Veritabanı Programlama
Bölüm 9
Index, Fonksiyon

Dr. Öğr. Üyesi Murat TAŞYÜREK (kayubmprogramlama1@gmail.com)

7 Aralık 2023

Kayseri Üniversitesi, Bilgisayar Mühendisliği Bölümü

Index

- **Index** diğer bir adıyla veritabanı dizini, veri ve dizinin veri yapısını koruyan ve ek depolama alanı maliyetiyle bir veritabanı tablosundaki veri alma işlemlerinin hızını artıran bir veri yapısıdır.
- **Index**, veritabanı arama motorlarının veri sorgulamayı hızlandırmak amacıyla kullandığı özel veri yapılarıdır.
- **Indexleme** işleminde, istenen tablodaki istenilen sütun verilerini işaretlernir.
- **Indexleme** işleminin daha iyi anlamak için kitaplardaki index/içindekiler kısmı göz önüne alınabilir. Bu işlem ile kitabın bütün kısmı aramak yerine direk ilgili adrese gildilir.

Neden index

- Sql'de indexe sahip olmayan tablolara **heap** adı verilir.
- **Heap** bir tabloda bir select çektiğimizde, sql tablodaki kaydı bulabilmek için bütün kayıtları bir bir dolaşır.
- Kaydı bulsa dahi, birden fazla kayıt olma ihtimaline karşı tabloyu dolaşmaya devam eder.
- Bu işlem sql için ciddi bir performans kaybına yol açar. Bu işlemi küçük boyutlu bir tabloda yapmak kolaydır.
- Ancak artan veri miktarına göre bu işlem vakit kaybettirir

Index oluşturma

- Oluşturulan index tablo veya diğer veri tabanı nesneleri gibi görünmez sadece sorguları hızlandırmak için kullanılır.
- İndex veri tabanı tasarımı sırasında veya daha sonra ihtiyaç halinde bir tabloya eklenebilir.
- SQL Primary Key ve SQL Unique özellikleri de bir indextir.
- Veri tabanına yapılan her ekleme, güncelleme ve silme işleminden sonra index yeniden inşa edileceğinden (**rebuild**) fazla kullanılmayan sütunlar için index oluşturmak veya gereğinden fazla index oluşturmak performans düşürür.

Index Oluşturma

- En sade hali

```
CREATE INDEX index1 ON schema1.table1 (column1);
```

- Detaylı hali

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
    ON <object> ( column [ ASC | DESC ] [ ,...n ] )
    [ INCLUDE ( column_name [ ,...n ] ) ]
    [ WHERE <filter_predicate> ]
    [ WITH ( <relational_index_option> [ ,...n ] ) ]
    [ ON { partition_scheme_name ( column_name )
         | filegroup_name
         | default
         }
    ]
    [ FILESTREAM_ON { filestream_filegroup_name | partition_scheme_name | "NULL" } ]

