

# İNDEKSLERLE ÇALIŞMAK

## 8

Bu bölüme kadar, veritabanı programlama adına birçok farklı özellik ve yetenek inceledik. Veritabanı tablolarını oluşturduk. Bu tablolara veriler eklemeyi, güncellemeyi, silmeyi inceledik. Verileri seçerken farklı teknik ve sorgulama tekniklerini inceledik. Ancak, SQL Server'ın bu verileri nasıl depoladığını ve verilere ulaşırken kullandığı yöntemlere değinmedik. Veriler tablolarda tutulur ve seçme, güncelleme, silme gibi komutlar ile yönetilir. Bu işlemler T-SQL geliştiricileri tarafından gerçekleştirilir.

Temel işlemler gerçekleştiren bir T-SQL geliştirici için verinin nasıl depolandığı ve arka planda nasıl yönetildiği çok önemli değildir. Ancak, bir veritabanı yöneticisi ya da ileri seviye bir T-SQL programcısı için verinin nasıl depolandığı, SQL Server'ın bu verilere nasıl ulaştığı, performans için hangi algoritma ve teknikleri kullandığı çok önemlidir.

Hayatın bize öğrettiği bir gerçek vardır. Hangi yolun daha kestirme olduğunu bilmek için, tüm yolların özelliklerini ve teknik bilgilerine hakim olmalısın.

Bu bölümde, SQL Server'ın veriyi daha performanslı kullanabilmek ve sorgu sonuç hızını artırmak için kullandığı İndeks mimarisini inceleyeceğiz.

İndeks kavramı, veritabanı programlamaya yeni başlayanlar için anlaşılması kolay olmayan bir mimari kavramdır. Yeni başlayanlar için şunu söyleyebilirim ki; İndeks mimarisini anlamakta zorlanmak gayet normaldir.

Tüm ilişkisel veritabanı yönetim sistemlerinde var olan İndeks kavramını anlatmak için nesnelleştirmek en iyi yöntemdir. İndeksler gerçek hayatta birçok farklı şekilde karşımıza çıkan ve gerçek hayat sorunlarına bulduğumuz çözümlerden ibarettir. Eskiden, telefon kulübelerinde ve evlerde telefon rehberleri vardı. Bu rehberlerde tüm telefon numaraları yazardı ve çok kalın bir kitaptı diyebilirim. Bu rehberlerde kayıtlı on binlerce telefon numarası arasında istenen kişi ya da kurumun telefon numarasını bulmak oldukça zor ve zaman gerektiren bir işti. Telefon rehberinde yüz bin telefon numarası olduğunu düşünelim. Bu telefon numaralarının belli kurallara göre değil de, karmaşık olarak yazıldığını ve bir telefon numarası aradığınızı hayal edin! Çok zor olmaz mıydı? Telefon rehberindeki numaraları daha hızlı bulmak için bir yol olmalıydı. Bu sorunu çözmek için telefon rehberini üreten firmalar telefonları kişi ya da kurumların isimlerinin baş harfine göre, alfabetik olarak sıralamayı ve harflere göre kategorilendirme yöntemini kullanmaya başladılar. Bu yöntem hem daha anlaşılabilir bir rehber oluşmasını sağladı hem de rehberden telefon arayan kişiler aradıklarını daha hızlı bulabilir oldular.

Veriye hızlı ulaşım sadece bilgisayar dünyasında değil, gerçek hayatta da kullanılan ve gerekli olan önemli bir gereksinimdir. Veritabanı ve İndeks mimarisini anlamak için bir diğer gerçek hayat örneği kütüphane modelidir. Bir kütüphanede binlerce ya da on binlerce kitap olabilir. Bu kitaplar belirli algoritmalara göre düzenlenmeli ve raflardaki yerini almalıdır. Aksi halde ne olur?

Bir kütüphane çalışanı, her yeni gelen kitabı, kütüphanede boş gördüğü raflara rastgele eklediğini düşünelim. 500 yıllık bir tarihi kitap ile yeni basılmış bir edebiyat romanının raflarda yan yana durması ne kadar doğrudur? Kütüphaneden kitap almaya gelen birisi, kütüphaneciden bir kitap istediğinde, kütüphaneci bu kitabı bulabilmek için tüm kitapları taraması gerekir. Ancak, kütüphanedeki her kitap bir düzene göre sıralanmış olsa, yazar isimlerine göre alfabetik olarak, edebiyat, tarih vb. şekilde kategorilendirilmiş ve yıllara göre sıralama yapılsaydı, istenen her kitap hızlı bir şekilde kolaylıkla bulunabilirdi. Bu kütüphanedeki yazar isimlerine göre, tarih ve kategorilere göre sıralama işleminin veritabanındaki karşılığı İndekslerdir. İndeksler, veriye hızlı, performanslı ve kolay ulaşmak için kullanılırlar.

## SQL SERVER DEPOLAMA

SQL Server’da İndeks ve performans kavramlarının anlaşılabilmesi için öncelikle veri depolama mimarisi incelenmelidir. SQL Server’da veriler farklı katmanlar ve nesnelerin birbirlerinin hiyerarşik bir parçası olarak yönetilir. Hiyerarşinin en altındaki katman ile en üstteki katman arasında farklı birçok nesne ve katman vardır. Bu nesnelerin bazılarının üzerinde çalışabildiğimiz için anlaşılması kolaydır. Bazı nesneler ise doğrudan erişilemese de erişilebilir durumdayken, bir kısım nesneler de SQL Server’ın sistemi tarafını ilgilendirdiği için dışarıdan erişime tamamen kapalı ve gizlidir. SQL Server, herhangi bir performans ve iyileştirme yapmadan da performanslı çalışabilecek şekilde tasarlanmıştır. Ancak, mimari açıdan geliştirilen yöntemler ve performans artırıcı nesneler ile SQL Server’dan daha yüksek verim alınabilir.

Bu tür iyileştirmelere, geliştiricinin hazırladığı tablo ve veri yapısını SQL Server’a doğru olarak tanıtarak verinin daha performanslı şekilde yönetilmesi denilebilir.

Veri performansı artırıcı konulara değinmeden önce, SQL Server’ın veri depolama için kullandığı yöntem ve mimari katmanları inceleyelim.

### VERİTABANI

Birçok veritabanında kullanılan bu terim bazı büyük veritabanlarında farklı isimlendirilebilir. Veritabanı, aynı amaç için oluşturulan tablo, veri ve nesneleri kapsayıcı ana nesnedir. Fiziksel anlamda dosyalarda tutuluyor olsa da aslında fiziksel değil, mantıksal bir kavramdır.

SQL Server’da veriler, veritabanı bazında ele alınarak işlenir ve güvenliği sağlanır. Performans için gerekli tüm düzenlemeler de bu ana kapsayıcı nesnenin bir parçasıdır.

### DOSYA

SQL Server, içerisindeki tüm nesne ve verileriyle birlikte fiziksel, yani disk’te bulunan dosyalarda tutulur. Bu dosyalar temel olarak ikiye ayrılır.

