AHMET YESEVİ ÜNİVERSİTESİ
BİLİŞİM SİSTEMLERİ VE MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ YÜKSEK LİSANS
BİLİŞİM SİSTEMLERİ SEMİNERİ

SQL TUNİNG (PERFORMANS DÜZENLEME) TEKNİKLERİ

HAZIRLAYAN

102173019 SUAT ÜSTKAN

suat87@hotmail.com

DANIŞMAN

GÜLSER DONDURMACI

Kasım 2011

Özet

Bu çalışmanın amacı, Sql Server mimarisinde performansı doğrudan etkileyen unsurların incelenmesidir. Sql Index mimarisi, sql sorgularının yazımı, performans çalışmaları için gerekli araçların kullanımına ilişkin ayrı ayrı bilgilendirmelere yer verilmiştir.

Araçtırmamızda Sql Server 2008 ve 2005 database motorları dikkate alınmıştır. Microsoft Sql Server araçları kullanılmıştır.

Çalışmanın sonucunda, Sql mimarisinin çalışma prensiplerinin sorgu yazımını ve performansı direk etkilediği tespit edilmiştir. Bu tespitler, diğer ölçüm araçları ile doğrulukları kanıtlanmıştır.

Anahtar Kelimeler: Performans, Sql, Optimizasyon, Sorgu, Düzenleme

Abstract

The aim of this work is to observe the factors which are influencing directly the performance of development architecture. Informations about Sql Index architecture, writing of sql string, using method of necessary instruments to performance studies are also included.

In our research we considered Sql Server 2008 and 2005 database engines, Microsoft Sql Server instrumets are used.

In conclusion of our work we idetified that the functioning rules of Sql architecture are inluencing directly the performance and query develope.

These anchorages are verified also with other performance reading instruments.

Keywords: Performantion, Sql, Optimization, Query, Tuning

İçindekiler

1. PERFORMANS MİMARİSİ

1.1 Index Mimarisi

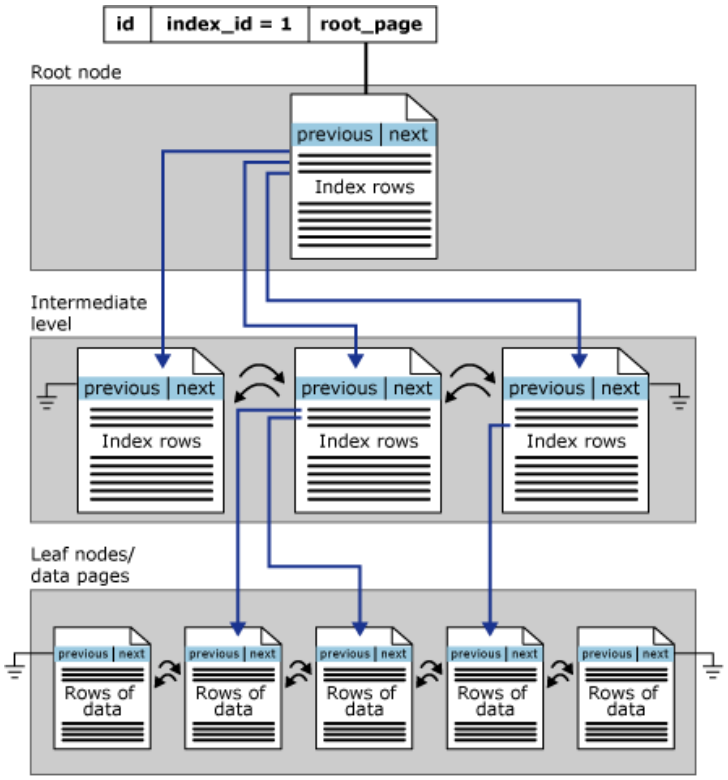
Sql Server' da Indexler B-Tree (Balanced Tree) şeklinde düzenlenmiştir. Bu ağaç yapısı *root node, index node, leaf node* tan oluşmaktadır. Leaf node bu ağaç yapısının en alt düzeyini yani yaprak düzeyini temsil eder.

Leaf Node, data page' ler içerir.
Index Node, Index row'lar içerir.

1.1.1 Clustered Index

Verileri kümesel şekilde sıralanması amacı ile kullanılmaktadır. Depolama alanı üzerinde fiziksel bir sıralama anlamına gelmekte, bundan dolayı bir tabloda bir tane *Clustered Index* oluşturulabilir. Clustered Index bir tabloda istenilen kayıda direk ulaşmayı sağlayacak olan sütuna verilmelidir. Primary Key içeren tablolarda otomatik olarak Clustered Index oluşturulmaktadır. Fakat kullanmış olduğunuz yapıda, primary key kullanmıyor iseniz, istisnasız tek kırılım ile ulaşabileceğiniz sütuna Clustered Index verilmelidir.

Örneğin: Bir telefon defterinde, kayıt edilen kişilere karşılık Identity bir alan üretiliyorsa, bu birincil alana ait bir Clustered Index oluşturmak pointer' ın tüm tabloyu gezmeden index aracılığı ile istediği kaydı direk bulmasını sağlamaktadır.

