Bölüm

11

Veritabanı Performansı Ayarlama ve Sorgu Optimizasyonu

Öğrenme Hedefleri

Bu bölümü tamamladıktan sonra şunları yapabileceksiniz:

11-1 Veritabanı performans ayarlamasına dahil olan prosedürleri tanımlama

11-2 Bir DBMS'nin SQL sorgularını üç aşamasının her birinde nasıl işlediğini açıklayın

11-3 Dizinlerin veri erişimini hızlandırmadaki rolünü açıklama

11-4 Kural tabanlı bir optimize edici ile maliyet tabanlı bir optimize edici arasındaki farkı ayırt etme

11-5 Verimli SQL kodu yazmak için kullanılan bazı yaygın uygulamaları tanımlama

11-6 Optimum performans için sorguların nasıl formüle edileceğini ve VTYS'nin nasıl ayarlanacağını açıklamak

ÖN İZLEME

Veritabanı performans ayarlaması kritik bir konudur, ancak genellikle veritabanı müfredatında çok az yer alır. Sınıflarda kullanılan çoğu veritabanında tablo başına yalnızca birkaç kayıt bulunur. Sonuç olarak, sorgu sürecinin verimliliği göz önünde bulundurulmadan SQL sorgularının amaçlanan görevi yerine getirmesine odaklanılmaktadır. Aslında, en verimli sorgu ortamı bile, yalnızca 20 veya 30 tablo satırı (kayıt) olduğunda, en az verimli sorgu ortamına göre gözle görülür bir performans artışı sağlamaz.
sorgulanmaktadır. Ne yazık ki, sorgu verimliliğine dikkat edilmemesi, sorgular on milyonlarca kayıt üzerinde yürütüldüğünde gerçek dünyada kabul edilemeyecek kadar yavaş sonuçlar verebilir. Bu bölümde, daha verimli bir sorgu ortamı oluşturmak için neler gerektiğini öğreneceksiniz.

Veri Dosyaları ve Mevcut Formatlar

	MS Erişim	Oracle	MS SQL	MySQL
Ch11_SaleCo	Evet	Evet	Evet	Evet

Veri Dosyaları cengage.com adresinde mevcuttur

Not

Bu kitap *veritabanlarına* odaklandığından, bu bölüm yalnızca *veritabanı* performansını doğrudan etkileyen faktörleri kapsamaktadır. Ayrıca, performans ayarlama teknikleri DBMS'ye özgü olabileceğinden, bu bölümdeki materyal her koşulda uygulanamayabilir ve tüm DBMS türleri için geçerli olmayabilir. Bu bölüm, veritabanı performans ayarlama konularının genel olarak anlaşılması için bir temel oluşturmak ve uygun performans ayarlama stratejilerini seçmenize yardımcı olmak üzere tasarlanmıştır. (Veritabanınızın ayarlanmasıyla ilgili en güncel bilgiler için veritabanı sağlayıcısının belgelerine başvurun).

11-1 Veritabanı Performans Ayarlama Kavramları

Bir veritabanı sisteminin ana işlevlerinden biri, son kullanıcılara zamanında yanıtlar sağlamaktır. Son kullanıcılar, aşağıdaki sırayı kullanarak bilgi üretmek için sorgular kullanarak DBMS ile etkileşime girer:

1. Son kullanıcı (istemci-uç) uygulaması bir sorgu oluşturur.
2. Sorgu DBMS'ye (sunucu ucu) gönderilir.
3. DBMS (sunucu ucu) sorguyu yürütür.
4. VTYS, elde edilen veri setini son kullanıcı (istemci-uç) uygulamasına gönderir.

Son kullanıcılar sorgularının mümkün olduğunca çabuk sonuç vermesini bekler. Bir veritabanının performansının iyi olduğunu nasıl ? İyi veritabanı performansını değerlendirmek zordur. Sorgu yanıt süresinin 1,06 saniye olmasının yeterince iyi olup olmadığını nasıl anlarsınız? Kötü veritabanı performansını tespit etmek iyi veritabanı performansını tespit etmekten daha kolaydır; bunun için son kullanıcının yavaş sorgu sonuçlarıyla ilgili şikayette bulunması yeterlidir. Ne yazık ki, aynı sorgu bir gün iyi performans gösterirken iki ay sonra o kadar iyi performans göstermeyebilir. Son kullanıcı algıları ne olursa olsun, *veritabanı performansının amacı sorguları mümkün olduğunca hızlı yürütmektir*. Bu nedenle, veritabanı performansı yakından izlenmeli ve düzenli olarak . **Veritabanı performans ayarlaması**, veritabanı sisteminin yanıt süresini azaltmak, yani son kullanıcı sorgusunun VTYS tarafından minimum sürede işlenmesini sağlamak için tasarlanmış bir dizi faaliyet ve prosedürü ifade eder.

veritabanı performans ayarı
Bir veritabanı sisteminin yanıt süresini azaltmak, yani son kullanıcı sorgusunun DBMS tarafından minimum işlenmesini sağlamak için tasarlanmış faaliyetler ve prosedürler .

Bir sorgunun bir sonuç kümesi döndürmesi için gereken süre, geniş kapsamlı olma ve ortamlar ve satıcılar arasında değişiklik gösterme eğiliminde olan birçok faktöre bağlıdır. Genel olarak, tipik bir DBMS'nin performansı üç ana faktör tarafından kısıtlanır: CPU işlem gücü, mevcut birincil bellek (RAM) ve giriş/çıkış (sabit disk ve ağ) verimi. Tablo 11.1 bazı sistem bileşenlerini listeler ve daha iyi sorgu performansı elde etmek için genel yönergeleri özetler.

Doğal olarak sistem, donanım ve yazılım kaynakları en iyi şekilde optimize edildiğinde en iyi performansı gösterecektir. Ancak gerçek dünyada sınırsız kaynak norm değildir; iç ve dış kısıtlamalar her zaman mevcuttur. Bu nedenle, sistem bileşenleri mevcut (ve genellikle sınırlı) kaynaklarla mümkün olan en iyi verimi elde etmek için optimize edilmelidir, bu nedenle veritabanı performans ayarlaması önemlidir.

Bir sistemin performansına ince ayar yapmak bütünsel bir yaklaşım gerektirir. , her birinin optimum düzeyde çalıştığından ve darboğazların oluşmasını en aza indirmek için yeterli kaynağa sahip olduğundan emin olmak için *tüm* faktörler kontrol edilmelidir. Veritabanı tasarımı, veritabanı sisteminin performans verimliliğini belirlemede çok önemli bir faktör olduğundan, bu kitabın mantrasını tekrarlamakta fayda vardır:

İyi veritabanı performansı, **iyi veritabanı tasarımıyla başlar.** *Hiçbir ince ayar, kötü tasarlanmış bir veritabanının iyi tasarlanmış bir veritabanı kadar iyi performans göstermesini sağlayamaz*. Bu, özellikle son kullanıcının eski veritabanlarından gerçekçi olmayan performans artışları beklediği mevcut veritabanlarını yeniden tasarlarken geçerlidir.

Tablo 11.1 Daha İyi Sistem Performansı için Genel Yönergeler

	Sistem Kaynakları	Müşteri	Sunucu
Donanım	CPU	Mümkün olan en hızlı dört çekirdekli CPU veya üstü Sanallaştırılmış istemci masaüstü teknolojileri de kullanılabilir.	Mümkün olan en hızlı Çoklu işlemciler (dört çekirdekli teknoloji veya üstü) Ağa bağlı bilgisayar kümesi Sanallaştırılmış sunucu teknolojisi kullanılabilir.
	RAM	İşletim sistemi belleğinin diske takas edilmesini önlemek için mümkün olan maksimum değer	İşletim sistemi belleğinin diske takas edilmesini önlemek için mümkün olan maksimum değer
	Depolama	Yeterli boş sabit disk alanına sahip hızlı SATA sabit disk Daha yüksek hız için katı hal sürücüleri (SSD'ler)	Çoklu yüksek hızlı, yüksek kapasiteli diskler Hızlı disk arabirimi (SAS/SCSI/Firewire/Fiber Kanal RAID yapılandırması verim için optimize edilmiştir Daha yüksek hız için katı hal sürücüleri (SSD'ler) İşletim sistemi, DBMS ve veri alanları için ayrı diskler
	Şebeke	Yüksek hızlı bağlantı	Yüksek hızlı bağlantı
Yazılım	İşletim sistemi (OS)	Daha geniş adres alanları için 64 bit işletim sistemi En iyi istemci uygulama performansı için ince ayar	Daha geniş adres alanları için 64 bit işletim sistemi En iyi sunucu uygulama performansı için ince ayar
	Şebeke	En iyi verim için ince ayarlı	En iyi verim için ince ayarlı
	Uygulama	İstemci uygulamasında SQL'i optimize etme	DBMS sunucusunu en iyi performans için optimize edin

İyi ve verimli bir veritabanı tasarımını ne oluşturur? Performans ayarı açısından bakıldığında, veritabanı tasarımcısı, tasarımın veritabanının bütünlüğünü ve optimum performansını garanti eden VTYS'deki özellikleri kullandığından emin olmalıdır. Bu bölüm, uygun veritabanı sunucusu yapılandırmasını seçerek, dizinleri kullanarak, tablo depolama organizasyonunu ve veri konumlarını anlayarak ve en verimli SQL sorgu sözdizimini uygulayarak veritabanı performansını optimize etmenize yardımcı olacak temel bilgileri sağlar.

11-1 a Performans Ayarlama: İstemci ve Sunucu

Genel olarak, veritabanı performans ayarlama faaliyetleri istemci tarafındakiler ve sunucu tarafındakiler olarak ikiye ayrılabilir.

- İstemci tarafında amaç, sunucu ucunda minimum miktarda kaynak kullanarak en kısa sürede doğru yanıtı döndüren bir SQL sorgusu oluşturmaktır. Bu hedefe ulaşmak için gereken faaliyetler genellikle **SQL performans ayarlaması** olarak adlandırılır.
- Sunucu tarafında, DBMS ortamı, mevcut kaynakları en iyi şekilde kullanırken, istemcilerin isteklerine mümkün olan en hızlı şekilde yanıt vermek için uygun şekilde yapılandırılmalıdır. Bu hedefe ulaşmak için gereken faaliyetler genellikle **VTYS performans ayarlaması** olarak adlandırılır.

SQL performans ayarı
Sunucu ucunda minimum miktarda kaynak kullanarak en kısa sürede doğru yanıtı döndüren bir SQL sorgusu oluşturmaya yardımcı olacak faaliyetler.

DBMS performans ayarlama
Mevcut kaynakları en iyi şekilde kullanarak müşterilerin taleplerinin mümkün olan en kısa sürede ele alınmasını sağlamaya yönelik faaliyetler.

Çevrimiçi İçerik
İstemciler ve sunucular hakkında daha fazla bilgi edinmek istiyorsanız, www.cengage.com adresindeki Ek F, İstemci/Sunucu Sistemleri bölümüne bakın.

DBMS uygulamalarının tipik olarak iki katmanlı bir istemci/sunucu yapılandırmasından daha karmaşık olduğunu unutmayın. Ağ bileşeni, mesajların istemciler ve sunucular arasında iletilmesinde kritik bir rol oynar; bu özellikle dağıtık veritabanlarında önemlidir. Ancak bu bölümde tamamen optimize edilmiş bir ağ varsayılmaktadır ve bu nedenle veritabanı bileşenlerine odaklanılmaktadır. Bir istemci ön ucu, uygulama ara yazılımı ve veritabanı sunucusu arka ucundan oluşan çok katmanlı istemci/sunucu ortamlarında bile performans ayarlama faaliyetleri, herhangi iki bileşen noktası arasında mümkün olan en hızlı yanıt süresini sağlamak için sıklıkla alt görevlere bölünür. Veritabanı yöneticisi, veritabanı trafiğinin ağ altyapısında verimli bir şekilde akmasını sağlamak ağ grubuyla yakın işbirliği içinde çalışmalıdır. Çoğu veritabanı sisteminin coğrafi olarak dağınık kullanıcılara hizmet verdiği düşünüldüğünde bu daha da önemlidir.

Bu bölüm, istemci tarafındaki SQL performans ayarlama uygulamalarını ve sunucu tarafındaki VTYS performans ayarlama uygulamalarını kapsamaktadır. Ancak, ayarlama süreçlerini öğrenmeye başlamadan önce, VTYS mimari bileşenleri ve süreçleri ve bu süreçlerin son kullanıcıların isteklerine yanıt vermek için nasıl etkileşimde bulunduğu hakkında daha fazla bilgi edinmeniz gerekir.

11-1 b VTYS Mimarisi

Bir DBMS'nin mimarisi, bir veritabanını yönetmek için kullanılan süreçler ve yapılar (bellekte ve kalıcı depolamada) ile temsil edilir. Bu süreçler belirli işlevleri yerine getirmek için birbirleriyle işbirliği yapar. Şekil 11.1'de temel VTYS mimarisi gösterilmektedir.