- **Veri Dosyası (\*.mdf)**

Veri dosyası, birincil ve vazgeçilmez veritabanı dosyasıdır. Veritabanının tüm nesne ve verilerini içerisinde barındırır. Varsayılan olarak dosya uzantısı .mdf

olsa da bu bir zorunluluk değildir. Farklı uzantılarda olabilir ve sorunsuz çalışır. Ancak, önerilen tabi ki bu dosya uzantısı ile kullanılmasıdır.

Veri katmanı için yapılan iyileştirmeler ve nesneler bu dosya içerisinde saklanır.

#### • Log Dosyası (\*.ldf)

Veritabanının ikincil ve diğer bir dosyası log dosyalarıdır. Log dosyaları **.ldf** (*log data file*) uzantısına sahiptir. Veritabanındaki işlemler için önemli bir yere sahiptir. Log dosyası olmadan veritabanı işlem yapmaz. Birçok bölümde log dosyası üzerinde farklı bazı işlemler gerçekleştirmiştik ve birçok durumda log dosyalarının önemine değışmiştik.

Log dosyaları veritabanı işlemleri için işlem loglarını tutmakla görevlidir. Aslında bazı durumlarda veritabanının can simididir diyebiliriz.

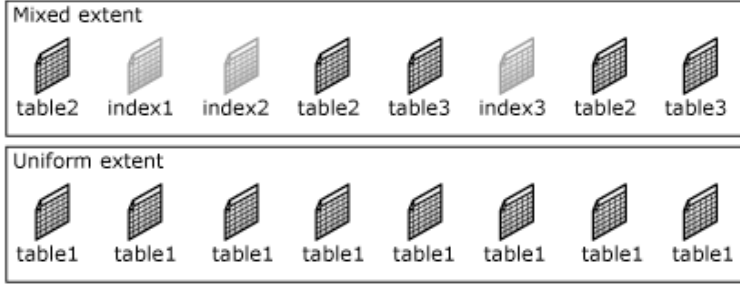
Veri ve log dosyaları birbirini tamamlayan bir bütünün iki parçasıdır. İkisi de gerekli dosyalardır. Büyük verilere sahip veritabanlarında .ndf gibi dosyalar da vardır. Bunlar, daha sonradan veritabanına eklenirler. Ancak temel anlamda en gerekli dosyalar veri ve log dosyalarıdır.

Büyük verileri yönetmek için tasarlanan profesyonel veritabanı uygulamalarında veri güvenliği ve performansı yüksek öneme sahiptir. Öyle ki, bazı durumlarda güvenliği artırmak için, performanstan ödün verilmesi gereken durumlar söz konusu olabilir. Performans ve güvenlik için temel olarak yapılan ilk işlemler gene bu dosyalar üzerinde gerçekleştirilir.

Bir sistemin disk'inin yazma ve okuma hızı standart olarak bellidir. Veri ve log dosyalarının her ikisinde de yazma ve okuma işlemi gerçekleştirilir. Her iki dosyanın da bir disk üzerinde tutulması hem performans hem de güvenlik açısından büyük bir risktir. Disk aynı anda hem ldf hem de mdf dosyalarında yazma ve okuma işlemi gerçekleştireceği için diskin yazma ve okuma hızı otomatik olarak yarı yarıya bölünmüş olacaktır. Bu duruma işletim sistemi ve diğer programların kullandığı yazma ve okuma işlemlerini de ekleyince, performans olumsuz anlamda etkilenecektir. Aynı zamanda, iki dosyanın aynı disk üzerinde tutulması güvenlik riskidir. Disk, sistemin çalışma anında herhangi bir sebepten dolayı arıza yaparsa veri ve log dosyası aynı anda kaybedilmiş olunur. Doğru olan, en azından işletim sistemi, veri dosyası ve log dosyası için ayrı ayrı, üç farklı disk kullanılması ve ayarların buna göre gerçekleştirilmesidir.

## EXTENT

Tablolar ve İndeksler için kullanılan temel depolama birimidir. Yani, veritabanı içinde ayrılan birim alanıdır. Sekiz adet bitişik 64K'lık veri page'lerinden oluşur. Page kavramını detaylıca inceleyeceğiz.



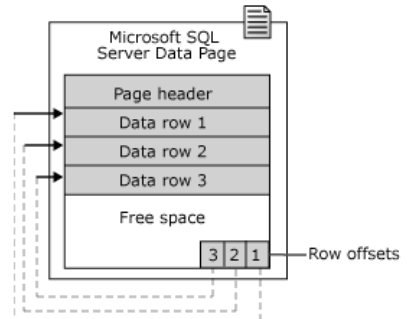
Extent dolduğunda, bir sonraki kayıt, kayıt boyutu kadar yeni bir extent'te yer alır. SQL Server extent'leri belirli durumların gerçekleşmesiyle otomatik olarak ayırır. Bu durumda her extent dolduğunda yeni bir extent oluşturmak yerine, daha önceden oluşturulmuş bir extent'e geçilir. Bu nedenle herhangi bir bekleme olmaz. Diğer extent'ler sıranın kendisine gelmesini bekler.

## PAGE

Page'ler, extent'ler içerisinde bulunan, extent'lerin birim alanlarıdır. Her extent içerisinde 8 page bulunur. Her extent'teki page sayısı aynıdır. Yani her extent sekizlik sistem ile çalışır.

Extent'ler yapısal olarak belirli bir kapsayıcıdır. Extent içerisindeki page'ler veri satırlarını içeren kapsayıcılardır. Her extent'in içerdiği page sayısı sekiz olsa da, her page içerisindeki satır sayısı aynı değildir. Page'ler veri satırlarını kapsayan en alt kapsayıcı olduğu için, veri büyüklüğüne göre değişen satırlara sahiptir. Bir satır, sadece bir page içerisinde olabilir.

Extent ve page'ler ile ilgili açıklamaları daha iyi anlayabilmek için, yukarıdaki extent grafiğini inceleyerek, içerdiği 8 ayrı page'i inceleyebilirsiniz. Her page içerisindeki yöntemi anlamak için de soldaki page grafiğini inceleyebilirsiniz.



İç içe hiyerarşi halinde çalışan bu yapılar, birbirini tamamlayarak SQL Server veritabanı ve içerdiği nesnelerin yönetilmesini sağlar.

Page'ler veriler ile üst hiyerarşi arasında son kapsayıcı katmandır. Tüm veri ve nesneleri doğru ve performanslı yönetebilmek için kendi içerisinde farklı alt parçalara sahiptir.

Verilerin extent ve page'ler içerisindeki yönetimi, hangi page ve extent'lerde boş alan olduğu, bunların sıralaması, İndekslerin page'ler içerisinde saklanması, metin ve BLOB verilerin nasıl depolanacağı gibi kısımlar bu alt page'ler ile yönetilir.

Alt page'lere kısaca değinerek hangi amaç ile kullanıldığını inceleyelim. Bu konular SQL Server mimarisi açısından önemli olsa da derin anlamda bilinmesi gereken konular değildir. Bu nedenle, özetleyerek inceleyeceğiz.