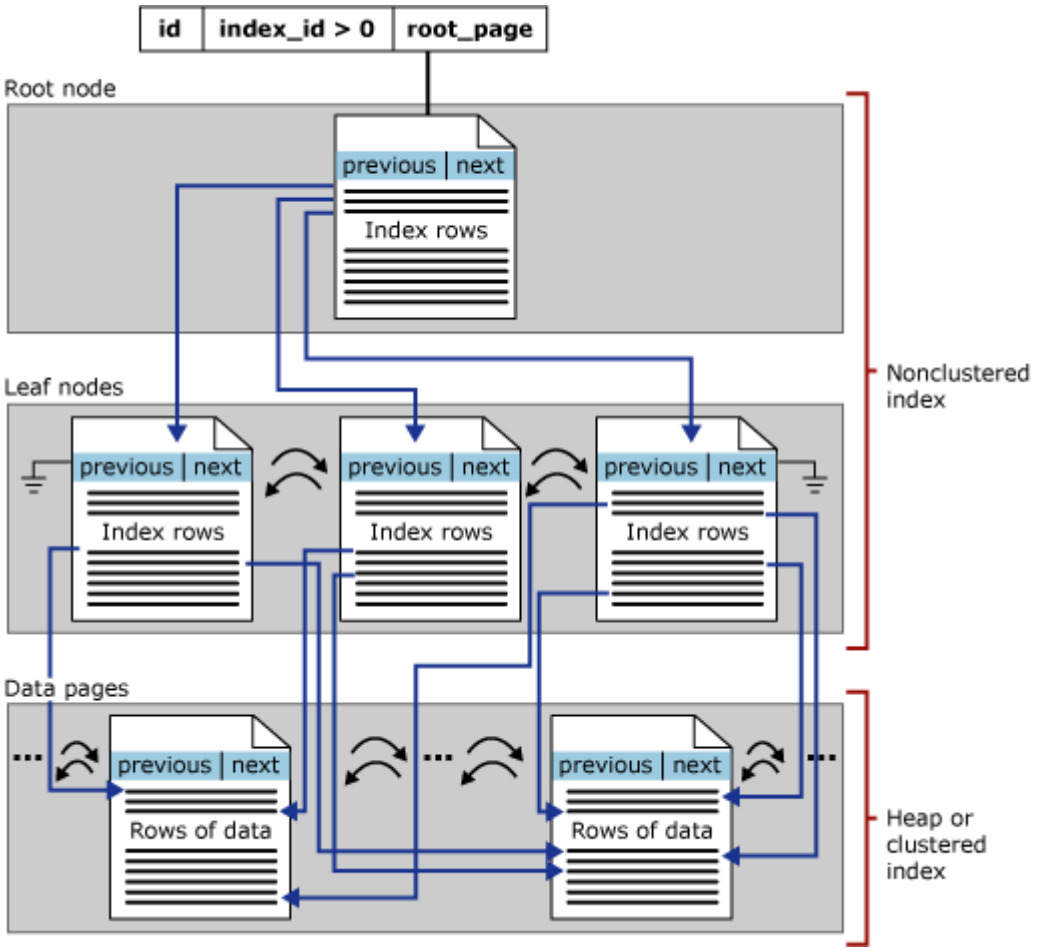


Resim – 1 Clustered Index Mimari Tasarımı

1.1.2 NonClustered Index

Temelde B-Tree yapısını kullanır fakat, Clustered Index' ten ayıran belli başlı özellikleri vardır. Clustered Index' ler stored based (Fiziksel İşaretleme Tabanlı) bir yapı değildir. Mantıksal olarak bir sıralama kullanmaktadır. Index Node' a kadar bir kırılım mevcuttur. Tüm tablo, bir sayfa numarası ve satır numarası olarak anlamlandırılmaktadır. Mantıksal olan bu anlamlandırma NonClustered Index anlamını taşımaktadır.

NonClustered Index yapısında verileri ayırt edici özelliklerine göre sıralamak, ve bu sıralamayı o dataya ulaşabilmek için kolaylaştırıcı olarak kullanılır.



Resim – 2 Clustered Index Mimari Tasarımı

1.1.3 Unique Index

Benzersiz verilerin gruplanması amacı ile kullanılmaktadır. Yinelenen kayıt oluşmasını engellemek amacı ile kullanılmaktadır. Örneğin, bir tabloda OgrenciNo sütunu benzersiz olmalı ve başka hiç bir kayıt için eşit olmamalı durumu var ise, bu sütun için *Unique Index* oluşturulabilir. Bir sütunun benzersiz olması Identity ile yapılabilmekle beraber, özel bir durum içeriyorsa kullanılabilir. Örneğin: OgrenciNo: 0527.320.56 gibi değerler içeriyorsa, böyle bir değeri Auto Identity yapmak mümkün değildir. Unique index tam da bu kullanım içindir. Bu durumda dataya karakteristik bir özellik atanmış olur.

1.2 T-Sql Komutlarının Uygun Hale Getirilmesi

1.2.1 Sabit Disk İstatistikleri:

Sorgu disk performanslarını gözlemlemek için “SET STATISTICS IO ON” cümleciği ile istatistiklerin görüntülenebildiği ve bu işlem bittikten sonra “SET STATISTICS IO OFF” ile kapatılabildiği bir performans değerlendirme kriteridir. İstatistiklerin kapatılması, hafıza da yer kaplamaması açısından önemlidir. ON olan her durum için istatistik ürettiğinden, kullanılmayacak durumlarda kapatılmalıdır.

SET STATISTICS IO ON ifadesi ile

Scan count, logical reads, physical reads, read-ahead reads, lob logical reads, lob physical reads, lob read-ahead reads değerlerine ait sonuçar görüntülenebilir.

Ör:

```
SET STATISTICS IO ON
Select * FROM Personel
SET STATISTICS IO OFF
```

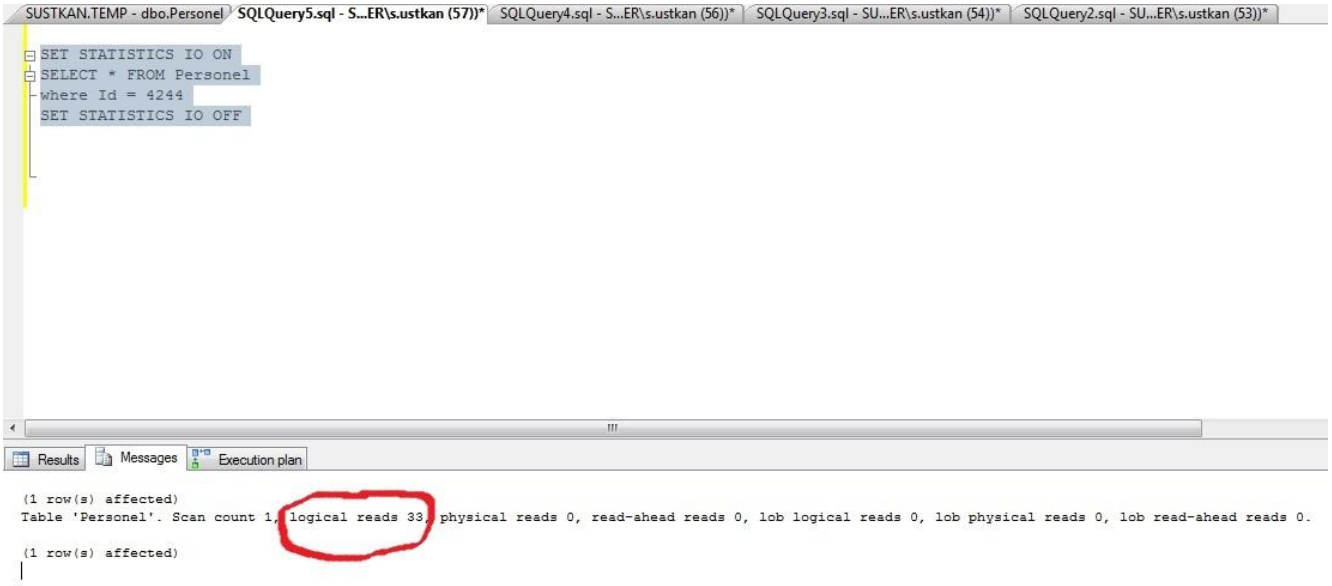
Cümlesinin ürettiği sonuç için verilen istatistikler aşağıdaki gibidir.

(5000 row(s) affected)

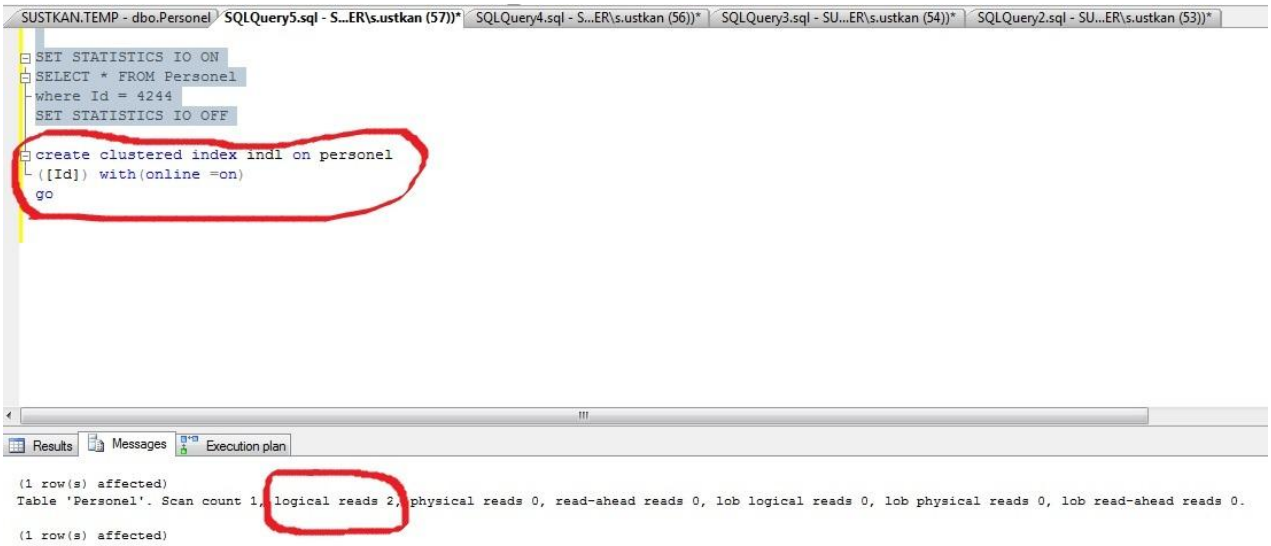
Table 'Personel'. Scan count 1, logical reads 35, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Üretilen bu sonucu anlamalndırmak istersek, logical read değerinin minimize edilmesi amaç olarak görülebilir. Ne kadar az logical read, o kadar çok performans demektir.

