nested Loop را در کنار یک Index Seek، برروی ایندکسی که با الگوی Multiple Columns ایجاد کرده‌ایم، خواهیم داشت.

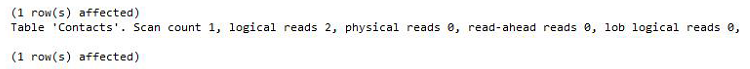
[](https://www.dotnettips.info/file/image?name=41.png)

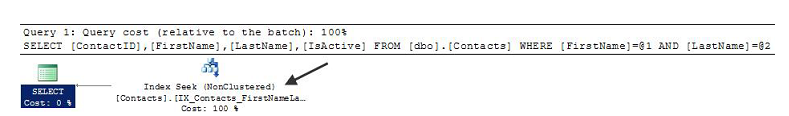
[](https://www.dotnettips.info/file/image?name=42.png)

الگوی index covering پیشنهاد می‌کند ستونی را هم که در قسمت شرطی مطرح نمی‌شود، به عنوان ستونی اصلی در ایندکس، نگهداری کنیم؛ به شکل زیر:

|  |
| --- |
| CREATE INDEX IX\_Contacts\_FirstNameLastNameIsActive ON dbo.Contacts(FirstName, LastName,IsActive) |

ایندکس غیر خوشه دار بالا، 3 فیلدی را که قرار است در بخش شرطی مطرح شوند، یا به عنوان خروجی Select برگردانده شوند، در بر می‌گیرد. سپس کوئری قبلی را دوباره اجرا میکنیم. به نتایج زیر خواهیم رسید:

[](https://www.dotnettips.info/file/image?name=43.png)

[](https://www.dotnettips.info/file/image?name=44-9457f5136e354e0784065651736a8246.png)

باز هم هزینه‌ی Key Lookup حذف شده و این بار از ایندکس جدید ما استفاده شده و تعداد Page‌های خوانده شده هم به 2 کاهش یافته است.  
این الگو در بیشتر سناریو‌ها کاملا مفید بوده و پتانسیل افزایش کارآیی را در بیشتر سناریو‌ها دارد. اما در سال‌های اخیر از زمانیکه امکانات جدیدی در Sql Server 2005 به بعد ایجاد شد، از استفاده‌ی آن کاسته شده است. با وجود این امکانات جدید که در الگوی بعد به آن خواهیم پرداخت، می‌توان ستون‌های اضافی را در ایندکس‌ها، Include کنیم و نیازی نیست که جزء ستون‌های اصلی ایندکس باشند.

**Included Columns**

الگوی Included Columns درواقعا پسر عموی الگوی Covering Index می‌باشد. در این الگو از عبارت INCLUDE در ایجاد یا تغییر ایندکس استفاده می‌شود و از این طریق امکان این را مهیا می‌کند تا یکسری ستون که جز ستون‌های اصلی ایندکس نیستند هم در ایندکس غیر خوشه دار ما افزوده شوند و حتی در قسمت شرطی هم مطرح شوند. این عمل خیلی شبیه به نگهداری دیتا‌های غیر کلیدی در یک ایندکس خوشه دار می‌باشد و این همان تفاوت اصلی بین دو الگو مطرح شده است.

اگر کوئری زیر را اجرا کنیم:

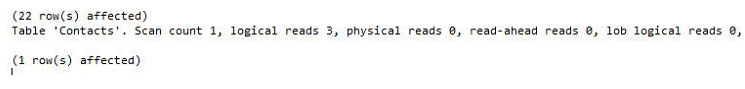
|  |
| --- |
| SET STATISTICS IO ON;  SELECT ContactID,FirstName,LastName,EmailAddress FROM dbo.Contacts WHERE FirstName = 'Catherine';  SET STATISTICS IO OFF; |

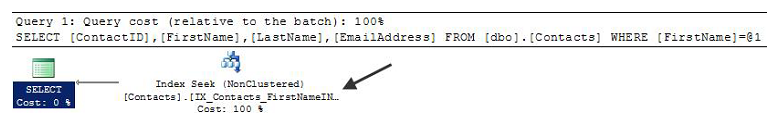
68 Page خوانده شده خواهیم داشت که حاصل یک Index Seek بر روی ایندکس IX\_Contacts\_FirstName می‌باشد و برای واکشی بقیه ستون‌ها هم یک Key Lookup بر روی ایندکس خوشه دار در پلن مشخص خواهد بود.