## VERİ PAGE

Verinin saklandığı veri page'dir. Veri page, verilerin tutulması için genel bir page olduğu için, BLOB veri tipindeki verileri de kapsamalı gerekir. BLOB veriler, BLOB page'de saklanır. Ancak BLOB page'deki adresi işaret eden 16 baytlık bir pointer da veri page'de saklanır. Bu sayede, veriler ile BLOB veri arasındaki bağ koparılmamış olur.

BLOB verileri varsayılan olarak büyük veri olarak nitelendirilir. Ancak, eğer metinsel veri page genişliğine sığacak şekilde kısa ise, BLOB veri ile birlikte saklanır. BLOB veri kısa değilse, BLOB page'de saklanır ve 16 baytlık bir pointer ile adresi de veri ile birlikte saklanır.

## İNDEKS PAGE

İndeks verisinin tutulduğu page'lerdir. Clustered İndeksin non-leaf page'lerini ve non-clustered İndeksin leaf ile non-leaf page'lerini içerisinde barındırır.

## BLOB PAGE

**BLOB** (*Binary Large Object*) verileri depolamak için kullanılır. BLOB veriler, çok geniş Text, Ntext veriler ile Image, Video gibi verilerin Binary olarak saklanmasını sağlar. Bu türden verileri işlemeyi ve yönetmeyi **İleri Sorgulama Teknikleri** bölümünde detaylıca incelemiştik. BLOB veriler 2GB'a kadar geniş veriyi depolayabilecek kapasitededir. Page'lerin kapasitesini göz önünde

bulundurunca, bir page'e sığma ihtimali yoktur. Bu nedenle, birden fazla page'de yer alabilecek verilerdir. Page yönetimi karmaşık olduğundan ve anlık olarak page sıralaması değişebileceği için, bu kadar çok page'in ardışık olarak yer alması düşünülemez. Bu nedenle, SQL Server BLOB verilerde, page'lerin ardışık olmasını garanti etmez.

## **GAM, SGAM, PFS**

Kısaltma olarak belirtilen page tiplerinin açılımı aşağıdaki gibidir.

**GAM** (*Global Allocation Map*)

**SGAM** (*Shared Global Allocation Map*)

**PFS** (*Page Free Space*)

Extent ve page'ler verileri tutan yapılardır. Page'lere sürekli veri yazma ihtiyacı olduğu için SQL Server, daha önceden gerektiği ayırdığı extent ve page'lerin boş ve dolu olma durumlarını kontrol eder. Bu kontrol işlemi gerçekleştirerek için de GAM, SGAM ve PFS page'lerini kullanır. Bu page'ler hangi extent ve page'lerin boş ve dolu olduğu bilgisini saklarlar. Bu sayede, SQL Server, hangi extent ve page'lere veri yazılabileceğini bilir.

## **BCM**

SQL Server'da dışarıdan veri alma ve tablonun Truncate edilmesi gibi işlemlere bulk işlemleri denir. Truncate ile ilgili bölümde anlattığımız gibi, bu işlem ile silinen veriler hızlı olarak page'lerden kopararak silinir. Bu nedenle, log tutulmaz ve işlemler geri alınamaz. Log tutma işlemi sadece operasyondan kaynaklı temel olarak gerçekleştirilir. Ancak verileri geri alacak derecede özel bilgiler loglanmaz.

BCM (Bulk Changed Map), bulk işlemleriyle değiştirilmiş büyüklükleri izleyen sayfalar bütünüdür. BCM, değişikliklerin özelliğiyle ilgilenmez ve veritabanı yedekleme işleminde daha fazla seçenek sunulması için kullanılır.

## **DCM**

**DCM** (*Differential Changed Map*)'de BCM gibi yedekleme işlemlerinde yardımcı rolünü oynar. BCM yedekleme işlemleri için kullanılırken, DCM'de alınan son yedekten sonraki değişiklikler için kullanılır.

## PAGE SPLIT

Page'in sınırlarından page konusunda bahsetmiştik. Page dolmaya başladığında, page bölünür. Bu işleme **Page Split** denir.

## SATIRLAR

Satır kavramı, veritabanı tablolarının her bir satırını ifade eder. Her satır maksimum 2044 sütundan oluşabilir. Yukarıda bahsettiğimiz alt yapı mimarisinden dolayı 8060 karakter sınırını geçemezler. Satırlar, bütün olarak tabloları oluşturur. Tablolar ve diğer nesneler ise veritabanını oluşturur.

## İNDEKSLER NERELERDE KULLANILIR?

SQL Server'da İndekslerin performans için kullanıldığından bahsettik. Teorik anlamda öyledir. Ancak, nesnel olarak hangi durumlarda İndeksler kullanılır kısaca bunları inceleyelim.

- En basit tabir ile İndeksler, SQL Server'ın veri ve tablo yapısını doğru tanıması için kullanılır. SQL Server'ın doğru ve hızlı performans sergileyebilmesi için bu özellik bile yeterlidir.
- **Primary Key** ile noktasal sorguları hızlandırmak için kullanılabilir. Bu sorgu tipi, **WHERE** ile gerçekleşir ve tam olarak hangi kaydın istendiği belirtildiği için, veriye en hızlı ulaşılabilen durumlardan biridir.

Örneğin;

---

```
SELECT Name FROM Production.Product WHERE ProductID = 1;
```

---

- Çok örneklediğimiz veri tekrarından kaçınmak için kullanılır. Bu bölümün Unique İndeks kısmında ve Constraint bölümünün Unique Constraint kısmında bu konuyu detaylarıyla inceledik.
- **ORDER BY** sıralama işlemlerinde, sıralamayı daha hızlı yapabilmek için kullanılır.
- Aralık sorgulama işlemlerinde kullanılır. Bu yöntem **WHERE** ile filtrelemeye benzer. Tek farkı, belirlenmiş aralıklardaki kayıtların getirilmesini sağlamaktır.

Bunlar ve daha birçok sebeple İndeksler kullanılabilir.



# İNDEKSLERİ ANLAMAK

İndeks mimarisini incelerken birçok alt yapı ve özellikten bahsettik. İndeksler, veri depolama mimarisinin veriler üzerindeki bir performans parçasıdır diyebiliriz. SQL Server ile verinin doğru yönetilmesi ve hızlı sorgulama gerçekleştirmek için kullanılırlar.

Bu bölüm ve sonrasındaki bölümlerde, İndeksler hakkında teorik bilgilere az girerek, uygulama tarafında İndekslerin oluşturulması ve yönetilmesini inceleyeceğiz.

## CLUSTERED İNDEKS

Clustered İndeks'te tabloda yer alan kayıtlar, fiziksel olarak İndeks tanımlı olan sütuna göre dizilirler. Clustered İndeks'lerin kaydedildiği sayfalar ile gerçek veri aynı seviyededir. Bu nedenle, doğru sütunlar üzerinde oluşturulan clustered İndeksler hızlı sonuç döndürürler.