Şekil 11.1'de aşağıdaki bileşenlere ve işlevlere dikkat edin:

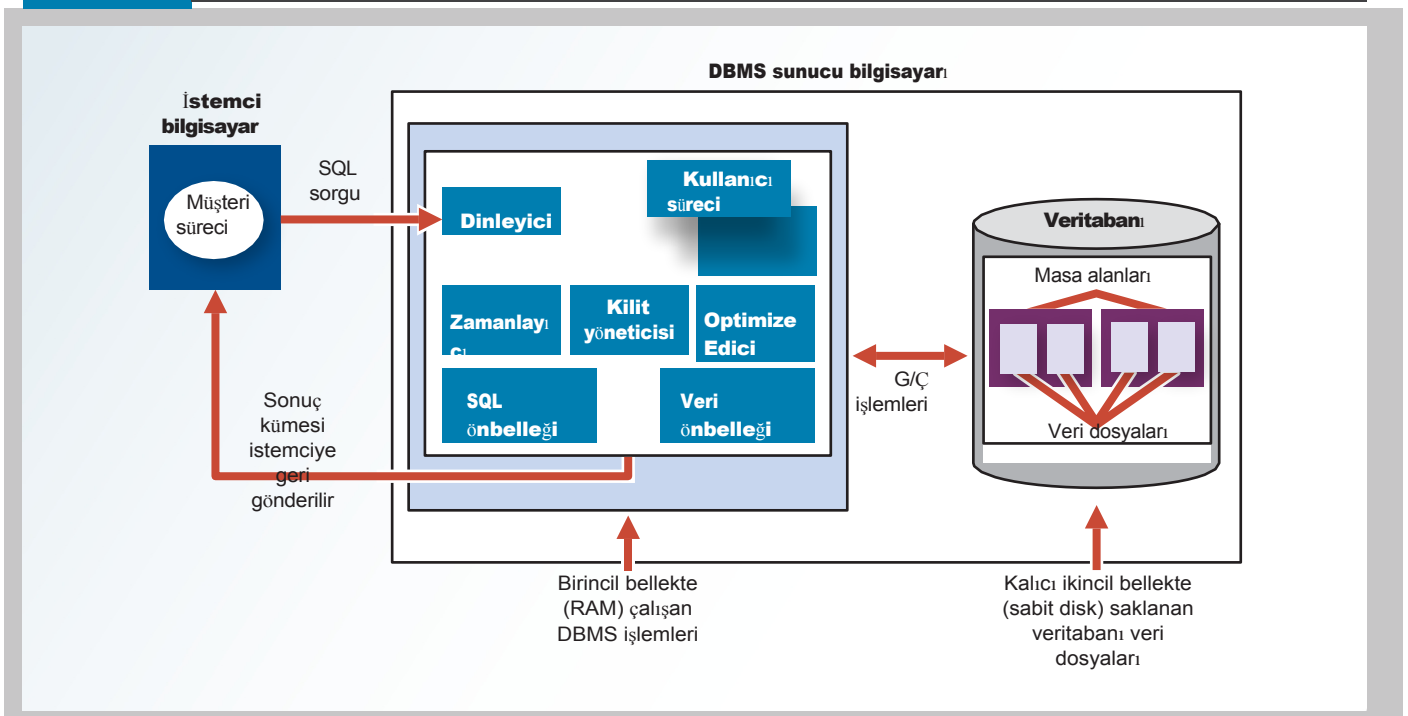
- Bir veritabanındaki tüm veriler **veri dosyalarında** saklanır. Tipik bir kurumsal veritabanı normalde birkaç veri dosyasından oluşur. Bir veri dosyası tek bir tablodaki satırları içerebileceği gibi birçok farklı tablodaki satırları da içerebilir. Bir veritabanı yöneticisi (DBA) veritabanını oluşturan veri dosyalarının başlangıç boyutunu belirler; ancak veri dosyaları gerektiğinde DBMS tarafından otomatik olarak genişletilebilir.
- Bazen "sayfa" olarak da adlandırılan **veri blokları**, veritabanındaki atanabilir depolama alanının en küçük mantıksal birimleridir. Bir veri dosyası içindeki fiziksel depolama alanı mantıksal olarak bir dizi veri bloğuna bölünmüştür. Örneğin, 10 MB boyutunda fiziksel bir veri dosyası oluşturulursa, bu 10 MB mantıksal olarak her biri 8 K boyutunda 1.280 veri bloğuna veya her biri 16 K boyutunda 640 veri bloğuna yapılandırılabilir. Veritabanı bileşenlerinin yapılandırılmasının bir parçası da kullanılacak veri bloklarının mantıksal boyutunu ayarlamaktır. Yaygın veri bloğu boyutları genellikle 2 K blok ile 32 K blok arasında değişir, ancak sizin özel DBMS'nizde başka boyutlar da mevcut olabilir.
- **Kapsamlar**, veritabanı nesnelerine tahsis edilen bitişik veri blokları (sayfalar) koleksiyonlarıdır. Bir kapsamdaki tüm veri blokları aynı veri dosyasında olmalıdır. Bir veritabanı nesnesi oluşturulduğunda, bu nesnenin verilerini tutmak için bir veya daha fazla uzantı tahsis edilir. Örneğin, MySQL'de bir tablo oluşturulduğunda, varsayılan olarak 16 K sayfa kullanılır ve tabloya başlangıçta 64 sayfadan oluşan bir kapsam tahsis edilir. Bu nedenle, ilk veri satırı depolanmadan önce tabloya kullanması için 1 MB depolama alanı verilir. Tablo 1 MB'den fazla depolama alanına ihtiyaç duyulacak şekilde büyürse, DBMS 64 sayfalık bir kapsam daha tahsis eder. Tablo çok büyükse veya çok hızlı büyüyorsa, DBMS tablo büyüdükçe performansı artırmaya yardımcı olmak için bir seferde iki veya daha fazla kapsam tahsis edebilir. Birçok DBMS, DBA'nın veritabanı nesneleri büyüdükçe tahsis edilen uzantıların boyutunu yapılandırmasına izin verir.

veri dosyası
Bir veritabanının verilerini depolayan adlandırılmış bir fiziksel depolama alanı. Bir veya daha fazla depolama konumunda farklı bir dizinde bulunabilir. Bir veritabanındaki tüm veriler veri dosyalarında saklanır. Tipik bir kurumsal veritabanı normalde birkaç veri dosyasından oluşur. Bir veri dosyası, bir veya daha fazla tablodan satırlar içerebilir.

veri bloğu
Veritabanındaki atanabilir depolama alanının en küçük mantıksal birimi. Sayfa olarak da bilinir.

kapsamlar
Bir DBMS ortamında, oluşturulduğunda veya genişletildiğinde bir veritabanı nesnesine atanan bitişik veri blokları veya sayfaları koleksiyonunu ifade eder.

Şekil 11.1 Temel VTYS Mimarisi



masa alanı
Bir DBMS'de, ilgili verileri gruplamak için kullanılan bir veya daha fazla veri dosyasından oluşan mantıksal bir depolama alanı. *Dosya grubu* olarak da bilinir.

dosya grubu
Masa alanına bakın.

veri önbelleği
RAM'de en son erişilen veri bloklarını saklayan paylaşılan, ayrılmış bir bellek alanı. *Tampon önbellek* olarak da adlandırılır.

tampon önbellek
Bkz. *veri önbelleği*.

SQL önbelleği
Tetikleyiciler ve işlevler dahil olmak üzere en son yürütülen SQL deyimlerini veya PL/SQL yordamlarını depolayan paylaşılan, ayrılmış bir bellek alanı. *Prosedür önbelleği* olarak da adlandırılır.

prosedür önbelleği
SQL önbelleğine bakın.

giriş/çıkış (I/O) talebi
Bilgisayar aygıtlarına veya aygıtlarından veri okuyan veya yazan düşük seviyeli bir veri erişim işlemi.

- Veri dosyaları genellikle dosya gruplarında veya tablo alanlarında gruplandırılır. Bir **tablo alanı** veya **dosya grubu**, benzer özelliklere sahip verileri depolayan birkaç veri dosyasının mantıksal bir gruplamasıdır. Örneğin, veri sözlüğü tablosu verilerinin depolandığı bir *sistem* tablosu alanınız, *kullanıcı* tarafından oluşturulan tabloları depolamak için bir *kullanıcı veri* tablosu alanınız, tüm dizinleri tutmak için bir dizin tablosu alanınız ve geçici sıralama, gruplama vb. işlemleri yapmak için bir geçici tablo alanınız olabilir. Her yeni veritabanı oluşturduğunuzda, DBMS otomatik olarak minimum bir tablo alanı kümesi oluşturur.
- **Veri önbelleği** veya **tampon** önbellek, RAM'de en son erişilen veri bloklarını saklayan paylaşılan, ayrılmış bir bellek alanıdır. Veri dosyalarından okunan veriler, veriler okunduktan sonra veya veri dosyalarına yazılmadan önce veri önbelleğinde saklanır. Veri önbelleği ayrıca sistem katalog verilerini ve dizinlerin içeriğini de önbelleğe alır.
- **SQL önbelleği** veya **yordam önbelleği**, tetikleyiciler ve işlevler de dahil olmak üzere en son çalıştırılan SQL deyimlerini veya PL/SQL depolayan paylaşılan, ayrılmış bir bellek alanıdır. (PL/SQL prosedürleri, tetikleyiciler ve SQL fonksiyonları hakkında daha fazla bilgi edinmek için Bölüm 8, Gelişmiş SQL'i inceleyin). SQL önbelleği son kullanıcı tarafından yazılan SQL'i depolamaz. Bunun yerine, SQL önbelleği SQL'in DBMS tarafından yürütülmeye hazır "işlenmiş" bir versiyonunu saklar.
- Verilerle çalışmak için DBMS'nin verileri kalıcı depolama alanından alması ve RAM'e yerleştirmesi gerekir. Başka bir deyişle, veriler veri dosyalarından alınır ve veri önbelleğine yerleştirilir.
- Verileri kalıcı depolama alanından (veri dosyaları) RAM'e (veri önbelleği) taşımak için DBMS G/Ç istekleri yayınlar ve yanıtları bekler. **Giriş/çıkış (G/Ç) isteği**, bellek, sabit diskler, video ve yazıcılar gibi bilgisayar aygıtlarına veri okuyan veya bu aygıtlardan veri yazan düşük seviyeli bir veri erişim işlemidir. Bir G/Ç disk okuma işleminin, yalnızca bir satırdan yalnızca bir öznitelik kullanacak olsanız bile, genellikle birden fazla satır içeren tüm bir fiziksel disk bloğunu kalıcı depolama alanından veri önbelleğine aldığını unutmayın. Fiziksel disk bloğu boyutu işletim sistemine bağlıdır ve 4 K, 8 K, 16 K, 32 K, 64 K ya da daha büyük olabilir. Ayrıca, koşullara bağlı olarak, bir DBMS tek bloklu bir okuma isteği veya çok bloklu bir okuma isteği yayınlayabilir.
- Veri önbelleğindeki verilerle çalışmak, veri dosyalarındaki verilerle çalışmaktan çok daha hızlıdır çünkü DBMS'nin verileri almak için sabit diski beklemesi gerekmez; veri önbelleğinde çalışmak için sabit disk I/O işlemlerine gerek yoktur.
- Performans ayarlama faaliyetlerinin çoğu G/Ç işlemlerinin sayısını en aza indirmeye odaklanır çünkü G/Ç işlemlerini kullanmak veri önbelleğinden veri okumaktan çok daha yavaştır. Örneğin, bu yazı yazılırken RAM erişim süreleri 5 ila 70 nanosaniye arasında değişirken, manyetik sabit disk erişim süreleri 5 ila 15 milisaniye ve SSD erişim süreleri 35 ila 100 mikrosaniye arasında değişmektedir. Bu da sabit disklerin RAM'den birkaç kat daha yavaş olduğu anlamına gelmektedir.

Şekil 11.1'de bazı tipik DBMS süreçleri de gösterilmektedir. Süreçlerin sayısı ve isimleri satıcıdan satıcıya değişse de, işlevsellik benzerdir. Aşağıdaki süreçler Şekil 11.1'de gösterilmektedir:

- *Dinleyici*. Dinleyici işlemi istemcilerin isteklerini dinler ve SQL isteklerinin diğer DBMS işlemlerine işlenmesini sağlar. Bir istek alındığında, dinleyici isteği uygun kullanıcı sürecine iletir.
- *Kullanıcı*. DBMS her istemci oturumunu yönetmek için bir kullanıcı süreci oluşturur. Bu nedenle, VTYS'de oturum açtığınızda, size bir kullanıcı işlemi atanır. Bu işlem sunucuya gönderdiğiniz tüm istekleri yönetir. Oturum açmış her istemci için en az bir tane olmak üzere çok sayıda kullanıcı işlemi vardır.
- *Zamanlayıcı*. Zamanlayıcı işlemi SQL isteklerinin eşzamanlı yürütülmesini düzenler. (Bkz. Bölüm 10, İşlem Yönetimi ve Eşzamanlılık Kontrolü.)
- *Kilit yöneticisi*. Bu işlem, disk sayfaları da dahil olmak üzere veritabanı nesnelerine yerleştirilen tüm kilitleri yönetir. (Bkz. Bölüm 10.)
- *İyileştirici*. İyileştirici işlemi SQL sorgularını analiz eder ve verilere erişmenin en verimli yolunu bulur. Bölümün ilerleyen kısımlarında bu süreç hakkında daha fazla bilgi

edinece
ksiniz.

11-1 c Veritabanı Sorgu Optimizasyon Modları

Sorgu optimizasyonu için önerilen algoritmaların çoğu iki ilkeye dayanmaktadır:

- En hızlı yürütme süresini elde etmek için optimum yürütme sırasının seçimi
- İletişim maliyetlerini en aza indirmek için erişilecek sahaların seçimi

Bu iki prensip dahilinde, bir sorgu optimizasyon algoritması, *çalışma modu* veya *optimizasyonunun zamanlaması* temelinde değerlendirilebilir.

Çalışma modları manuel veya otomatik olarak sınıflandırılabilir. **Otomatik sorgu optimizasyonu**, DBMS'nin kullanıcı müdahalesi olmadan en uygun maliyetli erişim yolunu bulması anlamına gelir. **Manuel sorgu optimizasyonu**, optimizasyonun son kullanıcı veya programcı tarafından seçilmesini ve programlanmasını gerektirir. Otomatik sorgu optimizasyonu son kullanıcı açısından açıkça daha arzu edilir bir durumdur, ancak bu kolaylığın bedeli VTYS'ye yüklediği ek yükün artmasıdır.