علاوه بر ایندکس‌های ایجاد شده‌ی در مراحل قبل، حال یک ایندکس غیر خوشه‌ای را با استفاده از الگوی INC ایجاد می‌کنیم:

|  |
| --- |
| CREATE INDEX IX\_Contacts\_FirstNameINC ON dbo.Contacts(FirstName)  INCLUDE (LastName, IsActive, EmailAddress); |

دوباره کوئری قبلی را اگر اجرا کنیم، نتایج به دست آمده، به شرح زیر خواهد بود:

[](https://www.dotnettips.info/file/image?name=51.png)

[](https://www.dotnettips.info/file/image?name=52.png)

این بار از ایندکس جدید ایجاد شده استفاده شده و تعداد Page‌های خوانده شده، به 3 کاهش یافته است. با توجه به انعطاف پذیری این الگو می‌توان از اندک افزایشی که در تعداد Page‌های خوانده شده نسبت به الگوی ایندکس پوشش دهنده وجود دارد، چشم پوشی کرد.  
در مثال‌های قبل چندین ایندکس بر روی جدول Contacts ایجاد کرده‌ایم که 4 مورد از آنها به صورت اختصاصی بر روی فیلد FirstName بوده است. باید توجه کرد این ایندکس‌ها نیاز به فضا و نگهداری در مواقع ویرایش رکورد‌های جدول خواهند داشت. لذا این هزینه‌ها اثر منفی برروی تمام عملیاتی خواهند داشت که روی جدول انجام می‌شود.

الگوی INC می‌تواند این مشکل را برطرف کند. برای مثال با استفاده از آن می‌توان ایندکس‌های تولید شده‌ی در مراحل قبل را بر روی FirstName، توسط یک ایندکس نیز پوشش داد. لذا ایندکس‌های قبلی را حذف کرده و با یکسری کوئری، مشخص خواهیم کرد که گفته‌ی ما صحت دارد:

|  |
| --- |
| IF EXISTS(SELECT \* FROM sys.indexes WHERE object\_id = OBJECT\_ID('dbo.Contacts')  AND name = 'IX\_Contacts\_FirstNameLastName')  DROP INDEX IX\_Contacts\_FirstNameLastName ON dbo.Contacts  GO  IF EXISTS(SELECT \* FROM sys.indexes WHERE object\_id = OBJECT\_ID('dbo.Contacts')  AND name = 'IX\_Contacts\_FirstNameLastNameIsActive')  DROP INDEX IX\_Contacts\_FirstNameLastNameIsActive ON dbo.Contacts  GO  IF EXISTS(SELECT \* FROM sys.indexes WHERE object\_id = OBJECT\_ID('dbo.Contacts')  AND name = 'IX\_Contacts\_FirstName')  DROP INDEX IX\_Contacts\_FirstName ON dbo.Contacts  GO |

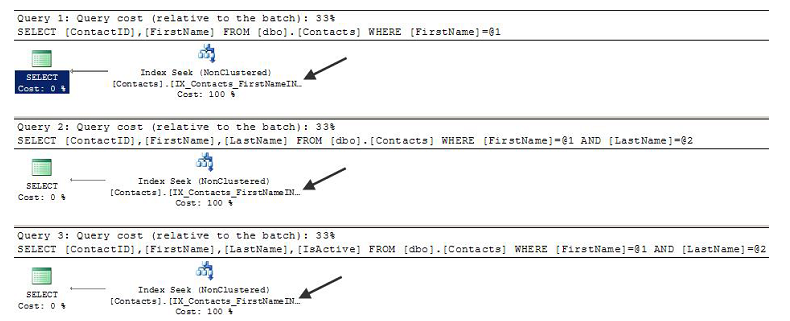
با کدهای بالا ایندکس‌هایی را که بر روی FirstName ایجاد شده بودند، حذف کرده و این بار تمام کوئری‌های مطرح شده‌ی در مراحل قبل را یکبار دیگر اجرا می‌کنیم:

|  |
| --- |
| SET STATISTICS IO ON;  SELECT ContactID,FirstName FROM dbo.Contacts WHERE FirstName = 'Catherine';  SELECT ContactID,FirstName,LastName FROM dbo.Contacts WHERE FirstName = 'Catherine' AND LastName = 'Cox';  SELECT ContactID,FirstName,LastName,IsActive FROM dbo.Contacts WHERE FirstName = 'Catherine' AND LastName = 'Cox';  SET STATISTICS IO OFF; |