İndeksler tanımlanmadan önce veritabanı yapısı ve kullanım yoğunluğu hesaplanarak iyi bir analiz süreci geçirilmelidir. Bir clustered İndeks oluşturmak için tablodaki en yoğun sorgu alması ön görülen ya da istatistikler ile en çok sorgu aldığını bilinen sütunlar seçilmelidir. AdventureWorks veritabanındaki Product tablosu üzerinde bir clustered İndeks tanımlamak için ProductID ya da Name sütunu kullanılabilir. Bu iki sütun da yoğun kullanılan sütunlardır. Ürünleri isimlerine göre aramanın yoğun olduğu bir mimari de Name sütunu üzerinde bir clustered İndeks oluşturmak yararlı olabilir. Ancak AdventureWorks veritabanı mimarisinde, en yoğun kullanılan sütun ProductID sütunudur. Bir tabloda sadece tek bir clustered İndeks oluşturulabilir. Bu nedenle, clustered İndeks oluştururken, sorgu ve tablo yapısı iyi analiz edilmeli ve seçici davranılmalıdır.

Clustered İndeks olarak belirlenen sütunların tekil, yani benzersiz değerlere sahip olması gerekir. Constraint işlemlerinden hatırlayacağınız üzere veri üzerinde tekilleştirmeyi sağlamak için Unique Constraint ya da Primary Key Constraint kullanılabilirdi. Bu constraint'ler ile veriyi tekilleştirme sağlanabilir. Performans ve İndeks mimarisi için verinin tekil olması gerekse de, genel kullanımda durum böyle değildir. Genelde tekrarlayan verilerin bulunduğu sütunlar üzerinde, veriye erişimi hızlandırmak için clustered İndeks oluşturulur. SQL Server, bu durumu arka planda çözmektedir. SQL Server, clustered İndeks oluşturulan sütun için 4Byte'lık tekilleştirici (*identifier*) kullanır. Bu sayede,

sütun içerisinde verinin tekilliği sağlanmamış görünse bile, arka planda SQL Server için veri tekil yani benzersizdir.

SQL Server, İndeks ihtiyacını aslında kendisi belirler. Programcı öneri olarak bazı İndeksler oluşturur. Ancak SQL Server, hangi İndeksin, hangi durumlarda ve ne şekilde kullanacağına kendisi karar verir.

## CLUSTERED İNDEKS TARAMASI (SCAN)

Sorgularda herhangi bir koşul yoksa kullanılır. Koşul ve sıralama işlemlerinin olmadığı durumlarda, İndekslere bakılmaksızın tüm tablo içeriği taranarak sonuç döndürülür. Bu yöntem **Table Scan** denir. İndeks kullanılmayan tablolarda da bu tarama yöntemi kullanılır.

## CLUSTERED İNDEKS ARAMASI (SEEK)

Sorgularda **WHERE** gibi bir koşul varsa kullanılır. Amacı, İndeksler üzerinden koşul ile belirtilen kayıtların bulunması ve sorgu içerisinde kullanılacak şekilde verilerin getirilmesini sağlamaktır.

## NON-CLUSTERED İNDEKS

Non-Clustered İndeksler, Clustered İndeksler gibi fiziksel değil, mantıksal olarak dizme işlemi gerçekleştirirler. Non-Clustered İndeksler, Clustered İndekslerin yardımcılarıdır diyebiliriz. Bir tablo üzerinde sadece bir clustered İndeks tanımlanabilirken, non-clustered İndeksten 999 adet tanımlanabilir. Bir Non-Clustered İndeks verilere doğrudan erişemez. Ancak, Heap üzerinden ya da bir Clustered İndeks üzerinden verilere erişebilir.

# SQL SERVER İNDEKS TÜRLERİ

SQL Server'da İndeksler, kullanım alanlarına ve teknik özelliklerine göre farklı türlere ayrılırlar. Bu İndeks türleri aşağıdaki gibidir.

## UNIQUE İNDEKS

Verinin tekilliğini garanti etmek için kullanılır. Örneğin; email ile üyelik gerektiren durumlarda email adresinin benzersiz olması gerekir. Bu senaryoda, tablodaki email sütununu Unique İndeks olarak tanımlamak sütundaki email bilgisinin benzersiz olmasını garanti eder.

Unique İndeks, hem veri tekrarını engeller hem de veri çekme hızını artırır.

## SÜTUNA KAYITLI (COLUMNSTORE) İNDEKS

Columnstore İndeks'ler, her bir sütuna ait verileri aynı sayfaya devam ettirerek sık kullanılan sorgularda performans artışı sağlar. Sık kullanılan sorgularda hızlı okuma gerçekleştirir.

SQL Server 2012 ile birlikte gelen bu yeni İndeks'in veriler üzerinden bir kısıtlaması vardır. Columnstore İndeks tanımlanmış bir tabloda sadece okuma yapılabilir. Tablodaki veri üzerinde ekleme, güncelleme ve silme işlemi gerçekleştirebilmek için İndeks pasifleştirilmelidir (*disable*).

## PARÇALI İNDEKS

İndeksleri farklı fiziksel dosya gruplarına dağıtarak sorgu performansını artırmak için kullanılır. SQL Server 2005 ile gelen bu İndeks türü; Clustered ya da Non-Clustered olabilir.

## EKLENTİ SÜTUNLU İNDEKS

İndeks yapısının en uç sayfalarında gerçek verilerde tutarak sorgu performansını hızlandırmak için kullanılan bu İndeks türü SQL Server 2005 ile birlikte gelmiştir.

Eski tip veri tipleri olan **TEXT**, **NTEXT** ve **IMAGE** türünden sütunlar eklenti sütun olarak kullanılamazlar. Ayrıca, bir eklenti sütunlu İndeks'in boyutu her satır için 900 baytı geçemez.

## XML İNDEKS

XML veriler ile native olarak tam uyumlu olan SQL Server, XML sütunlar için de sorgu performansını artırmak için İndeks oluşturmaya destek verir. XML'in yapısı gereği İndeks kullanımı da biraz farklıdır.

XML İndeks konusu XML bölümünde detaylıca anlatılmaktadır.

## KARMA (COMPOSITE) İNDEKS

16 sütun ya da toplam uzunlukları 900 baytı geçmemek üzere, birden fazla alanı kapsayan İndekstir.

## KAPSAM (COVERING) İNDEKS

Bir sorgunun **WHERE** kısmını da dahil ederek, seçilen sütunlar ile birlikte bir İndeks olarak tanımlanmasına denir. Diğer İndeks yöntemlerine göre farklı bir amaca hizmet ettiği ve kapsadığı verinin karmaşıklığı nedeniyle performans olarak yavaş olsa da, aynı işlemin Kapsam İndeksi kullanılmamış haline göre daha performanslıdır.

## FİLTRELİ İNDEKS

Adından da anlaşılacağı gibi, bir sütundaki tüm kayıtları İndekslemek yerine, sadece belirlenen kurala uya satırları İndekslemek amacı ile kullanılır. Filtrelİ İndeksler, SQL Server 2008 ile birlikte gelmiştir.