Sorgu optimizasyon algoritmaları, optimizasyonun ne zaman yapıldığına göre de sınıflandırılabilir. Bu zamanlama sınıflandırması içerisinde sorgu optimizasyon algoritmaları statik veya dinamik olabilir.

- **Statik sorgu optimizasyonu** derleme zamanında gerçekleşir. Başka bir deyişle, sorgu VTYS tarafından derlendiğinde en iyi optimizasyon stratejisi seçilir. Bu yaklaşım, SQL ifadeleri C# veya Python gibi prosedürel programlama dillerine gömüldüğünde yaygındır. Program derleme için VTYS'ye gönderildiğinde, veritabanına erişmek için gerekli planı oluşturur. Program yürütüldüğünde, DBMS veritabanına erişmek için bu planı kullanır.
- **Dinamik sorgu optimizasyonu** yürütme zamanında gerçekleşir. Veritabanı erişim stratejisi program çalıştırıldığında tanımlanır. Bu nedenle erişim stratejisi, veri tabanı hakkındaki en güncel bilgiler kullanılarak çalışma zamanında VTYS tarafından dinamik olarak belirlenir. Dinamik sorgu optimizasyonu verimli olmasına rağmen, maliyeti çalışma zamanı işlem yükü ile ölçülür. En iyi strateji sorgu her çalıştırıldığında belirlenir; bu aynı programda birkaç kez gerçekleşebilir.

Son olarak, sorgu optimizasyon teknikleri, sorguyu optimize etmek için kullanılan bilgi türüne göre sınıflandırılabilir. Örneğin, sorgular istatistiksel tabanlı veya kural tabanlı algoritmalara dayalı olabilir.

- İstatistiksel **tabanlı** bir **sorgu optimizasyon algoritması,** veritabanı hakkında istatistiksel bilgiler kullanır. İstatistikler, boyut, kayıt sayısı, ortalama erişim süresi, hizmet verilen istek sayısı ve erişim haklarına sahip kullanıcı sayısı gibi veritabanı özellikleri hakkında bilgi sağlar. Bu istatistikler daha sonra DBMS tarafından en iyi erişim stratejisini belirlemek için kullanılır. İstatistiksel tabanlı optimize edicilerde, bazı DBMS'ler optimize edicinin ilk satırı veya son satırı alma süresini en aza indirmeye çalışması gerektiğini belirtmek için bir hedef belirlenmesine izin verir. İlk satıra ulaşma süresinin en aza indirilmesi genellikle işlem sistemlerinde ve etkileşimli istemci ortamlarında kullanılır. Bu durumlarda amaç, ilk birkaç satırı kullanıcıya mümkün olduğunca çabuk sunmaktır. Daha sonra, DBMS kullanıcının veriler arasında gezinmesini beklerken, sorgu için diğer satırları getirebilir. Son satırın alınmasını en aza indirmek için optimizer hedefinin ayarlanması genellikle gömülü SQL'de ve saklı prosedürlerin içinde yapılır. Bu durumlarda, tüm veriler alınana kadar kontrol, çağıran uygulamaya geri aktarılmaz; bu nedenle, kontrolün geri döndürülebilmesi için son satıra kadar tüm verilerin mümkün olduğunca çabuk alınması önemlidir.
- İstatistiksel bilgiler DBMS tarafından yönetilir ve iki farklı moddan birinde oluşturulur: dinamik veya manuel. **Dinamik istatistik oluşturma modunda**, DBMS her veri erişim işleminden sonra istatistikleri otomatik olarak değerlendirir ve günceller. **Manuel istatistik oluşturma modunda**, istatistikler DB2 DBMS'ler tarafından kullanılan IBM'in RUNSTAT komutu gibi kullanıcı tarafından seçilen bir yardımcı program aracılığıyla periyodik olarak güncellenmelidir.

otomatik sorgu optimizasyonu
Bir DBMS'nin bir sorgunun yürütülmesi için en verimli erişim yolunu bulduğu bir yöntem.

manuel sorgu optimizasyonu
Son kullanıcının veya programcının bir sorgunun yürütülmesi için erişim yolunu gerektiren bir çalışma modu.

statik sorgu optimizasyonu
Bir veritabanına erişim yolunun derleme zamanında önceden belirlendiği bir sorgu optimizasyon modu.

dinamik sorgu optimizasyonu
Veritabanı hakkındaki en güncel bilgileri kullanarak çalışma zamanında SQL erişim stratejisini belirleme işlemi.

istatistiksel **tabanlı sorgu optimizasyon algoritması**
Bir veritabanı hakkında istatistiksel bilgiler kullanan bir sorgu optimizasyon tekniği. VTYS daha sonra en iyi erişim stratejisini belirlemek için bu istatistikleri kullanır.

dinamik istatistiksel üretim modu
Bir DBMS'de, her veri erişim işleminden sonra veritabanı erişim istatistiklerini otomatik olarak değerlendirme ve güncelleme yeteneği.

manuel istatistik oluşturma modu
Sorgu için istatistiksel veri erişim bilgisi oluşturma modu optimizasyon. Bu moddaDBA veri erişim istatistiklerini oluşturmak için periyodik olarak bir rutin ; örneğin IBM DB2 veritabanında RUNSTAT komutu.

Kural tabanlı sorgu optimizasyon algoritması
Bir sorguyu yürütmek için en iyi yaklaşımı belirlemek üzere önceden kuralları ve noktaları kullanan bir sorgu optimizasyon tekniği.

- **Kural tabanlı** bir **sorgu optimizasyon algoritması**, en iyi sorgu erişim stratejisini belirlemek için bir dizi kullanıcı tanımlı kurala dayanır. Kurallar son kullanıcı veya veritabanı yöneticisi tarafından girilir ve genellikle genel niteliktedir.

Veritabanı istatistikleri sorgu optimizasyonunda çok önemli bir rol oynadığından, bu konu bir sonraki bölümde daha ayrıntılı olarak incelenmektedir.

11-1 d Veritabanı İstatistikleri

veritabanı istatistikleri Sorgu optimizasyonunda, aşağıdakilerle ilgili ölçümler
Bir tablodaki satır sayısı, kullanılan disk bloğu sayısı, maksimum ve ortalama satır uzunluğu, her satırdaki sütun sayısı ve satır sayısı gibi veritabanı nesneleri her sütundaki farklı değerlerin sayısı. Bu tür istatistikler, veritabanı özelliklerinin anlık görüntüsünü sağlar.

Sorgu optimizasyonunda önemli bir rol oynayan bir diğer DBMS işlemi de veritabanı istatistiklerinin toplanmasıdır. **Veritabanı istatistikleri** terimi, kullanılan işlemci sayısı, işlemci hızı ve mevcut geçici alan gibi veritabanı nesneleri hakkında bir dizi ölçümü ifade eder. Bu tür istatistikler, veritabanı özelliklerinin anlık görüntüsünü sağlar.

Bu bölümün ilerleyen kısımlarında öğreneceğiniz gibi, VTYS bu istatistikleri sorgu işleme verimliliğini artırma konusunda kritik kararlar almak için kullanır. Veritabanı istatistikleri DBA tarafından manuel olarak ya da DBMS tarafından otomatik olarak toplanabilir. Örneğin, birçok DBMS satıcısı istatistik toplamak için SQL'de ANALYZE komutunu destekler. Buna ek olarak, birçok tedarikçinin istatistik toplamak için kendi rutinleri vardır. Örneğin, IBM'in DB2'si RUNSTATS prosedürünü kullanırken, Microsoft'un SQL Server'ı UPDATE STATISTICS prosedürünü kullanır ve başlatma parametrelerinde Auto-Update ve Auto-Create Statistics seçeneklerini sunar. DBMS'nin çeşitli veritabanı nesneleri hakkında toplayabileceği ölçümlerin bir örneği Tablo 11.2'de gösterilmektedir.

Tablo 11.2 Örnek Veritabanı İstatistik Ölçümleri

Veritabanı Nesnesi	Örnek Ölçümler
Masalar	Satır sayısı, kullanılan disk bloğu sayısı, satır uzunluğu, her sütun sayısı, her sütundaki farklı değer sayısı, her sütundaki maksimum değer, her sütundaki minimum değer ve dizinleri olan sütunlar
İndeksler	Dizin anahtarındaki sütunların sayısı ve adı, dizindeki anahtar değerlerinin sayısı, dizin anahtarındaki farklı anahtar değerlerinin sayısı, bir dizindeki anahtar değerlerinin histogramı ve dizin tarafından kullanılan disk sayfalarının sayısı
Çevre Kaynakları	Mantıksal ve fiziksel disk bloğu boyutu, veri dosyalarının konumu ve boyutu ve veri dosyası başına genişletme sayısı

Nesne istatistikleri mevcutsa, DBMS bunları sorgu işlemede kullanacaktır. Yeni DBMS'lerin çoğu (Oracle, MySQL, SQL Server ve DB2 gibi) istatistikleri otomatik olarak toplar; diğerleri ise DBA'nın istatistikleri manuel olarak toplamasını gerektirir. Veritabanı nesne istatistiklerini manuel olarak oluşturmak için her DBMS'nin kendi komutları vardır.

Oracle'da ANALYZE <TABLE/INDEX> object_name COMPUTE STATISTICS;

MySQL'de ANALYZE TABLE <table_name> kullanın;

SQL Server'da UPDATE STATISTICS <object_name> kullanın; burada nesne adı bir tablo veya görünümü ifade eder.

Örneğin, VENDOR tablosu için istatistik oluşturmak için şunu kullanırsınız:

Oracle'da: ANALYZE TABLE VENDOR COMPUTE STATISTICS;

MySQL'de: TABLO SATICISINI ANALIZ ET;

SQL Server'da: ISTATISTIK SATICISINI GÜNCELLE;

Bir tablo için istatistik oluşturduğunuzda, ilgili tüm dizinler de analiz edilir. Ancak, aşağıdaki komutu kullanarak tek bir dizin için istatistik oluşturabilirsiniz; burada VEND_ NDX dizinin adıdır:
VEND_NDX DIZININI ANALIZ ET ISTATISTIKLERI HESAPLA;

SQL Server'da UPDATE STATISTICS <tablo_adı> <index_adı> komutunu kullanın. Örnek bir komut UPDATE STATISTICS VENDOR VEND_NDX; şeklinde olabilir.

Veritabanı istatistikleri sistem kataloğunda özel olarak belirlenmiş tablolarda saklanır. Veritabanı nesneleri için istatistiklerin, özellikle de veritabanı istatistiklerinin periyodik olarak yeniden oluşturulması yaygındır.

sık sık değişime tabi olan nesneler. Örneğin, bir video kiralama VTYS'niz varsa, sisteminiz muhtemelen günlük video kiralamalarını depolamak için bir RENTAL tablosu kullanacaktır. Bu RENTAL tablosu ve ilişkili indeksleri, siz günlük kiralamaları ve iadeleri kaydettikçe sürekli ekleme ve güncellemelere tabi olacaktır. Bu nedenle, geçen hafta oluşturduğunuz RENTAL tablosu istatistikleri tablonun bugünkü halini tam olarak göstermez. İstatistikler ne kadar güncel olursa, DBMS'nin belirli bir sorguyu yürütmek için en hızlı yolu doğru şekilde seçme şansı o kadar artar.

Artık DBMS süreçlerinin ve bellek yapılarının temel mimarisini ve DBMS tarafından toplanan veritabanı istatistiklerinin önemini ve zamanlamasını bildiğinize göre, DBMS'nin bir SQL sorgu isteğini nasıl işlediğini öğrenmeye hazırsınız.

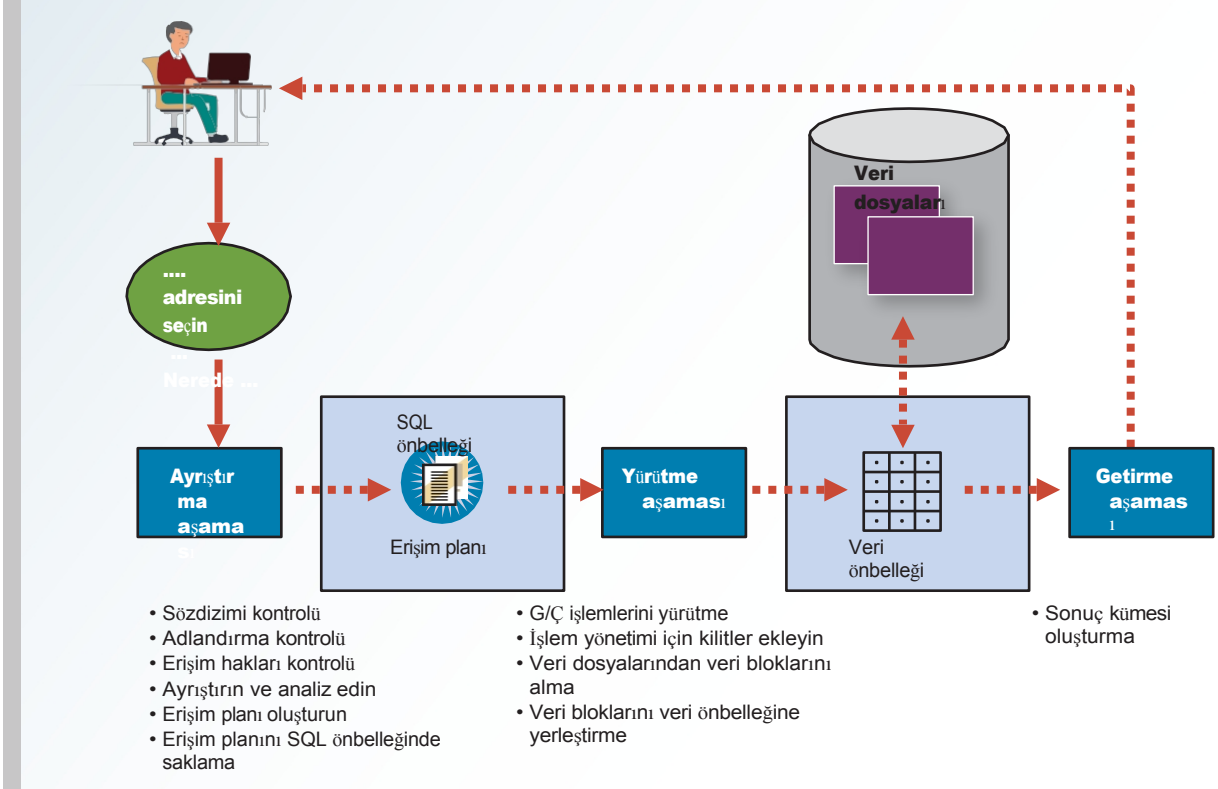
11-2 Sorgu İşleme

İstemcinin SQL ifadesi alındığında VTYS sunucu ucunda ne olur? Basit bir ifadeyle, VTYS bir sorguyu üç aşamada işler:

1. *Ayrıştırma*. DBMS SQL sorgusunu ayrıştırır ve en verimli erişim/çalıştırma planını seçer.
2. *Yürütme*. DBMS, seçilen yürütme planını kullanarak SQL sorgusunu yürütür.
3. *Getirme*. DBMS verileri getirir ve sonuç kümesini istemciye geri gönderir.