Logical Read değerlerinde Cluster Index’ lerin önemini vurgulamak için, aşağıdaki örnekte 2 durum ele alınmıştır. Clustered Index oluşturmadan sorgu performansı ve Clustered Index sonrası sorgu performansları göserilmiştir.



Resim – 3 Statistics IO Değerleri (Index oluşturulmadan logical read değeri : 33)



Resim – 4 Statistics IO Değerleri (Clustered Index oluşturulduktan sonra logical read değeri : 2)

1.2.2 İşlemci İstatistikleri

Sorgunun işlemci ve zaman performanslarının incelenmesi amacıyla kullanılmaktadır.

SET STATISTICS TIME ON, SET STATISTICS TIME OFF ifadeleri ile kullanılmaktadır.

CPU Time ve elapsed time ölçümleri yapılabilmektedir.

İlk örnekte kullandığımız cümlenin aynısını da Time için işlettiğimiz zaman aldığımız sonuçlar aşağıdaki şekildedir.

A . Clustered Index Olmama Durumu:

(5000 row(s) affected) SQL Server Execution Times: CPU time = 0 ms, elapsed time = 205 ms.

B . Clustered Index Durumu:

(5000 row(s) affected) SQL Server Execution Times: CPU time = 0 ms, elapsed time = 132 ms.

Index olmama durumunda çalışma zamanının daha uzun sürdüğünü görmekteyiz.

1.2.3 Sorgu İzleme İstatistikler

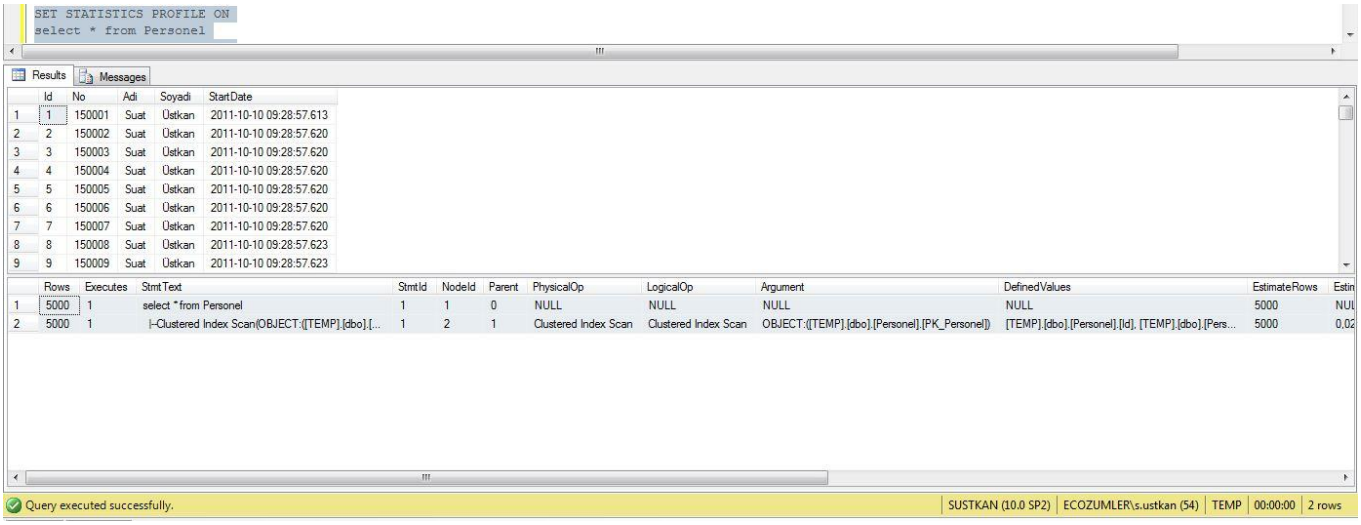
Çalıştırılan sql cümlesinin Profiler edilmiş sonuç bilgilerini döndürmektedir. Aynı diğer statistics' lerde olduğu gibi bunda da ON- OFF mode ta çalışmak performans sağlayacaktır. Buradan bir çok profile bilgisine erişmek mümkündür. Tuning çalışmalarında en çok kullanılan, Rows ve Executes sütunları olacaktır.

Executes : Operatörün kaç kez execute edildiği bilgisini verir.

Rows: Her bir operatör için kaç satır döndürdüğü bilgisi mevcuttur.

Aynı zamanda buradan LogicalOP, EstimateIO, EstimateCPU değerlerine ulaşmak mümkündür.

Örnek bir çıktı bilgisi aşağıda mevcuttur.



Resim – 5 Statistics Profile Değerleri

1.2.4 Deyimler,in Kullanımı

Sorgu yazımında, Table ve Index Scan' a sebep olabilen bazı durumlar vardır. Bunlardan bazıları, NOT, <>, NOT IN, NOT LIKE, OR ifadeleri tercih edilmeyen ifadelerdir.

Büyük datalar ile ilgili işlemler gerçekleştiriliyorsa, her detay düşünülmelidir.

Bu ifadelerdeki ortak nokta, diğer DataPage' ler ile karşılaştırmak zorunda kalmalarındandır. Ayırt edici özellikleri karşılaştırma sonucunda kalan kümeden oluşmaktadır.

Örneğin: Eşit olmama durumunu kontrol edebilmek için tüm datayı karşılaştırmak gerekmektedir.

Örneğin: IN ifadesi ile kullanılan bir expression' da, tüm datayı IN içindeki ifade ile karşılaştırmaya ihtiyaç duymaktadır.

Select * ifadesi kullanımı yerine, ihtiyaç olan sütunlar çekilmelidir. Hem network trafiği açısından hemde execution planda yaratacağı kalabalık yük nedeniyle tercih edilmemektedir.

1.2.5 Fonksiyon Kullanımları

Where kriteri içerisinde User Defined Function kullanılmış ise, Index' ler geçersiz sayılır. Bunun yerine data olarak bu bilgiye erişmeli, elde ettiğimiz data Where kriteri içerisinde diğer işleçler ile süzülmelidir. Kolonlarda function kullanmak yerine, Değişkenlerde function kullanımı tercih edilmeli.

Örnek - 1: WHERE Convert(Nvarchar(12), SutunAdi) = @Degisken
Örnek - 2:WHERE SutunAdi = substring(@Degisken,1,8)

Örnek 1 tercih edilmeyen bir durumdur. Örnek 1 yerine Örnek 2 kullanılması daha performanslı bir sorgu elde etmemizi sağlayacaktır.

