SQL SERVER'DA YAPILMASI VE YAPILMAMASI GEREKENLER

Özgür BELÜL

SQL SERVER'DA YAPILMASI VE YAPILMAMASI GEREKENLER

- SQL Server tabanlı bir projenin yöneticisiyseniz ve bu sizin ilk projenizse,
- Access'ten SQL'e yeni geçiş yaptıysanız,
- SQL Serverınızla ilgili performans problemleriniz var ve bir sonraki adımda ne yapacağınızı bilemiyorsanız,
- Basit olarak SQL Server ve Database Access Layer (DAL) çözümleri hakkında dizayn fikirleri öğrenmek istiyorsanız bu makale tam size göre.

SQL Server kullanmıyorsanız bile bu makaledeki bilgilerin bir çoğu diğer database yönetim sistemlerine de uygulanabilir.

Bu makale T-SQL bilginiz olduğu ve T-SQL komutlarını MSDN'de nasıl bulacağınızı bildiğiniz farz edilerek yazılmıştır.

Araçlarınızı Tanıyın

Bu ipucunu dikkate almanızı tavsiye ederim çünkü bu makalede okuyacağınız en iyi ipucu. Bir çok SQL Server programcısı T-SQL komutlarının büyük bir kısmını ve SQL Server'ın sahip olduğu etkili araçları bilmezler.

"Hiç kullanmayacağım SQL komutlarını öğrenmek için günlerimi harcamalı mıyım?" diye düşünüyor olabilirsiniz. Cevap tabikide hayır. Ancak en azından bir haftasonununuzu ayırıp MSDN'de T-SQL komutları arasında gezinebilirsiniz. Buradaki amaç neler yapabileceğiniz ve neler yapamayacağınız hakkındaki bilgilerinizi artırmak.
Böylece, bir gün gerektiğinde, "Böyle bir şeyler MSDN'de görmüştüm" diyebilir, MSDN'e bakarak komutu ve tam yazımını bulabilirsiniz.

Cursor kullanmayın

Tekrarlıyorum, Cursor kullanmayın. Cursor kullanmak bütün bir sistem performansını katletmenin en iyi yoludur. Bir çok yeni başlayan programcı cursor kullanır ve performansı ne kadar düşürdüklerini farketmezler. Cursorlar memory kullanırlar, tabloları tuhaf şekillerde kilitlerler ve yavaştırlar. Hepsinden de kötüsü yapabileceğiniz performans optimizasyonunun bir çoğunu yok ederler.

Kullandığınız her bir FETCH komutunun bir SELECT komutu kullanmakla yaklaşık aynı performansa sahip olduğunu biliyor musunuz? Bu da demektir ki cursor'da 10.000 tane kayıt varsa yaklaşık 10.000 tane SELECT sorgusu gerçekleştirecektir. Eğer veritabanı üzerinde yapacaklarınızı bir kaç SELECT, UPDATE ya da DELETE sorgusu ile hallederseniz veritabanınız çok daha hızlı çalışacaktır.

Normalizasyon Yapın

Genelde veritabanlarını normalize etmemenin iki sebebi vardır: Birincisi performans kaygısı, ikincisi ise tembellik.

İkincisinin bedelini er ya da geç ödersiniz. Performans konusuna gelince, yavaş olmayan bir veritabanını zaten optimize etmenize gerek yoktur. Bir çok programcı veritabanlarını yavaşlayacak kaygısıyla de-normalize ederler. Tahminlerinin aksine genellikle kendi veritabanı dizaynları daha yavaştır. Veritabanı yönetim sistemleri normalize edilmiş veritabanları ile kullanılmak üzere dizayn edilmişlerdir, bu sebeple normalizasyon her zaman aklınızın bir köşesinde olsun. Bununla birlikte aşırı normalizasyon yapmaktan da kaçının. Herşeyin fazlası zarar.

SELECT * FROM... Sorgusunu Kullanmamaya Çalışın
Buna alışması zor olabilir ancak sorgularınızda sadece ihtiyacınız olacak kolonları belirtmeye çalışın.

Böylelikle;

Memory tüketiminiz ve bandwith kullanımınız azalacaktır.

Güvenlik önlemlerinizi daha rahat alabileceksiniz.

Query Optimizer'a INDEXlerdeki bütün gerekli kolonları okuyabilmesi için şans vermiş olacaksınız.

Verilerinize nasıl erişildiğini ve erişileceğini bilin.