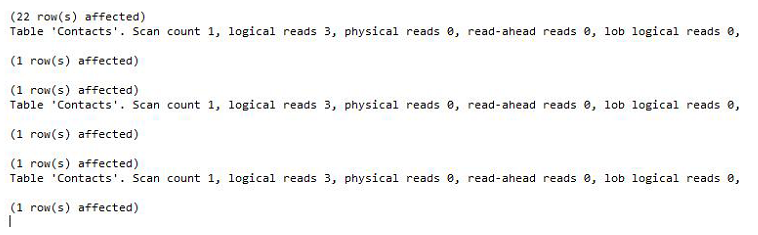
دو نکته‌ای که باید به آنها توجه کرد:

1. کوئری‌ها بالا در مقایسه با الگوهای قبلی به چه شکلی اجرا خواهند شد؟
2. توجه کردن به تعداد Page‌های خوانده شده

در جواب مورد اول، Sql Server از عملیات Index Seek برای فیلترینگ برروی FirstName استفاده کرده و اگر ستون دیگری هم در بخش شرطی کوئری آورده شده، باز هم از این نوع عملیات استفاده شده است. به عنوان مثلا در دو کوئری بعد، LastName هم در بخش شرطی مطرح شده است‌. دلیل این کار که باز هم از Index Seek استفاده می‌شود این است که بعد از اعمال فیلترینگ بر روی FirstName، حالا یکسری رکورد در اختیار داریم که اتفاقا به LastName آنها هم دسترسی هست و فقط رکورد‌ها براساس آن مرتب نشده اند و نیازی نیست به ایندکس خوشه دار دسترسی داشته باشیم. لذا می‌توان همینجا بر روی این فیلد هم فیلترینگ را اعمال کرد. به پلن زیر توجه کنید:

[](https://www.dotnettips.info/file/image?name=53.png)

در جواب مورد دوم، با اینکه حدود 50% افزایش در تعداد Page‌های خوانده شده نسبت به حالتی که به صورت جدا از هم برای هر کوئری خاص یک ایندکس در نظر گرفته بودیم، داشته‌ایم ولی این تغییر کارآیی نمی‌تواند ساخت 4 ایندکس را به جای 1 ایندکس که تمام آنها را پوشش می‌دهد، توجیه کند! در حالیکه ما به کارآیی مورد نظر خود دست یافته‌ایم.

[](https://www.dotnettips.info/file/image?name=54.png)

در نتیجه الگوی INC هنگام ساخت ایندکس‌های غیر خوشه دار خیلی مهم است و باید به آن توجه زیادی کرد. بیشتر در مواقعی‌که نیاز است عملیات Lookup را حذف کنید و سرعت خواندن و کارآیی اجرای کوئری را افزایش دهید، این الگو مناسب خواهد بود. همچنین با کاهش تعداد ایندکس‌ها برای پوشش دادن ایندکس‌های لازم برای کوئری‌ها مشابه، باید توجه کرد که باز هم نسبت به حالتی که هیچ ایندکس غیر خوشه داری ایجاد نشده، کارآیی افزایش می‌یابد.

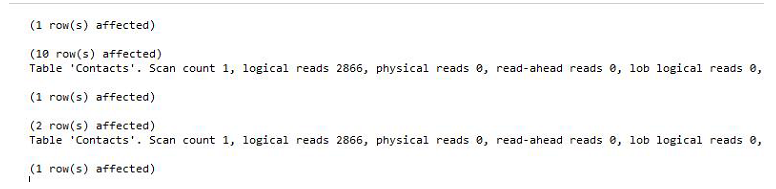
**Filtered Indexes**

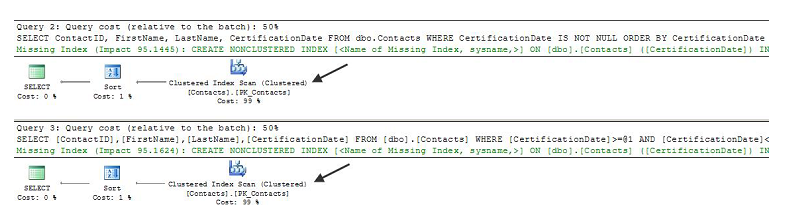
ممکن است در برخی از جداول دیتابیس، یکسری رکوردهایی با مقدار‌هایی که به ندرت یا هرگز از آنها در یک برنامه‌ی کاربردی استفاده نخواهد شد، ذخیره شده باشند. در این مواقع، حذف آنها از نتیجه‌ی خروجی کوئری‌ها می‌تواند خیلی مفید باشد. یا در مواقعی می‌توان از این مورد برای مشخص کردن یک زیر مجموعه‌ی از داده‌های جدول، برای ایجاد ایندکس استفاده کرد. همچنین می‌توان به جای کوئری زدن بر روی میلیون‌ها رکورد موجود در جدول، ایندکس‌ها را طوری ایجاد کرد که پوشش دهنده‌ی بخشی از دیتای چند میلیونی باشند.