1.2.6 Değişken Tanımlama

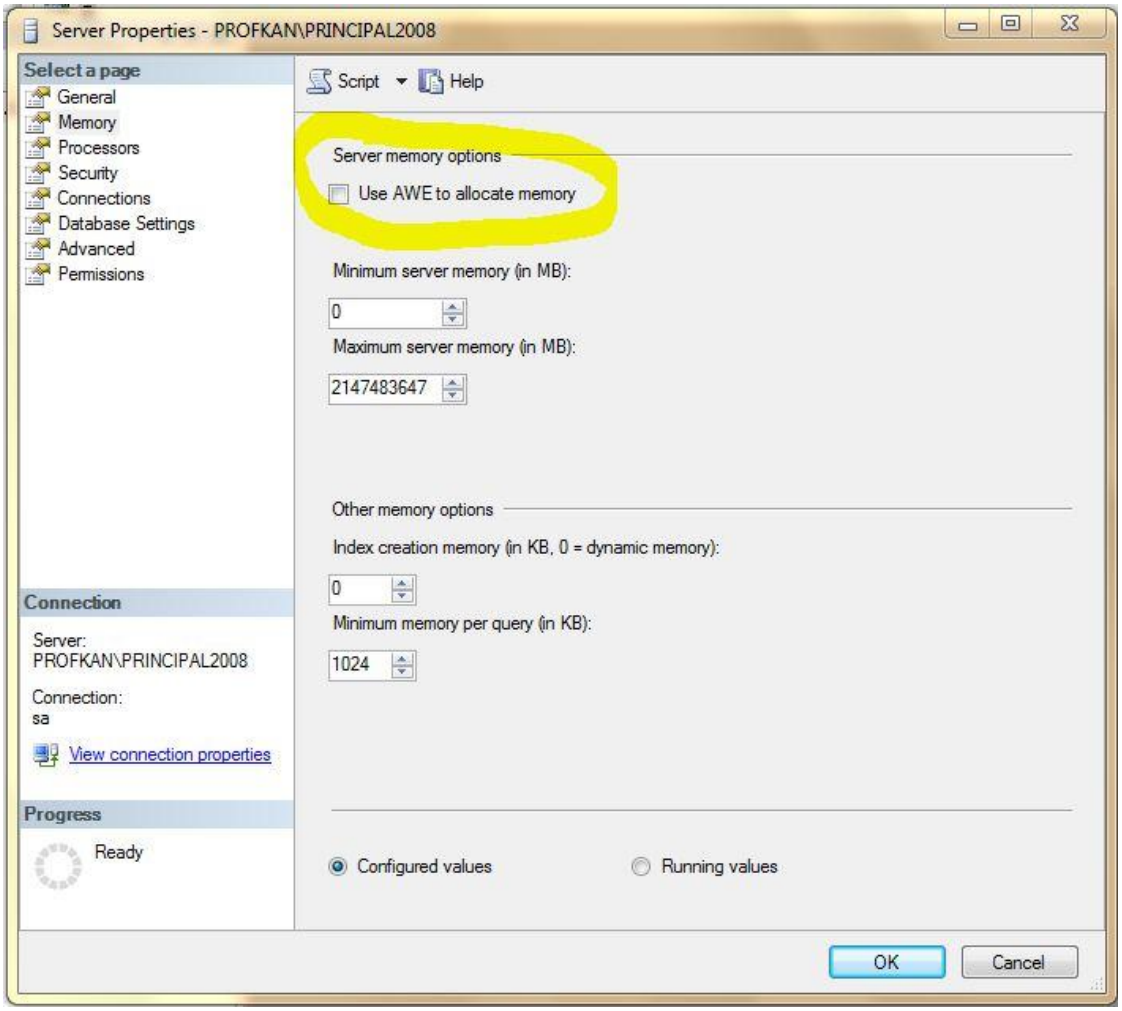
Değişken tanımlamalarında gözden kaçan noktalardan biri, Stored Procedure ve User Defined Function' larda, declare edilen değişken tip ve boyutu ile, veritabanındaki tip ve boyutun birbiri ile uyumsuz olması, performans kaybına yol açmaktadır. Yapılması gerekli işlem, column properties lerinde tanımlanan tip ve uzunluk ile, stored proceudre ve funtion' larda aynısı kullanılmalıdır.

Gereksiz hafıza meşgul edilmemeli, ihtiyaç duyulan yer kadar ayrılmalıdır.

1.3 Server ve Instance Performans Ayarlamaları

Bilindiği üzere, Sql Server' da Max performansı ölçebilmek için, Disk IO işlemlerini azaltıp, Memory üzerinden read işlemleri yapılmalıdır. Böylece Hafızadan verilerin daha hızlı bir şekilde alınıp işlenmesi, performansı doğrudan etkileyen bir konudur.

Sql Server Instance Properties bölümünde Memory ayarları burada önem kazanmaktadır.
"Use AWE to allocate memory" kutusunun işaretlenmesi Maksimum Memory kullanımına izin vermektedir. Zaman içerisinde Sql Server çok fazla Memory işgal etmeye başlayacaktır. Fakat buradaki aşırı yükleme, anlık olarak düşünülmemeli, Sql Server' ın mimarisi gereği Memory' den çalışacak şekilde ayarlanmıştır.



Resim – 6 Server Instance Memory Options

2. VERİTABANI TASARIMI

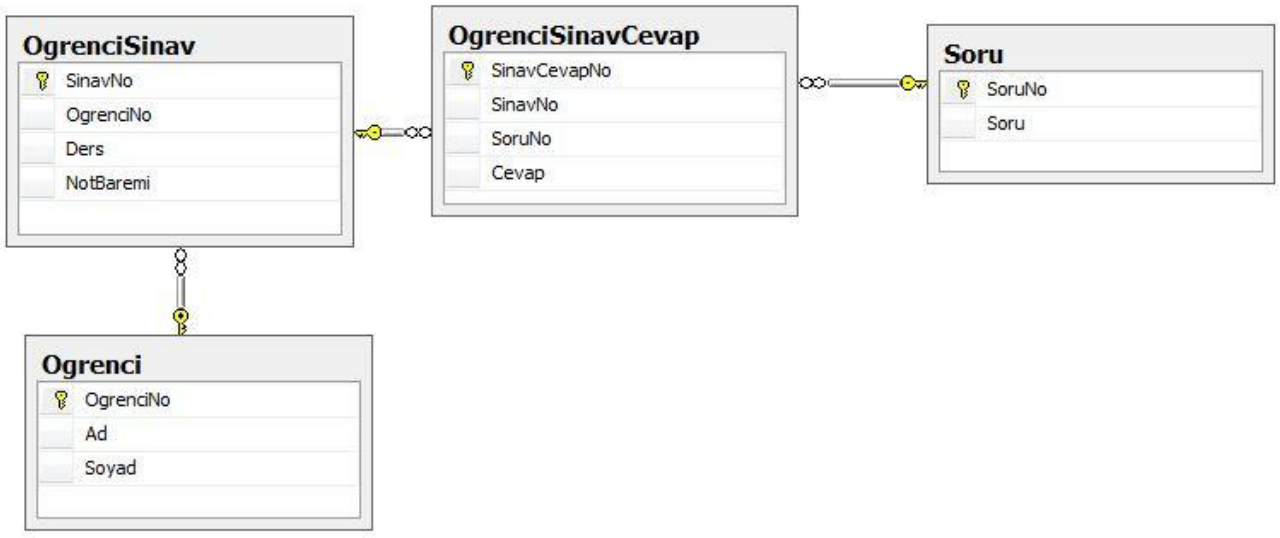
2.1 Normalizasyon

İlişkisel veritabanı modelinde birbirleri ile bağlantılı kümeler mevcuttur. Bu bağlantılar sayesinde, bir bilgiye ulaşabilmek için o kümeye kadar olan tüm bağlantılardan geçmek mümkündür.

