PERFORMANS VE SORGU OPTİMİZASYONU

19

Bilgisayar dünyasında en karmaşık ve sinir uçları belli olmayan iki konudan biri performans, diğeri güvenlik kavramıdır. SQL Server mimarisini, birçok bölümde, farklı katmanlarını inceleyerek irdeledik. Temel ve ileri seviye T-SQL programlama konularını tüm detaylarıyla inceledik. Ancak, performans kavramını tam olarak kavrayabilmek için bu bilgiler yeterli değildir. Bunun nedeni; performans kavramının farklı açılardan irdelenmesi ve optimize edilerek yönetilmesi gereken bir çok farklı iş katmanına sahip olmasıdır.

Bir veritabanı performansı artırma işlemine en az aşağıdaki açılardan bakmak gerekir.

- Müşterinin performans denilince anladığı ve istediği nedir?
- Veritabanı doğru tasarlandı mı?
- Sistem donanım performansı yeterli mi? (İşlemci, Ram, HDD vb.)
- T-SQL bazında nesne ve sorgular performansa uygun şekilde kullanıldı mı?
- Sistem ağ trafiği veri iletişim performansı açısından uygun mu?
- Veritabanı dağıtık mimariye uygun tasarlandı mı?
- Bakım ve yedeklemeler performansı dengeleyecek şekilde mi yapılıyor?
- Veritabanını kullanan istemcilerin sorgu performans durumu nedir?

Bu sorular sadece akla gelen en temel konular içindir. Her bir sorunun detayına girildiğinde birer kitap halini alabilir.

Biz SQL Server'da performans konusunu derinlemesine incelemek yerine, genel kapsamda neler yapabileceğimize bakacağız. Teorik anlamda zihinlerde bir başlangıç noktası ve kavramsal model oluşturmak için bu bölümde çeşitli konulara değineceğiz.

Performans Ayarı Ne Zaman Yapılmalıdır?

Performans ayarı projeye başlangıç tarihidir. Dikkat ederseniz T-SQL geliştirmeye başlangıç değil, projeye başlangıç! Performans durumu, projenin büyüklüğü ve hedefleri genellikle doğru orantılıdır. Küçük çaplı projelerde de performans gerekir. Ancak, bu gereklilik çok kritik değildir. Büyük veritabanı mimarilerinde en ufak performans sorunu katlanarak büyük bir soruna dönüşür. Bunun nedeni; büyük sistemlerin daha yüksek sorgu sayılarına ve kullanıcılara sahip olmasıdır.

Bir projeye başlandığında gereklilikler zaten hazırlanmış olduğu varsayılır. Bu durumda, bu veritabanının ne için tasarlandığı biliniyordur. Öncelik sırası veritabanı tasarımına aittir. Donanım performansının, gerçek veri yükü ile çalışmaya başlayana kadar normal olması yeterlidir. Veritabanı tasarımı, geliştirmeden önceki en önemli unsurdur. Eğer veritabanı tasarımında kritik hatalar yapılırsa, geliştirme sürecinde sürekli yama yapmak ve düzenlemeler gerçekleştirmek için geri dönmek ve mimari değiştirilmek zorunda kalınabilir. Ayrıca, ilişkisel veritabanlarında veritabanı tasarımı, doğrudan sorgu performansı üzerinde etkilidir.

Doğru tasarlanan veritabanında, artık T-SQL geliştirme mimarisi açısından performanslı sorgular geliştirilmelidir. İndeks ve constraint'ler doğru kullanılmalı, veri tekrarından kaçınılmalıdır. Bu aşamada gerçekleştirilmesi gereken T-SQL performans konularında çok detaylara girmeyeceğiz. Ancak, T-SQL'de kullanılan tüm sorgular performanslı değildir. Veri istatistiklerini takip ederek bu konuda tecrübe edinmeniz ve düzenli olarak performans iyileştirmeleri yapmanız gerekir.

Veritabanı performans iyileştirmeleri projenin başlangıcıyla başlar ve proje tamamıyla kullanımdan kalkmadan bitmez.

Donanım

Daha önce proje analizi ve yönetim tecrübesi olmayanlar için donanım kısmındaki keskin uçları anlamak kolay olmayabilir. Aslında bir geliştirici için çok kritik ve uzmanlık gerektiren bir konu değildir. Fakat genel anlamda bir proje için hayati duruma sahiptir.

Veritabanı ve yazılımlar için donanım, para-kasa ilişkisine benzer. Veri, yani veritabanı ya da yazılım para ise; donanım da bunları koruyacak ve hasarsız saklanmasını sağlayacak kasadır.

Nesneleştirmeyi daha da ilginç hale getirirsek, donanım insan vücudu ise, veritabanı bu vücudun beynidir. Beynin performanslı çalışması ve güvenliğinin sağlanabilmesi için vücudun güçlü ve sağlıklı olması gerekir.

Donanım yatırımları projeye mali açıdan büyük yük oluşturabilir. Ancak depoladığınız verinin fiyatı nedir?

Bir online alışveriş sitesini senaryolaştıralım. Binlerce ürüne ve günlük on binlerce kullanıcı girişine, binlerce satış grafiğine sahipsiniz. Yazılım ve veritabanı sorunsuz çalışıyor. Ancak donanım bu kapasiteyi kaldıramayacak özelliklere sahipse, yazılım ve veritabanı göçme senaryoları gerçekleşebilir. Sisteme erişim sağlanamayabilir ve bu sorunların alışveriş sistemine maliyeti hem maddi hem de müşterilerin güvenini kaybetmek gibi paha biçilemez sonuçlara yol açabilir.

İş bu kadarıyla da bitmez. Veritabanı ve yazılımı garantiye almak için bazen farklı sunucular üzerinden hizmet verilmesi gerekir. Bu durum donanım maliyetlerini tabi ki artıracaktır. Örneğin; bir video sitesinin de yazılım ve veritabanı gereksinimi yüksektir. Kullanıcı bilgileri, özel veriler ve video bilgileri tamamen veritabanında tutulur. Video işlemleri yüksek trafik ve donanım tüketirler. Birçok sunucu hatasına da yol açabilirler. Bu tür durumlarda, veritabanı ve yazılımı farklı bir donanım üzerinde barındırmanız gerekecektir.

Ayrıca sistemin lokal mi, yoksa global bir ağ içerisinde mi barındırılması konusu da önemlidir. Bunun için, sistemin sunucu olarak kullanılıp kullanılmayacağı daha önceden belirlenmelidir. Veritabanı sunucuları şirket bünyesinde bir sunucu odasında tutulabileceği gibi, özel bir veri merkezinde de tutulabilir. Bu iki seçenekten doğru olanı seçmek önemlidir. Sadece firma içerisindeki bir

veritabanını barındıracak bir sunucunun firma bünyesinde tutulması doğru olabilir. Ancak bir alışveriş sistemi örneğinde, global bir ağ oluşması nedeniyle, yüksek trafik ve veri kapasitesi, güvenlik ve elektrik kesintisi gibi riskler nedeniyle veri merkezinde barındırma fikri daha doğru olacaktır.

Özetlemek gerekirse, donanımın belirlenmesi için şu sorular sorulabilir.

- Sunucu üzerinde işlem yoğunluğu I/O mu, yoksa işlemci üzerinde mi gerçekleşecek?
- Sistem çökerse, risklerim neler ve veri kaybı senaryolarına hazır mıyım?
- Öncelik performans mı, yoksa verinin güvenliği mi?
- Donanım, bir sunucu olarak mı kullanılacak?
- Sunucu ise, bu sunucu bulunulan yerde mi yoksa bir veri merkezinde mi bulunacak?
- Sunucu uzakta ise, en kritik göçme senaryolarında müdahale edilecek teknik ve eleman var mı?
- Veritabanının bulunduğu sunucu tamamen çökerse aynı anda aynı veriye sahip yeni bir sunucunun devreye girmesini sağlayacak aynalama işlemi gerçekleştirildi mi?

I/O ve CPU Yoğunluğu ve RAM Kullanımı

SQL Server ya da herhangi bir veritabanının en yoğun kullandığı donanım unsuru işlemcidir. Bir makinede birden fazla yoğun işlem gerçekleştiren yazılım çalışıyorsa, bunların her biri işlemciyi daha etkin kullanmak için rekabet edeceklerdir. Günümüzde işlemci maddi açıdan daha ucuz ve teknik olarak daha gelişmiş özelliklere sahip olduğundan, SQL Server çalışan makinede birden fazla işlemcinin olması bir şart olarak görülmelidir.

Tek işlemci çalışan bir sistemde, durum "SQL Server ve diğer yazılımlar" olarak düşünülmelidir. Bir sistemde sadece SQL Server çalışmaz. Geliştirme ortamı bile olsa, bir sistemde en az iki işlemci olmalıdır. En azından, diğer yazılım ve işlemlerin SQL Server'ın kullanacağı işlemci gücünü tüketmemesi sağlanmış olur.

SQL Server farklı sürümlerinde farklı işlemci kullanım yeteneklerine sahiptir. Standard Edition sürümü dört işlemciyi desteklerken, Enterprise Edition sürümü donanımınızın en üst sınırına kadar işlemci desteği sağlar. Genel olarak geliştirme ortamı ve ücretsiz seçenek olan Express Edition ise tek işlemci kullanabilir. Tek işlemci kullanması durumunu önemsenmelidir. Makinede bir işlemci varsa, Express Edition, maksimum kullanabileceği işlemci gücünü, tüm sistemdeki yazılım ve servisler ile paylaşmış olur. İki işlemci olursa birini SQL Server kullanırken diğer sistem yazılımları ikinci işlemciyi kullanabilir.

CPU'nun ayrılmaz parçası olarak RAM kullanımını düşünebiliriz. RAM, bir veritabanı için olmazsa olmaz ve yüksek boyutlarda olması önerilen gereksinimdir.

Veritabanı boyutu, kullanıcı sayısı, veri büyüklüğü, sorguların karmaşıklığı (gruplama, join'ler vb.) gibi farklı parametrelere göre RAM ihtiyacı belirlenir.

Bir kullanıcının sisteme bağlanması yaklaşık 24K RAM kullanır. Her sorgunun sonucunda alınan veri RAM'de bir alan kaplar. Sistem açık kaldığı ve çalıştığı sürece bu kullanım sürekli artar. Veritabanı üzerinde mantıksal işlemlerin yoğun olduğu durumlarda bu durum daha da önem kazanır. Dışarıdaki bir kaynaktan veri alan (data import) bir veritabanı var ise bu ihtiyaç daha da artar. Yani, veritabanının kullanım yöntemi ve yoğunluğuna göre bir RAM kullanımı gerçekleşir.

SQL Server'ın hızlı ve performanslı I/O işlemleri gerçekleştirebilmesi için yüksek RAM bir zorunluluktur.

Çökme Senaryoları

Veritabanı uygulamaları, yazılım alanında en kritik alt yapı çalışmalarıdır. İşlem yoğunluğu veritabanı üzerinden gerçekleşen bir uygulamada veritabanının ya da üzerinde bulunduğu sunucunun çökmesi sistemin tamamen kilitlenmesiyle eş değerdir. Kullanılan istemci uygulamada veritabanı yoğunluklu bir işlem olmasa bile, sadece veritabanı üzerinden kullanıcı girişi yapması kritiklik derecesini ortaya koyacaktır. Kullanıcılar uygulamaya giriş yapmak için, yani yazılıma erişebilmek için bile veritabanının çalışıyor ve kullanıcı bilgilerini sorgulayarak yazılıma erişim izni vermesi gerekiyor.

Tüm şirket yazılımlarının tek bir veritabanı kullanan büyük bir şirket yönetim uygulaması olduğunu ve tüm departmanların yetkileri dahilinde bu yazılıma erişerek günlük işlemlerini (veri girişi, raporlama, girdi-çıktı vb.) yaptıklarını düşünelim. Veritabanının çökmesi, yönetim yazılımları çalışmayacağı için tüm şirket çalışanlarının sorun çözülünceye kadar tatil yapması anlamına gelebilir. Sorun sadece iş yapamamak değildir. Çalışanların saat ücreti üzerinden hesaplama yapıldığında, bu hatanın, maddi açıdan da ne kadar büyük bir maliyeti olacağı hesaplanabilir.

Çökme riskleri özel bir analiz süreci ile hesaplanmalıdır. Veri, kendisini kullanan kullanıcılar ve firma açısından ne kadar önemlidir? Çok kritik seviyede önemlilik arz eden senaryolarda, çeşitli önlemler alınarak çökme sonucu oluşacak veri kaybı ve iş kaybı en aza indirilebilir.

Çökme sonucunda oluşacak sorunları en aza indirmek için aşağıdakiler yapılabilir.