SQL DDL deyimlerinin (CREATE TABLE gibi) işlenmesi, DML deyimlerinin gerektirdiği işlemden farklıdır. Aradaki fark, bir DDL durumu veri sözlüğü tablolarını veya sistem kataloğunu güncellerken, bir DML durumu (SELECT, INSERT, UPDATE veya DELETE) çoğunlukla son kullanıcı verilerini manipüle eder. Şekil 11.2'de sorgu işleme için gereken genel adımlar gösterilmektedir. Her adım aşağıdaki bölümlerde ele alınmaktadır.

Şekil 11.2 Sorgu İşleme



11-2 a SQL Ayrıştırma Aşaması

Optimizasyon süreci, sorguyu daha küçük birimlere ayırmayı (ayrıştırmayı) ve orijinal SQL sorgusunu orijinal SQL kodunun biraz farklı bir versiyonuna, ancak tamamen eşdeğer ve daha verimli bir versiyona dönüştürmeyi içerir. *Tamamen eşdeğer*, optimize edilmiş sorgu sonuçlarının her zaman orijinal ile aynı olduğu anlamına gelir. *Daha verimli* olması, optimize edilmiş sorgunun neredeyse her zaman orijinal daha hızlı çalışacağı anlamına gelir. (*Neredeyse* her zaman daha hızlı çalıştığını unutmayın çünkü daha önce açıklandığı gibi bir veritabanının performansını birçok faktör etkiler. Bu faktörler arasında ağ, istemci bilgisayarın kaynakları ve aynı veritabanında eşzamanlı olarak çalışan diğer sorgular yer alır). Sorguyu çalıştırmanın en verimli yolunu belirlemek için DBMS daha önce öğrendiğiniz veritabanı istatistiklerini kullanabilir.

sorgu iyileştirici
SQL sorgularını analiz eden ve verilere erişmenin en verimli yolunu bulan bir DBMS işlemi. Sorgu iyileştirici, sorgu için erişim veya yürütme planı oluşturur.

SQL ayrıştırma faaliyetleri, SQL sorgusunu analiz eden ve verilere erişmenin en verimli yolunu bulan **sorgu iyileştirici** tarafından gerçekleştirilir. Bu işlem, sorgu işlemede en çok zaman alan aşamadır. Bir SQL sorgusunun ayrıştırılması, SQL sorgusunun olduğu birkaç adım gerektirir:

- Sözdizimi uyumluluğu için onaylandı
- Tablo adlarının ve sütun adlarının doğru olduğundan emin olmak için veri sözlüğüne göre doğrulanır
- Kullanıcının uygun erişim haklarına sahip olduğundan emin olmak için veri sözlüğüne göre doğrulanır
- Analiz edilmiş ve daha atomik bileşenlere ayrıştırılmıştır
- Tamamen eşdeğer ancak daha verimli bir SQL sorgusuna dönüştürülerek optimize edilmiştir
- En verimli yürütme veya erişim planı belirlenerek yürütme için hazırlanır

erişim planı
Uygulama derleme zamanında oluşturulan ve bir VTYS tarafından yönetilen bir dizi talimat. Erişim planı, bir uygulamanın sorgusunun çalışma zamanında veritabanına nasıl erişeceğini önceden belirler.

SQL ifadesi dönüştürüldükten sonra, DBMS genellikle erişim planı veya yürütme planı olarak bilinen planı oluşturur. Bir **erişim planı**, bir SQL ifadesinin ayrıştırılmasının sonucudur; bir DBMS'nin sorguyu yürütmek ve sonuç kümesini en verimli şekilde döndürmek için kullanacağı bir dizi adımı içerir. İlk olarak, DBMS SQL önbelleğinde sorgu için erişim planının zaten var olup olmadığını kontrol eder. Eğer , DBMS zaman kazanmak için erişim planını yeniden kullanır. Eğer yoksa, iyileştirici çeşitli planları değerlendirir ve ardından hangi dizinlerin ve birleştirme işlemlerinin en iyi nasıl gerçekleştirileceğine karar verir. Sorgu için seçilen erişim planı daha sonra SQL önbelleğine yerleştirilir ve kullanım ve gelecekte yeniden kullanım için hazır hale getirilir.

Erişim planları DBMS'ye özgüdür ve istemcinin SQL sorgusunu, verileri fiziksel veri dosyalarından okumak ve sonuç kümesini oluşturmak için gereken bir karmaşık I/O işlemine dönüştürür. Tablo 11.3'te Oracle RDBMS için bazı I/O işlemleri gösterilmektedir. Çoğu DBMS, veri kümelerine erişirken ve bunları değiştirirken benzer türde işlemler gerçekleştirir.

Tablo 11.3 Örnek DBMS Erişim Planı G/Ç İşlemleri

Operasyon	Açıklama
Tablo taraması (tam)	Tüm tabloyu sırayla, ilk satırdan son satıra kadar, her seferinde bir satır okur (en yavaş)
Tablo erişimi (satır kimliği)	Satır kimliği değerini kullanarak bir tablo satırını doğrudan okur (en hızlı)
Dizin taraması (aralık)	Satır kimliklerini elde etmek için önce dizini okur ve ardından tablo satırlarına doğrudan erişir (tam tablo taramasından daha hızlıdır)
Dizin erişimi (benzersiz)	Bir tablo, bir sütunda benzersiz bir dizine sahip olduğunda kullanılır
İç içe döngü	İç içe döngü stili kullanarak bir değerler kümesini okur ve başka bir değerler kümesiyle karşılaştırır (yavaş)
Birleştirme	İki veri setini birleştirir (yavaş)
Sırala	Bir veri kümesini sıralar (yavaş)

Tablo 11.3'te, bir satır kimliği kullanarak tablo erişiminin en hızlı yöntem olduğuna dikkat edin. Satır kimliği, kalıcı depolama alanına kaydedilen her satır için benzersiz bir tanımlamadır; satıra doğrudan erişmek için kullanılabilir. Kavramsal olarak, bir satır kimliği, arabanızı bir havaalanı otoparkına park ettiğinizde aldığınız bir fişe benzer. Park fişi bölüm numarasını ve lot numarasını içerir. Bu bilgileri kullanarak, her bölümü ve lotu aramadan doğrudan arabanıza gidebilirsiniz.

11-2 b SQL Yürütme Aşaması

Bu aşamada, erişim planında belirtilen tüm G/Ç işlemleri yürütülür. Yürütme planı çalıştırıldığında, erişilecek veriler için -gerekirse- uygun kilitler edinilir ve veriler veri dosyalarından alınarak DBMS'nin veri önbelleğine yerleştirilir. Tüm işlem yönetimi komutları, sorgu işlemenin ayrıştırma ve yürütme aşamaları sırasında işlenir.

11-2 c SQL Getirme Aşaması

Ayrıştırma ve yürütme aşamaları tamamlandıktan sonra, belirtilen koşul(lar)la eşleşen tüm satırlar alınır, sıralanır, gruplandırılır ve toplanır (gerekirse). Getirme aşaması sırasında, elde edilen sorgu sonuç kümesinin satırları istemciye döndürülür. VTYS, geçici verileri depolamak için geçici tablo alanı kullanabilir. Bu aşamada, veritabanı sunucusu sonuç kümesi satırlarının sunucu önbelleğinden istemci taşınmasını koordine eder. Örneğin, belirli bir sorgu sonuç kümesi 9.000 satır içerebilir; sunucu ilk 100 satırı istemciye gönderir ve ardından tüm sonuç kümesi istemciye gönderilene kadar istemcinin bir sonraki satır talep etmesini bekler.

11-2d Sorgu İşleme Darboğazları

Sorgu işlemenin temel amacı, belirli bir sorguyu en az kaynakla mümkün olan en hızlı şekilde yürütmektir. Gördüğünüz gibi, bir sorgunun yürütülmesi, DBMS'nin sorguyu işbirlikçi bir şekilde yürütülecek bir dizi birbirine bağlı I/O işlemine ayırmasını gerektirir. Bir sorgu ne kadar karmaşıksa, işlemler de o kadar karmaşıktır, bu da darboğazların daha olası olduğu anlamına gelir. Bir **sorgu işleme darboğazı**, bir G/Ç işleminin işlenmesinde ortaya çıkan ve tüm sistemin yavaşlamasına neden olan bir gecikmedir. Aynı şekilde, bir sistem ne kadar fazla bileşene sahipse, bileşenler arasında o kadar fazla arayüz gerekir ve bu da darboğaz olasılığını artırır. Bir DBMS içinde, tipik olarak beş bileşen darboğazlara neden olur:

sorgu işleme darboğazı
Sorgu optimizasyonunda, bir G/Ç'nin işlenmesinde ortaya çıkan gecikme tüm sistemin neden olan işlem.

- *CPU*. DBMS'nin CPU işlem gücü, sistemin beklenen iş yüküyle eşleşmelidir. Yüksek CPU kullanımı, işlemci hızının gerçekleştirilen iş miktarı için yavaş olduğunu gösterebilir. Ancak, yüksek CPU kullanımına arızalı bir bileşen, yeterli RAM olmaması (CPU bellek bloklarını değiştirmek için çok fazla zaman harcar), kötü yazılmış bir aygıt sürücüsü veya hileli bir işlem gibi başka faktörler de neden olabilir. Bir CPU darboğazı sadece DBMS'yi değil sistemde çalışan tüm süreçleri etkileyecektir.
- *RAM*. DBMS, veri önbelleği ve SQL önbelleği gibi belirli kullanımlar için bellek ayırır. RAM, işletim sistemi ve DBMS dahil olmak üzere çalışan tüm işlemler arasında paylaşılmalıdır. Yeterli RAM mevcut değilse, kıt RAM için rekabet eden bileşenler arasında veri taşımak bir darboğaz yaratabilir.
- *Sabit disk*. Darboğazların diğer yaygın nedenleri sabit disk hızı ve veri aktarım hızlarıdır. Mevcut sabit disk depolama teknolojisi geçmişe kıyasla daha fazla depolama kapasitesi sağlamaktadır; ancak sabit disk alanı son kullanıcı verilerini depolamaktan daha fazlası için kullanılmaktadır. Mevcut işletim sistemleri sabit diski *sanal bellek* için de kullanmaktadır; bu da daha acil görevler için RAM'de yer açmak amacıyla RAM alanlarının gerektiğinde sabit diske kopyalanması anlamına gelmektedir. Bu nedenle, daha fazla sabit disk depolama alanı ve daha hızlı veri aktarım hızları darboğaz olasılığını azaltır.
- *Ağ*. Bir veritabanı ortamında, veritabanı sunucusu ve istemciler bir ağ üzerinden bağlanır. Tüm ağlar, tüm istemciler arasında paylaşılan sınırlı miktarda bant genişliğine sahiptir. Birçok ağ düğümü ağa aynı anda eriştiğinde, darboğazlar olasıdır.
- *Uygulama kodu*. Tüm darboğazlar sınırlı donanım kaynaklarından kaynaklanmaz. Darboğazların en yaygın iki kaynağı, yetersiz uygulama kodu ve kötü tasarlanmış

veritabanları. Altta yatan veritabanı tasarımı sağlam olduğu sürece, düşük kaliteli kod, kod optimizasyon teknikleriyle iyileştirilebilir. Ancak hiçbir kodlama, kötü tasarlanmış bir veritabanının daha iyi performans göstermesini sağlayamaz.