Normalizasyon işlemlerinin önemli noktaları:
Veri bütünlüğü : büyük veritabanları ve çok sık transaction işlemleri yapılan noktalarda önem kazanmaktadır.
Veri Tekrarı: Aynı verileri birden fazla tabloda tutmak yerine, ilişkisel model ile merkezi yerden yönetmektir.
Veriler Arası İlişki: Mantıksal sonuç çıkarılması gereken tablolar arasında, birbirleri ile bağlantılı ortak özelliklere sahip olmalıdır.

Örnek olarak; Ogrenci, OgrenciSinav, OgrenciSinavCevap tablolarımız olsun. OgrenciSinav tablomuzda bir Ogrenciye bağlı Sınavların genel bilgilerini tutuyor olalım. OgrenciSinavCevap tablomuzda ise, sınavlara bağlı sorulara ne cevap verildiğini tutuyor olalım.
Öğrencinin tüm sorularda hangi cevaplarını verdiğini sorgulayan bir cümle yazmak istediğimiz zaman, ilişkisel veritabanı modelinden kaynaklı olarak, sırası ile: Ogrenci → OgrenciSinav → OgrenciSinavCevap tablosuna kadar gitmemiz gerekmekte ve aldığımız bu tabloyu, Hangi sorular olduğunu bulabilmemiz için Soru tablosu ile çarpmak gerekmektedir.

Aşağıdaki şekil, bunu daha rahat açıklayacaktır.



Resim – 7 Normalizasyon yapılmış veritabanı diagramı

Performans açışısında normalizasyonun incelendiği durumda ise; kullanım çeşidine, veri miktarına, data tipine bağlı olarak değişiklilik göstermektedir.

Temel olarak verilerin ilişkisel olması, tablolarımızdaki Indexlerimizin kullanılmasına sebep olmaktadır. Fakat kimi durumlarda, ilişkiler üzerinden gitmek performans kaybına sebep olabilmektedir.

Yukarıdaki senaryoyu örnek alırsak, OgrenciSinavCevap tablosunda hangi cevabın hangi öğrenciye ait olduğunu bulabilmemiz için, OgrenciSinav tablosu ile kartezyen çarpılması gerekmektedir.

Çözüm Önerisi: Özet tablolar oluşturulmalıdır.
Herhangi bir raporlama işlemleri yapıyorsak, yukarıdaki diagramda bulunan 4 tabloyu kartezyen çarparak sonuç üretmek doğru bir yöntem değildir. Her sorgulama işlemi için bir performans kaybı oluşmaktadır.

Bu durumda yapılması gereken, belirli periyotlarda bu 4 tablodan anlamlı bir veri oluşturarak, tek tablo üzerinde özet olarak birleşmektedir. Burada göz önünde bulundurulması gereken nokta, özet tablo değişen datalar içerebildiğidir.

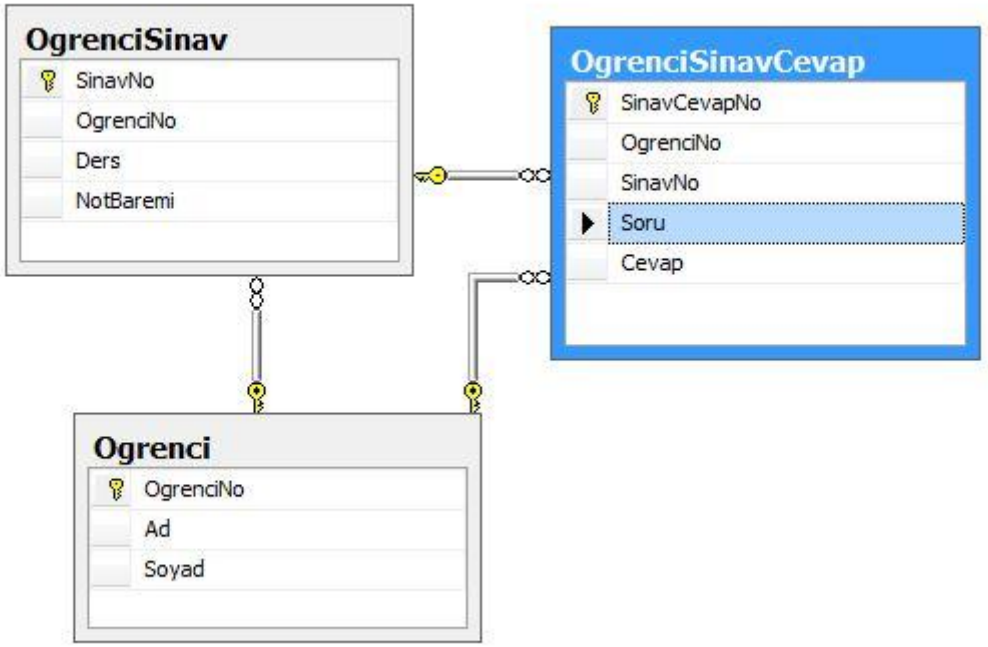
Eğer anlık olarak değişen dataya ihtiyaç duyuyorsak bunun çözümü ise, bir sonraki Denormalization başlığı altındadır.

2.2 DeNormalizasyon

Denormalizasyon veribütünlüğü kavramı ile ters düşsede , performans adına transcation tablolardan çalışılması gereken durumlarda, performansı çok ciddi yönde etkileyen unusurlar arasındadır.

Aşağıdaki diagramdan da görecek olursak, Ogrenci tablosu ile OgrenciSinavCevap tablosu arasında doğrudan bir ilişki mevcuttur. Buda belki milyonlarca cevap arasından, ikinci bir tabloya ihtiyaç duymadan Ogrenciye ait tüm cevapları listelemeye yaramaktadır.

Aynı işlem, Soru tablosu içinde yapılmıştır. Her kayıt için soru tablosu ile kartezyen çarpımı engellemek için sorunun direk ismi tablo üzerinde tutulmuştur. Bu tip bir tablo çok daha fazla bir performans sağlayacaktır.



Resim – 8 Denormalizasyon yapılmış veritabanı diagramı

3. ARAÇLAR

3.1 Çalışma Planlar (Execution Plan)

Sql Server'da bir sorgunun hangi yöntemleri izleyerek çalıştırıldığının, grrafiksel, metinsel anlamda istatistiklerini veren araçtır. Çalışma Planlarından aşağıdaki istatistik bilgilerini elde etmek mümkündür.