- Veritabanı ve işletim sistemi farklı disk'lerde bulundurulmalı.
- Veritabanı yedekleri belirli aralıklarla otomatik olarak alınmalı.
- Veritabanı yedeğinin işletim sistemi ve veritabanının bulunduğu diskten, hatta mümkünse farklı bir sunucuda bulundurulmalı.
- Veritabanı, dışarıdan erişimlerde belirli IP aralıkları haricinde erişime kapalı olmalı.(Sadece şirket IP'leri gibi)
- Veritabanının bulunduğu sunucu ile istemci arasında yük dengeleme yazılımı kullanılabilir.
- Veritabanının bulunduğu sunucudan farklı bir sunucu ile arasında aynalama yaparak, verinin anlık olarak farklı lokasyonlardaki iki ayrı sunucuya yazılması sağlanabilir.
- **Aynalama:** Birincil olan sunucu üzerinden gerçek işlemler yürütülürken, diğer sunucular, birincil sunucudan anlık işlemlerin kopyasını otomatik olarak alır ve anlık yedek veri oluşmasını sağlar. Birincil sunucu da çökme yaşandığında, kopya olan ikincil sunucu anında devreye girerek sistemin ayakta kalmasını sağlar.

Daha da ötesinde, veri merkezinde oluşabilecek bağlantı ya da genel saldırı gibi durumlara karşı da veritabanının korumaya alınması gerekir. Genel olarak

global saldırılarda, veri merkezi hizmetlerini askıya alarak kendini otomatik olarak güvenlik kalkanı içerisine hapseder. Bu durumda verilere ulaşamazsınız. Ancak yasal olarak da yapacağınız bir şey yoktur. Çünkü veri merkezinin genel güvenlik politikaları gereği alt yapısını koruma hakkı vardır. Bu tür durumlarda sorun yaşamamak için, yedek ya da aynalama yapılan sunucuları aynı veri merkezinde barındırmak yerine, farklı lokasyondaki hatta farklı karasal bağlantı hattına sahip bir veri merkezinde tutmak mükemmele en yakın çözüm olabilir.

Çökme senaryolarını hesaplarken asla olasılıklar için imkansız denilmemelidir. Yağmur yağması sonucu, ulusal ve uluslararası hizmet veren bir GSM operatörünün veri merkezine su basması ve büyük veri kaybı yaşanması bir örnek olabilir. Tüm telefon faturalarının 10 günden fazla olan bir kısmının veri kaybı sonucu yok olması nedeniyle, kullanıcılara hediye edildiğinin söylenmesi (!) gibi bir durumla karşılaşmamak için tüm olasılıkları hesaplamak gerekir.

Kim bilir, belki veri merkezine meteor düşer!

Online Sistemler Üzerinde SQL Server

SQL Server, lisanslı olarak web siteleri ve online veritabanı hizmetlerinde en çok kullanılan veritabanıdır. Bir web uygulaması ya da uzak sunucuya bağlanan veritabanı uygulaması geliştirildiğinde kullanılacak veritabanı çözümüdür.

Bu işlem iyi bir hizmet esnekliği sağlasa da, verinin internet üzerinde olması nedeniyle birçok sorun ve riski de beraberinde getirir.

Online Güvenlik

Veritabanının uzak sunucuda olması başlı başına bir güvenlik riskidir. Verinin güvenliği artık sizin elinizde değildir. Uzaktaki sistemin network ve güvenlik yöneticisinin yetenekleri ile sınırlı bir güvenliğe sahipsiniz. Ayrıca verilerinize sizin kadar özen göstermeyeceklerine emin olabilirsiniz. Uzak sunucuya bağlanarak işlem yapabilmeniz için port açılması gerekir. Port açmak güvenlik açısından risklidir. İyi monitör edilmeli ve yönetilmelidir. Uzaktan bağlanılacak sistemde güvenlik ayarlarının iyi bir network uzmanı tarafından yapılması gerekir. İyi bir yazılımcı ya da veritabanı uzmanı olmak iyi bir network uzmanı olmak anlamına asla gelmez. SQL Server'ın yönetildiği sistem ve ağ doğru yönetilmiyorsa, sizin veritabanı tarafında aldığınız güvenlik önlemlerinin bir önemi yoktur.

Online Performans

İstemciden uzan bir sunucuya bağlanarak veritabanı işlemi yapmaya başlamak, veritabanına ilk bağlanırken bile performansın yavaşlaması anlamına gelir. Veritabanı ve veri yoğunluğuna göre değişecek bu performans sorunu, bir tabloyu `SELECT` ettiğinizde kendisini gösterecektir. Bağlantınız ne kadar yüksek olursa olsun yerel ağ'dan hızlı olamaz. Bu nedenle, geliştirilen veritabanında tüm sorgu ve nesneler daha da küçültülmeli ve gereksiz verilerin filtrelenmesine daha da önem gösterilmelidir. Sorgular optimize edilmeli, gereksiz sunucu bağlantılarından kaçınılmalı, tablo yapıları parçalara bölünerek ağ trafiğini daha az kullanacak şekilde tasarlanmalıdır.

Online Destek

Uzak sunucu hizmeti alınan firma sizin işinize genelde sizin kadar önem göstermez. Veritabanında ya da sunucu tarafında gerçekleşecek bir hata sonrasında hizmet alınan yerden anında çözüm odaklı destek almak çok önemlidir. Ticari bir çalışma için hazırlanan web sitesinde, kullanıcıların işlem yapması için hazırlanan veritabanında, bir bağlantı hatası oluşması ve bu sorunun çözülmesinin 1 gün sürdüğü senaryosunu düşünmek bile can sıkıcıdır. Daha da ötesinde, uzak sistemlerde veritabanı yedeğini hizmet alınan firma üstlenir. Yedeklerin bulunduğu sunucu ile hizmet verdikleri sunucunun diskinin yandığı gibi en kötü senaryolara hazırlıklı olunmalıdır.

Yasal olarak sorumluluğun karşı tarafta olması, belki paha biçilemez olarak gördüğünüz verilerin geri getirilememesi durumunda bir anlam ifade etmeyecektir. Hizmet alınan firma, veritabanı ve yazılım yedeklerini tutuyor olsa da, verilerin güvenliği açısından kendi yedeklerinizi farklı fiziksel ortamda tutmanız önerilir.

Sorgu Optimizasyonu

Performans işlemlerinin bir diğer unsuru yazılım taraflı performans geliştirmeleridir. Veritabanı tasarımından başlayan yazılım performans unsurları, veritabanı programlama ile devam eder. Donanımsal kısımda çok sık değişiklik gerekmez. Ancak, yazılımsal tarafta hiç bitmeyecek bir performans iyileştirmesi gerekir. Bir sorgu ilk geliştirildiğinde çok yoğun kullanılmayabilir, fakat kullanıcıların tercihleriyle birlikte sorgu kullanımı artabilir. Bu durumda sorgunun daha hızlı çalışması, sorgu sonucundaki verinin daha hızlı getirilmesi

ve sorgunun ulaştığı tablonun tasarımında geliştirme düzenlemeleri yapma ihtiyacı doğabilir. Bu işlem bir döngüdür ve sürekli devam eder.

Minimum Kuralı

Veritabanı ve uygulama için temel kurallardan birisidir. İstemci uygulama tarafından bir veri tablosunun tamamını çekmek çoğu zaman anlamsız ve gereksiz kayıtların sunucudan, network aracılığıyla istemciye ulaşması anlamına gelir. İstemci uygulamanın isteği, tablodaki 10 kaydın 4 sütunu ise, istemciye 20 adet sütun ve içerdiği 100 kaydı iletmek istemci ve sunucu performansı açısından olumsuz bir işlem olacaktır.

Sunucudan istenen her kayıt hem istemcide, hem de sunucuda bellek ve I/O kullanımına sebep olur. Gerekli olan verinin haricinde, fazladan gönderilen her veri bu kaynakları fazladan tüketmek anlamına gelecektir. Performans bakış açısıyla incelenen bir sistemde, istemci yazılım formlarında, veritabanından çekilen ve kullanılmayan her veri performans israfıdır.

Örneğin, kullanıcının hiç kullanmadığı bir veriyi, veritabanından çekerek uygulamadaki arayüzde görüntülemek bir kaynak israfıdır. Uygulama formları performans işlemleri için tasarlanmalı ve bu bakış açısına göre ilgili veriler listelenmelidir.

Genellikle uygulamalar, sık kullandığı statik veriyi cache bellekte tutarlar. Veritabanında sürekli değişmeyen bir verinin, sürekli veritabanından istenmesi gereksizdir. Örneğin; bir web uygulamasında, ComboBox ile ülkeler listelenirken, bu ülke listesinin her sayfa yenilenmesinde veritabanından çekilmesine gerek yoktur. Uygulama başladığında bir kez veritabanından çekilen ülke listesi, cache belleğe alınır ve sonraki kullanımlarda sürekli cache bellekten statik veri olarak okunur. Cache bellek, doğru kullanıldığında kaynak tüketimini azaltır ve performansa olumlu anlamda katkı sağlar.

Geçici Tablolar

Genel olarak, hızlı sorgular geliştirmek için kullanılır. Birden fazla tablo birleştirilerek elde edilen bir sorgu senaryosunu ele alalım. Çok sütunlu ve yüz bin kayıt bulunan bir tabloda, sütunlardan birinin tarih bilgisini tuttuğunu düşünelim. Bu tablo üzerinde, Kasım ayına ait bilgiler sürekli olarak sorgulanıyorsa ve bu tarihteki veriler üzerinde sürekli farklı filtre, gruplama gibi

işlemler gerçekleştiriliyorsa, Kasım ayına ait verileri geçici bir tabloda tutmak ve sonrasında gerçekleşecek tüm sorguları, bu geçici tablo üzerinde yaparak, diğer yüz bin kayıtlı ana veri tablosunu sorgulamadan, daha az veri üzerinde sorgu gerçekleştirmek gerekir.

Bu örnek tam olarak şuna benzer; bir büyük kutu içerisinde tüm eşyaların karışık olarak saklanması ve sonrasında, her seferinde kutu içeriği aranarak bir eşyanın bulunması mı kolay, yoksa büyük kutu içerisine giysi, mücevher, oyuncak gibi eşyaların ayrılarak küçük kutulara konulması ve her arama isteğinde ilgili kutudan aramak mı daha kolaydır? Tabi ki küçük kutularda, ilgili olduğumuz şeyleri aramak daha kolay olacaktır. Bu işlemde, büyük kutu yüz bin kayıtlık veri tablosu iken, küçük kutu, Kasım ayına ait kayıtlardır.

Filtreleme, Gruplama ve Sayfalama

Uygulama tarafından belirtilen bazı özel istekler vardır. Bunlar, veriyi gruplamak, sayfalamak ya da filtrelemek gibi veri üzerinde kaynak ve süre açısından ek yük gerektiren işlemlerdir. Bu tür istekler gerçekleştirildiğinde, öncelikle tam olarak ne istendiği belirlenmelidir. Uygulama tarafından gelen istek, maksimum 100 satırdan oluşabilecek bir sorgu içeriyorsa, 100 kaydın üzerinde gönderilen her kayıt ve gereksiz sütun ek kaynak tüketeceği için performansı olumsuz yönde etkileyecektir.

Veri katmanı ile uygulama katmanı arasında iş yükü dağılımının doğru yapılması gerekir. Veri üzerinde filtreleme, gruplama ya da sayfalama benzeri işlem gerçekleştirilecekse, bu işlemlerin veritabanında yapılması gerekir. Örneğin; 1000 satırlık bir veri üzerinde bir sayfalama işlemi gerçekleştirilecekse, bu kayıt satırlarının tamamını uygulama tarafına göndererek sayfalamanın uygulama katmanında yapılmasını sağlamak network trafiğini ve farklı bazı performans parametrelerini olumsuz etkileyecektir. Sayfalama için 1000 kayıt isteniyor olabilir. Ancak bu kayıtlar 100 eşit sayfaya bölündüğünde, uygulama katmanında iki ya da üçüncü sayfadaki verinin kullanıcı tarafından talep edileceği belli değildir. Henüz gereksinim duyulmayan bir veri, uygulamaya gönderilmemelidir.

Ayrıca bu tür işlemleri gerçekleştirmek için, indeksler tanımlanmalı ve örneğin, sayfalama işlemini gerçekleştirecek bir prosedür oluşturularak, kullanıcının, daha önceden sorgu planı hazırlanmış ve derlenmiş olan bu prosedüre,

sayfa bilgilerini ileterek, belirlediği sayfa aralıklarındaki bilgileri elde etmesi sağlanabilir.