[ ; ]
```

Index Ne Zaman Kullanılmalı

Index genellikle performans açısından fayda sağlar. Ancak, index'lenen sütunun doğru seçilmelidir.

- SQL sorgu süreniz istenilen süreden daha uzun sürüyorsa
- SQL koşulları içerisinde sürekli kullanılan bir sütun var ise
- Sorgulanan sütunda çok farklı değerler var ise ve fazla **NULL** değer içermiyorsa
- Bir veya daha fazla sütun, sıklıkla bir WHERE ifadesiyle veya bir join işlemi ile birlikte kullanılıyorsa (iki sutuna birlikte index tanımlanır)

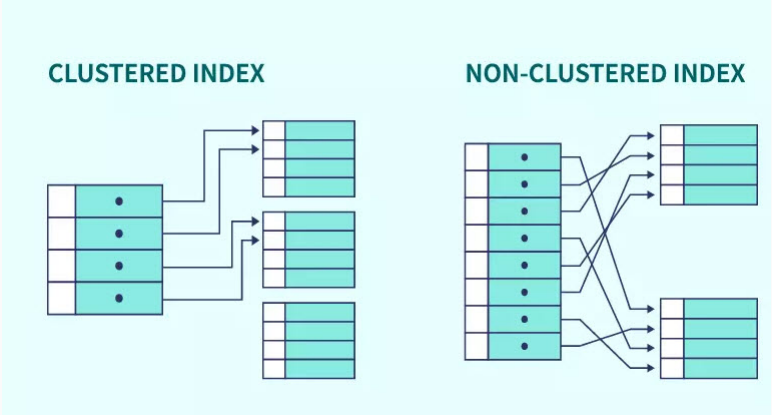
Index Ne Zaman Kullanılmamalı

Index ana amacı veritabanının performansını arttırmaktır ve bazı durumlarda kullanmamak daha faydalı olabilir.

- Küçük tablolar için kullanımı önerilmez
- Sorgu koşulu olarak **sık kullanılmayan sütunlar** için kullanımı önerilmez.
- Çok fazla **NULL** verisi bulunan tablolarda kullanımı önerilmez.
- Yazma oranı, okuma oranından çok daha fazla olan tablolarda önerilmez.
- Sık ya da büyük toplu güncelleme veya ekleme işlemlerine maruz kalan tablolarda kullanımına dikkat edilmelidir.

Index Türü

SQL Server'da **Clustered** ve **Non-Clustered Index** olmak üzere 2 farklı Index yapısı mevcuttur.



Clustred index

- **Clustred index** , veriyi sql'de fiziksel olarak sıraya sokan yapıdır.
- Tablolarımıza tanımladığımız her bir Primary key aslında otomatik olarak bir Clustred index yapısıdır.
- Çünkü tablolarımız bu primary key'ye göre fiziksel olarak sıralanır.
- Her tabloda yalnızca 1 adet clustered index olabilir.
- Tablodaki bir clustered index primary key olabileceği gibi aynı zamanda birden fazla kolonun birleşiminden oluşan bir yapı da olabilir (composite clustered index)

Non-Clustred index

- **Non-Clustered Index** veriyi fiziksel değil mantıksal olarak sıralar.
- Bu index mantığında verinin kendisi değil nerede olduğu bilgisi tutulur.
- Tablo üzerinde en fazla 999 tane non-clustered index tanımlanabilir.
- **Non-clustered index**'ler veriye doğrudan erişemez. Heap üzerinden ya da bir clustered index üzerinden erişebilir.
- Bu index'i oluştururken sorgumuzun koşul kısmında sık kullandığımız kolonlardan oluşturulması gerekir

Index Örnek

- **Sales** şemasındaki **SalesOrderDetail** tablosunda satış detay bilgileri tutulmaktadır.
- Bu tabloda **UnitPrice** değerine göre bir sorgulama yapalım ve Index performansını araştıralım.
- **set statistics time on** komutu t-sql'de istatistik zamanı açar
- **set statistics time off** komutu t-sql'de istatistik zamanı kapatır
- **SalesOrderDetail** tablosunda index olmadan **UnitPrice** sütünü üzerinde sorgulama yapıldığına sonuç

Index olmadan Sonuç

```
set statistics time on

SELECT * FROM Sales.SalesOrderDetail Where UnitPrice between 200 and 1000

set statistics time off
```

100 %

Results | Messages

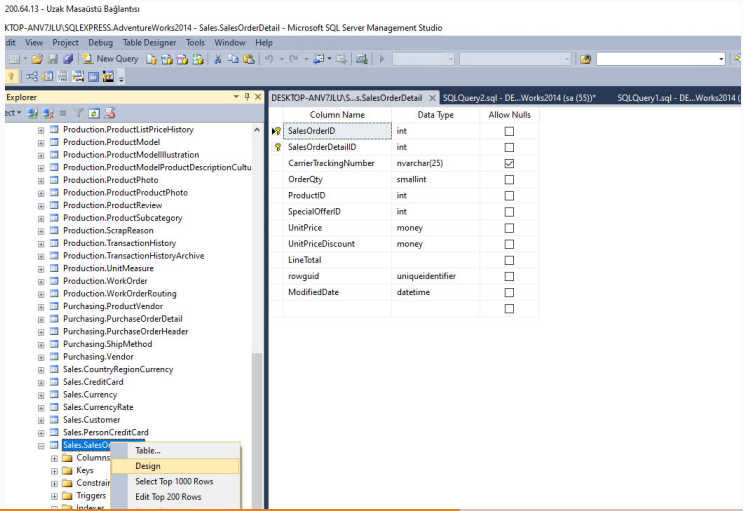
```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

(27138 row(s) affected)