## FULL-TEXT İNDEKS

Full-Text Search özelliği için tasarlanan bu İndeks türü sadece **char**, **nchar**, **varchar**, **nvarchar**, **varbinary(max)**, **image** ve **XML** veri tipindeki sütunlar üzerinde tanımlanabilir. Full-Text İndeksler kavramını anlayabilmek için Full-Text Search özelliğini bilmek gerekir.

Özetlemek gerekirse, genellikle arama motoru amacı ile kullanılırlar. Doküman gibi yoğun içeriğe sahip veriler üzerinde dil kurallarından bağımsız hızlı sorgulamalar yapmak için kullanılır.

# İNDEKS OLUŞTURMAK

Buraya kadar, SQL Server depolama ve İndeks mimarisini inceledik. Artık bir İndeks oluşturabilmek için gereken temel bilgilere sahibiz. İndeks oluşturmak için gerekli söz dizimi en temel anlamda basittir. Ancak, İndeks kavramı çok geniş ve bir çok farklı özelliklere sahiptir. Bu nedenle, detaylı söz dizimi biraz karmaşık gelebilir.

Öncelikle basit İndeks oluşturma söz dizimini inceleyeceğiz. Sonrasında detaylı söz dizimi özelliklerini sırasıyla inceleyeceğiz.

### Söz Dizimi:

---

```
CREATE Indeks_tipi INDEX Indeks_ismi
ON tablo_ismi(sutun_ismi)
```

---

Basit söz diziminde her şey açıktır. Nesne oluşturmak için kullanılan **CREATE** ile başlayarak, önce İndeks tipi belirleniyor, daha sonra da bu İndekse bir isim veriyoruz. Bir sonraki satırda ise bu İndeksin hangi tablo üzerindeki hangi sütun için oluşturulacağını belirtiyoruz.

## İNDEKS OLUŞTURURKEN KULLANILAN İFADELER

- **indeks\_tipi**: **CLUSTERED**, **NONCLUSTERED** ya da **UNIQUE CLUSTERED** olarak belirtilir. Tip belirtilmezse varsayılan olarak **NONCLUSTERED** kullanılır.
- **indeks\_ismi**: İndekse verilen isimdir. İndeks tiplerinin baş harflerine göre kısa isimler alabilir (CL gibi).
- **tablo\_ismi**: İndeksin üzerinde tanımlandığı tablo ya da view'in ismi.
- **sutun\_ismi**: Tablo ya da view'de İndekslenmesi istenen sütun ya da sütunların isimleri.

## AYRINTILI İNDEKS SÖZ DİZİMİNİ ANLAMAK

İndeksler temel anlamda basit olduğu gibi birçok ileri seviye özelliğe sahiptir. Bu özelliklerin bazılarını az kullanabileceğiniz gibi bazılarını muhtemelen hiç kullanmayacaksınız. Ancak, İndekslerin güç ve yeteneklerinin farkında olabilmek için teorik anlamda öğrenilmesinde yarar vardır.

İndekslere ait özellikleri sırasıyla inceleyelim.

### ASC/DESC

İndeksin artarak ya da azalarak sıralanmasını belirtmek için kullanılır. **ASC** (*Ascending*) diğer SQL işlemlerinde olduğu gibi artan şekilde sıralama yapar ve varsayılan özelliktir. **DESC** (*Descending*) ise tersi yönde sıralama yaparak azalan şekilde sıralar.

Bir veriyi artan, diğer veriyi azalan şekilde sıralamak mümkündür. Bu durumda bir veri fiziksel olarak artan şekilde sıralanırken diğer veri fiziksel olarak azalan şekilde sıralanacaktır.

#### Söz Dizimi:

---

[ ASC | DESC ]

---

## INCLUDE

İndeks türlerini anlatırken Kapsam İndekslerinden bahsetmiştik. **INCLUDE**, **Kapsam** (*Covering*) İndeksleri desteklemek amacı ile SQL Server 2005'de söz dizimine eklenmiştir. Bu yöntemde, gerekli veriler zaten İndekste yer aldığı için gerçek veri page'lerine tekrar erişmeye gerek yoktur. Gerçek veri page'lere gidilmeye gerek olmaması I/O performansı açısından faydalı bir özelliktir.

### Söz Dizimi:

---

```
INCLUDE (column [ ,... n ] )
```

---

## WITH

Kendisinden sonra gelen özelliklerin kullanılmasını sağlar.

## PAD\_INDEX

İndeks oluşturulduğunda non-leaf seviye page'lerin yüzdesel olarak nasıl dolu olarak kabul edileceğini belirler.

### Söz Dizimi:

---

```
PAD_INDEX = { ON | OFF }
```

---

## FILLFACTOR

Page'lerin yoğunluğunu ayarlamak için kullanılır. İndeks ilk oluşturulurken, page'ler varsayılan olarak tam doluluk durumundan iki kayıt eksik olarak doldurulur.

Yüzdesel olarak, page'in dolu kabul edilmesi gereken oran belirtilebilir. Bunun için **FILLFACTOR** değerini 1-100 arasında bir değer verilir. Bu değer sadece İndeks oluşturulurken değiştirilebilir. Mevcut bir İndeksin **FILLFACTOR** değeri üzerinde değişiklik yapılamaz.

### Söz Dizimi:

---

```
FILLFACTOR = fillfactor
```

---

## IGNORE\_DUP\_KEY

Unique İndeks ile veri tekrarını önlediğinde, veri ekleme işlemlerinde veri tekrarını sağlayacak bir değer bulunduğunda işlem hata verir. Bir transaction içerisinde bu işlemle karşılaşılırsa veri ekleme işlemi hata vereceği için transaction'da tamamlanamayacak ve sonlandırılacaktır.

Örneğin; transaction içerisindeki bir veri ekleme işleminde böyle bir hatanın oluşması sonucunda, transaction'ın sonlanması istenmeyebilir. **IGNORE\_DUP\_KEY** özelliği; hata mesajının seviyesini düşürerek bir uyarı mesajı halinde verilmesini sağlar. Böylelikle bir hata ile karşılaşmadığı için transaction sonlanmayacak ve devam edecektir. Ancak, transaction sonlanmasa bile veri ekleme işlemi gerçekleştirilemeyecektir. Yani veri ekleme gerçekleşmez, ama transaction'da sonlandırılmaz.

### Söz Dizimi:

---

```
IGNORE_DUP_KEY = { ON | OFF }
```

---

## DROP\_EXISTING

Oluşturulmak istenen bir İndeks adı ile aynı isimde, yeni bir İndeks oluşturulmak istendiğinde eski İndeksi silip yeni İndeksi aynı isimle oluşturur. Gereksiz gibi görülebilir. Ancak, bazı durumlarda İndeksleri silmek hiç kolay olmayacaktır. **DROP\_EXISTING** özelliği bu tür durumlarda performanslı bir şekilde eski İndeksi siler ve aynı isimle yeniden oluşturur.