Filtreleme işlemlerinde, uygulama katmanına, gerçekte kullanıcının ihtiyacı olan veriyi göndermek için koşullar doğru belirlenmelidir. Örneğin; kullanıcı adının baş harfi "*A*" ve üçüncü harfi "*İ*" olan kayıtların istendiği bir senaryoda, baş harfi "*A*" olan tüm kayıtların uygulama katmanına gönderilmesi, sistem kaynaklarını fazladan kullanmak anlamına gelecektir. Kullanıcının isteğinde, sadece 5 kayıt dönecekken, uygulama katmanına 100 kayıtlık bir sonuç döndürülmesi anlamsızdır.

Gruplama işlemlerinde, uygulama katmanı genel olarak, belirli ve az sayıda sonuç bekler. Gruplama isteği tam olarak bilinmeli ve bu isteğin karşılanacağı şekilde gruplama sorgusu hazırlandıktan sonra, uygulama tarafına iletilmelidir. Diğer sorgu yöntemlerinde olduğu gibi, gruplama işlemlerinde de yapılan istekle ilgili olmayan hiç bir kayıt uygulama katmanına iletilmemelidir.

İNDEKS

Veritabanı yönetimi ve performanslı çalışması için en çok ihtiyaç duyulan veritabanı nesnesidir. SQL Server, sorguların daha performanslı çalışabilmesi için indeksleri kullanır. SQL Server, tüm sorgularda kullanmak için arka planda bazı özel tanımlamalar ve veri tanıma parametreleri kullanır. Çeşitli veritabanı istatistikleriyle birlikte hangi tablo, sütun ve sorguların daha çok kullanıldığını öğrenerek kendisi performans artırıcı önlemler alabilir. Bir T-SQL geliştiricisi, veritabanı sorgu performansını artırmak için çok kullanılan ve filtrelemeler gerçekleştirilen kayıtlar için indeks tanımlayabilir. İndeks kullanımı performans artırmak için kullanılsa da, doğru tasarlanmayan indeks mimarisi, verinin olduğundan daha yavaş çalışmasına da sebep olabilir. Sorgularda kullanılan **`WHERE`** ile filtreleme, **`JOIN`** ile birleştirme işlemleri, **`ORDER BY`** ile sıralama işlemleri için indeksler tanımlanmalıdır.

İndeksler ile ilgili aşağıdaki öneriler dikkate alınmalıdır.

- Tabloların veritabanı sunucusu tarafından doğru taranabilmesi için, her tabloda bir **`Primary Key`** bulunmalı.
- Sık kullanılan **`WHERE`** filtrelemelerinde kullanılan alanlarda **`NonClustered`** indeks kullanılmalı.

- Sık kullanılan ve genellikle beraber **WHERE** ile filtrelenen sütunlar, birlikte indekslenmeli.
- **JOIN** ile birleştirilen tablolarda **Foreign Key** alanların **NonClustered** olarak indekslenmeli.

İndekslerin nerede, ne zaman kullanılması gerektiğini ve performans etkilerini görmek için **çalıştırma planı** (*execution plan*) incelenebilir. Ayrıca SQL Server, Index Tuning Wizard ile sizin için bazı indeks önerileri sunar.

Filtreli İndeks (Filtered Index)

SQL Server 2008 ile birlikte gelen, optimize edilmiş bir **NonClustered** indeks'tir. Filtreli indeks oluşturulurken kullanılan **WHERE** filtresi nedeniyle indeks key'i, tüm verileri değil, sadece belirlenen filtreye uygun veriyi içerir. Sorgular genel olarak **WHERE** ya da benzeri koşullar ile filtrelenerek elde edilir. Bu nedenle, filtreli sorgular üzerinde indeks oluşturmak daha performanslı olabilir. Filtreli indeksler bakım maliyetini düşüreceği gibi, disk üzerinde de daha az yer kaplar.

Sorgu Performansını Artırır

Filtreli indeksler daha az veri içerdiği için indeks üzerinden yapılacak tarama işlemleri normal indekslere göre daha hızlı yapılacaktır. Ayrıca bu işlem, verinin veritabanına daha doğru işlemesi için tanıtılması anlamına gelir. Bu durumda veritabanı, oluşturacağı istatistik ile veriyi daha iyi işleyecek parametrelere sahip olacaktır.

Bakım Maliyetlerini Düşürür

Tablo üzerinde DML işlemleri arttıkça, indeks **fragmante** (*parçalanma*) olur. Filtreli indeks, daha az veri içerdiği için fragmante sorunu olasılığı daha da düşük olacaktır. Ayrıca, fragmante olsa bile, bakım işlemi filtreli indekslerde daha az veri üzerinde yapılacağı için, bakım maliyeti daha az olur.

Disk Depolama Maliyetini Düşürür

Normal indekslere göre filtreli indeksler, daha az veri içerdiği için, kapladığı disk alanı da düşük olacaktır. Bu durum, sadece disk kotası ile ilgili değil, I/O, CPU ve RAM üzerinde de maliyetleri azaltmak için faydalıdır.

View

Veritabanında en çok kullanılan sorgu komutu, veri seçme için kullanılan **SELECT** komutudur. Hem en çok hem de en karmaşık sorgularda bu komut kullanılır. Bu nedenle SQL Server, veri seçme işlemlerini daha performanslı hale getirebilmek için birçok ek özellik sunar. View (*görünüm*) nesneleri, filtreli, filtresiz ya da **JOIN** ile birleştirilmiş bir veri kümesine takma isim vererek, gerçek bir tablo gibi sorgulanmasını sağlar. Kitabın view bölümünde derinlemesine incelediğimiz bu konu için, performanslı sorgular hazırlanması gerekir.

Bir tabloda tüm veriler seçilecek ise, view kullanılmasının amacı nedir? View kullanmak için gerçek ihtiyaçlara sahip olunmalıdır. Örneğin; bir **JOIN** işlemi sonucunda oluşturulacak verinin, tekrar tekrar yazılmaması için view ile isimlendirerek hızlı erişmek doğru bir tercih olabilir. Ancak herhangi bir filtreleme yapmadan, sadece bir tablonun 3 sütununun değerini verecek bir view oluşturmak anlamlı ve doğru değildir. Çünkü oluşturulan her view, aslında veri ile uygulama arasına yerleştirilen ara bir katmandır. Ara katmanlar her zaman ek iş yükü getirir ve işlemin az ya da çok, yavaşlamasına sebep olur.

View kullanımının performans üzerindeki etkisini inceledik. Bazı durumlarda view kullanmanın performansı artıracağı senaryolarda gerçekleşebilir. Bu tür durumlarda, view içerisindeki sorguyu akıllıca hazırlamak gerekir. Örneğin; bir view içerisinde sıralama işlemi yapmak anlamsızdır. Çünkü view bir tablo gibi sorgulanabilir. Geliştirici, view içerisinde sıralamayı A şeklinde yaparsa, view'i kullanan kişi sıralamayı B'ye göre yapmak isteyebilir. Bu durumda, view içerisinde yapılan sıralama işlemi, kaynak ve işlem süresini gereksiz şekilde kullanıyor olacaktır. Aynı zamanda, view'in kullanılacağı yerde yapılması gereken dönüştürme ya da hesaplama gibi işlemlerin de view içerisinde yapılmaması gerekir.

Constraint

Constraint'ler, veritabanı için değer nesnelerdir. Ancak her şeyin bir bedeli vardır. Veri girişi sırasında kullanılan bir Check Constraint'in görevi verinin, istenen şarta uygunluğunu kontrol etmektir. Bu da veri girişinde yavaşlamaya neden olacaktır. Veri girişinde, zorunlu kalınmadığı taktirde, bu tür kontrollerden kaçınılmalıdır. Veritabanı katmanında yapılacak kontrolleri, uygulama katmanında yapmak bir çözüm olabilir. Örneğin; bir kullanıcı

tablosunda email bilgisinin benzersiz olması gerekir. Email bilgisi kontrolden geçirilmelidir. Bu kontrol, veritabanı katmanında Check Constraint ile yapılabileceği gibi, uygulama katmanında birkaç satır kod ile de yapılabilir. Veri ekleme performansı düşünülüyorsa, bu işlemin uygulama katmanında yapılması gerekir.

Trigger

Trigger'lar etkili ve ileri seviye bir veritabanı özelliğidir. Trigger'lar otomatik olarak tetiklendiği için, bazı sorunları da beraberinde getirebilir. Trigger kullanımı konusunda dikkatli davranılmalıdır. Her veritabanı nesnesinde olduğu gibi, trigger'lar için de performans artırıcı ve doğru kullanmayı sağlayacak bazı püf nokta ve öneriler vardır. Bu bölümde, trigger tabanlı performans konularına değineceğiz.

Trigger'lar Reaktif'tir

Proaktif, bir olaydan önce gerçekleşmeyi gerektirirken, reaktif bir olaydan sonra gerçekleşmesi anlamına gelir. Trigger başlatıldıktan sonra tüm sorgu çalışır ve transaction log bilgisi tutulur. Ancak, trigger'larda da her şey yolunda gitmeyebilir. Trigger işlemini gerçekleştiren transaction'da bir hata meydana geldiğinde, tüm transaction işlemleri `ROLLBACK` ile geri alınır. Trigger performansı ile trigger gövdesinde tanımlanan SQL kod bloğunun büyüklüğü doğru orantılıdır. Trigger ne kadar çok iş yapıyorsa, hata sonucundaki `ROLLBACK` işlemi o kadar uzun sürecek ve performansı olumsuz yönde etkileyecektir.

Trigger'larda Ortak Zamanlılık

Trigger ile trigger'ı başlatan ifade dolaylı olarak transaction'ın birer parçasıdır. Trigger'ı başlatan ifade, açık bir transaction olmasa bile, SQL Server için tek ifadeli bir transaction'dır. Her ikisini de bir bütün olarak işler. Bir transaction parçaları olduğu için ROLLBACK ile işlem geri alınması durumunda tüm yapılan işlemler geri alınır. Trigger bölümünde seviye kavramından bahsettik. Trigger'lar 32 seviyeye kadar derinliği desteklemektedir. Her trigger ve bu trigger'ların tetiklenmesini sağlayan SQL işlemlerinin kod ve işlem yoğunluğuna göre, ROLLBACK işleminin ne kadar zor olacağı hesaplanabilir.

Trigger Genişliği

Trigger genişliği kavramı genel olarak tüm SQL sorgularını kapsayan bir performans sorunudur. Bir kod bloğunun çalışma süresi ve performansı içerisinde barındırdığı işlemlerin ve kodların yoğunluğu ile ilgilidir. Kendisini tetikleyen bir transaction'ın parçası olarak çalışan trigger'lar için sorgu hızı ve kod yoğunluğu önemli bir etkendir.

Bir Stored Procedure tarafından tetiklenen trigger tamamlanmadan, Stored Procedure içerisindeki transaction'da tamamlanmamış olacaktır. Bu durumda performansı sadece trigger açısından değil, trigger'ı tetikleyen transaction ile birlikte düşünmek gerekir. Bir trigger tetikleyen Stored Procedure'un performans analizleri yapılırken, düşük performans ile çalışması durumunda, yapılan prosedür incelemelerinde kod bloğunun performansı yüksek olabilir. Ancak, tetiklediği trigger'ın performansı düşük olduğu için bu durum prosedürün performansını da olumsuz yönde etkileyecektir.

Trigger kullanırken, tetikleyen transaction ile trigger gövdesini mantıksal olarak birlikte ele almak gerekir.

Trigger'lar ve Rollback

Trigger'ları anlatırken, özellikle geri alma (`ROLLBACK`) işleminin önemine değindik. `AFTER` trigger gibi önce işlemi yaparak, sonrasında trigger işlemlerini gerçekleştiren trigger'larda bir hata oluşma durumunda ne kadar işlemin geri alınacağı iyi hesaplanmalıdır. Bu sorunun önüne geçmek T-SQL geliştiricinin görevidir. Trigger'ı tetikleyen transaction'ın ve trigger içerisindeki işlemlerin hata olasılıklarını önceden hesaplamalı ve bu olası sorunlara karşı önlem alınmalıdır.

Trigger performansı açısından en önemli unsurlardan biridir. Genel kavram olarak, trigger içerisinde `ROLLBACK` işlemine gerek bırakmadan geliştirme yapmak gerektiği söylenebilir.

Performans İçin İstatistiksel Veri Kullanımı

Bir iş için performans artırımı gerçekleştirmek için, o işin veri parametreleri hakkında bilgi sahibi olunmalıdır. İstatistiksel bilgilere sahip olunmadan

performans işlemleri gerçekleştirilemez. Kısmen gerçekleştirilse bile, gerçekten performans sağlanıp sağlanamadığı konusunda gerekli bilgiler elde edilemez. Veritabanında performans analizi yapabilmek için gerekli tüm istatistikler SQL Server tarafından tutulurlar. Bu istatistikleri kullanarak, hangi sorgunun nasıl çalıştığı, sorgu planı, sorgu süresi, kaynak tüketimi, I/O ve CPU kullanımı, kullanılan bellek miktarı ve sistemde ayrılan bellek, disk, CPU miktarı gibi bilgilere erişilebilir. Sistem kaynakları hakkında bilgi sahibi olunmadan bir sistemin performansını analiz etmek mümkün değildir.