بله همانطور که از نام این الگو نیز مشخص است، هدف آن کاهش تعداد رکوردهایی است که در ایندکس نگهداری می‌شوند. به دو کوئری زیر توجه کنید:

|  |
| --- |
| SET STATISTICS IO ON;  SELECT ContactID,FirstName,LastName,CertificationDate FROM dbo.Contacts WHERE CertificationDate IS NOT NULL  ORDER BY CertificationDate;  SELECT ContactID,FirstName,LastName,CertificationDate FROM dbo.Contacts WHERE CertificationDate BETWEEN '20050101' AND '20050201'  ORDER BY CertificationDate;  SET STATISTICS IO OFF; |

در کوئری اول به دنبال رکورد هایی هستیم که CertificationDate آنها نال می‌باشد و در دومی هم به دنبال آنهایی هستیم که در یک بازه‌ی زمانی قرار دارند. از آمار و پلن زیر مشخص است که چون هیچ ایندکس غیر خوشه داری بر روی CertificationDate ایجاد نشده‌است، از Index Scan برروی ایندکس خوشه دار استفاده شده است که حاصل آن خوانده شدن 2866 عدد Page می‌باشد!

[](https://www.dotnettips.info/file/image?name=61.png)

[](https://www.dotnettips.info/file/image?name=62.png)

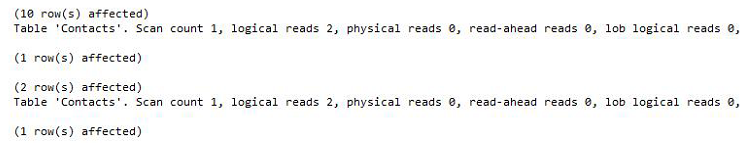
زمانیکه مقدار آن نال باشد، استفاده نخواهد شد. آیا عقل سلیم قبول می‌کند که این مقادیر نال را در ایندکس نگهداری و رکوردهایی با مقادیر نال داشته باشیم؟ برای پیاده سازی این الگو باید از عبارت Where به هنگام ساخت ایندکس‌های غیر خوشه‌ای استفاده کنیم.  
 توجه کنید که امکان استفاده از مقادیر متغیر در بخش Where، وجود ندارد.

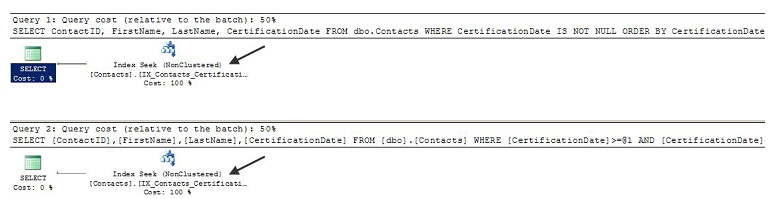
نکته‌ی بعدی این است که نمی‌توان مقایسه‌های پیچیده را در این مورد استفاده کرد. برای مثال استفاده از LIKE و BETWEEN امکان پذیر نیست.

این بار با استفاده از الگوی Filtered Indexes یک ایندکس غیر خوشه‌ای را بر روی ستون CertificationDate ایجاد می‌کنیم:

|  |
| --- |
| CREATE INDEX IX\_Contacts\_CertificationDate ON dbo.Contacts(CertificationDate)  INCLUDE (FirstName, LastName)  WHERE CertificationDate IS NOT NULL; |

حال دوباره دو کوئری قبلی را اجرا می‌کنیم. آمار و پلن زیر نشان می‌دهند که این بار فقط 2 عدد Page خوانده شده است و عملیات به Index Seek بر روی ایندکس جدید تغییر کرده است.