### Söz Dizimi:

---

```
DROP_EXISTING = { ON | OFF }
```

---

## STATISTIC\_NORECOMPUTE

İndeksler için istatistikler hayati öneme sahiptir. Bir İndeksin performanslı ve doğru yöntemle çalışabilmesi için Query Optimize, İndeksin istatistik bilgilerini kullanır. Bu bilgiler otomatik olarak SQL Server tarafından güncellenir. **STATISTIC\_NORECOMPUTE** özelliği kullanılarak bu otomatik gerçekleşen istatistik güncelleme işlemini kendinizin yapmak istediğinizi belirtmeniz anlamına gelir. Bu özellik kullanılırsa SQL Server istatistik güncelleme işlemini otomatik gerçekleştirmez ve geliştiricinin takip etmesini ve güncellemesini bekler.

Tablodaki veri ve tablo özellikleri değiştiğinde bu istatistiklerin güncellenmesi gerekir. El ile istatistik güncellemek için `UPDATE STATISTICS` komutu çalıştırılmalıdır. İndeksler ile ilgili istatistik konusunu bölümün ilerleyen kısımlarında detaylıca inceleyeceğiz. Ancak, güncelleme işlemini SQL Server'a bırakmanız önerilir.

#### Söz Dizimi:

---

```
STATISTICS_NORECOMPUTE = { ON | OFF }
```

---

## **SORT\_IN\_TEMPDB**

Tempdb ve disk okuma-yazma işlemleriyle ilgili bir özelliktir. İndeksleri barındıran Tempdb, veritabanının bulunduğu fiziksel sürücüden farklı bir yerde depolandığı durumlarda bu özellik kullanılır.

`SORT_IN_TEMPDB` özelliği veritabanı yöneticiliği ile ilgili bir konu olduğu için bu detayları bu kitabın konusu dışındadır.

#### Söz Dizimi:

---

```
SORT_IN_TEMPDB = { ON | OFF }
```

---

## **ONLINE**

`ONLINE` özelliği, tabloyu erişime açmak olarak özetlenebilir. Kullanıcıların İndeks ya da tabloya erişimini engelleyen herhangi bir İndeks oluşturulamamasını sağlar.

Bu özellik, sadece SQL Server Enterprise Edition tarafından etkin olarak kullanılabilir. Daha küçük SQL Server versiyonlarında `ONLINE` özelliğini kullanılabilsede herhangi bir etkisi olmayacaktır.

#### Söz Dizimi:

---

```
ONLINE = { ON | OFF }
```

---



## ALLOW PAGE/PAGE LOCKS

İndeksin kilit biçimlerine izin verip vermeyeceğini belirler. Kilitler ile ilgili bölümde anlatılan tüm konuları incelediğinizde bu özelliğin ne kadar ileri seviye ve ustalık isteyen bir özellik olduğunu anlayabilirsiniz. Bu tür ileri seviye özellikleri kullanmak için veritabanı programlama ve yönetim alanlarında ileri seviye bilgi ve tecrübeye sahip olmanız gerekir. Aksi halde, içinden çıkılmaz bir hal alabilir.

### Söz Dizimi:

---

```
ALLOW_ROW_LOCKS = { ON | OFF }
```

---

## MAXDOP

SQL Server'da her bir işlem için kullanılan ve işlemler için kaç işlemcinin kullanılacağını belirleyen **maksimum paralellik derecesi** adında bir sistem ayarı vardır. **MAXDOP** özelliği ile paralellik derecesi ayarlanabilir.

### Söz Dizimi:

---

```
MAXDOP = paralellik_olcusu
```

---

## ON

Performans ile ilgili ve ileri seviye bir SQL Server konusudur. İndekslerin verinin bulunduğu diskten farklı bir diskte saklanması için kullanılır.

## İNDEKSLER HAKKINDA BİLGİ EDİNMEK

Tablo ve view üzerindeki İndeksleri takip etmek önemlidir. Oluşturulan İndekslerin incelenmesi için SQL Server bir sistem prosedürü kullanır.

### Söz Dizimi:

---

```
sp_helpindex 'tablo_yada_view_ismi'
```

---

**Production.Product** tablosu üzerindeki İndeksleri inceleyerek bilgi edinelim.

```
EXEC sp_helpindex 'Production.Product';
```

	index_name	index_description	index_keys
1	AK_Product_Name	nonclustered, unique located on PRIMARY	Name
2	AK_Product_ProductNumber	nonclustered, unique located on PRIMARY	ProductNumber
3	AK_Product_rowguid	nonclustered, unique located on PRIMARY	rowguid

Sorgu sonucunda gelen sütunlar:

- **index\_name**: İndeksin adı.
- **index\_description**: İndeksin yapısı hakkında oluşturulan açıklama.
- **index\_keys**: İndeksin hangi sütunlar üzerinde oluşturulduğu. İndeks keyleri.

İndeksler hakkında bilgi almak için **sys.indexes** sistem kataloğu da kullanılabilir.

```
SELECT * FROM sys.indexes;
```

object_id	name	index_id	type	type_desc	is_unique	data_space_id	ignore_dup_key	is_primary_key	is_unique_constraint	fill_factor	is_padded	is_disabled	is_hypothetical
1	3	clst	1	1	CLUSTERED	1	1	0	0	0	0	0	0
2	5	clust	1	1	CLUSTERED	1	1	0	0	0	0	0	0
3	6	clst	1	1	CLUSTERED	1	1	0	0	0	0	0	0
4	7	clust	1	1	CLUSTERED	1	1	0	0	0	0	0	0
5	7	nc	2	2	NONCLUSTERED	1	1	0	0	0	0	0	0
6	8	NULL	0	0	HEAP	0	1	0	0	0	0	0	0
7	9	clst	1	1	CLUSTERED	1	1	0	0	0	0	0	0
8	17	cl	1	1	CLUSTERED	1	1	0	0	0	0	0	0
9	17	nc	2	2	NONCLUSTERED	1	1	0	0	0	0	0	0
10	17	nc2	3	2	NONCLUSTERED	1	1	0	0	0	0	0	0

**sys.indexes** ile İndeksler hakkında tüm detaylı bilgi alınabilir.

İndekslerin ayrıntılı söz dizimini inceleyerek güç ve yeteneklerini öğrendik. İndeksler hakkında bilgi edinmeyi ve incelemeyi öğrendik. Şimdi, en temel İndeks söz dizimini kullanarak bir kaç örnek yaparak inceleyelim.

İndeks örneğinde kullanılacak **Personeller** tablosunun yapısı aşağıdaki gibidir.

dbo.	Personeller
Columns	
PersonelID	(int, not null)
KullaniciAd	(varchar(20), not null)
Email	(varchar(50), null)
Sehir	(varchar(30), null)
KayitTarih	(smalldatetime, not null)

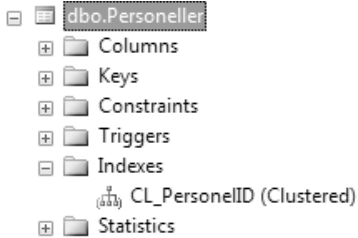
**Personeller** tablosunun **PersonelID** sütunu üzerinde **Clustered Indeks** tanımlayalım.