Bu bölümde, SQL Server'da veri ve donanım alt yapısı ile ilgili istatistiksel verileri incelemeyi, kullanmayı ve analiz etme konularını işleyeceğiz.

Performans İçin DMV ve DMF Kullanımı

SQL Server'da veri yönetimi ve verinin sistem tarafındaki bilgi ve istatistiklerini elde edebilmek için geliştirilen view (DMV) ve fonksiyonlardır (DMF). DMV ve DMF'ler disk üzerinde değil, bellek üzerinde kümülatif olarak veri tutarlar. Bu nedenle bu tür sorgular ile elde edilen veriler, SQL Server servisi başlatıldıktan sonra toplanan veriler olduğu için, istatistik açısından kesin kanaate varılamayabilir. DMV ve DMF ile alınan verilerin istatistiksel olarak kritik derecede kullanılabilmesi için SQL Server servisinin uzun süredir çalışıyor olması gerekir.

Açılımları aşağıdaki gibidir:

- DMV (*Dynamic Management View*)
- DMF (*Dynamic Management Function*)

Bu bölümde, DMV ve DMF kullanarak, performans için gerekli bilgileri elde etmeye çalışacağız. Bölümde DMF ve DMF'nin derinliklerine inmemekle birlikte, kritik konulardaki DBA ve geliştiriciye sağlayacağı önemli bilgileri elde edeceğiz.

Bu bölümle birlikte, bir başlangıç yapabileceğiniz DMV ve DMF'ler ile indeksler hakkında bilgiler alarak, kullanılan ve kullanılmayan ya da eksik indeksleri tespit etmek gibi performans açısından gerekli bilgiler elde edebilirsiniz.

DMV ve DMF Hangi Amaç İle Kullanılır?

DMV ve DMF ile sistem tarafındaki problemleri belirleme, performans ayarlarının gözlenmesi ve sistemin izlenmesi gibi istatistiksel veri takibi gerçekleştirilebilir

- **Performans:** Sistemdeki bozuk indeks, eksik indeks ve yüksek I/O işlemi gerçekleştiren sorgular belirlenerek bunların düzenlenmesi için kullanılabilir.
- **Problemleri Gidermek:** Sistem üzerinde bekleyen ya da çalışması engellenen işlemlerin belirlenmesi için kullanılabilir.
- **Sistemi İzlemek:** Sistemde hangi kullanıcının ne tür işlem yaptığını belirlemek ya da execution planların incelenmesi gibi işlemler için kullanılabilir.

DMV ve DMF'ler yapısal olarak basit olmakla birlikte, `JOIN`'li kullanımı ve elde edilen verilerin sistem taraflı teknik bilgiler içermesi nedeniyle karmaşık gelebilir. Bu yöntemler ile elde edilen verileri zaman içerisinde tecrübe edinerek anlayabilir ve ihtiyaçlarınız doğrultusunda farklı kombinasyonlar oluşturarak geliştirebilirsiniz.

Şimdi, temel olarak bazı örnekler gerçekleştirerek DMV ve DMF'lerin yeteneklerini inceleyelim.

Bağlantı Hakkında Bilgi Almak

SQL Server'a gerçekleştirilen bağlantı ile ilgili bilgileri elde etmek için kullanılır.

```
SELECT C.session_id,
 C.auth_scheme,
 C.last_read,
 C.last_write,
 C.client_net_address,
 C.local_tcp_port,
 ST.text AS lastQuery
FROM sys.dm_exec_connections C
CROSS APPLY sys.dm_exec_sql_text(C.most_recent_sql_handle) ST
```

	session_id	auth_scheme	last_read	last_write	client_net_address	local_tcp_port	lastQuery
1	51	NTLM	2013-02-15 15:29:03.460	2013-02-15 15:29:03.463	<local machine>	NULL	CREATE PROCEDURE [dbo].[CleanEventRecords] @M...
2	52	NTLM	2013-02-15 15:30:38.637	2013-02-15 15:30:38.637	<local machine>	NULL	declare @BatchID uniqueidentifier ...
3	53	NTLM	2013-02-15 15:30:21.240	2013-02-15 15:30:21.240	<local machine>	NULL	select value_in_use from sys.configurations where configur...
4	54	NTLM	2013-02-15 15:30:40.270	2013-02-15 15:30:40.270	<local machine>	NULL	SELECT C.session_id, C.auth_scheme, C.last_read, C.l...

- **session_id**: Bağlantının kullandığı SessionID değerini verir.
- **auth_scheme**: Bağlantının SQL Server'mı, Windows Authentication'mı olduğu bilgisini verir.
- **last_read**: Bağlantının en son okuma yaptığı zaman.
- **last_write**: Bağlantının en son yazma yaptığı zaman.
- **client_net_address**: Bağlantıyı yapan makinenin IP adresi.
- **local_tcp_port**: Bağlantının hangi port üzerinden yapıldığı bilgisi.
- **lastQuery**: Bağlantının üzerinden çalıştırılan en son sorgu.

Session Hakkında Bilgi Almak

Session hakkında bilgi almak için **sys.dm_exec_sessions** isimli DMV kullanılabilir. Sisteme ne zaman login olunduğu, session'ı açan program ve makinenin bilgisi gibi bir çok bilgi sunar.

```
SELECT login_name, COUNT(session_id) AS session_count, login_time
FROM sys.dm_exec_sessions
GROUP BY login_name, login_time;
```

	login_name	session_count	login_time
1	sa	1	2013-02-15 10:43:09.967
2	sa	7	2013-02-15 10:43:09.980
3	sa	1	2013-02-15 10:43:11.767
4	sa	1	2013-02-15 10:43:12.250
5	sa	2	2013-02-15 10:43:12.297
6	sa	2	2013-02-15 10:43:40.847
7	sa	1	2013-02-15 10:43:49.000

Veritabanı Sunucusu Hakkında Bilgi Almak

CPU, I/O ve RAM ile ilgili kararlar vermek ve performans analizi yapabilmek için veritabanı sunucusu hakkında bazı bilgilere sahip olmak gerekebilir. Veritabanı sunucusunun ne zaman başlatıldığı, en son ne zaman başlatıldığı, CPU adeti ve fiziksel hafıza gibi daha bir çok sunucu ile ilgili temel seviye bilgileri elde etmek için aşağıdaki DMV kullanılır.

```
SELECT * FROM sys.dm_os_sys_info;
```

Sorgu sonucunda listelenen sütunları inceleyelim.

- **sqlserver_start_time**: SQL Server servisinin ne zaman başlatıldığı.
- **ms_ticks**: Veritabanı sunucusunun en son başlatıldığı zamandan beri geçen süre(milisaniye).
- **sqlserver_start_time_ms_ticks**: SQL Server servisi başladığında ms_ticks süresi.
- **cpu_count**: Sunucunun sahip olduğu CPU sayısı.
- **physical_memory_kb**: Sunucunun fiziksel hafıza boyutu.
- **virtual_memory_kb**: Sunucunun sanal hafıza boyutu.
- **max_workers_count**: Maksimum kaç thread'in oluşturulabileceği bilgisi.
- **scheduler_count**: Kullanıcı işlerini yapacak scheduler bilgisi.
- **scheduler_total_count**: Toplam scheduler adeti.

Performans analizini gerçekleştirirken SQL Server'ın ne kadar süredir çalıştığını öğrenmek gerekebilir. Şimdi, SQL Server'ın kaç dakikadır çalıştığını bulalım.

```
SELECT
DATEDIFF(MINUTE, sqlserver_start_time, CURRENT_TIMESTAMP) AS UpTime
FROM sys.dm_os_sys_info;
```

Aynı işlemi gerçekleştirecek bir başka yöntem de, **ms_ticks** ve **sqlserver_start_time_ms_ticks** sütunlarını kullanmaktır.

	UpTime
1	289

```
SELECT (ms_ticks-sqlserver_start_time_ms_ticks)/1000/60 AS UpTime
FROM sys.dm_os_sys_info;
```

Fiziksel Bellek Hakkında Bilgi Almak

Daha önce sunucu ve dosyalar ile ilgili DMV kullanımlarını inceledik. SQL Server kullanımında performansı etkileyen en önemli unsurlardan biri de bellek kullanımıdır. Fiziksel bellek ile ilgili istatistiksel bilgileri elde etmek için **sys.dm_os_sys_memory** isimli DMV kullanılır.

```
SELECT * FROM sys.dm_os_sys_memory;
```

- **total_physical_memory_kb**: Sunucuda bulunan toplam fiziksel bellek değeri.
- **available_physical_memory_kb**: Sunucuda bulunan fiziksel belleğin ne kadarının boşta olduğu bilgisi.
- **totalpage_file_kb**: Page File'ın toplam ulaşabileceği boyut.
- **available_page_file_kb**: Kullanılabilecek Page File boyutu.
- **system_memory_state_desc**: Bellek durumuna göre, aşağıdaki açıklama mesajlarından birini gösterir.

  - Available physical memory is high

  - Available physical memory is low

  - Physical memory usage is steady

Her sistemde değişken değerlere sahip olacak bellek değerlerini almak için gerekli DMV sorgusunu hazırlayalım.

```
SELECT
 total_physical_memory_kb/1024 AS [ToplamFizikselBellek],
 (total_physical_memory_kb-available_physical_memory_kb)/1024
          AS [KullanılanFizikselBellek],
 available_physical_memory_kb/1024 AS [KullanılabilirFizikselBellek],
 total_page_file_kb/1024 AS [ToplamPageF(MB)],
 (total_page_file_kb - available_page_file_kb)/1024 AS[KullanılanPageF(MB)],
 available_page_file_kb/1024 AS [KullanılabilirPageF(MB)],
 system_memory_state_desc
FROM
 sys.dm_os_sys_memory;
```

	ToplamFizikselBellek	KullanılanFizikselBellek	KullanılabilirFizikselBellek	ToplamPageF(MB)	KullanılanPageF(MB)	KullanılabilirPageF(MB)	system_memory_state_desc
1	8172	3359	4812	16343	4801	11541	Available physical memory is high

Aktif Olarak Çalışan İstekleri Sorgulamak

SQL Server üzerinde, şuan aktif olan istekleri sorgulamak için sys.dm_exec_requests isimli DMV kullanılır. Bu isteklerin durumunu, sorgu metnini ve hangi sorguda beklediği gibi birçok bilgi elde edilebilir.

- **dbname**: Session'ın çalıştığı veritabanının ismi.
- **login_name**: Session'ın hangi kullanıcı adı ile açıldığı bilgisi.

- **host_name**: Session'ın makine bilgisi.
- **text**: Çalışmaya devam eden sorgunun metnini gösterir.
- **statemnent_text**: Çalışmaya devam eden sorgunun şuan çalışmakta olan kısmını gösterir. 100 satırlık bir sorgu çalışıyorsa, çalışma sırası 45. satırda ise, buradaki sorguları gösterecektir.
- **blocking_session_id**: Başka bir session tarafından bir session bloklandıysa, bu session'ı bloklayan session'ın ID değeri gelir.
- **status**: Sorgunun şu anki durumunu gösterir.
- **wait_type**: Sorgu şu an bloklu durumdaysa, bu blok'un tipini verir.
- **wait_time**: Sorgu şu an bloklu durumdaysa, ne kadar zamandır (*milisaniye*) bloklu olduğunu verir.
- **percent_complete**: İşlem gerçekleşirken, işlemin yüzde kaçında olduğunu verir.
- **estimated_completion_time**: İşlem gerçekleşirken, işlemin tamamlanmasına kaç milisaniye kaldığını verir.