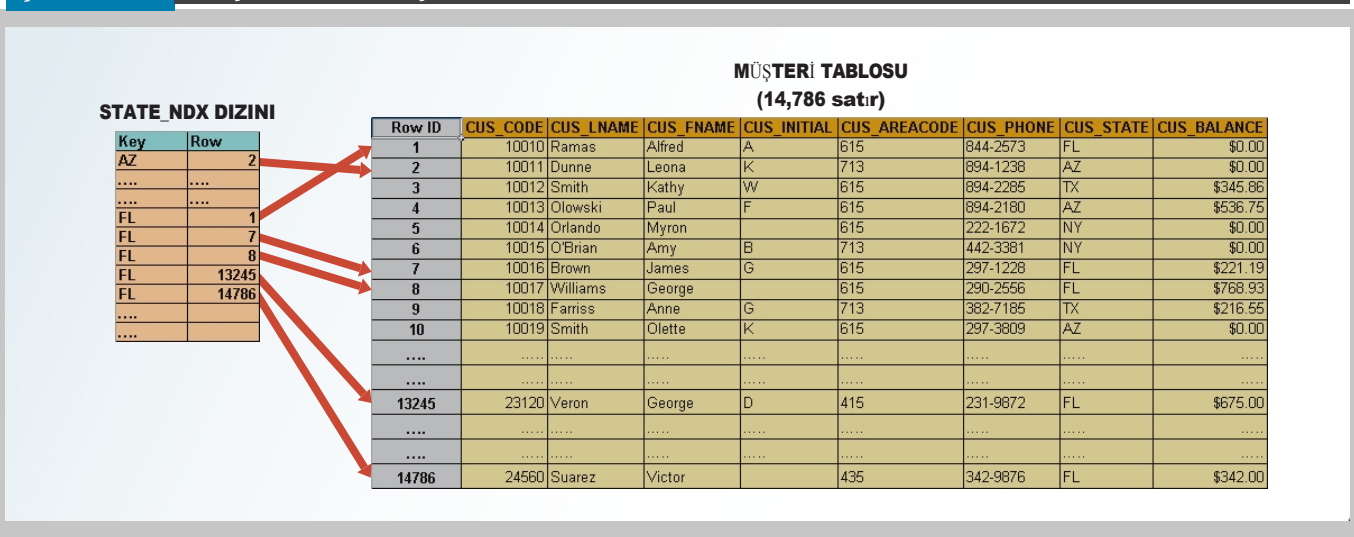
Darboğazlar, veri tabanı kaynaklarının (CPU, RAM, sabit disk, indeksler, kilitler, tamponlar, vb.) kullanımı için rekabet eden birden fazla veritabanı işleminin sonucudur. Bu bölümde daha önce öğrendiğiniz gibi, bir DBMS işlemlerini gerçekleştirmek için işlemler, , kilitler, tablo alanları, dizinler ve günlük dosyaları gibi birçok bileşen ve yapı kullanır. Bu kaynaklar veritabanında yürütülen tüm işlemler tarafından kullanılır ve bu nedenle işlemler genellikle bu kaynaklar için rekabet eder. İşlemlerin çoğu (hepsi olmasa da) tablolardaki veri satırlarıyla çalıştığından, en tipik darboğazlardan biri aynı veri satırları için rekabet eden işlemlerden kaynaklanır. Bir başka yaygın çekişme kaynağı da paylaşılan bellek kaynakları, özellikle de paylaşılan tamponlar ve kilitlerdir. Veri güncelleme işlemlerini hızlandırmak için DMBS verileri önbelleğe almak üzere tamponlar kullanır. Aynı zamanda, verilere erişimi yönetmek için DBMS kilitleri kullanır. Bu tıkanıklıkların nasıl önleneceğini ve veritabanı performansının nasıl optimize edileceğini öğrenmek bu bölümün ana odak noktasıdır.

11-3 Dizinler ve Sorgu Optimizasyonu

İndeksler veri erişimini hızlandırmada çok önemlidir çünkü arama, sıralama, toplama işlevlerini kullanma ve hatta birleştirme işlemlerini kolaylaştırırlar. Veri erişim hızındaki iyileşme, bir dizinin dizin anahtarını ve işaretçileri içeren sıralı bir değerler kümesi olması nedeniyle gerçekleşir. İşaretçiler, gerçek tablo satırları için satır kimlikleridir. Kavramsal olarak, bir veri dizini bir kitap dizinine benzer. Bir kitap dizini kullandığınızda, dizin anahtarına benzer bir sözcüğü ararsınız. Sözcüğe, sözcüğün kullanıldığı bir veya daha fazla sayfa numarası eşlik eder; bu numaralar işaretçilere benzer.

Dizin taraması tam tablo taramasından daha verimlidir çünkü dizin verileri önceden sıralanmıştır ve veri miktarı genellikle çok daha azdır. Bu nedenle, arama yaparken, DBMS'nin bir tabloya erişmek için dizini kullanması, bir tablodaki tüm satırları sırayla taramaktan neredeyse her zaman daha iyidir. Örneğin, Şekil 11.3, 14.786 satırlı bir CUSTOMER tablosunun ve CUS_STATE özniteliğindeki STATE_NDX dizininin dizin gösterimini göstermektedir.

Şekil 11.3 Müşteri Tablosu için Dizin Gösterimi



Aşağıdaki sorguyu gönderdiğinizi varsayalım:

```
SEÇİNİZ                    CUS_NAME, CUS_STATE
FROM                       MÜŞTERİ
NEREDE                     CUS_STATE 5 'FL';
```

Dizin yoksa, DBMS bir tam tablo taraması gerçekleştirecek ve tüm 14.786 müşteri satırını okuyacaktır. STATE_NDX dizininin oluşturulduğunu (ve analiz edildiğini) varsayarsak, DBMS otomatik olarak 'FL'ye eşit bir duruma sahip ilk müşteriyi bulmak için dizini kullanacak ve ardından dizindeki satır kimliklerini kılavuz olarak kullanarak sonraki tüm CUSTOMER satırlarını okumaya devam edecektir. Sadece beş satırın CUS_STATE 5 'FL' koşulunu karşıladığını varsayarsak, dizine beş erişim ve verilere beş erişim olmak üzere toplam 10 G/Ç erişimi olacaktır. DBMS, kriterleri karşılamayan müşteri satırları için yaklaşık 14.776 G/Ç talebini okumaktan kurtulacaktır. Bu çok fazla CPU döngüsü demektir!

Eğer indeksler bu kadar önemliyse, neden her tablodaki her sütunu indekslemiyoruz? Basit cevap, bunu yapmanın pratik olmadığıdır. Her tablodaki her sütunu indekslemek, özellikle tabloda çok sayıda nitelik ve satır varsa veya çok sayıda ekleme, güncelleme ve silme işlemi gerektiriyorsa, DBMS'yi indeks bakım işlemleri açısından aşırı yorar.

Bir dizin ihtiyacını belirleyen ölçütlerden biri, dizinlemek istediğiniz sütunun veri seyrekliğidir. **Veri seyrekliği**, bir sütunun sahip olabileceği farklı değerlerin sayısını ifade eder. Örneğin, STUDENT tablosundaki bir STU_SEX sütunu yalnızca iki olası değere sahip olabilir: M veya F; bu nedenle, bu sütunun düşük seyrekliğe sahip olduğu söylenir. Buna karşılık, öğrencinin doğum tarihini saklayan STU_DOB sütunu birçok farklı tarih değerine sahip olabilir; bu nedenle, bu sütunun yüksek seyrekliğe sahip olduğu söylenir. Seyrekliğin bilinmesi, bir dizin kullanımının uygun olup olmadığına karar vermenize yardımcı olur. Örneğin, düşük bir sütunda arama yaptığınızda, tablo satırlarının yüksek bir yüzdesini okumanız ; bu nedenle, dizin işleme gereksiz bir iş olabilir. Bölüm 11-5'te, bir dizinin ne zaman önerileceğini nasıl belirleyeceğinizi öğreneceksiniz.

veri seyrekliği
Değerlerin sütun dağılımı veya bir sütunun sahip olabileceği farklı değerlerin sayısı.

Çoğu DBMS, dizinleri aşağıdaki veri yapılarından birini kullanarak uygular:

- **Hash indeksi**. Bir karma dizin, karma değerlerin sıralı bir listesine dayanır. Bir anahtar sütunundan bir hash değeri oluşturmak için bir hash algoritması kullanılır. Bu değer, hash tablosundaki bir girdiye işaret eder ve bu da veri satırının gerçek konumuna işaret eder. Bu tür bir dizin, eşitlik koşullarına dayalı basit ve hızlı arama işlemleri için iyidir; örneğin, LNAME5 "Scott" ve FNAME5 "Shannon".

karma indeks
Sıralı bir karma değerler listesine dayalı bir dizin.

- **B-ağacı dizini**. B-ağacı dizini, baş aşağı bir ağaç olarak düzenlenmiş sıralı bir veri yapısıdır. (Bkz. Şekil 11.4.) Dizin ağacı verilerden ayrı olarak saklanır. B-ağacı dizininin alt düzey yaprakları gerçek veri satırlarının işaretçilerini içerir. B-ağacı dizinleri "kendi kendine dengelidir", yani dizindeki herhangi bir satıra erişmek yaklaşık olarak aynı miktarda zaman alır. Bu, veritabanlarında kullanılan varsayılan ve en yaygın dizin türüdür. B-ağacı dizini esas olarak sütun değerlerinin nispeten az sayıda tekrar ettiği tablolarda kullanılır.

B-ağacı endeksi
Baş aşağı bir ağaç olarak düzenlenmiş sıralı bir veri yapısı.

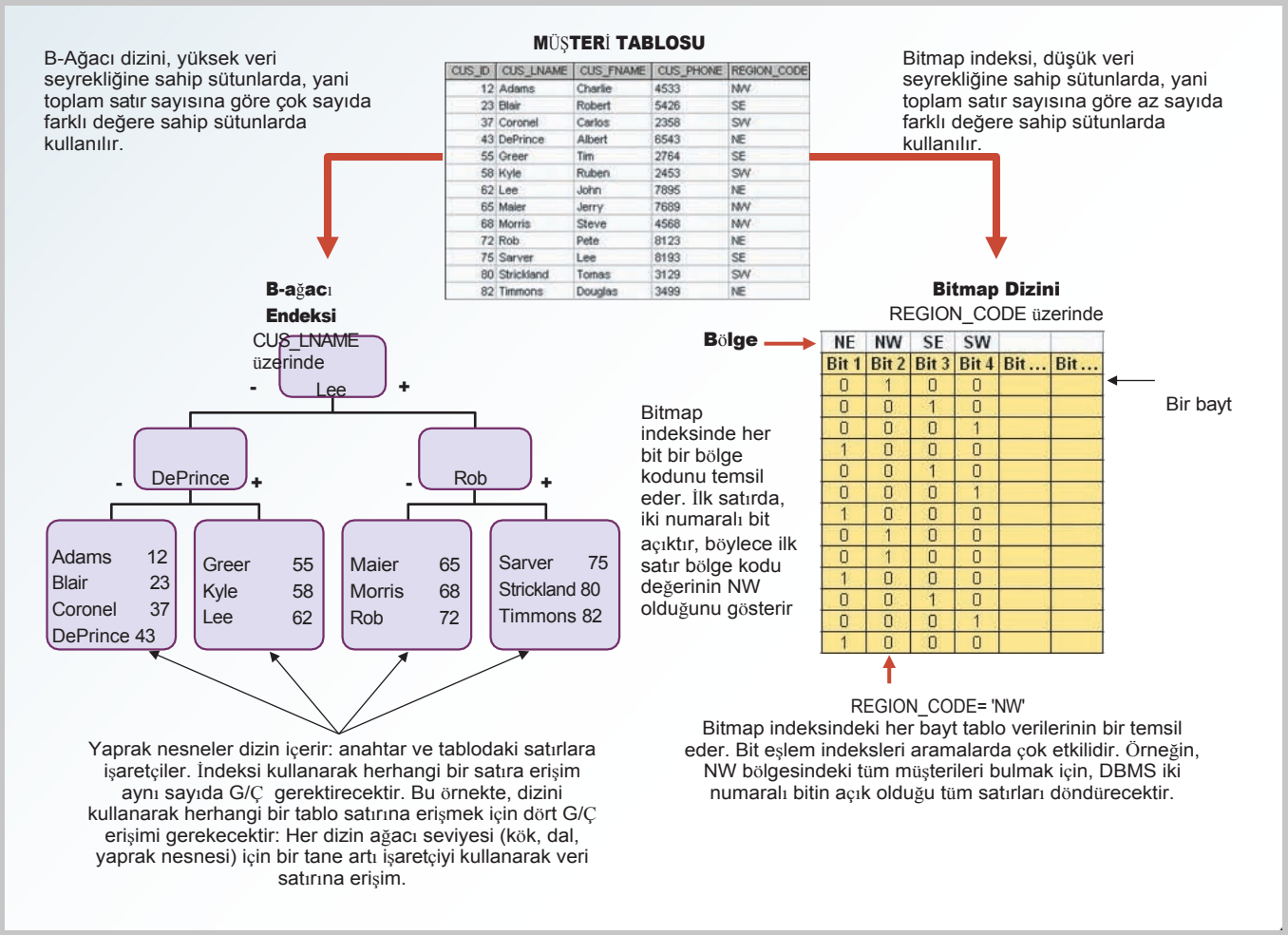
- **Bit eşlem indeksi**. Bir bit eşlem dizini, bir değerin veya koşulun varlığını temsil etmek için bir bit dizisi (0'lar ve 1'ler) kullanır. Bu indeksler çoğunlukla veri ambarı uygulamalarında, az sayıda sütun değerinin birçok kez tekrar ettiği çok sayıda satır içeren tablolarda kullanılır. (Bkz. Şekil 11.4.) Bitmap indeksleri, verilerini depolamak için bayt yerine bit kullandıklarından B-ağacı indekslerinden daha az alan kullanma eğilimindedir.

bitmap dizini
Temsil etmek için bir bit dizisi (0'lar ve 1'ler) kullanan bir dizin bir değerin veya koşulun varlığı.