Sağlam bir INDEX dizaynı veritabanınız için yapabileceğiniz en iyi şeylerden bir tanesidir. Şunu unutmayın ki tablonuza her bir INDEX eklediğinizde işler SELECT sorgusu tarafında hızlanacaktır fakat INSERT ve DELETE sorguları tarafında yavaşlayacaktır.

INDEXleri yaratırken ve düzenlerken yapılacak bir çok iş vardır. SELECT sorgularınızı hızlandırmak için bir tablonuza bir kaç INDEX birden eklerseniz bir süre sonra INDEX'leri güncellerken kilitlenme (LOCK) durumlarının uzun sürdüğünü göreceksiniz.

Burada sormanız gereken tablonuz ile ne yapılacağıdır. Veri okunacak mı? Yoksa yazılacak mı? Bu cevaplaması zor bir sorudur. Özellikle DELETE ve UPDATE söz konusu olduğunda. Çünkü bu sorgular genelde öncelikle bir SELECT.. WHERE.. sorgusu gerçekleştirip ondan sonra tabloyu güncellerler.

Her kolon üzerinde INDEX yaratmayın.

Çünkü bu işe yaramaz. Öncelikle INDEXlerin tablo erişimlerini nasıl hızlandırdıklarını anlamak gerekir. INDEXleri bir kritere bağlı olarak tablolar üzerinde bölümler (partition) yaratmanın hızlı bir yolu olarak ele alabilirsiniz. Mesela müşteri bilgilerinin tutulduğu bir tablonuz var ve bu tabloda da cinsiyet bilgilerinin tutulduğu bir kolonunuz var. Eğer cinsiyet kolonu üzerinde bir INDEX yaratırsanız bu durumda iki bölümünüz olacaktır: Bay ve Bayan. Bu durumda , örneğin 1.000.000 satırı olan bir tabloda nasıl bir optimizasyon yapmayı düşünürsünüz?

INDEXlerinizde her zaman farklı bilgilerin çok olduğu kolonları ilk sırada kullanın ve INDEXlerinizi farklı bilgilerin çok olduğu kolondan farklı bilgilerin az olduğu kolona doğru sıralayarak dizayn edin.

Transaction kullanın

Özellikle çalışma süresi uzun olan sorgularda transaction kullanın. Verilerle çalışırken bir süre sonra stored procedurelerin çökmesine sebep olacak beklenmedik durumlarla karşılacaksınız. İşler ters gittiğinde transaction sizi kurtaracaktır.

Deadlock hakkında bilgi edinin

Tablolarınıza her zaman aynı sırada erişin. Eğer A tablosunu ve daha sonra B tablosunu kilitliyorsanız, bütün stored procedurelerde bunu aynı sırada yapın. Eğer kazara önce B tablosunu kilitler ve başka bir prosedürde ise A tablosunu kilitlerseniz bir gün DEADLOCK olacağından şüpheniz olmasın. Stored procedurler ve transactionlarla çalışırken bununla karşılaşabilirsiniz. Eğer kilitleme sırası dikkatli bir şekilde dizayn edilmemişse DEADLOCKları bulmak zor olabilir.

Büyük kayıtları tek seferde açmayım

Bir combobox'ı 100.000 tane kayıtla doldurmaya çalışmak bir hatadır. Öncelikle kullanıcınız doğru olan kaydı bulmak için 100.000 tane kayıt arasında gezinmekten nefret edecektir. Burada daha iyi bir kullanıcı arabirimi tasarımı gerekecektir çünkü ideal olan en fazla 200 kaydı aynı anda kullanıcıya göstermektir. Mümkün olan durumlarda daha az kaydı kullanıcıya göstermek tercih edilmelidir.

Server taraflı cursor kullanmayın

Mecbur kalmadıkça cursor kullanmayın. Özellikle ne yaptığınızdan emin değilseniz server tarafında kesinlikle cursor kullanmayın. Client tarafında kullanacağınız cursorlar sıklıkla (her zaman değil) network'e ve server'a daha az yük bindirir ve kilitleme zamanlarını kısaltır.

Sorgularınızda Parametre Kullanın

Parametre kullanmamak SQL Injection saldırılarına davetiye çıkarmak gibi ciddi güvenlik açıkları yaratmanın yanısıra bir tane yerine birbirine benzer sorguları saklayacak olan SQL Server'ın cache sistemini çöplüğe çevirecektir. Sorgularınızda parametre kullanmayı öğrenin.