Bir sorgu ekranı açarak aşağıdaki döngü oluşturacak sorguyu çalıştıralım.

```
DECLARE @test INT = 0;
WHILE 1 = 1
SET @test = 1;
```

Döngü devam ederken, yeni bir sorgu ekranı açarak aşağıdaki sorguyu çalıştıralım.

```
SELECT DB_NAME(er.database_id) AS DB_ismi,
 es.login_name,
 es.host_name,
 st.text,
      SUBSTRING(st.text, (er.statement_start_offset/2) + 1,
        ((CASE er.statement_end_offset
          WHEN -1 THEN DATALENGTH(st.text)
          ELSE er.statement_end_offset
          END - er.statement_start_offset)/2) + 1) AS ifade_metni,
   er.blocking_session_id,
   er.status,
```

```
    er.wait_type,
    er.wait_time,
    er.percent_complete,
    er.estimated_completion_time
FROM sys.dm_exec_requests er
     LEFT JOIN sys.dm_exec_sessions es ON es.session_id = er.session_id
     CROSS APPLY sys.dm_exec_sql_text(er.sql_handle) st
     CROSS APPLY sys.dm_exec_query_plan(er.plan_handle) qp
WHERE er.session_id > 50 AND er.session_id != @@SPID;
```

	DB_ismi	login_name	host_name	text	ifade_metni	blocking_session_id	status	wait_type	wait_time	percent_complete	estimated_completion_time
1	AdventureWorks2012	dijibil-pc\dijibil	DIJIBIL-PC	DECLARE @test INT = 0; WHILE 1 = 1 SET @test =...	WHILE 1 = 1	0	running	NULL	0	0	0

Cache'lenen Sorgu Planları Hakkında Bilgi Almak

SQL Server, çalıştırılan tüm SQL sorguları (adhoc, prosedür, trigger, view vb.) için hazırlanan sorgu planlarını saklar. Aynı sorgunun bir sonraki çalıştırılmasında, saklanan bu sorgu planlarını kullanarak performans elde eder.

Cache'lenen sorgu planları hakkında bilgi almak için **`sys.dm_exec_cached_plans`** isimli DMV kullanılır.

```
SELECT * FROM sys.dm_exec_cached_plans;
```

Sorgu sonucunda listelenen sütunları inceleyelim.

- **`usecounts`**: Cache'lenen sorgu planının, cache'lendikten sonra kaç defa kullanıldığını gösterir.
- **`size_in_bytes`**: Cache'lenen sorgu planının hafızada kaç bayt yer kapladığını gösterir.
- **`objtype`**: Cache'lenen nesnenin tipini (Adhoc, Proc, Trigger vb.) verir.
- **`plan_handle`**: Cache'lenen nesnenin planını verir.

Cache'lenmiş nesnelerden, cache'lendikten sonra 3 den fazla kullanılanları listeleyelim.

```
SELECT CP.usecounts,
 ST.text,
 QP.query_plan,
 CP.cacheobjtype,
 CP.objtype,
 CP.size_in_bytes
FROM sys.dm_exec_cached_plans CP
 CROSS APPLY sys.dm_exec_query_plan(CP.plan_handle) QP
 CROSS APPLY sys.dm_exec_sql_text(CP.plan_handle) ST
WHERE CP.usecounts > 3
ORDER BY CP.usecounts DESC;
```

	usecounts	text	query_plan	cacheobjtype	objtype	size_in_bytes
1	1737	declare @BatchID uniqueide...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	Compiled Plan	Adhoc	229376
2	1737	declare @BatchID uniqu...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	Compiled Plan	Adhoc	147456
3	841	() select table_id, item_guid, oplsn_fseqno, oplsn_bOf...	NULL	Compiled Plan	Prepared	32768
4	841	() select table_id, item_guid, oplsn_fseqno, oplsn_bOf...	NULL	Compiled Plan	Prepared	32768
5	841	() select table_id, item_guid, oplsn_fseqno, oplsn_bOf...	NULL	Compiled Plan	Prepared	32768
6	841	() select table_id, item_guid, oplsn_fseqno, oplsn_bOf...	NULL	Compiled Plan	Prepared	32768
7	841	() select table_id, item_guid, oplsn_fseqno, oplsn_bOf...	NULL	Compiled Plan	Prepared	32768

Stored Procedure kullanırken, sorgu performansını hesaplamak için sorgu planlarını incelemek ve gerekli T-SQL müdahalelerini prosedür içerisinde gerçekleştirmek gerekir. Şimdi, cache'lenen Stored Procedure'lerin sorgu planlarını elde edelim.

```
SELECT
 DB_NAME(st.dbid) AS DB_ismi,
 OBJECT_SCHEMA_NAME(ST.objectid, ST.dbid) AS sema_ismi,
 OBJECT_NAME(ST.objectid, ST.dbid) AS nesne_ismi,
 ST.text,
 QP.query_plan,
 CP.usecounts,
 CP.size_in_bytes
FROM sys.dm_exec_cached_plans CP
 CROSS APPLY sys.dm_exec_query_plan(CP.plan_handle) QP
 CROSS APPLY sys.dm_exec_sql_text(CP.plan_handle) ST
WHERE ST.dbid <> 32767
AND CP.objtype = 'Proc';
```

	DB_ismi	sema_ismi	nesne_ismi	text	query_plan	usecounts	size_in_bytes
1	msdb	dbo	fn_syspolicy_is_automation_enabled	CREATE FUNCTION fn_syspolicy_is_automation_enabled() ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	1	147456
2	ReportServer	dbo	CleanExpiredEditSessions	CREATE PROC [dbo].[CleanExpiredEditSessions] @Max...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	29	524288
3	ReportServer	dbo	CleanExpiredJobs	CREATE PROCEDURE [dbo].[CleanExpiredJobs] AS SET ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	29	57344
4	ReportServer	dbo	CleanOrphanedSnapshots	CREATE PROCEDURE [dbo].[CleanOrphanedSnapshots] ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	29	1556480
5	ReportServer	dbo	CleanExpiredCache	CREATE PROCEDURE [dbo].[CleanExpiredCache] AS SE...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	29	147456
6	ReportServer	dbo	DeleteExpiredPersistedStreams	CREATE PROCEDURE [dbo].[DeleteExpiredPersistedStrea...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	29	73728
7	ReportServer	dbo	CleanExpiredSessions	CREATE PROCEDURE [dbo].[CleanExpiredSessions] @Se...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	29	532480

Yukarıdaki sorguyu çalıştırdıktan sonra listelenen sonucu inceleyin. Daha sonra, aşağıdaki gibi iki prosedür çalıştırarak tekrar sorgulayın. Kendi çalıştırdığınız prosedürün sorgu planını sonuçlar içerisinde göreceksiniz.

Cache'lenen Sorgu Planların Nesne Tipine Göre Dağılımı

Cache'lenen sorguların hangi nesne tipinde oldukları SQL Server'da kayıtlıydı. Bu nesne tipi değerlerini kullanarak sorgu planlarının nesne tipine göre gruplanmasını ve hangi nesne tipinde kaç adet sorgu planı olduğu hesaplanabilir.

```
SELECT CP.objtype,COUNT(*) AS nesne_toplam
FROM sys.dm_exec_cached_plans CP
GROUP BY CP.objtype
ORDER BY 2 DESC;
```

	objtype	nesne_toplam
1	View	90
2	Adhoc	58
3	Prepared	41
4	Proc	19
5	Check	6
6	UsrTab	1

Parametre Olarak Verilen sql_handle'ın SQL Sorgusunu Elde Etmek

`sys.dm_exec_sql_text` isimli DMF'ye, parametre olarak verilen sql_handle'ın, SQL sorgu metnine ulaşılabilir. Örneğin; SQL Server'da aktivitelerin takip edilmesini sağlayan **Activity Monitor**'de bir **Session ID** belirlenerek, bu process'in SQL sorgu metnine ulaşılabilir.

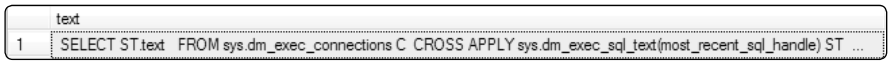
```
SELECT ST.text
FROM sys.dm_exec_requests R
```

```
CROSS APPLY sys.dm_exec_sql_text(sql_handle) ST
WHERE session_id = @@SPID;
```

	text
1	SELECT ST.text FROM sys.dm_exec_requests R CROSS APPLY sys.dm_exec_sql_text(sql_handle) ST WHERE session_id = @...

Aktif olan bağlantıların uyguladıkları son SQL script'leri de öğrenilebilir.

```
SELECT ST.text
FROM sys.dm_exec_connections C
CROSS APPLY sys.dm_exec_sql_text(most_recent_sql_handle) ST
WHERE session_id = @@SPID;
```



	text
1	SELECT ST.text FROM sys.dm_exec_connections C CROSS APPLY sys.dm_exec_sql_text(most_recent_sql_handle) ST ...

sql_handle bilgisine erişmek ya da **CROSS APPLY** ile birbirine bağlamak için aşağıdaki DMV'ler kullanılabilir.

```
sys.dm_exec_sql_text
sys.dm_exec_requests
sys.dm_exec_cursors
sys.dm_exec_xml_handles
sys.dm_exec_query_memory_grants
sys.dm_exec_connections
```

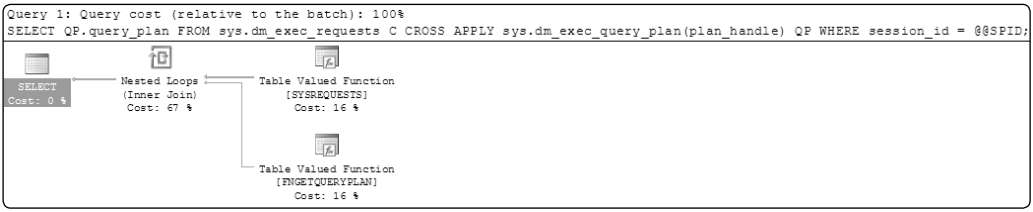
Parametre Olarak Verilen plan_handle'ın Sorgu Planını Elde Etmek

plan_handle olarak verilen sorgunun XML formatında sorgu planını elde etmek için kullanılır.

Aktif olarak devam eden process'lerin sorgu planlarını elde edelim.

```
SELECT QP.query_plan
FROM sys.dm_exec_requests C
CROSS APPLY sys.dm_exec_query_plan(plan_handle) QP
WHERE session_id = @@SPID;
```

Sorgu sonucunda gelen XML bağlantı açıldığında sorgu planı, aşağıdaki gibi görsel olarak görüntülenecektir.



Sorgu İstatistikleri

sys.dm_exec_query_stats isimli DMV kullanılarak sorgu istatistikleri elde edilebilir. Bu DMV ile, cache'lenmiş sorguların CPU süreleri (*worker time*), toplam çalışma süreleri (*elapsed time*), fiziksel okuma (physical read), mantıksal okuma (*logical read*), mantıksal yazmalar (*logical write*) gibi bir çok parametre analiz edilerek sorunlu sorgular üzerinde iyileştirme çalışmaları yapabilmek için SQL Server yöneticisi ve geliştiricilerine istatistiksel bilgiler verir.

```
select * from sys.dm_exec_query_stats;
```

Sorgu sonucunda 5 ana bilgi gelmektedir. Bunlar;

- **workertime**: Sorgunun CPU üzerinde geçirdiği süre.
- **elapsedTime**: Sorgunun toplam süresi.
- **physicalread**: Diskten yapılan okuma miktarı.
- **logicalread**: Bellekten yapılan okuma miktarı.
- **logicalwrite**: Belleğe yapılan yazma miktarı.

Bu istatistik özelliklerinin her biri için 4 farklı istatistik sütunu yer alır. Örneğin; workertime özelliği için, **execution_count**'un 10 olduğunu düşünürsek;

- **min_worker_time**: Sorgunun 10 kez çalışmasında, CPU üzerinde minimum zamanı geçirenin kaç microsaniye olduğunu gösterir.
- **max_worker_time**: Sorgunun 10 kez çalışmasında, CPU üzerinde maximum zamanı geçirenin kaç microsaniye olduğunu gösterir.
- **last_worker_time**: **Last_Execution_Time** olarak verilen zamanda, çalışan sorgunun CPU üzerinde kaç microsaniye iş yaptığını gösterir.

- **total_worker_time**: Sorgunun 10 kez çalışmada toplamda kaç microsaniye CPU üzerinde iş yaptığını gösterir.

Şimdi birkaç örnek yaparak sorgu üzerinde deneyelim.