---

```
CREATE CLUSTERED INDEX CL_PersonelID
ON Personeller(PersonelID);
```

---

İndeks oluşturulduktan sonra, **Personeller** tablosu içerisinde **Indexes** kısmında görülebilir.



Yukarıdaki Indeks, varsayılan Indeks oluşturma yöntemi ile şu şekilde oluşturulabilirdi.

---

```
CREATE INDEX NC_PersonelID
ON Personeller(PersonelID);
```

---

İndeks söz diziminde Indeks tipini açıklarken eğer tip belirtilmezse, varsayılan olarak **NONCLUSTERED** Indeks oluşturulacağı belirtilmişti. **CREATE INDEX** kullanımı varsayılandır. Indeks ismine de dikkat edilirse **NC** yani **NONCLUSTERED**'in baş harflerinden oluşmaktadır.

İndeks söz dizimi ayrıntılarını açıklarken, ilk sırada açıklanan **ASC/DESC** ile sıralı Indeks oluşturma yöntemine de bir örnek verelim.

---

```
CREATE INDEX NC_PersonelID
ON Personeller(PersonelID ASC);
```

---

## UNIQUE İNDEKS OLUŞTURMAK

İndeksteki verilerin tekrarını önlemek için kullanılır ve **UNIQUE** deyimi ile tanımlanır. **UNIQUE** Indeks clustered ya da nonclustered olabilir. Bir Primary Key Constraint ya da Unique Constraint oluşturulduğu zaman, SQL Server ilgili sütun için bir Unique Indeks tanımlar.

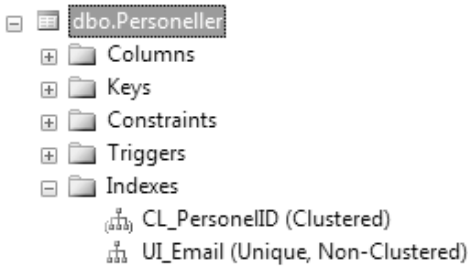
Personellerin email bilgilerini tutuyoruz. Her personel için benzersiz bir email adresi olması gerekir. Veri girişi sırasında kullanıcıdan benzersiz bir email adresi girmesini istiyoruz. Bu nedenle **Email** sütunu üzerinde bir Unique Indeks oluşturulalım.

---

```
CREATE UNIQUE NONCLUSTERED INDEX UI_Email
ON dbo.Personeller (Email)
ON [PRIMARY];
```

---

**UI\_Email** isimli Indeks oluşturulduktan sonra tablodaki **Indexes** kısmının görünümü aşağıdaki gibidir.



Unique Constraint ile Unique Indeks oluşturulabildiği gibi, Primary Key Constraint ile de Unique Indeks oluşturulabilir. Prmiary Key Constraint kullanarak bir Unique Indeks oluşturulalım.

## KAPSAM (COVERING) INDEKS OLUŞTURMAK

Karma Indeks ile ilgili açıklamayı, Indeks Türleri kısmında yapmıştık. Anlaşılması zor olabilecek bir Indeks olduğu için, örnek senaryo üzerinde inceleyelim.

Kitaptaki sorguları genel olarak **Production.Product** tablosu ile gerçekleştirdik. Ayrıca, tüm sütunlarını almak yerine 3 ya da 4 sütunu ile işlem yaptık. Sorgu sayısı yüksek olduğuna göre, gerçek uygulamalarda bu sorguların performansını artırmak için bir Indeks tanımlamak gerekir. Birden fazla sütundan oluşan bir sorgu olduğuna göre en uygun Indeks tipi **Kapsam Indeks** (*Covering Index*) olabilir.

Sorgumuz aşağıdaki gibidir:

---

```
SELECT Name, ProductNumber, ListPrice FROM Production.Product;
```

---

Kapsam İndeksini tanımlayalım.

---

```
CREATE INDEX CV_Product
ON Production.Product (Name, ProductNumber, ListPrice);
```

---

## EKLENTİ SÜTUNLU İNDEKS OLUŞTURMAK

İndekse, bütün sütunları anahtar olarak vermek yerine, sadece arama kriterlerini anahtar olarak verip, geri kalan sütunları eklenti sütun olarak belirleyecek bir İndeks oluşturulabilir.

---

```
CREATE INDEX CV_SalesDetail
ON Sales.SalesOrderDetail (SalesOrderID)
INCLUDE (OrderQty, ProductID, UnitPrice)
```

---

## FİLTRELİ İNDEKS OLUŞTURMAK

SQL Server'da tablonun sadece belirli şartlara uygun satırları üzerinde İndeks tanımlayarak, İndeks boyutunu azaltmak ve sorgu süresini kısaltarak performansı artırmak mümkündür.

Aşağıdaki sorgu ile alınacak filtreli veri için bir İndeks oluşturacağız.

---

```
SELECT ProductID, Name, Color FROM Production.Product
WHERE Color IS NOT NULL
```

---

Seçili filtreli veriyi İndekslemek için aşağıdaki gibi bir İndeks oluşturalım.

---

```
CREATE INDEX FI_Product
ON Production.Product (ProductID, Name)
WHERE Color IS NOT NULL;
```

---

Filtreli İndeksler, sadece **nonclustered** türünden İndeksler için geçerlidir.

## İNDEKS YÖNETİMİ

Doğru İndeks oluşturmak zaman ve tecrübe gerektirir. Bir o kadar, oluşturulan bu İndeksleri yönetmek, istatistikler, İndekslerin yeniden derlenmesi, kapatılması ya da açılması, seçeneklerinin değiştirilmesi de bir o kadar zaman ve tecrübe gerektiren uzmanlıklardır.

Bu bölümde, tüm bu konuları inceleyeceğiz.

## İNDEKSLER ÜSTÜNDE DEĞİŞİKLİK YAPMAK

İndeksler oluşturulduktan sonra veri ve tablo yapısı sürekli değiştiği için belli aralıklarla performans iyileştirilmeleri yapılmalıdır. Bu işlemlerin gerçekleşmesi için kullanılan bazı yöntemler vardır. Bir İndeksin yeniden derlenmesi ya da seçeneklerinin değiştirilmesi gibi işlemler için İndeksler üzerinde değişiklikler yapılmalıdır.

### REBUILD: İNDEKSLERİ YENİDEN DERLEMEK

Bir İndeksi silip yenisini oluşturarak kapladığı alanı azaltmak ve yeniden yapılandırmak için kullanılır.

#### Söz Dizimi:

---

```
ALTER INDEX ALL ON tablo_ismi
REBUILD(secenekler)
```

---

**Production.Product** tablosundaki İndeksi **REBUILD** ile yeniden derleyerek **FILLFACTOR** özelliğini düzenleyelim.

---

```
ALTER INDEX ALL ON Production.Product
REBUILD WITH(FILLFACTOR = 90)
```

---

Aşağıdaki gibi aynı anda birden fazla özellikte düzenleme işleminde seçenek olarak kullanılabilir.