- SET SHOWPLAN_ALL ON
- SET SHOWPLAN_TEXT ON
- SET SHOWPLAN_XML ON
- SET STATISTICS XML ON
- SET STATISTICS PROFILE ON
- SET STATISTICS IO ON
- SET STATISTICS TIME ON

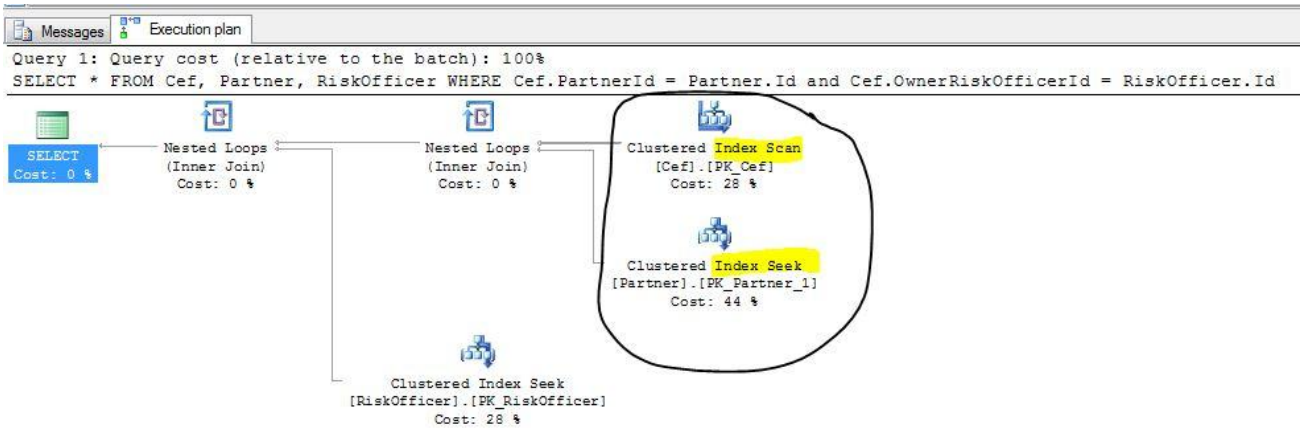
Sql Server Management Studio' da bulunan araç çubuklarından erişilebilir.

Resim – 9 Designer' da Execution Plan butonu

Kısa yol tuşu olarak, Ctrl + L kullanılabilir.

Execution plan okunarak, tüm performans problemleri tespit edilebilir, denilebilir. Çünkü execution plan, veritabanı motorunun çalışmasına ait istatistikleri toplar. Çeşitli durumlarda bu istatistikleri daha detaylı bir şekilde alabiliriz.

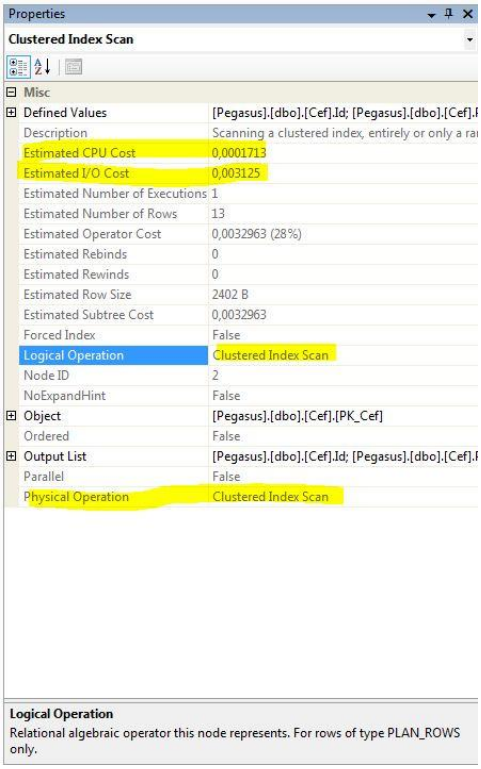
Execution plan' da, hedeflenen her zaman *Index Seek* durumudur. Scan durumlarının hiç biri tercih edilmez. Çünkü scan durumunda, tüm tablo dolaşılmak zorundadır.



Resim 10 – Execution Plan çıktısı

Leaf node ve data page olarak düşünürsek, Seek durumlarında direk Leaf' ler aracılığı ile hareket edilirken, scan durumlarında, data page' lere kadar kırılım yaşanır.

Her bölüm kendi içinde ayrı ayrı değerlendirilebilir. Planları ayrı ayrı alınabilir, bunun için, Node' a sağ tıklayıp properties bölümünden aşağıdaki detaylar listelenebilir.



Resim 11 – Execution planlarından alınan sonuçlara ait properties

Execution Plan okuyabilmek, kısa sorgular için uygunsa da, query karmaşıklaştıkça, okunması zor ve anlam çıkartılması zorlaşır. Bu durumda, böl-parçala-yönet yaklaşımı uygulanarak, daha küçük parçalarda sorgulara ayrılırlar. Böylece aynı performas yaklaşımını daha küçük şekilde uygulanabilir olacaktır. Fakat burada atlanmaması gereken bir nokta, büyük resmi görmekten kaçınılmamalıdır. Kimi durumlarda verinin çeşitliliğine bakılarak, küçük sorgular, büyük performans problemleri çıkartabilmektedir.

3.2 Sql Server Profiler

Sql Server Management Studio ile gelen Sql Server Profiler tool' u, veritabanı motorunun o an işlediği tüm komutları izleyebilmemize yarar. Profiler özelleştirilebilir bir araçtır. İhtiyaç duyduğunuz işleme göre, izleyeceği alanlar belirlenebilir.

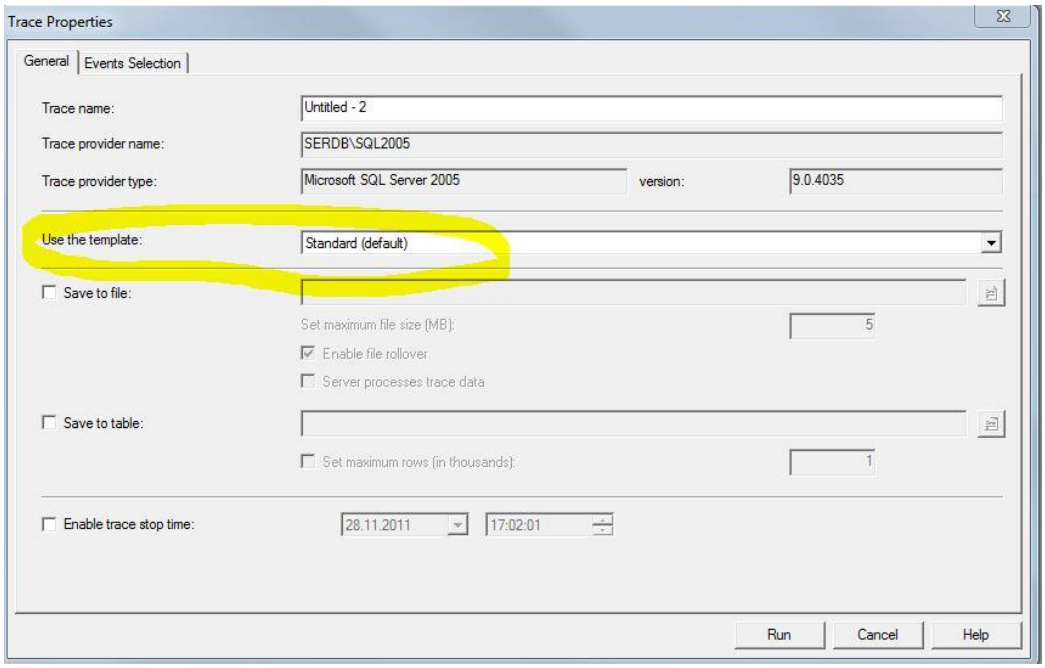
Profiler yardımı ile, veritabanında memory harcayan , CPU harcayan veya çalışma süresi uzun süren sorgular var ise çok rahatlıkla bulunabilir.

Profiler kullanılırken, Events Selection tab' ında bulunan trace'e dahil edilecek kavramları seçmek sonuca ulaşmayı ve problemi diğerlerinden ayırt etmeye yarar.