Sorgularınızı her zaman büyük veritabanları ile test edin

Bu genel bir problemdir. Programcılar küçük test veritabanları ile çalışırlar ancak son kullanıcılar büyük veri tabanları kullanırlar. Bu bir programcı hatasıdır. Küçük veri tabanında farkedilmeyen bir problem büyük veritabanlarında sıkıntı yaratabilir ve genellikle performans problemleri farkedildiğinde geç kalınmıştır.

INSERT sorgusu ile BULK DATA import etmeyin

Başka bir şansınız yoksa insert sorguları ile bulk data import etmeyin. Data Transformation Services ya da BCP kullanın. Her iki türlü de daha esnek ve hızlı bir çözüm elde etmiş olursunuz.

Timeoutları dikkatle izleyin

Bir veritabanından sorgu çekerken timeoutları dikkatle izleyin. Timeout süresi genellikle 15-30 saniye gibi kısa bir süredir. Unutmayın ki, özellikle veritabanınız büyüdüğünde raporlama sorguları bu süreden daha uzun sürebilir.

Eş zamanlı güncellemeleri engelleyin.

Bazen iki kullanıcı aynı kaydı aynı anda değiştiriler. Kayıt eklerken son yazan kullanıcı kazanır ve diğer kullanıcının yaptığı güncellemelerin bir kısmı kaybolur. Bu durumu kontrol altına almak kolaydır: Bir zaman kolonu yaratın (timestamp) ve zaman kolonunu veri kaydetmeden önce kontrol edin. Eğer bir çelişki varsa kullanıcıya uyarı mesajı gönderin.

Bir detay tablosuna veri girerken SELECT max(ID) from Master sorgusunu kullanmayın

Bu da sıklıkla yapılan bir hatadır ve aynı anda iki kullanıcı veri giriyorsa bir işe yaramaz. SCOPE_IDENTITY, IDENT_CURRENT ve @@IDENTITY sorgularından birini kullanın. (Bkz. MSDN). Mümkün olan durumlarda @@IDENTITY kullanmamaya çalışın. Trigerlarınızda hataya sebep olabilir.

Nullabe kolonlardan sakının

Nullable kolonlardan sakının. Her bir kolonda ve her bir satırda extra byte tutarlar. Bununla birlikte Data Access Layer'a kod yazmak zorlaşır çünkü o kolona her erişiminizde kontrol ettirmeniz gerekmektedir.

Bazı durumlarda NULLable kolonlar şöyle kullanılmaktadır:

MüşteriAdı1

MüşteriEmail1

MüşteriAdı2

MüşteriEmail2

MüşteriAdı3

MüşteriEmail3

Bu yapılmaması gerekenler için iyi bir örnektir. Bunu yapmayın. Tablonuzu normalize edin. Hem daha esnek hem de daha hızlı olacak ayrıca NULLabla kolonları azaltacaktır.

Kolonlarda TEXT veri tipini (datatype) kullanmamaya çalışın

Gerçekten büyük bir veri için kullanmayacaksanız veri tiplerinde TEXT'i kullanmayın. TEXT veri tipi esnek değildir, yavaştır ve yanlış kullanıldığında gereksiz yer kaplar. Bazen VARCHAR aynı işi daha performanslı yaptırır.

Geçiçi tablolar kullanmayın

Çok gerekmedikçe geçici tablolar kullanmayın. Bunun yerine subquery tercih edin. Genellikle bir subquery geçici bir tablonun yerini tutabilir. Geçici tablolar ek yük bindirirler ve özellikle COM+'da program yazarken başınızı ağrıtırlar çünkü COM+ veritabanı bağlantı havuzu kullanır ve geçici tablolar sonsuza dek kalır. SQL Server 2000'de stored procedure içerisindeki küçük tablolar için memoryde çözüm sağlayan TABLE veri tipi gibi alternatifler de vardır.

Query Execution Plan okumayı öğrenin

SQL Server Query Analyzer'ın nasıl çalıştığı, sorgu ve INDEX dizaynının onun vasıtasıyla performansı nasıl etkilediği hakkında fazlaca bilgi edinin.

Referential Integrity kullanın

Bu büyük bir zaman kazancı olacaktır. Primary Keylerinizi, unique constraintlerinizi ve foreign keylerinizi tanımlayın. Server üzerinde yarattığınız her bir validation ileri de size zaman kazandıracaktır.

Görüşmek üzere .

Özgür Belül