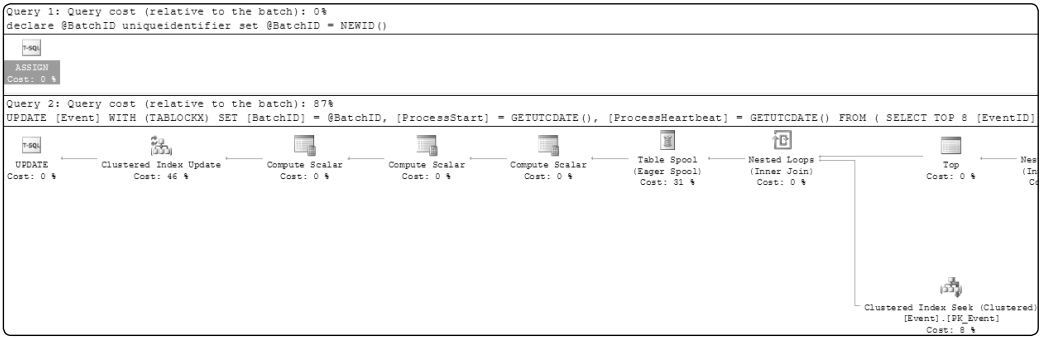
En fazla CPU tüketen ilk 20 sorguyu listeleyelim.

```
SELECT
     q.[text],
     SUBSTRING(q.text, (qs.statement_start_offset/2)+1,
        ((CASE qs.statement_end_offset
           WHEN -1 THEN DATALENGTH(q.text)
           ELSE qs.statement_end_offset
          END - qs.statement_start_offset)/2) + 1) AS ifade_metni,
     qs.last_execution_time,
     qs.execution_count,
     qs.total_worker_time / 1000000 AS toplam_cpu_zaman_sn,
     qs.total_worker_time / qs.execution_count / 1000 AS avg_cpu_zaman_ms,
     qp.query_plan,
     DB_NAME(q.dbid) AS veritabani_ismi,
       q.objectid,
       q.number,
       q.encrypted
FROM
    (SELECT TOP 20
          qs.last_execution_time,
          qs.execution_count,
     qs.plan_handle,
          qs.total_worker_time,
          qs.statement_start_offset,
          qs.statement_end_offset
    FROM sys.dm_exec_query_stats qs
    ORDER BY qs.total_worker_time desc) qs
CROSS APPLY sys.dm_exec_sql_text(plan_handle) q
CROSS APPLY sys.dm_exec_query_plan(plan_handle) qp
ORDER BY qs.total_worker_time DESC;
```

	text	ifade_metni	last_execution_time	execution_count	toplam_cpu_zaman_sn	avg_cpu_zaman_ms
1	declare @BatchID uniqueidentif...	UPDATE [Event] WITH (TABLOCKX) ...	2013-02-15 15:41:18.780	1777	0	0
2	declare @BatchID uniqueidentifier ...	UPDATE [Notifications] WITH (TABLOCKX) ...	2013-02-15 15:41:18.780	1777	0	0
3	declare @BatchID uniqueidentifier ...	select top 8 -- Notification data ...	2013-02-15 15:41:18.780	1777	0	0
4	SELECT CP.usecounts, ST.text, QP.query_plan, CP.c...	SELECT CP.usecounts, ST.text, QP.query_plan, CP....	2013-02-15 15:34:43.443	1	0	261
5	(@_msparam_0 nvarchar(4000),@_msparam_1 nvarchar(400...	SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], sp....	2013-02-15 15:30:37.083	1	0	216
6	(@_msparam_0 nvarchar(4000))SELECT CAST(COLLATION...	SELECT CAST(COLLATIONPROPERTY(name, 'LCID') AS i...	2013-02-15 15:41:14.020	12	0	17
7	(@_msparam_0 nvarchar(4000))SELECT SCHEMA_NAME(u...	SELECT SCHEMA_NAME(udf.schema_id) AS [Schema], u...	2013-02-15 15:30:35.507	1	0	204

Sorgu sonucunda hangi sorguların kaç kez çalıştırıldığı, çalıştırma süresi ve sorgunun SQL kodları gibi bir çok farklı parametre hakkında bilgi alabiliyoruz.

Bu sonuçlarda herhangi bir kaydın sorgu planına ulaşmak için **query_plan** sütunundaki XML veriye Mouse ile tıklanması yeterli olacaktır. Aşağıdaki gibi bir ekran görüntülenir.



SQL Server'da bir başka en çok ihtiyaç duyulan istatistik ise, en çok I/O işlemi yapan sorguların bulunmasıdır. Bu sorgular üzerinde performans iyileştirmeleri yapmak için incelemeler gerçekleştirilir.

En fazla I/O gerçekleştiren ilk 20 sorguyu listeleyelim.

```
SELECT
     q.[text],
     SUBSTRING(q.text, (qs.statement_start_offset/2)+1,
        ((CASE qs.statement_end_offset
           WHEN -1 THEN DATALENGTH(q.text)
           ELSE qs.statement_end_offset
         END - qs.statement_start_offset)/2) + 1) AS ifade_metni,
     qs.last_execution_time,
     qs.execution_count,
     qs.total_logical_reads AS toplam_mantiksal_okuma,
     qs.total_logical_reads/execution_count AS avg_mantiksal_okuma,
     qs.total_worker_time/1000000 AS total_cpu_time_sn,
     qs.total_worker_time/qs.execution_count/1000 AS avg_cpu_zaman_ms,
     qp.query_plan,
     DB_NAME(q.dbid) AS veritabani_ismi,
       q.objectid,
```

```
        q.number,
        q.encrypted
FROM
    (SELECT TOP 20
          qs.last_execution_time,
          qs.execution_count,
     qs.plan_handle,
          qs.total_worker_time,
          qs.total_logical_reads,
          qs.statement_start_offset,
          qs.statement_end_offset
    FROM sys.dm_exec_query_stats qs
    ORDER BY qs.total_worker_time desc) qs
CROSS APPLY sys.dm_exec_sql_text(plan_handle) q
CROSS APPLY sys.dm_exec_query_plan(plan_handle) qp
ORDER BY qs.total_logical_reads DESC;
```

	text	ifade_metni	last_execution_time	execution_count	toplam_mantiksal_okuma	avg_mantiksal_okuma	total_cpu_time_sn
1	(@_msparam_0 nvarchar(4000),@_msparam_1 nvarchar(400...	SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], sp....	2013-02-15 15:30:45.720	1	80947	80947	0
2	(@_msparam_0 nvarchar(4000),@_msparam_1 nvarchar(400...	SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], sp....	2013-02-15 15:30:37.083	1	80442	80442	0
3	(@_msparam_0 nvarchar(4000),@_msparam_1 nvarchar(400...	SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], sp....	2013-02-15 15:37:48.563	1	80403	80403	0
4	(@_msparam_0 nvarchar(4000))SELECT SCHEMA_NAME(u...	SELECT SCHEMA_NAME(udf.schema_id) AS [Schema], ud...	2013-02-15 15:30:35.507	1	36390	36390	0
5	(@_msparam_0 nvarchar(4000))SELECT SCHEMA_NAME(v...	SELECT SCHEMA_NAME(v.schema_id) AS [Schema], v.na...	2013-02-15 15:30:34.863	1	17246	17246	0
6	declare @BatchID uniqueidentifier ...	select top 8 -- Notification data ...	2013-02-15 15:43:58.783	1793	14428	8	0
7	(@_msparam_0 nvarchar(4000),@_msparam_1 nvarchar(400...	SELECT clmns.name AS [Name], clmns.column_id AS [ID], ...	2013-02-15 15:41:14.487	210	6049	28	0

Sorgu sonucunda en çok I/O işlemi yapan ilk 20 sorgu listelendi. Bilgiler arasında sorgunun ne kadar çalıştırıldığı, ne kadar I/O işlemi yaptığı (**`total_logical_read`**) ve işlemin ne kadar sürdüğü gibi bilgiler yer alır.

Bu tür sorgular, sorguların performansını artırmak için çok önemlidir. Yüksek kaynak tüketen sorgularda genel olarak, performansı olumsuz etkileyecek sorgu kullanımı mevcuttur. Bu yanlış geliştirmeden kaynaklı sorguları düzenlemek performansı artıracaktır.

İşlemlerdeki Bekleme Sorunu Hakkında Bilgi Almak

İşlemler birçok durumda beklemeler yapar. Sebebini tam olarak anlayabilmek için, bazı sistem özelliklerinden bilgi almak gerekir. Network I/O işlemlerinde mi bekleme var yoksa CPU gibi donanım yetersizliği nedeniyle mi? Bunu anlayabilmek için **`sys.dm_os_wait_stats`** isimli DMV kullanılır.

```
SELECT * FROM sys.dm_os_wait_stats;
```

	wait_type	waiting_tasks_count	wait_time_ms	max_wait_time_ms	signal_wait_time_ms
1	MISCELLANEOUS	0	0	0	0
2	LCK_M_SCH_S	0	0	0	0
3	LCK_M_SCH_M	11	9	2	0
4	LCK_M_S	14	223482	37195	2
5	LCK_M_U	0	0	0	0
6	LCK_M_X	0	0	0	0
7	LCK_M_IS	0	0	0	0

Sorgu sonucunda dönen sütunları inceleyelim.

- **`wait_type`**: Bekleme tipi.
- **`waiting_tasks_count`**: Bu bekleme ile kaç kez karşılaşıldı.
- **`wait_time_ms`**: Toplam bekleme süresi (milisaniye).
- **`max_wait_time_ms`**: Bekleme tipi için gerçekleşen maksimum bekleme süresi.
- **`signal_wait_time_ms`**: Bekleme tipinin signaled edildiği zaman ile çalışmaya başladığı zaman arasında geçen sürenin toplamı.

`sys.dm_os_wait_stats` isimli DMV, kümülatif olarak veri toplayan bir DMV'dir. Bellekte çalışan bu tür bir view'in topladığı verileri temizleme ve yeniden istatistiksel bilgiler toplamaya başlaması sağlanabilir. Bunun için iki yöntem vardır. Bunlar, SQL Server servisinin restart edilmesiyle bellek üzerindeki tüm işlemlerin sıfırlanması ya da aşağıdaki gibi topladığı veriyi temizlemekti.

```
DBCC SQLPERF('sys.dm_os_wait_stats', CLEAR);
```

```
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

Disk Cevap Süresi Hakkında Bilgi Almak

Performans için önemli noktalardan biri de **Disk Response Time** (*disk cevap süresi*)'dır. SQL Server isteğine, disk'in ne kadar gecikmeli olarak cevap verdiğini gösteren bir ifadedir. SQL Server açısından önerilen değer 10 milisaniyenin altında olmasıdır.

Bu bilgiyi bize sunan DMV'nin adı **`sys.dm_io_virtual_file_stats`**'dır.

Bu DMV iki parametre alır. Bunlar;

- **DB ID**: Sadece ID değeri verilen veritabanının dosyalarını sorgular.
- **File ID**: Bir veritabanının sadece belirli dosyalarının response time'larına bakmak için kullanılır. Örneğin; bir veritabanının sadece log file'ının response time'larına bakmak için log file'ın **File ID** değeri parametre olarak verilir.

```
SELECT * FROM sys.dm_io_virtual_file_stats(null, null);
```

Parametreleri **NULL** olarak belirterek sorguladığınızda aşağıdaki gibi bir sonuç listelenecektir.

	database_id	file_id	sample_ms	num_of_reads	num_of_bytes_read	io_stall_read_ms	num_of_writes	num_of_bytes_written	io_stall_write_ms	io_stall	size_on_disk_bytes	file_handle
1	1	1	18217952	55	3457024	642	0	0	0	642	5111808	0x000000000000076C
2	1	2	18217952	10	372736	26	12	49152	46	72	1835008	0x0000000000000770
3	2	1	18217952	31	1851392	12	2	16384	1	13	8388608	0x00000000000000C10
4	2	2	18217952	5	503808	34	4	135168	3	37	524288	0x00000000000000C0C
5	3	1	18217952	80	5070848	18	2	16384	1	19	4259840	0x00000000000000C04
6	3	2	18217952	9	516096	18	8	40960	3	21	1310720	0x00000000000000C08
7	4	1	18217952	66	4087808	146	1	8192	2	148	17498112	0x00000000000000BA8

database_id: Veritabanı ID'si.

- **file_id**: File'ın ID'si.
- **sample_ms**: SQL Server'ın en son başladığı andan itibaren kaç milisaniye geçtiğini gösterir.
- **num_of_reads**: Dosyadan kaç kez okuma işlemi yapıldığını gösterir.
- **num_of_bytes_read**: Yapılan bu okumalarda toplam kaç bayt veri okunduğunu gösterir.
- **io_stall_read_ms**: Yapılan okuma işlemlerinde kullanıcının ne kadar beklediğini gösterir.
- **io_stall_write_ms**: Yapılan yazma işlemlerinde kullanıcının ne kadar beklediğini gösterir.
- **io_stall**: Okuma ve yazma işlemi için toplam bekleme süresini gösterir.
- **num_of_writes**: Dosyaya kaç kez yazma işlemi yapıldığını gösterir.
- **num_of_bytes_write**: Yazma işlemlerinde toplam kaç bayt veri yazıldığını gösterir.
- **size_on_disk_bytes**: Dosya'nın içerisinde gerçekten kullanılmakta olan kısmı bayt cinsinden ifade eder.

AdventureWorks veritabanı dosyalarının okuma, yazma gibi istatistiklerini gösterecek bir sorgu hazırlayalım.