Bir veritabanı tasarımcısı, önceki dizin özelliklerini kullanarak kullanılacak en iyi dizin türünü belirleyebilir. Örneğin, bir CUSTOMER tablosunun birkaç bin satıra sahip olduğunu varsayalım. CUSTOMER tablosunda sorgu amacıyla yoğun olarak kullanılan iki sütun vardır: Bir müşterinin soyadını temsil eden CUS_LNAME ve dört değerden (NE, NW, SW ve SE) birine sahip olabilen REGION_CODE. Bu bilgilere dayanarak sonuca varabilirsiniz:

- CUS_LNAME sütunu, tablodaki toplam satır sayısına kıyasla nispeten az sayıda tekrar eden birçok farklı değer içerdiğinden, bir B-ağacı dizini kullanılacaktır.
- REGION_CODE sütunu, tablodaki toplam satır sayısına kıyasla nispeten çok sayıda tekrar eden yalnızca birkaç farklı değer içerdiğinden, bir bit eşlem dizini kullanılacaktır. Şekil 11.4, önceki tartışmada kullanılan CUSTOMER tablosu için B-ağacı ve bit eşlem gösterimlerini göstermektedir.

Şekil 11.4 B-Ağacı ve Bitmap Dizin Gösterimi



Mevcut nesil DBMS'ler, DBMS'nin güncellenmiş veritabanı istatistiklerine sahip olması koşuluyla, belirli koşullar altında kullanılacak en iyi dizin türünü belirleyecek kadar akıllıdır. Hangi dizinin seçildiğine bakılmaksızın, DBMS belirli bir sorguyu yürütmek için en iyi planı belirler. Bir sonraki bölüm, sorgu iyileştiricinin yapması gereken seçimlerin türüne ilişkin basitleştirilmiş bir örnek üzerinden size yol gösterir.

11-4 Optimize Edici Seçenekleri

kural **tabanlı iyileştirici**
Kural tabanlı sorgu optimizasyon algoritmasına dayalı bir sorgu optimizasyon modu.

maliyet tabanlı optimize edici
Satır sayısı, mevcut dizinler, dizin seyrekliği vb. dahil olmak üzere erişilen nesneler hakkındaki istatistiklere dayalı bir algoritma kullanan bir sorgu optimizasyon modu

Sorgu optimizasyonu, sorgu işlemede ayrıştırma aşamasındaki merkezi faaliyettir. Bu aşamada, DBMS hangi indekslerin kullanılacağını, birleştirme işlemlerinin nasıl gerçekleştirileceğini, ilk olarak hangi tablonun kullanılacağını vb. seçmelidir. Her DBMS'nin verilere erişmenin en verimli yolunu belirlemek için kendi algoritmaları vardır. Sorgu iyileştirici iki moddan birinde çalışabilir:

- **Kural tabanlı** bir **iyileştirici**, bir sorguyu yürütmek en iyi yaklaşımı belirlemek üzere önceden belirlenmiş kuralları ve noktaları kullanır. Kurallar her SQL işlemine bir "sabit maliyet" atar; maliyetler daha sonra yürütme planının maliyetini vermek için toplanır. Örneğin, tam bir tablo taraması 10 set maliyetine sahipken, satır kimliğine göre bir tablo erişimi 3 set maliyetine sahiptir.
- **Maliyet tabanlı** bir en **iyileştirici**, bir sorguyu yürütmek için en iyi yaklaşımı belirlemek üzere erişilen nesneler hakkındaki istatistiklere dayanan gelişmiş algoritmalar kullanır. Bu durumda, en iyi duruma getirme işlemi, belirli bir yürütme planının toplam maliyetini belirlemek için işlem maliyetini, G/Ç maliyetlerini ve kaynak maliyetlerini (RAM ve geçici alan) toplar.

Optimize edicinin amacı, bir sorguyu yürütmek için alternatif yollar bulmak, her bir alternatifin "maliyetini" değerlendirmek ve ardından en düşük maliyetli olanı seçmektir. Sorgu optimize edicinin işlevini anlamak için basit bir örnek düşünün. Florida'da bulunan bir satıcı tarafından sağlanan tüm ürünleri listelemek istediğinizi varsayalım. Bu bilgiyi elde etmek için aşağıdaki sorguyu yazabilirsiniz:

```
SEÇİNİZ      P_CODE, P_DESCRIPT, P_PRICE, V_NAME, V_STATE
FROM         ÜRÜN, SATICI
NEREDE       PRODUCT.V_CODE 5 VENDOR.V_CODE VE
             VENDOR.V_STATE 5 'FL';
```

Ayrıca, veritabanı istatistiklerinin aşağıdakileri gösterdiğini varsayın:

- PRODUCT tablosunda 7.000 satır vardır.
- VENDOR tablosunda 300 satır vardır.
- On satıcı Florida'da yer almaktadır.
- Bin ürün Florida'daki satıcılardan geliyor.

Optimize edicinin sadece ilk iki öğeyi kullanabileceğini belirtmek önemlidir. İkinci iki maddenin optimize edicinin yapması gereken seçimleri gösterdiği varsayılmaktadır. İlk iki maddedeki bilgilerle donanmış olan optimize edici, verilere erişmenin en verimli yolunu bulmaya çalışacaktır. En verimli erişim planının belirlenmesinde birincil faktör I/O maliyetidir. (DBMS'nin her zaman G/Ç işlemlerini en aza indirmeye çalıştığını unutmayın.) Tablo 11.4'te önceki sorgu için iki örnek erişim planı ve bunların ilgili G/Ç maliyetleri gösterilmektedir.

Tablo 11.4 Erişim Planları ve I/O Maliyetlerinin Karşılaştırılması

Plan	Adım	Operasyon	G/Ç İşlemleri	I/O Maliyeti	Sonuçlanan Set Satırları	Toplam I/O Maliyeti
A	A1	Kartezyen çarpım (ÜRÜN, SATICI)	7,000 1 300	7,300	2,100,000	7,300
	A2	A1'de eşleşen satıcı kodlarına sahip satırları seçin	2,100,000	2,100,000	7,000	2,107,300
	A3	A2 içinde V_STATE 5 'FL' ile satırları seçin	7,000	7,000	1,000	**2,114,300**
B	B1	VENDOR içinde V_STATE 5 'FL' ile satırları seçin	300	300	10	300
	B2	Kartezyen Çarpım (PRODUCT, B1)	7,000 1 10	7,010	70,000	7,310
	B3	B2'de eşleşen satıcı kodlarına sahip satırları seçin	70,000	70,000	1,000	**77,310**

Örneğin anlaşılmasını kolaylaştırmak için, Tablo 11.4'teki G/Ç İşlemleri ve G/Ç Maliyeti sütunları yalnızca VTYS'nin gerçekleştirmesi gereken G/Ç disk okuma sayısını tahmin etmektedir. Basitlik açısından, indeks olmadığı ve okunan her satırın G/Ç maliyetinin 1 olduğu varsayılmıştır. Örneğin, Adım A1'de, DBMS'nin PRODUCT VENDOR'ın Kartezyen çarpımını hesaplaması gerekir. Bunu yapmak için, VTYS'nin ÜRÜN'deki tüm satırları (7.000) ve SATICI'daki tüm satırları (300) okuması ve toplam 7.300 G/Ç işlemi gerçekleştirmesi gerekir. Tüm adımlarda aynı hesaplama yapılır. Tablo 11.4'te, Plan A'nın Plan B'den neredeyse 30 kat daha yüksek bir toplam G/Ç maliyetine sahip olduğunu görebilirsiniz. Bu durumda, iyileştirici SQL'i yürütmek için Plan B'yi seçecektir.

Not

Tüm DBMS'ler SQL sorgularını aynı şekilde optimize etmez. Oracle, bu bölümdeki çeşitli kısımlarda açıklanan yöntemlerden farklı şekilde ayrıştırır. DBMS uygulamanızın optimizasyon gereksinimlerini incelemek için her zaman belgeleri okuyun.

Doğru koşullar sağlandığında, bazı sorgular yalnızca bir dizin kullanılarak tamamen yanıtlanabilir. Örneğin, PRODUCT tablosunu ve P_QOH özniteliğinde P_QOH_NDX dizinini kullandığınızı varsayın. Bu durumda, SELECT MIN(P_QOH) FROM PRODUCT gibi bir sorgu, PRODUCT tablosu için herhangi bir veri bloğuna erişmeye gerek kalmadan, yalnızca P_QOH_NDX dizinindeki ilk giriş okunarak çözümlenebilir. (Dizinin varsayılan olarak artan sırada olduğunu unutmayın).

Bölüm 11-3'te düşük seyrekliğe sahip sütunların dizin oluşturmak için iyi adaylar olmadığını öğrendiniz. Ancak, bazı durumlarda düşük seyreklikli bir sütunda bir dizin yararlı olabilir. Örneğin, EMPLOYEE tablosunda 122.483 satır olduğunu varsayın. Şirkette kaç kadın çalışanın olduğunu öğrenmek istiyorsanız, aşağıdaki gibi bir sorgu yazarsınız:

```
SELECT COUNT(EMP_SEX) FROM EMPLOYEE WHERE EMP_SEX 5 'F';
```

EMP_SEX sütunu için bir dizininiz yoksa, sorgunun tüm EMPLOYEE satırlarını okumak için tam bir tablo taraması gerçekleştirmesi gerekir ve her tam satır ihtiyacınız olmayan öznitelikleri içerir. Ancak, EMP_SEX üzerinde bir dizininiz varsa, çalışan verilerine hiç erişmeye gerek kalmadan yalnızca dizin verileri okunarak sorgu yanıtlanabilir.

11-4a Optimize Edici Seçimlerini Etkilemek için İpuçlarını Kullanma

Optimize edici genellikle çoğu durumda çok iyi performans gösterse de, bazı durumlarda en iyi yürütme planını seçmeyebilir. Optimize edicinin mevcut istatistiklere göre karar verdiğini unutmayın. İstatistikler eskiyse, en iyileştirici en iyi yürütme planını seçmede iyi bir iş çıkaramayabilir. Güncel istatistikler olsa bile, optimize edicinin seçimi en verimli seçim olmayabilir. Bazen son kullanıcı geçerli SQL deyimi için iyileştirici modunu değiştirmek ister. Bunu yapmak için ipuçlarını kullanmanız gerekir. **Optimize edici ipuçları**, SQL komut metninin içine gömülü olan **optimize** edici için özel talimatlardır. Her DBMS'nin kendine özgü bir iyileştiricisi olduğu gibi, her DBMS'nin de kendine özgü bir iyileştirici ipuçları kümesi ve bu ipuçları için sözdizimi vardır. Tablo 11.5, DBMS'lerde kullanılan birkaç yaygın iyileştirici ipucunu göstermektedir.

iyileştirici ipuçları
SQL komut metninin içine gömülü olan sorgu iyileştirici için özel talimatlar.

Tablo 11.5 Optimize Edici İpuçları

İpucu	Kullanım
ALL_ROWS (Oracle)	En iyileştiriciye genel yürütme süresini en aza indirmesini, yani sorgu sonuç kümesindeki tüm satırları döndürmek için gereken süreyi en aza indirmesini söyler. Bu ipucu genellikle toplu mod işlemleri için kullanılır. Örneğin: SEÇİNİZ /*1 ALL_ROWS */ * FROM ÜRÜN NEREDE P_QOH < 10;
HIZLI *n* (SQL Server)	En iyileştiriciye ilk *n* sayıda satırı işlemek için gereken süreyi en aza indirmesini, yani sorgu sonuç kümesinde yalnızca ilk satır kümesini döndürmek için gereken süreyi en aza indirmesini söyler. Bu ipucu genellikle etkileşimli mod işlemleri için kullanılır. Örneğin: SEÇİNİZ * FROM ÜRÜN NEREDE P_QOH< 10 SEÇENEĞİ (HIZLI 3);
USE INDEX(name) (MySQL)	İyileştiriciyi bu sorguyu işlemek için P_QOH_NDX dizinini kullanmaya zorlar. Örneğin: SEÇİNİZ * FROM ÜRÜN KULLANIM ENDEKSI (P_QOH_NDX) NEREDE P_QOH < 10

Artık DBMS'nin SQL sorgularını nasıl işlediğini bildiğinize göre, sorgu iyileştiricinin çalışmasını kolaylaştırmak için dikkatinizi bazı genel SQL kodlama önerilerine çevirebilirsiniz.

11-5 SQL Performans Ayarlama

SQL performans ayarlaması istemci perspektifinden değerlendirilir. Bu nedenle amaç, verimli SQL kodu yazmak için kullanılan bazı yaygın uygulamaları göstermektir. Birkaç uyarıda bulunmak yerinde olacaktır:

- Mevcut nesil ilişkisel 'lerin çoğu sunucu ucunda otomatik sorgu optimizasyonu gerçekleştirir.
- Çoğu SQL performans optimizasyon tekniği DBMS'ye özgüdür ve bu nedenle aynı DBMS'nin farklı sürümleri arasında bile nadiren taşınabilir. Bu davranışın nedenlerinden biri de veritabanı teknolojilerindeki sürekli ilerlemedir.

Bu, bir SQL sorgusunun nasıl yazıldığı konusunda endişelenmemeniz gerektiği anlamına mı gelir, çünkü DBMS her zaman onu optimize edecektir? Hayır, çünkü iyileştirme için önemli bir alan vardır. (DBMS, sorgu yürütmenin özel koşulları tarafından dikte edilen belirli tekniklere odaklanmak yerine *genel* optimizasyon tekniklerini kullanır). Kötü yazılmış bir SQL sorgusu, performans açısından veritabanı sistemini dize getirebilir *ve genellikle de* getirecektir. Mevcut veritabanı performans sorunlarının büyük çoğunluğu kötü yazılmış SQL kodlarıyla ilgilidir. Bu nedenle, bir DBMS genel optimizasyon hizmetleri sağlasa da, dikkatle yazılmış bir sorgu neredeyse her zaman kötü yazılmış bir sorgudan daha iyi performans gösterir.