[](https://www.dotnettips.info/file/image?name=63.png)

[](https://www.dotnettips.info/file/image?name=64.png)

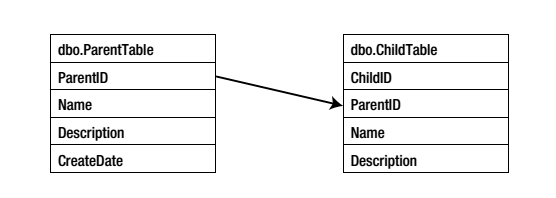
یکسری از مزایای نگهداری فقط زیر مجموعه‌ای از رکوردهای جدول در ایندکس، به شرح زیر است:

* کم شدن تعداد رکورد‌های ایندکس‌ها موجب کاهش تعداد Page‌های مورد نیاز برای ذخیره سازی آنها و در نتیجه کاهش حجم مورد نیاز برای ذخیره سازی خواهد شد.
* با توجه به مورد اول، اگر تعداد Page‌های برای نگهداری ایندکس کم باشند، لذا فرصت Fragmentation برای ایندکس کم خواهد بود و در نتیجه، هزینه و تلاش کمی برای نگهداری آن لازم است.
* زمانیکه تعداد مقادیر نگهداری شده‌ی در ایندکس محدود هستند، تعداد Page هایی که برای پیمایش نیاز است، کم خواهند بود و اینجاست که حتی Index Scan هم بروری آن خیلی بهینه‌تر از Index Scan بر روی ایندکس خوشه دار می‌باشد.
* شرایطی که می‌توان و باید از Filtered Indexes استفاده کرد:
* اگر لازم است بر روی یک ستون که به‌صورت نال‌پذیر است، ایندکس ایجاد کنید(دلایل آن پیش‌تر گفته شد).
* اگر لازم است برروی [Sparse Column](https://www.simple-talk.com/sql/performance/introduction-to-sql-server-filtered-indexes/)، یک ایندکس یکتا ایجاد کنید.
* مورد بعدی همان بحث کاهش تعداد رکوردهایی می‌باشد که در ایندکس ذخیره می‌شوند.

**Foreign Keys**

آخرین الگویی که به آن می‌پردازیم مربوط می‌شود به کلید خارجی. این مورد تنها الگویی است که به طور مستقیم به اشیاء موجود در طراحی دیتابیس مربوط می‌باشد. کلید‌های خارجی گاهی مواقع می‌توانند باعث بروز مشکلی کارآیی شوند، بدون آنکه کسی متوجه این دخالت در کارآیی باشد.

از آنجائیکه کلید خارجی یک قید را بر روی مقادیر مجاز برای یک ستون مهیا می‌کند، لذا یک بررسی برای زمانیکه مقادیر نیاز به اعتبارسنجی دارند، وجود خواهد داشت. این اعتبارسنجی با توجه به شکل زیر دو نوع می‌باشد که به شرح زیر است:

[](https://www.dotnettips.info/file/image?name=71.png)

1. اعتبارسنجی بر روی جدول ParentTable
2. اعتبارسنجی بر روی جدول ChildTable

در مورد نوع اول، هر وقت که رکوردهای جدول ChildTable تغییر کند، در این صورت مقدار ParentID موجود جدول ChildTable با یک جستجو در جدول ParentTable اعتبارسنجی خواهد شد. از آنجایی که این کلید خارجی در جدول ParentTable یک کلید اصلی بوده، یک ایندکس خوشه دار بر روی آن ایجاد شده است و تأثیری در کاهش کارآیی نخواهد داشت.

در مورد نوع دوم، هروقت تغییراتی بر روی  ParentID موجود در جدول ParentTable داشته باشیم، نیاز است اعتبار سنجی بر روی جدول ChildTable انجام شود. برای مثال با حذف یک رکورد در جدول پدر، لازم است که جدول فرزند بررسی کند که آیا این ParentID در رکورد‌ها موجود استفاده شده است یا خیر؟ در این نوع از اعتبارسنجی، الگوی Foreign Key خود را نشان می‌دهد.

برای نشان دادن استفاده‌ی از این الگو، لازم است جداول مطرح شده‌ی در تصویر بالا را ایجاد کنیم:

|  |
| --- |
| CREATE TABLE dbo.Customer (  CustomerID INT ,  FillterData CHAR (1000),  CONSTRAINT PK\_Customer\_CustomerID PRIMARY KEY CLUSTERED (CustomerID)  );  CREATE TABLE dbo.SalesOrderHeader (  SalesOrderID INT ,  OrderDate DATETIME ,  DueDate DATETIME ,  CustomerID INT ,  FillterData CHAR (1000),  CONSTRAINT PK\_SalesOrderHeader\_SalesOrderID PRIMARY KEY CLUSTERED (SalesOrderID),  CONSTRAINT GK\_SalesOrderHeader\_CustomerID\_FROM\_Customer FOREIGN KEY (CustomerID) REFERENCES dbo.Customer (CustomerID)  ); |