```
SELECT db_name(mf.database_id) AS veritabani_ismi,
  mf.name AS mantiksal_dosya_ismi,
  io_stall_read_ms/num_of_reads AS cevap_okuma_suresi,
  io_stall_write_ms/num_of_writes AS cevap_yazma_suresi,
  io_stall/(num_of_reads+num_of_writes) AS cevap_suresi,
       num_of_reads, num_of_bytes_read, io_stall_read_ms,
  num_of_writes, num_of_bytes_written, io_stall_write_ms,
  io_stall, size_on_disk_bytes
FROM
 sys.dm_io_virtual_file_stats(DB_ID('AdventureWorks'), NULL) AS divFS
JOIN
 sys.master_files AS MF
 ON MF.database_id = divFS.database_id
 AND MF.file_id = divFS.file_id;
```

	veritabani_ismi	mantiksal_dosya_ismi	cevap_okuma_suresi	cevap_yazma_suresi	cevap_suresi	num_of_reads	num_of_bytes_read	io_stall_read_ms	num_of_writes	num_of_bytes_written	io_stall_write_ms	io_stall
1	AdventureWorks2012	AdventureWorks2012_Data	25	0	25	88	5578752	2255	1	8192	0	2255
2	AdventureWorks2012	AdventureWorks2012_Log	296	0	110	10	471040	2960	17	73728	13	2973

Veritabanının veri ve loglarını tutan MDF ve LDF uzantılı dosyaları hakkında istatistikleri elde ettik.

Bekleyen I/O İstekleri Hakkında Bilgi Almak

Bekleme durumundaki I/O istekleri hakkında bilgi almak için **sys.dm_io_pending_io_requests** isimli DMV kullanılır.

```
SELECT * FROM sys.dm_io_pending_io_requests;
```

	io_completion_request_address	io_type	io_pending_ms_ticks	io_pending	io_completion_routine_address	io_user_data_address	scheduler_address	io_handle	io_offset
1	0x000000037F5B83E8	network	18192226	1	0x000007FEF2F11C50	0x000000037F5B83E8	0x000000037F140040	0x0000000000000AB0	0

Bu DMV ile bekleme (*pending*) durumundaki I/O istekleri getirilir. Dolayısıyla, sorgu sonucu boş olarak gelirse, beklemede olan bir I/O isteği olmadığı anlamına gelecektir.

Sorgu sonucunda gelen sütunları inceleyelim.

- **io_type**: I/O isteğinin şuan hangi aşamada bekleme durumunda olduğunu gösterir.

- **io_pending_ms_ticks:** Toplam ne kadar süredir (*milisaniye*) bekleme durumunda olduğunu belirtir.
- **io_pending:** İsteğin gerçekten bekleme durumunda mı, yoksa işlem bitirilmiş olmasına rağmen SQL Server'a ulaştırılmadı mı durumunu gösterir. Değer 1 ise istek beklemede, 0 ise istek tamamlanmış ama SQL Server bu bilgiyi almamıştır.
- **io_handle:** I/O handle'ı.

Beklemede olan I/O isteklerini getirelim.

```
SELECT
   FS.database_id AS veritabani_id,
   db_name(FS.database_id) AS veritabani_ismi,
   MF.name AS logical_file_name,
   IP.io_type,
   IP.io_pending_ms_ticks,
   IP.io_pending
FROM sys.dm_io_pending_io_requests IP
LEFT JOIN sys.dm_io_virtual_file_stats(null, null) FS
     ON FS.file_handle = IP.io_handle
LEFT JOIN sys.master_files MF
     ON MF.database_id = FS.database_id
     AND MF.file_id = FS.file_id
```

	veritabani_id	veritabani_ismi	logical_file_name	io_type	io_pending_ms_ticks	io_pending
1	NULL	NULL	NULL	network	18217617	1

I/O istekleri ve bunların hızlı bir şekilde karşılanması performans açısından çok önemlidir. Bir SQL Server DBA'in başlıca görevlerinden biri bu performans parametrelerini takip etmektir.

DBCC SQLPERF İLE DMV İSTATİSTİKLERİNİ TEMİZLEMEK

DMV'ler, SQL Server servisi başlatıldıktan sonraki istatistikleri toplayan ve servis yeniden başlatılınca, bellekteki bu verileri temizleyen kümülatif veri tutan yapılardır.

DMV'lerin tuttuğu verileri servisi yeniden başlatmadan da temizlemek mümkündür. Bunun için DBCC SQLPERF kullanılabilir.

sys.dm_os_wait_stats isimli DMV verisini, **wait_time_ms** değerine göre listeleyelim.

```
SELECT * FROM sys.dm_os_wait_stats
WHERE wait_time_ms > 0;
```

	wait_type	waiting_tasks_count	wait_time_ms	max_wait_time_ms	signal_wait_time_ms
1	PAGEIOLATCH_SH	1	80	80	0
2	LAZYWRITER_SLEEP	188	188458	1016	0
3	ASYNC_NETWORK_IO	1466	433	14	97
4	SLEEP_TASK	513	95812	1027	1
5	SOS_SCHEDULER_YIELD	46756	124	1	93
6	LOGMGR_QUEUE	1455	189154	136	4
7	REQUEST_FOR_DEADLOCK_SEARCH	38	189992	4999	189992

```
DBCC SQLPERF('sys.dm_os_wait_stats', CLEAR);
```

```
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

Tekrar aynı listeleme sorgusu çalıştırıldığında, DMV'nin temizlenmiş olduğu görülecektir.

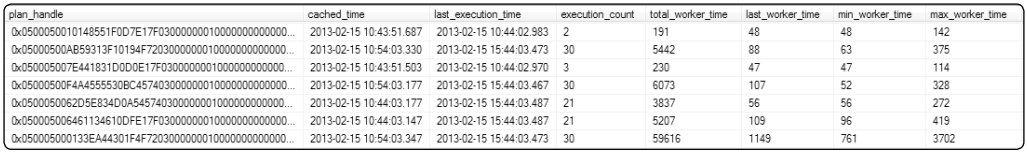
	wait_type	waiting_tasks_count	wait_time_ms	max_wait_time_ms	signal_wait_time_ms
1	LAZYWRITER_SLEEP	27	27065	1002	0
2	ASYNC_NETWORK_IO	232	46	3	7
3	SLEEP_TASK	68	13325	1025	0
4	SOS_SCHEDULER_YIELD	6573	13	0	9
5	LOGMGR_QUEUE	205	26651	133	0
6	REQUEST_FOR_DEADLOCK_SEARCH	6	29998	4999	29998
7	SQLTRACE_INCREMENTAL_FLUSH_SLEEP	6	24014	4002	0

Siz de, örnekteki yöntemi kullanarak **sys.dm_os_latch_stats** isimli ya da bir başka DMV'nin tuttuğu veriyi listeleyerek DBCC SQLPERF ile temizleyebilirsiniz.

Stored Procedure İstatistikleri Hakkında Bilgi Almak

Stored Procedure'lerin CPU ve I/O gibi kaynak tüketimleri hakkında bilgi alarak performans analizleri gerçekleştirmeyi sağlayan DMV'nin ismi **sys.dm_exec_procedure_stats**'dır.

```
SELECT * FROM sys.dm_exec_procedure_stats;
```



plan_handle	cached_time	last_execution_time	execution_count	total_worker_time	last_worker_time	min_worker_time	max_worker_time
0x05000500101485 51F0D7E17F0300000001000000000000...	2013-02-15 10:43:51.687	2013-02-15 10:44:02.983	2	191	48	48	142
0x05000500AB59313F10194F720300000001000000000000...	2013-02-15 10:54:03.330	2013-02-15 15:44:03.473	30	5442	88	63	375
0x05000500 7E441831D0D0E17F0300000001000000000000...	2013-02-15 10:43:51.503	2013-02-15 10:44:02.970	3	230	47	47	114
0x05000500F4A4555530BC45740300000001000000000000...	2013-02-15 10:54:03.177	2013-02-15 15:44:03.467	30	6073	107	52	328
0x05000500 62D5E834D0A5457403000000010000000000...	2013-02-15 10:44:03.177	2013-02-15 15:44:03.487	21	3837	56	56	272
0x05000500 6461134610DFE17F0300000001000000000000...	2013-02-15 10:44:03.147	2013-02-15 15:44:03.487	21	5207	109	96	419
0x05000500 0133EA44301F4F720300000001000000000000...	2013-02-15 10:54:03.347	2013-02-15 15:44:03.473	30	59616	1149	761	3702

En çok CPU kaynağı harcayan ilk 20 Stored Procedure'ü listeleyelim.

SQL Server'da çalıştırılan Stored Procedure'leri takip etmek, performans analizleri ve prosedürlerin performanslarını artırmak için gereklidir. Performans ve güvenlik nedeniyle kullanılan prosedürlerin performansını takip etmek için **`sys.dm_exec_procedure_stats`** isimli DMV kullanılır.

```
SELECT TOP 20
   DB_NAME(database_id) AS DB_ismi,
   OBJECT_NAME(object_id) AS SP_ismi,
   st.[text] AS SP_kod,
   qp.query_plan,
   cached_time AS ilk_calisma_zamani,
   last_execution_time,
   execution_count,
   ps.total_logical_reads AS toplam_mantiksal_okuma,
   ps.total_logical_reads / execution_count AS avg_mantiksal_okuma,
   ps.total_worker_time / 1000 AS toplam_cpu_zamani_ms,
   ps.total_worker_time / ps.execution_count/1000 AS avg_cpu_zamani_ms
FROM sys.dm_exec_procedure_stats ps
CROSS APPLY sys.dm_exec_sql_text(ps.plan_handle) st
CROSS APPLY sys.dm_exec_query_plan(ps.plan_handle) qp
WHERE DB_NAME(database_id)='AdventureWorks'
ORDER BY ps.total_worker_time DESC;
```

AdventureWorks veritabanını kullanarak bu sorguyu çalıştırdığım için sorgu sonucu boş geldi. Bunun nedeni;bilgisayarı başlattığımdan beri AdventureWorks veritabanında hiç bir prosedür çalıştırmamış olmamdır. DMV'ler servis başlatıldıktan sonra bu bilgileri tutuyordu. Farklı birkaç **`AdventureWorks`** veritabanı prosedürü çalıştırılarak prosedür istatistiklerine ulaşılabilir.

İki prosedür çalıştırarak istatistiklere ulaşalım.

	DB_ismi	SP_ismi	SP_kod	query_plan	ilk_calisma_zamani	last_execution_time	execution_count
1	AdventureWorks2012	pr_UrunAra	CREATE PROCEDURE pr_UrunAra @ProductName VARCHAR...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2013-02-15 15:53:15.933	2013-02-15 15:54:08.023	9
2	AdventureWorks2012	pr_ProcedureCall	CREATE PROC pr_ProcedureCall(@sp_ad VARCHAR(2000) ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2013-02-15 15:54:04.560	2013-02-15 15:54:05.490	3

toplam_mantiksal_okuma	avg_mantiksal_okuma	toplam_cpu_zamani_ms	avg_cpu_zamani_ms
624	69	512	56
48	16	172	57

Kullanılmayan Stored Procedure'lerin Tespit Edilmesi

Veritabanın için geliştirilen nesnelerin tamamı sürekli kullanılmayabilir. Bir geliştirme sırasında test için oluşturulan stored procedure ya da kendisini kullanan uygulamanın arayüzü iptal edilen ve artık gerek duyulmayan bir stored procedure olabilir. Bunlar zaman içerisinde gerçekleşebilecek olası düzenleme ihtiyaçlarını doğurur.

Kullanılmayan bir Stored Procedure'ü bulmak için de **`sys.dm_exec_procedure_stats`** isimli DMV kullanılabilir. Ancak, bu konuda dikkat edilmesi gereken bir husus var. DMV'ler disk değil, bellek üzerinde kümülatif olarak veri tutuyorlardı. Yani, bir DMV'nin tuttuğu veri veritabanı oluşturulmasından şu ana kadar değil, servis başlatıldığından şimdiki zamana kadar olan istatistiklerdir. Bu durumda bir prosedürün gerçekten kullanılıp kullanılmadığını anlayabilmek için SQL Server servisinin uzun zamandır çalışıyor olduğuna emin olunmalıdır. Ayrıca, her prosedür günlük ya da haftalık kullanılmaz. Özel görevleri olan prosedürler aylık ya da 6 aylık periyotlar ile kullanılıyor da olabilir.