---

```
REBUILD WITH(FILLFACTOR = 90, SORT_IN_TEMPDB = ON)
```

---

**REBUILD** işleminde kullanılabilecek seçenekler aşağıdaki gibidir:

- **FILLFACTOR**
- **SORT\_IN\_TEMPDB**
- **IGNORE\_DUP\_KEY**
- **STATISTICS\_NORECOMPUTE**

## REORGANIZE: İNDEKSLERİ YENİDEN DÜZENLEMEK

Boşalan İndeks sayfalarının atılmasını sağlar. Bu işlem İndeks performansı açısından yararlıdır. **REORGANIZE** deyimi sadece bu işlem için kullanıldığından, sadece kendisiyle ilgili olan **LOB\_COMPACT** seçeneğini kullanabilir.

Unique İndeks olarak oluşturulan **UI\_Email** İndeksini yeniden derleyelim.

---

```
ALTER INDEX UI_Email
ON dbo.Personeller
REORGANIZE WITH (LOB_COMPACT = ON);
```

---

## İNDEKSLERİ KAPATMAK


Çok sık rastlanan durum olmasa da, bazen tablolar üzerindeki İndeksleri bir süreliğine kullanıma kapatmak gerekebilir. Kritik bir konudur. Çünkü tablo üzerindeki Clustered İndeks kapatılırsa, tabloda en basit veri seçme sorgusu dahi kullanılamaz. Nonclustered bir İndeks kapatılırsa veri seçme açısından sorun teşkil etmeyecektir.

**DIJİBİL** veritabanında daha önce oluşturduğumuz **Personeller** tablosunda bir Clustered İndeks tanımlamıştık. Bu İndeksi kapatalım.

---

```
ALTER INDEX CL_PersonelID
ON Personeller
DISABLE
```

---

 **CL\_PersonelID (Clustered)**

CL ile tanımladığımız bir İndeks, yani Clustered İndeks. Bu durumda veri seçme işlemi gerçekleşmemesi gerekiyor. Şimdi **Personeller** tablosunda veri seçme işlemi yapalım.

---

```
SELECT * FROM Personeller;
```

---

Msg 8655, Level 16, State 1, Line 1

The query processor is unable to produce a plan because the index 'CL\_PersonelID' on table or view 'Personeller' is disabled.

Clustered İndeksin kapatılması sonucunda artık veri seçme işlemi için bile bir sorgu çalıştıramıyoruz.

Nonclustered, bir İndeksi kapatırken durum bu kadar katı değildir. **Production.Product** tablosu üzerinde oluşturulan **FI\_Product** isimli Nonclustered İndeksi kapatalım.

---

```
ALTER INDEX FI_Product
ON Production.Product
DISABLE
```

---

#### FI\_Product (Non-Unique, Non-Clustered, Filtered)

Kapatılan İndeksin bulunduğu tabloda veri çekme sorgusu gerçekleştirildiğinde herhangi bir hata vermeden sorgu sonucunu getirdiği görülebilir.

---

```
SELECT * FROM Production.Product;
```

---

Kapatılan İndeksleri açmanız önemlidir. Örneğin; Clustered İndeksi açmazsanız tablo üzerinde işlem yapamazsınız. **Personeller** tablosundaki Clustered İndeks ve **Production.Product** içerisindeki **FI\_Product** İndeksini açalım.

İki şekilde İndeks açılabilir. Bunlar;

---

```
ALTER INDEX CL_PersonelID
ON dbo.Personeller
REBUILD

ALTER INDEX FI_Product
ON Production.Product
REBUILD
```

---

## İNDEKS SEÇENEKLERİNİ DEĞİŞTİRMEK

İndeks söz dizimindeki seçenekleri anlatırken birçok farklı özellikten bahsettik. Bu seçeneklerin ayarları değiştirilebilmektedir.

**FI\_Product** İndeksinin **ALLOW\_ROW\_LOCKS** özelliğini değiştirmeden önce **sys.indexes** içerisinde **ALLOW\_ROW\_LOCKS** sütununun değerine bakalım.

---

```
SELECT Object_ID, Name, Index_ID, Type, type_desc, Allow_Row_Locks
FROM sys.indexes WHERE Name = 'FI_Product';
```

---

	Object_ID	Name	Index_ID	Type	type_desc	Allow_Row_Locks
1	1973582069	FI_Product	12	2	NONCLUSTERED	0



**FI\_Product** İndeksinin **ALLOW\_ROW\_LOCKS** özelliğini değiştirelim.

---

```
ALTER INDEX FI_Product
ON Production.Product
SET(ALLOW_ROW_LOCKS = ON);
```

---

**sys.indexes** içerisindeki **FI\_Product** İndeksinin **ALLOW\_ROW\_LOCKS** özelliği değerine tekrar bakalım.

	Object_ID	Name	Index_ID	Type	type_desc	Allow_Row_Locks
1	1973582069	FI_Product	12	2	NONCLUSTERED	1

Sisteminizde **ALLOW\_ROW\_LOCKS** değeri daha önceden **ON** olarak ayarlı ise, aradaki farkı göremeyebilirsiniz.

Farkı görmek için **SET (ALLOW\_ROW\_LOCKS = OFF)** ile özelliği kapatarak tekrar deneyebilirsiniz.

Bir İndeksin aşağıdaki seçenekleri değiştirilebilir.

- **ALLOW\_PAGE\_LOCKS**
- **ALLOW\_ROW\_LOCKS**
- **STATISTICS\_NORECOMPUTE**
- **IGNORE\_DUP\_KEY**

## İSTATİSTİKLER

SQL Server, sorgu performansı için istatistikler tutar. Bu istatistikleri SQL Server tutmamasını ve el ile takip etmek de sağlanabilir.

Bunun için İndeks özelliklerinden **STATISTICS\_NORECOMPUTE** özelliğini tekrar inceleyebilirsiniz.

İstatistikler diğer veritabanı nesneleri gibi **CREATE** ile oluşturulur. İstatistikler güncellenebilir ve silinebilirler.

## İSTATİSTİK OLUŞTURMAK

İndekslerde olduğu gibi İndekslerde olmayan bir sütun için bile istatistik oluşturulabilir.

**Söz Dizimi:**

---

```
CREATE STATISTICS istatistik_ismi  
ON {tablo_ismi | view_ismi}(sutun_ismi)
```

---

**Production.Product** tablosundaki **ProductID** sütunu için bir istatistik oluşturalım.

---

```
CREATE STATISTICS Statistic_ProductID  
ON Production.Product(ProductID);
```

---

## İSTATİSTİKLERİ SİLMEK

Kullanıcı tarafından oluşturulan istatistikler silinebilirler.

**Söz Dizimi:**

---

```
DROP STATISTICS {tablo_ismi | view_ismi}.(istatistik_ismi)
```

---

İstatistik silebilmek için, istatistik nesnesinin bağlı olduğu tablo ya da view adı ile birlikte belirtilmesi gerekir.

---

```
DROP STATISTICS Production.Product.Statistic_ProductID;
```

---