Aynı ekranda bulunan Column Filter' sekmesi ise, trace edilen sütunlarda aradığınız bir değer varsa, veya belirli bir veritabanını, belirli bir kullanıcıyı, belirli bir değer aralığında olan alanları seçmek için işinizi kolaylaştırıcı bir yapıdadır.

İlk olarak, Standart Profiler template' ini inceleyeceğiz.

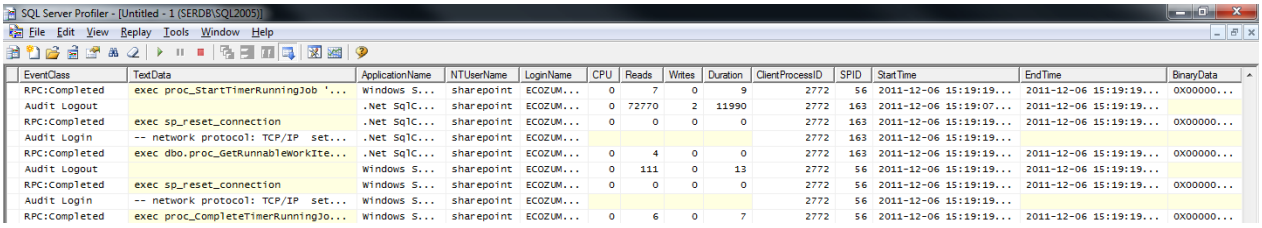
3.2.1 Standart Profiler



Resim 12- Standart Template Profiler

Standart profile' da alınabilen raporlama başlıkları aşağıdaki gibidir.

EventClass , TextData, ApplicationName, NTUserName, LoginName, CPU, Reads, Writes, Duration, ClientProcessID, SPID, StartTime, EndTime, BinaryData bilgileri alınabilir.

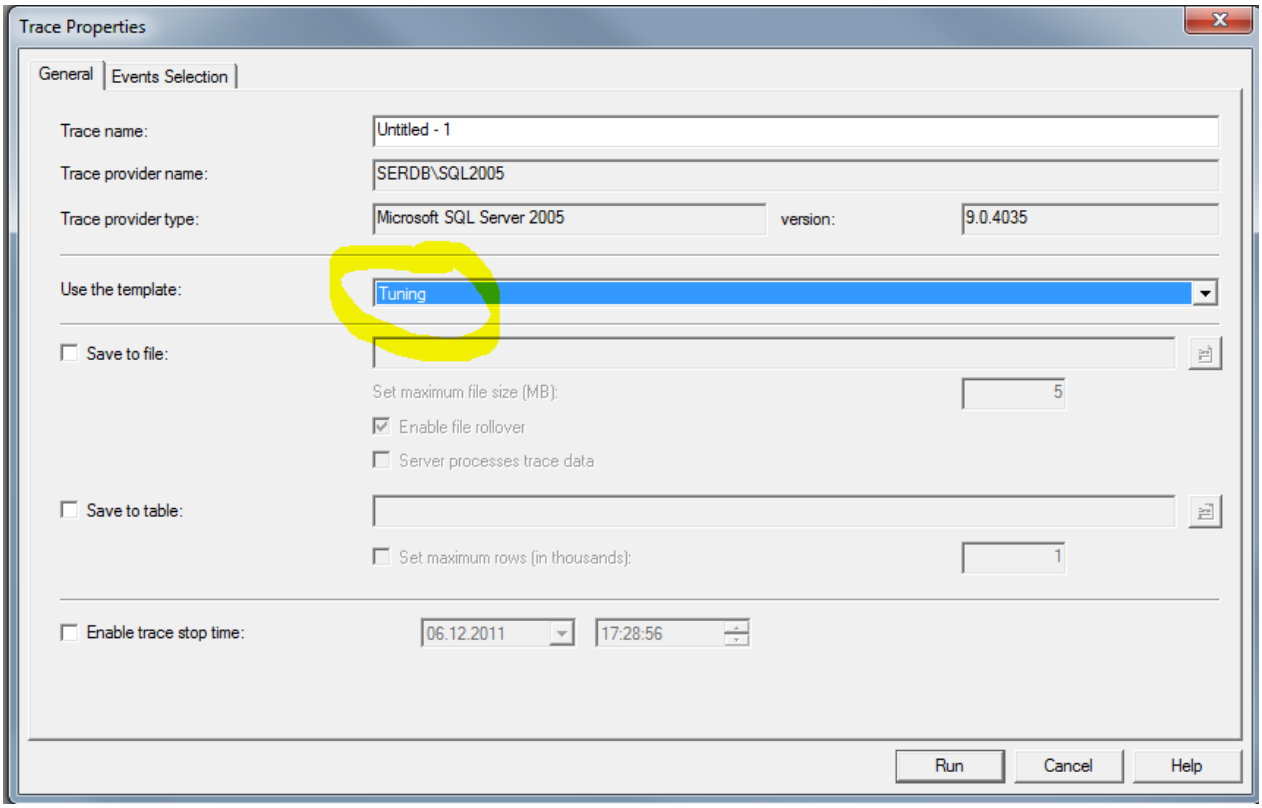


Resim 13 - Standart Profile Görüntüsü

3.2.2 Tuning Profiler

Tuning profiler daha çok performans düzenlemeleri üzerine sadece ihtiyaç olan bilgilerin gösterildiği bir template' tir. Temelde standart template ile arasında hiç bir fark yoktur. Sadece gösterilen sütunlar değişir.

Events Selection' da bulunan Columns Filter özelliklerinde filtreleme özelliği yapılabilmektedir.