`AdventureWorks` veritabanında servis başlatıldıktan sonra kullanılmayan prosedürleri listeleyelim.

```
SELECT SCHEMA_NAME(O.schema_id) AS sema_ismi,
       P.name AS nesne_ismi,
       P.create_date,
       P.modify_date
FROM sys.procedures P
LEFT JOIN sys.objects O ON P.object_id = O.object_id
WHERE P.type = 'P'
AND NOT EXISTS(SELECT PS.object_id
  FROM sys.dm_exec_procedure_stats PS
  WHERE PS.object_id = P.object_id
  AND PS.database_id=DB_ID('AdventureWorks'))
ORDER BY SCHEMA_NAME(O.schema_id), P.name;
```

	sema_ismi	nesne_ismi	create_date	modify_date
1	dbo	AdayEkle	2013-02-08 02:51:23.963	2013-02-08 02:51:23.963
2	dbo	Gsp_Create_GenericCursor	2013-02-02 23:47:27.533	2013-02-02 23:47:27.533
3	dbo	KitapEkle	2013-02-08 04:28:24.383	2013-02-08 04:28:24.383
4	dbo	pr_CreateDB	2013-02-03 02:00:34.637	2013-02-03 02:00:34.637
5	dbo	pr_HataBilgisiGetir	2013-02-05 13:07:51.123	2013-02-05 13:07:51.123
6	dbo	pr_HataGoster	2013-02-05 13:35:04.720	2013-02-05 13:35:04.720
7	dbo	pr_InsertLocation	2013-02-04 22:06:19.727	2013-02-04 22:34:29.737

Bu sorgu sonucunda gelen liste üzerinde, veritabanı programcısı ya da SQL Server DBA'i inceleme yaparak, hangi prosedürlerin kullanım ömrünü doldurduğunu bilebilir. Sonuç olarak, geliştirilen prosedürler ihtiyaçlar doğrultusunda DBA ya da veritabanı programcısı tarafından oluşturulmaktadır.

Trigger İstatistikleri

Trigger bölümünde ve trigger ile ilgili performans konularını incelediğimiz bu bölümde, trigger'ın tetiklendiği transaction'ın bir parçası olduğundan bahsettik. Tetikleyici ve tetiklenen trigger arasında bir performans ortaklığı vardır. Bir Stored Procedure içerisindeki transaction'ın tamamlanması için, tetiklediği trigger'ın ta **ROLLBACK** olmadan tamamlanması gerekir. Bu durumda, birinin performansı diğerini etkileyecektir. Bu performans ortaklığının önemini kavradıktan sonra, trigger ile ilgili istatistikleri inceleyebiliriz.

Trigger istatistiklerini tutan DMV'nin ismi **sys.dm_exec_trigger_stats**'dır.

```
select * from sys.dm_exec_trigger_stats;
```

En çok CPU tüketen ilk 10 trigger'ı listeleyelim.

```
SELECT TOP 10
    DB_NAME(database_id) AS DB_ismi,
    SCHEMA_NAME(O.schema_id) AS tablo_sema_ismi,
    O.name AS tablo_nesne_ismi,
    T.name AS trigger_nesne_ismi,
         ST.[text] AS trigger_kod,
    QP.query_plan,
    cached_time AS ilk_calisma_zamani,
    last_execution_time,
    execution_count,
       PS.total_logical_reads AS toplam_mantiksal_okuma,
```

```
        PS.total_logical_reads / execution_count AS avg_mantiksal_okuma,
        PS.total_worker_time / 1000 AS toplam_cpu_zaman_ms,
        PS.total_worker_time / PS.execution_count / 1000 AS avg_cpu_zaman_ms
FROM sys.dm_exec_trigger_stats PS
LEFT JOIN sys.triggers T ON T.object_id = PS.object_id
LEFT JOIN sys.objects O ON O.object_id = T.parent_id
CROSS APPLY sys.dm_exec_sql_text(PS.plan_handle) ST
CROSS APPLY sys.dm_exec_query_plan(PS.plan_handle) QP
WHERE DB_NAME(database_id) = 'AdventureWorks'
ORDER BY PS.total_worker_time DESC;
```

Servisin son başladığı andan itibaren veri tutulmaya başlandığı için bu sorgu sonucunda hiç bir kayıt dönmemesi bizi şaşırtmamalı.

Örnek bir kaç trigger çalıştırarak yukarıdaki sorguyu tekrar inceleyelim.

`Production.WorkOrder` tablosu üzerinde bir güncelleştirme yapalım.

```
UPDATE Production.WorkOrder
SET StartDate = GETDATE(), EndDate = GETDATE(), DueDate = GETDATE()
WHERE WorkOrderID = 1
```

`Sales.SalesOrderHeader` tablosu üzerinde bir güncelleştirme yapalım.

```
UPDATE Sales.SalesOrderHeader
SET OrderDate = GETDATE(), DueDate = GETDATE(), ShipDate = GETDATE()
WHERE SalesOrderID = 75123;
```

Bu sorgular sonucunda **`UPDATE`** komutunu takip eden trigger'lar tetiklenecek ve bizim takip ettiğimiz istatistikler tarafından görünür hale gelecektir.

Yukarıdaki trigger istatistik sorgusu tekrar çalıştırıldığında istenen istatistikler elde edilecektir.

	DB_ismi	tablo_sema_ismi	tablo_nesne_ismi	trigger_nesne_ismi	trigger_kod	query_plan	ilk_calisma_zamani
1	AdventureWorks2012	Sales	SalesOrderHeader	uSalesOrderHeader	CREATE TRIGGER [Sales].[uSalesOrderHeader] ON ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2013-02-15 15:58:09.673
2	AdventureWorks2012	Production	WorkOrder	uWorkOrder	CREATE TRIGGER [Production].[uWorkOrder] ON [P...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2013-02-15 15:58:02.500

last_execution_time	execution_count	toplam_mantiksal_okuma	avg_mantiksal_okuma	toplam_cpu_zaman_ms	avg_cpu_zaman_ms
2013-02-15 15:58:10.400	2	10	5	0	0
2013-02-15 15:58:04.750	5	0	0	0	0

Kullanılmayan Trigger'ları Tespit Etmek

Kullanılmayan Stored Procedure'leri incelerken bahsettiğimiz durumların tamamı trigger'lar için de geçerlidir. Veritabanı geliştirildikten sonra, sürekli değişen ihtiyaçlar doğrultusunda, bazen daha önceden geliştirilmiş trigger'lar artık kullanılmamaya başlanabilir. Bu tür kullanılmayan trigger'lar takip edilmeli ve gerekiyorsa DROP edilmelidir.

```
SELECT SCHEMA_NAME(o.schema_id) AS tablo_sema_ismi,
       o.name AS tablo_nesne_ismi,
       t.name AS trigger_nesne_ismi,
       t.create_date,
       t.modify_date
FROM sys.triggers t
LEFT JOIN sys.objects o ON t.parent_id = o.object_id
WHERE t.is_disabled = 0 AND t.parent_id > 0
AND NOT EXISTS(SELECT ps.object_id
  FROM sys.dm_exec_procedure_stats ps
  WHERE ps.object_id = t.object_id
  AND ps.database_id = DB_ID('AdventureWorks'))
  ORDER BY SCHEMA_NAME(o.schema_id),o.name,t.name;
```

	tablo_sema_ismi	tablo_nesne_ismi	trigger_nesne_ismi	create_date	modify_date
1	dbo	Calisanlar	trgAfterInsert	2013-02-11 12:18:36.620	2013-02-11 12:18:36.620
2	dbo	Kullanicilar	trg_KullaniciSil	2013-02-11 22:34:56.673	2013-02-11 22:34:56.673
3	dbo	Makaleler	trg_MakaleSil	2013-02-11 21:14:32.040	2013-02-11 21:14:32.040
4	dbo	Personeller	trg_PersonelHatirlatici	2013-02-13 01:58:44.880	2013-02-13 01:58:44.880
5	dbo	vw_MusteriSiparisleri	trg_MusteriSiparisEkle	2013-02-13 10:36:56.003	2013-02-13 10:36:56.003
6	dbo	vw_MusteriSiparisleri	trg_MusteriSiparisSil	2013-02-13 10:42:51.687	2013-02-13 10:42:51.687
7	HumanResources	Employee	dEmployee	2012-03-14 13:14:55.383	2012-03-14 13:14:55.383

Sorgu sonucunda dönen kayıtlar, **AdvemtureWorks** veritabanında kullanılmayan trigger'lardır. Tekrar hatırlatmak gerekiyor ki, bu sorgu ile elde edilen kayıtlar SQL Server servisinin başladığı andan itibaren kullanılmayan trigger'lardır. SQL Server 1 gün önce mi başlatıldı, yoksa 1 yıl mı, bunu ancak SQL Server DBA bilebilir.

Açık Olan Cursor'leri Sorgulama

Bu kitabın cursor'ler ile ilgili bölümünde performans ile ilgili önemli bir konuya değinmiştik. Cursor'ler kullanılmak için açılmalı ve işi bitince kapatılmalıdır. Ancak kapatılmadığı taktirde sistem kaynaklarını tüketmeye devam edecektir. Sistemdeki açık cursor'leri takip edebilmek, bu cursor'leri kimin, ne zaman açtığını ve içerdiği sorguların neler olduğunu öğrenebilmek için **`sys.dm_exec_cursors`** DMF'si kullanılır.

```
SELECT * FROM sys.dm_exec_cursors(0);
```

Bu DMF, parametre olarak **`Session ID`** alır. Varsayılan kullanım **`0`** parametresidir. Sıfır parametresi ile kullanıldığında, sorgunun çalıştırıldığı veritabanı üzerindeki açık olan tüm cursor'leri listeler. Parametre olarak bir **`Session ID`** verilirse, bu **`Session ID`** için açık olan cursor'leri listeler.

Sorgu sonucunda dönen sütunları inceleyelim.

- **`session_id`**: Cursor'ın hangi session üzerinde açıldığını belirtir.
- **`cursor_id`**: Cursor'ın ID bilgisi.
- **`name`**: Cursor'ın ismi.
- **`properties`**: Cursor'ın hangi özellikler ile açıldığını belirtir.
- **`sql_handle`**: Cursor'ın **`sql_handle`**'ıdır.
- **`creation_time`**: Cursor'ın oluşturulma zamanını belirtir.
- **`fetch_status`**: Cursor'ın en son **`FETCH`** durumunu belirtir.
- **`dormant_duration`**: En son **`OPEN`** ya da **`FETH`** işleminden sonra geçen zamanı (*milisaniye*) belirtir.

Şuan sisteminizde açık cursor olmadığını varsayarak yeni bir cursor tanımlamalıyız. Şimdi, örnek olarak, belirli bir süre açık kalacak cursor tanımlayalım.

```
DECLARE @ProductID INT;
DECLARE @Name  VARCHAR(255);
DECLARE ProductCursor CURSOR FOR
   SELECT ProductID, Name FROM Production.Product WHERE ProductID < 5;
OPEN ProductCursor;
```

```
FETCH NEXT FROM ProductCursor INTO @ProductID, @Name;
WHILE @@FETCH_STATUS = 0
BEGIN
 WAITFOR DELAY '00:00:2';
 PRINT CAST(@ProductID AS VARCHAR) + '  -  ' + @Name;
 FETCH NEXT FROM ProductCursor INTO @ProductID, @Name;
END;
CLOSE ProductCursor;
DEALLOCATE ProductCursor;
```

Cursor açıkken aşağıdaki sorguyu çalıştırarak açık olan cursor'leri bulalım.

```
SELECT ec.cursor_id,
       ec.name AS cursor_name,
       ec.creation_time,
       ec.dormant_duration/1000 AS uyku_suresi_sn,
       ec.fetch_status,
       ec.properties,
       ec.session_id,
       es.login_name,
       es.host_name,
       st.text,
       SUBSTRING(st.text, (ec.statement_start_offset/2)+1,
        ((CASE ec.statement_end_offset
          WHEN -1 THEN DATALENGTH(st.text)
          ELSE ec.statement_end_offset
          END - ec.statement_start_offset)/2) + 1) AS ifade_metni
FROM sys.dm_exec_cursors(0) ec
CROSS APPLY sys.dm_exec_sql_text(ec.sql_handle) st
JOIN sys.dm_exec_sessions es ON es.session_id = ec.session_id
```

	cursor_id	cursor_name	creation_time	uyku_suresi_sn	fetch_status	properties	session_id	login_name	host_name	text
1	180150005	ProductCursor	2013-02-15 16:01:24.480	0	0	TSQL | Dynamic | Optimistic | Global (0)	54	dijibil-pc\dijibil	DIJIBIL-PC	DECLARE @ProductID INT; DECLARE @Name VARCHAR(...

Sorgu sonucuna baktığımızda, cursor **ID** ve isim bilgisi, oluşturulma zamanı, **FETCH** durumu, özellikleri, session'ın **ID** bilgisi, login ve bilgisayar adı ve cursor'ın içerdiği SQL sorgu metni gibi bilgiler listelenmektedir.