کد T-SQL بالا دو جدول مشتری و سفارش را ایجاد کرده و یک ارتباط یک به چند مابین آنها را از سمت مشتری به سفارش ایجاد می‌کند. برای انجام آزمایش خود، یکسری دیتای موجود را هم از جداول دیتابیس AdventureWorks2012 در جداول بالا درج می‌کنیم:

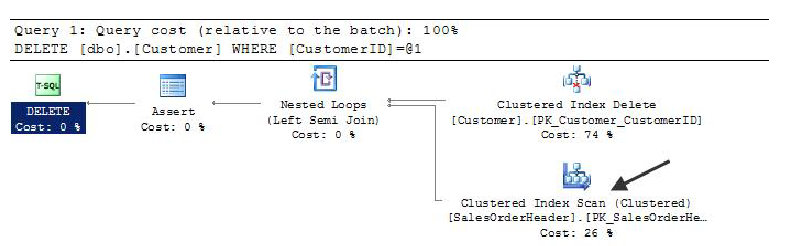
|  |
| --- |
| INSERT INTO dbo.Customer (CustomerID)  SELECT CustomerID FROM Sales.Customer;  INSERT INTO dbo.SalesOrderHeader (SalesOrderID, OrderDate, DueDate, CustomerID)  SELECT SalesOrderID,OrderDate,DueDate,CustomerID FROM Sales.SalesOrderHeader; |

در واقع می‌خواهیم نشان دهیم که در زمان تغییر یک رکورد از جدول Customers، چه اتفاقاتی می‌افتد. برای مثال این تغییر می‌تواند حذف یک رکورد باشد که به شکل زیر آن را انجام خواهیم داد:

|  |
| --- |
| SET STATISTICS IO ON;  DELETE dbo.Customer WHERE CustomerID = 701;  SET STATISTICS IO OFF; |

آمار و پلن زیر نشان می‌دهد که برای حذف یک رکورد در جدول مشتری، چون از عملیات Index Seek برروی ایندکس خوشه دار موجود برروی ستون CustomerID استفاده شده است، تنها 3 Page خوانده شده‌است؛ ولی برای اعتبارسنجی برروی جدول سفارش، با خواندن 4513 page و انجام عملیات Index Scan برروی ایندکس خوشه دار باعث کاهش کارآیی شده است.

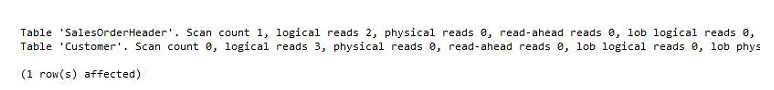
[https://www.dotnettips.info/file/image?name=72.png](https://www.dotnettips.info/file/image?name=72.png)

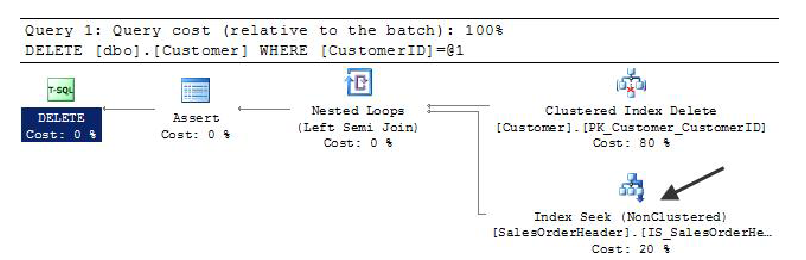
[](https://www.dotnettips.info/file/image?name=73.png)

برای پیاده سازی الگوی کلیدخارجی یک ایندکس غیر خوشه‌ای را بر روی CustomerID در جدول سفارشات ایجاد می‌کنیم:

|  |
| --- |
| CREATE INDEX IS\_SalesOrderHeader\_CustomerID ON dbo.SalesOrderHeader(CustomerID) |

اگر دوباره کوئری بالا را با یک CustomerID دیگر انجام دهیم، به نتایج بهتری دست خواهیم یافت. تعداد Page‌های خوانده شده‌ی برای اعتبارسنجی جدول سفارشات، به عدد 2 کاهش یافته است! و از یک عملیات Index Seek بر روی ایندکس ایجاد شده، استفاده شده است.

[](https://www.dotnettips.info/file/image?name=74.png)

[](https://www.dotnettips.info/file/image?name=75.png)