SQL veri işleme deyimleri INSERT, UPDATE, DELETE ve SELECT birçok farklı komutu içermesine rağmen, bu bölümdeki önerilerin çoğu SELECT deyiminin kullanımı ve özellikle de dizinlerin kullanımı ve koşullu ifadelerin nasıl yazılacağı ile ilgilidir.

11-5a Endeks Seçiciliği

İndeksler SQL performans optimizasyonunda kullanılan en önemli tekniktir. Önemli olan, bir dizinin ne zaman kullanılacağını bilmektir. Genel bir kural olarak, indekslerin kullanılması muhtemeldir:

- İndekslenmiş bir sütun WHERE veya HAVING cümlesinin arama kriterlerinde tek başına göründüğünde
- Dizinlenmiş bir sütun bir GROUP BY veya ORDER BY cümlesinde tek başına göründüğünde
- Dizinlenmiş bir sütuna bir MAX veya MIN işlevi uygulandığında
- İndekslenen sütundaki veri seyrekliği yüksek olduğunda

İndeksler, belirli bir koşula bağlı olarak büyük bir tablodan küçük bir satır alt kümesi seçmek istediğinizde çok kullanışlıdır. Eğer *seçimde kullanılan* sütun için bir indeks mevcutsa, DBMS bunu kullanmayı seçebilir. Amaç, yüksek seçiciliğe sahip indeksler oluşturmaktır. **Dizin seçiciliği**, bir dizinin sorgu işlemede kullanılma olasılığının bir ölçüsüdür. İndeksleri oluşturmak ve kullanmak için bazı genel yönergeler aşağıda verilmiştir:

endeks seçiciliği
Bir dizinin sorgu işlemede kullanılma olasılığının ölçüsü.

- *Bir WHERE, HAVING, ORDER BY veya GROUP BY cümlesinde kullanılan her bir öznitelik için dizinler oluşturun*. *Arama koşullarında kullanılan* tüm tek özniteliklerde dizinler oluşturursanız, DBMS tabloya tam tablo taraması yerine bir dizin taraması kullanarak erişir. Örneğin, P_PRICE için bir dizininiz varsa, P_PRICE> 10.00 koşulu, tüm tablo satırlarını sırayla taramak ve her satır için P_PRICE'ı değerlendirmek yerine dizine erişilerek çözülebilir. Dizinler, CUSTOMER.CUS_ CODE 5 INVOICE.CUS_CODE birleştirme ifadelerinde de kullanılır.
- *Küçük tablolarda veya düşük seyrekliğe sahip tablolarda dizin kullanmayın*. Unutmayın, küçük tablolar ve düşük seyreklikli tablolar aynı şey değildir. Düşük seyrekliğe sahip bir tablodaki bir arama koşulu yine de tablo satırlarının yüksek bir yüzdesini döndürebilir, bu da dizin işlemini çok maliyetli hale getirir ve tam tablo taramasını uygulanabilir bir seçenek haline getirir. Aynı mantığı kullanarak, az sayıda satıra ve az sayıda *özniteliğe* sahip tablolar için dizinler oluşturmayın - *bir sütunda benzersiz değerlerin varlığından emin olmanız gerekmedikçe*.

- *İyileştiricinin birleştirme işlemlerinde dizinleri kullanabilmesi için birincil ve yabancı anahtarları bildirin*. Tüm doğal birleştirmeler ve eski tarz birleştirmeler, birincil anahtarları ve yabancı anahtarları bildirirseniz fayda sağlayacaktır çünkü iyileştirici, birleştirme zamanında mevcut dizinleri kullanacaktır. (Bir PK veya FK, birincil anahtar veya yabancı anahtar bildirimi, bildirilen sütun için otomatik olarak bir dizin oluşturacaktır). Ayrıca, aynı nedenden dolayı, birleşimleri SQL JOIN sözdizimini kullanarak yazmak daha iyidir. (Bkz. Bölüm 7, Yapısal Sorgu Diline (SQL) Giriş).
- *PK veya FK dışındaki birleştirme sütunlarında dizin bildirme*. Birincil ve yabancı anahtarlar dışındaki sütunlarda birleştirme işlemleri gerçekleştiriyorsanız, bu sütunlarda dizin bildirmeniz daha iyi olabilir.

Performansı artırmak için her zaman bir dizin kullanamazsınız. Örneğin, bir sonraki bölümde Tablo 11.6'da gösterilen verileri kullanarak, P_MIN için bir dizin oluşturmak P_QOH> P_MIN * 1.10 arama koşuluna yardımcı olmayacaktır. Bunun nedeni, bazı DBMS'lerde, *tablo özniteliklerinde işlevler kullandığınızda dizinlerin göz ardı edilmesidir*. Ancak Oracle, SQL Server ve DB2 gibi büyük veritabanları artık fonksiyon tabanlı indeksleri desteklemektedir. **İşlev tabanlı** bir **dizin,** belirli bir SQL işlevini veya ifadesini temel alan bir dizindir. Örneğin, YEAR(INV_DATE) üzerinde bir dizin oluşturabilirsiniz. Fonksiyon tabanlı indeksler özellikle türetilmiş öznitelikle çalışırken kullanışlıdır. Örneğin, EMP_SALARY 1 EMP_COMMISSION üzerinde bir dizin oluşturabilirsiniz.

işlev tabanlı dizin Belirli bir SQL işlevini veya ifadesini temel alan bir dizin türü.

Kaç tane dizin oluşturmalısınız? Bir tablodaki her sütun için bir dizin oluşturmamanız gerektiğini tekrarlamakta fayda var. Çok fazla indeks, özellikle tablo binlerce satır içeriyorsa INSERT, UPDATE ve DELETE işlemlerini yavaşlatacaktır. Ayrıca, bazı sorgu iyileştiricileri, sorgunuz birçok farklı dizinli sütundaki koşulları kullansa bile, bir sorgu için yönlendirici dizin olarak yalnızca bir dizin seçecektir. İyileştirici hangi dizini kullanır? Maliyet tabanlı iyileştirici kullanıyorsanız, tablolara yeni satırlar eklendikçe veya tablolardan silindikçe yanıt zamanla değişecektir. Her durumda, tüm arama sütunlarında dizinler oluşturmalı ve ardından iyileştiricinin seçim yapmasına izin vermelisiniz. Dizin kullanımını sürekli olarak değerlendirmek önemlidir - izleyin, test edin, değerlendirin ve performans yeterli değilse iyileştirin.

11-5 b Koşullu İfadeler

Koşullu bir ifade normalde bir SQL ifadesinin WHERE veya HAVING cümleleri içine yerleştirilir. Koşullu ölçütler olarak da bilinen koşullu ifade, bir sorgunun çıktısını yalnızca koşullu ölçütlerle eşleşen satırlarla kısıtlar. Genel olarak, koşullu ölçütler Tablo 11.6'da gösterilen biçime sahiptir.

Tablo 11.6 Koşullu Kriterler

OPERAND1	KOŞULLU OPERATÖR	OPERAND2
P_FİYAT	>	10.00
V_STATE	=	FL
V_CONTACT	LIKE	Smith%
P_QOH	>	P_MIN * 1.10

Tablo 11.6'da, bir operandın şu şekilde dikkat edin:

- P_PRICE veya V_STATE gibi basit bir sütun adı
- Bir değişmez veya 10.00 değeri ya da 'FL' metni gibi bir sabit
- P_MIN * 1.10 gibi bir ifade

Aşağıda bahsedilen sorgu optimizasyon tekniklerinin çoğu optimize edicinin işini kolaylaştırmak için tasarlanmıştır. SQL kodunda verimli koşullu ifadeler yazmak için aşağıdaki yaygın uygulamalar kullanılır.

- Koşullu bir ifadede işlenen olarak basit sütunlar veya değişmezler kullanın; mümkün olduğunca işlevlerle koşullu ifadeler kullanmaktan kaçının. Tek bir sütunun içeriğini bir değişmezle karşılaştırmak, ifadelerle karşılaştırmaktan daha hızlıdır. Örneğin, P_PRICE > 10.00 ifadesi P_QOH > P_MIN * 1.10 ifadesinden daha hızlıdır çünkü DBMS'nin önce P_MIN * 1.10 ifadesini değerlendirmesi gerekir. İfadelerde fonksiyonların kullanılması da toplam sorgu yürütme süresine katkıda bulunur. Örneğin, koşulunuz UPPER (V_NAME) 5 'JIM' ise, V_NAME sütunundaki tüm adlar uygun büyük harflerle saklanıyorsa V_NAME 5 'Jim' kullanmayı deneyin.
- *Sayısal alan karşılaştırmaları karakter, tarih ve NULL karşılaştırmalarından daha hızlıdır*. Arama koşullarında, sayısal bir özniteliğin sayısal bir değişmezle karşılaştırılması, bir karakter özniteliğinin bir karakter değişmeziyle karşılaştırılmasından daha hızlıdır. Genel olarak, CPU sayısal karşılaştırmaları (tamsayı ve ondalık) karakter ve tarih karşılaştırmalarından daha hızlı işler. Dizinler null değerlere referansları saklamadığından, NULL koşulları ek işlem gerektirir ve bu nedenle tüm koşullu işlenenler arasında en yavaş olma eğilimindedir.
- *Eşitlik karşılaştırmaları genellikle eşitsizlik karşılaştırmalarından daha hızlıdır*. Örneğin, P_PRICE 5 10.00 daha hızlı işlenir çünkü DBMS sütundaki dizini kullanarak doğrudan bir arama yapabilir. Tam eşleşme yoksa, koşul yanlış olarak değerlendirilir. Bununla birlikte, bir eşitsizlik sembolü (>, >5, <, <5) kullanırsanız, DBMS isteği tamamlamak için ek işlem gerçekleştirmelidir, çünkü dizinde neredeyse her zaman "eşit" değerlerden daha fazla "daha büyük" veya "daha küçük" değer olacaktır. NULL haricinde, tüm karşılaştırma operatörleri arasında en yavaş olanı, V_CONTACT LIKE "%glo%" gibi joker karakter sembolleri içeren LIKE'dır. Ayrıca, "eşit değil" sembolünün (< >) kullanılması, özellikle verilerin seyrekliği yüksek olduğunda, yani eşit değerlerden çok daha fazla farklı değer olduğunda daha yavaş aramalar sağlar.
- *Mümkün olduğunda, koşullu ifadeleri değişmezleri kullanacak şekilde dönüştürün*. Örneğin, koşulunuz P_PRICE 2 10 5 7 ise, bunu P_PRICE 5 17 olarak değiştirin. Ayrıca, aşağıdaki gibi bileşik bir koşulunuz varsa:

  P_QOH< P_MIN VE P_MIN 5 P_REORDER VE P_QOH 5 10

  Okumak için değiştir:

  P_QOH 5 10 VE P_MIN 5 P_REORDER VE P_MIN> 10
- *Birden fazla koşullu ifade kullanırken, önce eşitlik koşullarını yazın*. Bunun önceki örnekte yapıldığına dikkat edin. Eşitlik koşullarının eşitsizlik koşullarına göre daha hızlı işlendiğini unutmayın. Çoğu RDBMS bunu sizin için otomatik olarak yapacak olsa da, bu ayrıntıya dikkat etmek sorgu iyileştiricinin yükünü hafifletir. Optimize edici, sizin zaten yapmış olduğunuz şeyi yapmak zorunda kalmayacaktır.
- *Birden fazla AND koşulu kullanıyorsanız, önce yanlış olma olasılığı en yüksek olan koşulu yazın*. Bu tekniği kullanırsanız, DBMS yanlış olarak değerlendirilen bir koşullu ifade bulur bulmaz diğer koşulları değerlendirmeyi durduracaktır. Birden fazla AND koşulunun doğru bulunması için tüm koşulların doğru olarak değerlendirilmesi gerektiğini unutmayın. Koşullardan biri yanlış olarak değerlendirilirse, tüm koşul kümesi olarak değerlendirilecektir. Bu tekniği kullanırsanız, DBMS gereksiz yere ek koşulları değerlendirerek zaman kaybetmeyecektir. Doğal olarak, bu tekniğin kullanımı veri kümesinin seyrekliği hakkında bilgi sahibi olmayı gerektirir. Örneğin, aşağıdaki koşul listesine bakın:

  P_PRICE> 10 VE V_STATE 5 'FL'

  Florida'da yalnızca birkaç satıcının bulunduğunu biliyorsanız, bu durumu şu şekilde yeniden yazabilirsiniz:

  V_STATE 5 'FL' VE P_PRICE> 10
- *Birden fazla VEYA koşulu kullanırken, doğru olma olasılığı en yüksek olan koşulu ilk sıraya koyun*. Bunu yaparak, VTYS bir doğru koşul bulur bulmaz kalan koşulları değerlendirmeyi durduracaktır.

doğru olarak değerlendirilen koşullu ifade. Birden fazla VEYA koşulunun doğru olarak değerlendirilmesi için koşullardan yalnızca birinin doğru olarak değerlendirilmesi gerektiğini unutmayın.

- *Mümkün olduğunda, NOT mantıksal işlecinin kullanımından kaçınmaya çalışın*. NOT mantıksal işleci içeren bir SQL ifadesini eşdeğer bir ifadeye dönüştürmek en iyisidir. Örneğin:

  NOT (P_PRICE> 10.00), P_PRICE <5 10.00 olarak yazılabilir.

  Ayrıca, NOT (EMP_SEX 5 'M') EMP_SEX 5 'F' olarak yazılabilir.