 SQL Server Execution Times:
   CPU time = 31 ms,  elapsed time = 148 ms.
```

Index olmadan Sonuç

- İstatistik verilerinde görüldüğü üzere CPU time ve toplam harcanan zaman gözükmektedir (işletim sisteminde o anda başka işlem çalışırsa bu zaman diliminde değişiklik gösterebilir)
- Aynı sütün üzerine bir index oluşturalım ve ona göre sorguyu tekrar çalıştıralım.
- Sıralı bir veri olmadığından nonclustered index oluşturacağız.
- Index iki şekilde oluşturulur. SSMS üzerinden veya kod üzerinden.
- Her iki örneği de inceleyelim.

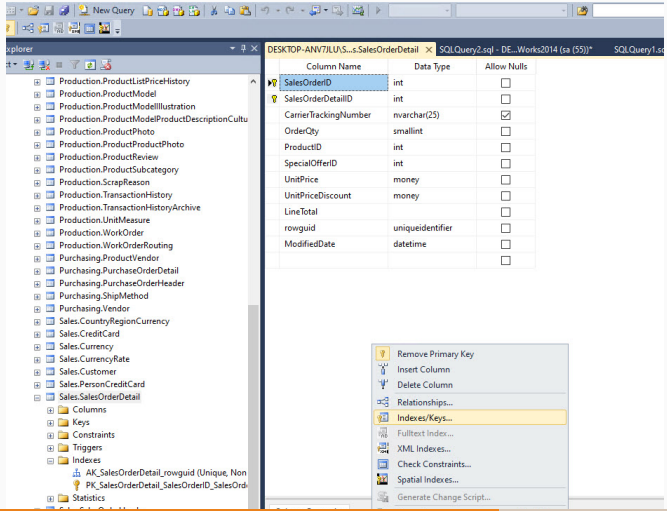
SSMS Index Oluşturma

Index oluşturulmak istenen tablo sağ tıklanarak **Design** menüsü tıklanır.



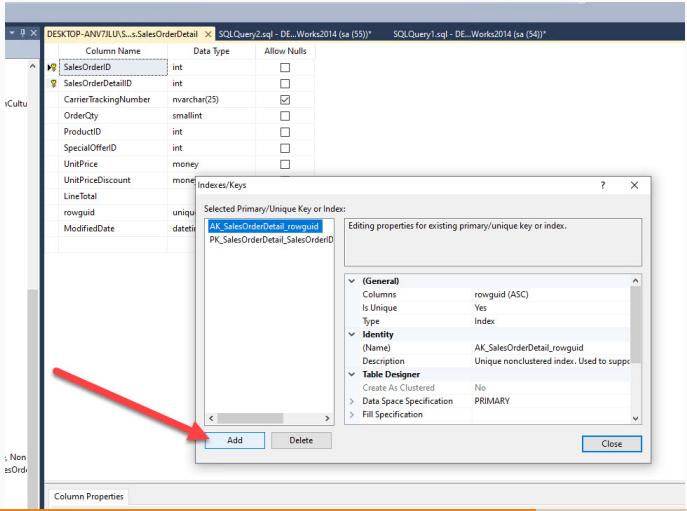
SSMS Index Oluşturma

Açılan pencere sağ tıklanarak **Indexes-Keys** menüsü tıklanır.



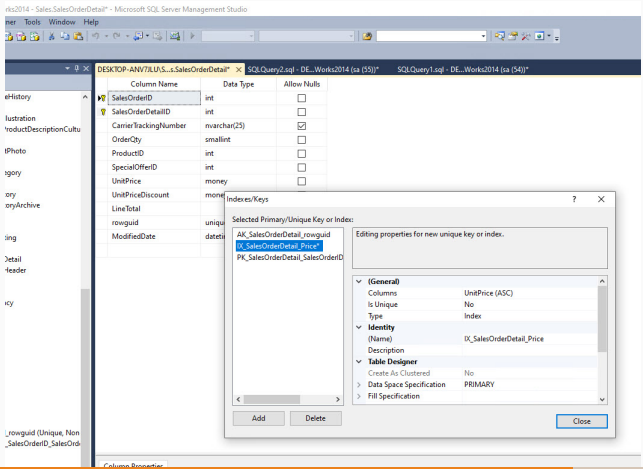
SSMS Index Oluşturma

Add butonuna tıklanarak yeni **Index** oluşturam ekranı açılır.



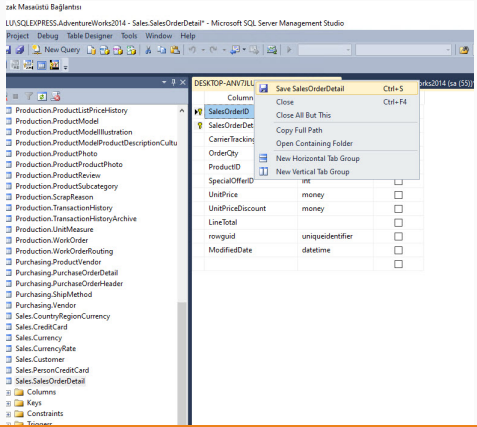
SSMS Index Oluşturma

Index oluşturma ekranda hangi sütün veya sütünlara göre index oluşturlacağı, adının ne olacağı ve diğer özellikler seçilir.



SSMS Index Oluşturma

SSMS tarafından **Index** oluşturulması için tablo üzerinde sağ tıklnarak kaydedilmesi gerekmektedir. Diğer bir ifade ile tabloda değişiklik yapılmıştır ve bunun kaydedilmesi gerekir.



T-SQL Index Oluşturma

İlgili index T-SQL kodu kullanılarak da oluşturabilir.