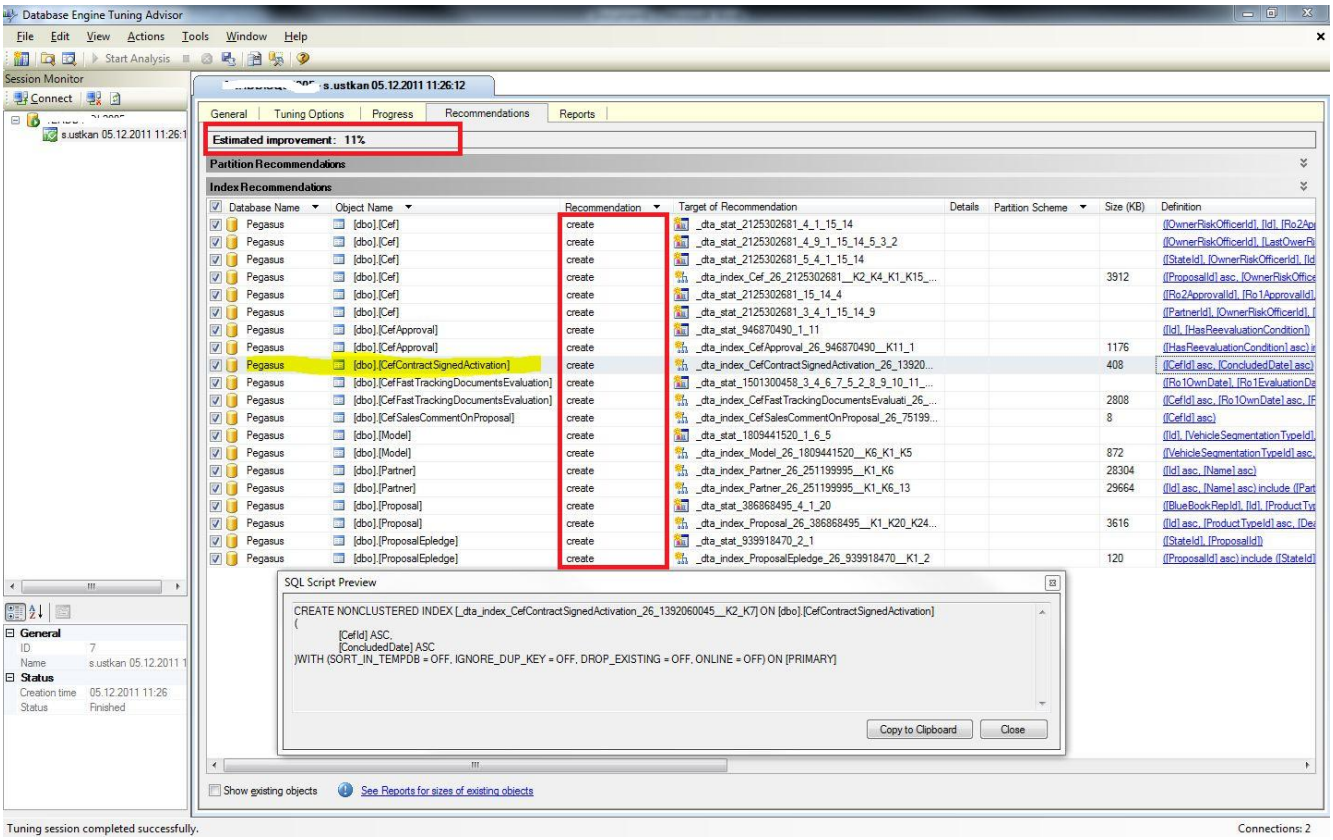


Resim 14 – Tuning Profiler

3.3 Database Engine Tuning Advisor

Bu tool'un kullanılabilmesi için Sys Admin yetkisine sahip olmak gerekmektedir.
Performans problemi yaşadığınız bir sql script dosyasını browse ederek, database motorunun öngöreceği bazı performans iyileştirmeleri yapılabilir. Bu kontroller çapraz şekilde yapılmalıdır. Sadece Tuning Advisor tool'unun önerdiği işlemleri sağlamak performans sağlamayabilir. Onun yerine Execution Plan' dan alınan sonuçlar ile birlikte, karşılaştırılmalıdır.

Aşağıdaki örnekte, verilen sorguya karşılık önerdiği index ler mevcuttur. Bunun neticesinde %11 bir kazanç olacağını bildirmiştir. Bu kazanımın tahmin olacağı unutulmamalıdır. Dataya ve tablo tipine göre, süre değişkenlik gösterebilmektedir.



Resim 15 – Tuning Advisor Rapor Sonucu

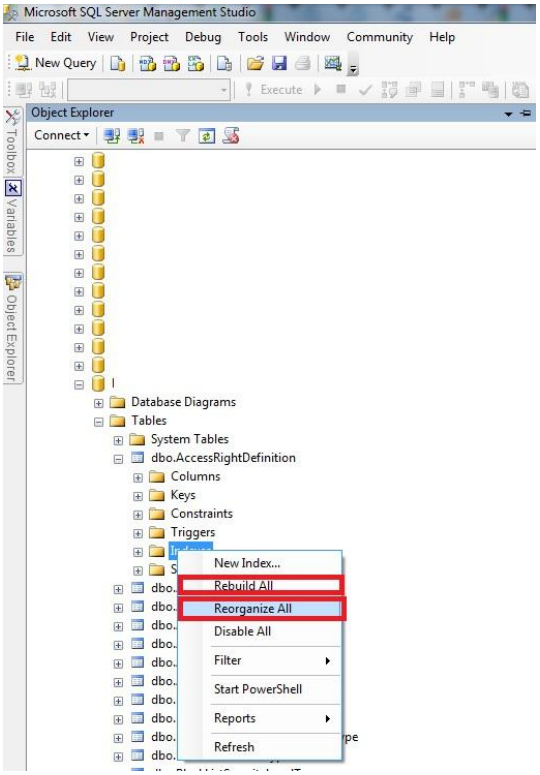
4. BAKIM

4.1 İndex Yenileme

Index' lerin sahip olduğu Fregmentation değerleri mevcuttur. Fregmentation; bellek, disk gibi kaynaklar üzerindeki birimlerin düzensiz parçalar (fragment) haline dönüşmesi ve bunun sonucunda düzenli boş alanın azalmasıdır.

SQL Server'da uzun süre veritabanı üzerinde yapılan silme, güncelleme, ekleme işlemleri sonucunda veritabanı dosyasında data veya index kayıtları arasında boş alanlar oluşur. Boş alanlar birbirleriyle ilişkili kayıtların arasında olduğu için herhangi bir kayda erişme durumunda dağılmış kayıtlar arasında fiziksel olarak dolaşılacağından performans sorunları yaşanmaktadır.

Index rebuild işlemi, arabirimden yapılabildiği gibi, TSQL komutları yazılarak ta yapılabilmektedir.



Resim 16 – Indexlerin yeniden oluşturulması

TSQL kullanarak Rebuild yapabilmek için, aşağıdaki tsql komutları kullanılarak yapılabilir.

```
USE VERITABANI_ADI;
GO
ALTER INDEX ALL ON VERITABANI_ADI.TABLOADI REBUILD
GO

USE AdventureWorks;
GO
ALTER INDEX ALL ON VERITABANI_ADI.TABLOADI REORGANIZE
GO
```

4.2 Log Dosyaları

Select Into, Create Index, Bcp, Bulk loading gibi transactional bilgi içeren tüm işlemler Sql Server tarafından loglanır. Fakat bu loglama zaman içerisinde Sql Server Log dosyaları olan .ldf uzantılı dosyaların fiziksel olarak artmasına sebep olmaktadır. Anlık verinin önemli olmadığı durumlarda .ldf dosyası shrink yöntemi ile boşaltılır. Böylece geçmişe ait loglar temizlenir ve fiziksel olarak dosya küçülür.

Bu dosyaların küçültülmesi T-SQL komutları ile yapılabilmektedir. Aşağıdaki komutlar ile log dosyaları temizlenebilmektedir.

- backup DB_NAME with truncate_only
- DBCC SHRINKDATABASE (DB_NAME, NOTRUNCATE)
- DBCC SHRINKDATABASE (DB_NAME, TRUNCATEONLY)

Kaynaklar

1) KALEN DELANAY, D. (2009), Sql Server Internals, Washington: Microsoft Press.
2) http://msdn.microsoft.com/en-us/library/ms177443.aspx (02/11/2011)
3) http://msdn.microsoft.com/en-us/library/aa933131(v=sql.80).aspx (02/11/2011)
4) http://msdn.microsoft.com/en-us/library/ms177484.aspx (02/11/2011)
5) http://msdn.microsoft.com/en-us/library/ms184361.aspx (02/11/2011)
6) http://www.ahmetkaymaz.com/2008/03/06/sql-server-fragmentation-defragmentation/
7) http://blog.sqlauthority.com/2007/12/22/sql-server-difference-between-index-rebuild-and-index-reorganize-explained-with-t-sql-script/
8) http://www.databasejournal.com/features/mssql/article.php/1466951/SQL-Server-Performance-Tuning-for-SQL-Server-Developers.htm
9) http://www.simple-talk.com/sql/performance/graphical-execution-plans-for-simple-sql-queries/
10) http://blog.sqlauthority.com/2007/12/22/sql-server-difference-between-index-rebuild-and-index-reorganize-explained-with-t-sql-script/ (28/11/2011)