```
CREATE NONCLUSTERED INDEX IX_SalesOrderDetail_Price
ON Sales.SalesOrderDetail
(
    UnitPrice ASC
)
```

Index olduğunda sorgu sonucu

SQLQuery4.sql - DE...Works2014 (sa (57))* DESKTOP-ANV7JLU\S...s.SalesOrderDetail SQLQuery2.sql - DE...Works2014 (sa (5

```
set statistics time on

SELECT * FROM Sales.SalesOrderDetail Where UnitPrice between 200 and 1000

set statistics time off
```

100 %

Results Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

(27138 row(s) affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 159 ms.
```

Index olan ve olmayan sonuç karşılaştırma

- Indeks olduğunda CPU time'ın çok azaldığını gördük (işletim sisteminde o anda başka işlem çalışırsa bu zaman değerler değişiklik gösterebilir)
- Veritabanınız ve bulunan kaydımızız çok fazla değil.
- Ancak büyük veritabanlarında çalışırken ve soru cevap süresi dakiları aştığında performans farkı daha net ortaya çıkacaktır.
- Index'i olması gerektiği gibi ve ihtiyaç olduğunda oluştururursanız sorgu performansınız artacaktır.

Fonksiyon

- Kullanıcı Tanımlı **Fonksiyonlar** (User Defined Functions), sorgu tekrarlarını önlemek amacı ile iş parçacıkları oluşturmak için kullanılır.
- Kullanıcı tanımlı **fonksiyonlar**, dışarıdan parametre alabilir, IF ELSE gibi akış kontrol ifadeleri içerebilirler. Parse edilir, derlenir ve tampon hafızadan çağrılabilirler.
- **Fonksiyonlar** istenilen deger tipinde dönüs yapabilir. INT, VARCHAR deger döndürebileceginiz gibi bir tablo da döndürebilirsiniz.
- Sql Serverda 4 çeşit fonksiyon kavramı vardır. Scalar-Valued Function, Table-Valued Function, Aggragate Function, System Function

Fonksiyonlar oluşturma

```
-- Transact-SQL Scalar Function Syntax
CREATE [ OR ALTER ] FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type [ NULL ]
 [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
[ ; ]
```

Örnek Fonksiyon

iki sayının toplomını hesaplayan bir fonksiyon oluşturalım ve o fonksiyona değerler gönderelim

```
--Tanımı
CREATE FUNCTION FN_IKISAYI_TOPLA(@a int, @b int)
RETURNS INT
AS
begin
Declare @degisken int
    set @degisken =@a+@b;
    return @degisken
end

--Kullanımı
SELECT dbo.FN_IKISAYI_TOPLA(3,5)
```

161 %

Results | Messages

	(No column name)
1	8

Asal Sayı Fonksiyon

```
ALTER FUNCTION FN_AsalSayiKontrol
(
    @Sayi INT
)
RETURNS VARCHAR(100)
BEGIN
    DECLARE @Rakam INT = 1, @Sayac INT = 0, @Sonuc NVARCHAR(100);

    WHILE (@Sayi >= @Rakam)
    BEGIN
        IF (@Sayi % @Rakam = 0)
        BEGIN
            SET @Sayac = @Sayac + 1;
        END;
        SET @Rakam = @Rakam + 1;
    END;
    set @Sonuc=CONVERT(varchar,@Sayi)
    IF (@Sayac > 2)
    BEGIN
        SET @Sonuc =@Sonuc+ N'-Asal Değil';
    END;
    ELSE
    BEGIN
        SET @Sonuc =@Sonuc+ N'-Asal';
    END;
    RETURN @Sonuc;
END;
```

Asal Sayı Fonksiyon Sonuçlar

```
--Kullanımı

SELECT dbo.FN_AsalSayiKontrol(3);
SELECT dbo.FN_AsalSayiKontrol(58);
SELECT dbo.FN_AsalSayiKontrol(11);
SELECT dbo.FN_AsalSayiKontrol(38);
```

133 %

Results | Messages

	(No column name)
1	3-Asal

	(No column name)
1	58-Asal Degil

	(No column name)
1	11-Asal

	(No column name)
1	38-Asal Degil

Örnek Fonksiyon

- Sales şemasında bulunan SalesOrderDetail tablosunda satış detay bilgileri tutulmaktadır.
- Production şemasında Product tablosunda ürün bilgileri tutulmaktadır.
- Bu iki tablo ProductID ile bir birine bağlıdır.
- Product tablosunda bulunan ürüntlerin toplam satış bedelini getiren fonksiyonu kodlayalım.

Fonksiyon

```
CREATE FUNCTION FN_ToplamSatisTutari
(
    @ProductID INT
)
RETURNS float
BEGIN
    DECLARE @toplamFiyat float;
    SELECT @toplamFiyat=SUM(UnitPrice) FROM Sales.SalesOrderDetail
    where ProductID =@ProductID;

    return @toplamFiyat;
END;
```

Sorgu sonucu

```
--Kullanımı
SELECT Product.ProductID,dbo.FN_ToplamSatisTutari(ProductID) as ToplamSatis
from Production.Product
```

161 %

Results | Messages

	ProductID	ToplamSatis
10	884	37925,8154
11	347	NULL
12	521	NULL
13	396	NULL
14	791	590313,36
15	775	519714,7101
16	848	NULL
17	438	NULL
18	715	57063,1712
19	492	NULL
20	781	1919478,5...
21	679	NULL
22	912	NULL
23	830	6905,448
24	950	4309,032
25	788	144481,0622

Örnek Tablo dönderen Fonksiyon

- Fonskiyonlarda int, varchar değerler dönderebileceğiniz gibi **Tablo** türünde değerler de elde edebilirsiniz.
- Tablo Döndüren Fonksiyon(SQL De temp tablo yada değişken tablo kullanımı bazen performans sorunlarına yol açabilir)
- Kullanımında dikkat edilmesi gerekir.
- Girilen bir ürün ID'sinden daha küçük ürün ID'sine sahip olan ürün Id ve ürün adlarını **Tablo** olarak döndüren fonksiyonu kodlayalım.

Table Fonksiyon

```
CREATE FUNCTION dbo.FN_GETUrunler
(
@UrunID INT
)
RETURNS @ResultTable TABLE
(
    UrunID INT,UrunAdi VARCHAR(50)
) AS BEGIN
        INSERT INTO @ResultTable
            SELECT ProductID, Name FROM Production.Product where ProductID < @UrunID;
RETURN
END
```

Table Fonksiyon Sonucu

```
SELECT * FROM dbo.FN_GETUrunler(3000)
```

161 %

Results | Messages

	UrunID	UrunAdi
1	1	Adjustable Race
2	879	All-Purpose Bike Stand
3	712	AWC Logo Cap
4	3	BB Ball Bearing
5	2	Bearing Ball
6	877	Bike Wash - Dissolver
7	316	Blade
8	843	Cable Lock
9	952	Chain
10	324	Chain Stays
11	322	Chainring
12	320	Chainring Bolts
13	321	Chainring Nut
14	866	Classic Vest, L
15	865	Classic Vest, M
16	864	Classic Vest, S
17	505	Cone-Shaped Race
18	323	Crown Race
19	504	Cup-Shaped Race
20	325	Decal 1