Not

Oracle sorguları burada açıklandığı gibi değerlendirmez. Bunun yerine, Oracle koşulları sondan başa doğru değerlendirir.

11-6 Sorgu Formülasyonu

Sorgular genellikle soruları yanıtlamak için yazılır. Örneğin, bir son kullanıcı size örnek bir çıktı verir ve bu çıktı biçimini eşleştirmenizi isterse, ilgili SQL'i yazmanız gerekir. İşi tamamlamak için, istenen çıktıyı oluşturmak için hangi sütunların, tabloların ve hesaplamaların gerekli olduğunu dikkatlice değerlendirmelisiniz. Bunu yapmak için, SQL kodunuzun odak noktası olacak veritabanı ortamını ve veritabanını iyi anlamanız gerekir.

Bu bölüm SELECT sorgularına odaklanmaktadır çünkü çoğu uygulamada karşılaşacağınız sorgular bunlardır. Bir sorgu formüle etmek için normalde şu adımları izlersiniz:

1. *Hangi sütunların ve hesaplamaların gerekli olduğunu belirleyin*. İlk adımda, hangi veri değerlerini döndürmek istediğinizi net bir şekilde belirlemeniz gerekir. Yalnızca adları ve adresleri mi döndürmek istiyorsunuz yoksa bazı hesaplamaları da dahil etmek istiyor musunuz? SELECT deyimindeki tüm sütunların tek bir değer döndürmesi gerektiğini unutmayın.
   a. Basit ifadelere mi ihtiyacınız var? Örneğin, toplam envanter maliyetini oluşturmak için fiyatı eldeki miktarla çarpmanız mı gerekiyor? DATE(), SYSDATE() veya ROUND() gibi bazı tek özellikli fonksiyonlara ihtiyacınız olabilir.
   b. Toplama fonksiyonlarına ihtiyacınız var mı? Ürüne göre toplam satışları hesaplamanız gerekiyorsa GROUP BY cümlesi kullanmalısınız. Bazı durumlarda, bir alt sorgu kullanmanız gerekebilir.
   c. Çıktınız için gereken ham verilerin ayrıntı düzeyini belirleyin. Bazen, herhangi bir tabloda kolayca bulunmayan verileri özetlemeniz gerekebilir. Bu gibi durumlarda, sorguyu birden fazla alt sorguya bölmeyi ve bu alt sorguları görünümler olarak depolamayı düşünebilirsiniz. Ardından, bu görünümleri birleştiren ve nihai çıktıyı oluşturan üst düzey bir sorgu oluşturabilirsiniz.
2. *Kaynak tabloları belirleyin*. Hangi sütunların gerekli olduğunu öğrendikten sonra, sorguda kullanılan kaynak tabloları belirleyebilirsiniz. Bazı nitelikler birden fazla tabloda görünür. Bu durumlarda, birleştirme işlemlerinin sayısını en aza indirmek için sorgunuzda en az sayıda tablo kullanmaya çalışın.
3. *Tabloların nasıl birleştirileceğini belirleyin*. Sorgu deyiminizde hangi tablolara ihtiyacınız olduğunu öğrendikten sonra, tabloları nasıl birleştireceğinizi doğru bir şekilde belirlemelisiniz. Çoğu durumda, bir tür iç birleştirme kullanacaksınız, ancak bazı durumlarda bir dış birleştirme kullanmanız gerekebilir.
4. *Hangi seçim kriterlerinin gerekli olduğunu belirleyin*. Çoğu sorgu bir tür seçim ölçütü içerir. Bu durumda, ölçütlerinizde hangi işlenenlerin ve işleçlerin gerekli olduğunu belirlemeniz gerekir. Karşılaştırma verilerin veri türünün ve ayrıntı düzeyinin doğru olduğundan emin olun.
   a. *Basit karşılaştırma*. Çoğu durumda, tek değerleri karşılaştıracaksınız - örneğin, P_PRICE > 10.

b. *Tek değerden birden çok değere*. Tek bir değeri birden fazla değerle karşılaştırıyorsanız, bir IN karşılaştırma operatörü kullanmanız gerekebilir; örneğin, V_STATE IN ('FL', 'TN', 'GA').

c. *İç içe karşılaştırmalar*. Diğer durumlarda, alt sorguları içeren bazı iç içe seçim kriterlerine sahip olmanız gerekebilir; örneğin, P_PRICE >5 (SELECT AVG (P_PRICE) FROM PRODUCT).

d. *Gruplandırılmış veri seçimi*. Diğer durumlarda, seçim kriterleri ham veriler için değil, toplu veriler için geçerli olabilir. Bu durumlarda HAVING cümlesini kullanmanız gerekir.

5. *Çıktının görüntüleneceği sırayı belirleyin*. Son olarak, gerekli çıktı bir veya daha fazla sütuna göre sıralanmış olabilirBu durumlarda ORDER BY cümlesini kullanmanız gerekir. ORDER BY cümlesinin DBMS için en yoğun kaynak gerektiren işlemlerden biri olduğunu unutmayın.

11-7 DBMS Performans Ayarlama

DBMS performans ayarlaması, birincil bellekteki DBMS süreçlerinin yönetilmesi (önbellekleme amacıyla bellek ayrılması) ve fiziksel depolamadaki yapıların yönetilmesi (veri dosyaları için alan ayrılması) gibi global görevleri içerir.

VTYS'nin performansına ince ayar yapmak, önceki bölümde incelenen çeşitli uygulamaların uygulanmasını da içerir. Örneğin, DBA, sorguların beklendiği gibi performans göstermesini sağlamak için geliştiricilerle birlikte çalışmalıdır - sorgu yanıt süresini hızlandırmak için dizinler oluşturmalı ve maliyet tabanlı optimize edicilerin ihtiyaç duyduğu veritabanı istatistiklerini oluşturmalıdır.

Sunucu ucunda DBMS performans ayarlaması, kullanılan parametrelerin ayarlanmasına odaklanır:

- *Veri önbelleği*. Veri önbelleği boyutu, önbellekten mümkün olduğunca çok sayıda veri isteğinin karşılanmasına izin verecek kadar büyük ayarlanmalıdır. Her DBMS'nin veri önbelleğinin boyutunu kontrol eden ayarları vardır; bazı DBMS'ler yeniden başlatma gerektirebilir. Bu önbellek tüm veri tabanı kullanıcıları arasında paylaşılır. Birincil bellek kaynaklarının çoğu veri önbelleğine tahsis edilecektir.
- *SQL önbelleği*. SQL önbelleği en son çalıştırılan SQL ifadelerini saklar (SQL ifadeleri iyileştirici tarafından ayrıştırıldıktan sonra). Genel olarak, bir veritabanına erişen birden fazla kullanıcısı olan bir uygulamanız varsa, *aynı* sorgu muhtemelen birçok farklı kullanıcı tarafından gönderilecektir. Bu gibi durumlarda, VTYS sorguyu yalnızca bir kez ayrıştıracak ve aynı erişim planını kullanarak birçok kez çalıştıracaktır. Bu şekilde, aynı sorgu için ikinci ve sonraki SQL istekleri, ayrıştırma aşamasını atlayarak SQL önbelleğinden sunulur.
- *Sıralama önbelleği*. Sıralama önbelleği, ORDER BY veya GROUP BY işlemlerinin yanı sıra dizin oluşturma işlevleri için geçici bir depolama alanı olarak kullanılır.
- *Optimize edici modu*. Çoğu DBMS iki optimizasyon modundan birinde çalışır: maliyet tabanlı veya kural tabanlı. Diğerleri, veritabanı istatistiklerinin mevcut olup olmadığına bağlı olarak optimizasyon modunu otomatik olarak belirler. Örneğin, maliyet tabanlı optimize edici tarafından kullanılan veritabanı istatistiklerinin oluşturulmasından DBA sorumludur. İstatistikler mevcut değilse, DBMS kural tabanlı bir iyileştirici kullanır.

Performans açısından bakıldığında, maliyetli disk erişimini en aza indirmek için tüm veritabanının birincil bellekte depolanması en uygunudur. Bu nedenle birçok veritabanı satıcısı ana ürünleri için bellek içi veritabanı seçenekleri sunmaktadır. **Bellek içi veritabanı** sistemleri, veritabanının büyük bir bölümünü (tamamını olmasa da) ikincil (disk) depolama yerine birincil (RAM) depolamada saklamak için tercih edilir. Bu sistemler, modern veritabanı uygulamalarının (İş Analitiği ve Büyük Veri gibi) artan performans talepleri, azalan maliyetler ve bileşenlerin (flash bellek ve katı hal sürücüleri gibi) teknolojik gelişmeleri nedeniyle popüler hale gelmektedir.

bellek içi veritabanı
Veritabanının büyük bir (tamamını olmasa da) ikincil (disk) depolama yerine birincil (RAM) depolamada saklamak için optimize edilmiş bir veritabanı.

özellikle kötü tasarlanmış veritabanları veya kötü yazılmış SQL ifadeleri ile karşılaşıldığında sorgu optimizasyonu ve performans ayarlama kurallarına tabidir.

Bellek içi veritabanları belirli pazarlarda kendine bir yer edinmiş olsa da, çoğu veritabanı uygulaması hala disk sürücülerinde depolanan verilere dayanmaktadır. Bu nedenle veri dosyalarının fiziksel depolama ayrıntılarını yönetmek, VTYS performans ayarlamasında önemli bir rol oynar. Veritabanlarının fiziksel depolanması için aşağıdaki genel önerilere dikkat edin:

G/Ç hızlandırıcı
Giriş/çıkış işlemlerinde verimi artırmak için kullanılan bir cihaz.

RAID
Redundant Array of Independent Disks'in kısaltmasıdır. RAID sistemleri, birkaç ayrı diskten sanal diskler (depolama birimleri) oluşturmak için birden fazla kullanır. RAID sistemleri performans iyileştirmesi, hata toleransı ve bu ikisi arasında bir denge sağlar.

- **I/O hızlandırıcıları** kullanın. Bu tür bir cihaz, veritabanını depolamak için flaş katı hal sürücüleri (SSD'ler) kullanır. Bir SSD'nin hareketli parçası yoktur ve bu nedenle G/Ç işlemlerini geleneksel döner disk sürücülerinden daha yüksek bir hızda gerçekleştirir. G/Ç hızlandırıcıları yüksek aktarım performansı oranları sunar ve tipik depolama sürücülerinin neden olduğu çekişmeyi azaltır.
- Hem performans artışı hem de hata toleransı sağlamak ve bunlar arasında bir denge kurmak için **RAID** (Redundant Array of Independent Disks) kullanın. Hata toleransı, arıza durumunda verilerin yeniden yapılandırılabileceği ve geri alınabileceği anlamına gelir. RAID sistemleri, birkaç ayrı diskten oluşan sanal diskler (depolama birimleri) oluşturmak için birden fazla disk kullanır. Tablo 11.7'de en yaygın RAID yapılandırmaları açıklanmaktadır.

Tablo 11.7 Yaygın RAID Seviyeleri

RAID Seviyesi	Açıklama
0	Veri blokları ayrı sürücüler üzerine yayılır. *Şeritli dizi* olarak da bilinir. Daha yüksek performans sağlar ancak hata toleransı yoktur. En az iki sürücü gerektirir.
1	Aynı veri blokları ayrı sürücülere yazılır (çoğaltılır). *Yansıtma* veya *dupleksleme* olarak da adlandırılır. Veri yedekliliği yoluyla daha yüksek okuma performansı ve hata toleransı sağlar. En az iki sürücü gerektirir.
3	Veriler ayrı sürücüler arasında şeritlenir ve eşlik verileri hesaplanır ve özel bir sürücüde saklanır. (Eşlik verileri, bozuk veya eksik verilerin yeniden oluşturulmasına izin veren özel olarak oluşturulmuş verilerdir). Eşlik verileri aracılığıyla iyi okuma performansı ve hata toleransı sağlar. En az üç sürücü gerektirir.
5	Veriler ve eşlik verileri ayrı sürücülerde şeritlenir. Eşlik verileri aracılığıyla iyi okuma performansı ve hata toleransı sağlar. En az üç sürücü gerektirir.
110	Veri blokları ayrı sürücülere dağıtılır ve yansıtılır (çoğaltılır). Bu düzenleme hem hız hem de hata toleransı sağlar. Bu, çoğu veritabanı kurulumu için önerilen RAID yapılandırmasıdır (maliyet bir sorun değilse).

- Disk çekişmesini en aza indirin. Sabit disk döngülerini en aza indirmek için bağımsız iğlere (dönen diskler) sahip birden fazla, bağımsız depolama birimi kullanın. Bir veritabanının, her biri belirli bir işleve sahip birçok tablo alanından oluştuğunu unutmayın. Buna karşılık, her tablo alanı, verilerin gerçekte depolandığı birkaç veri dosyasından oluşur. Bir veritabanı en azından aşağıdaki tablo alanlarına sahip olmalıdır:
  - *Sistem tablo alanı*. Bu, veri sözlüğü tablolarını depolamak için kullanılır. En sık erişilen tablo alanıdır ve kendi biriminde depolanmalıdır.
  - *Kullanıcı veri tablosu alanı*. Bu, son kullanıcı verilerini depolamak için kullanılır. Performans ve kullanılabilirliği dengelemek için gerektiği kadar kullanıcı veri tablosu alanı ve veri dosyası oluşturmalısınız. Örneğin, her uygulama ve her farklı kullanıcı grubu için farklı bir kullanıcı veri tablosu alanı oluşturabilir ve atayabilirsiniz, ancak bu her kullanıcı için gerekli değildir.
  - *Dizin tablosu alanı*. Bu, dizinleri depolamak için kullanılır. Her uygulama ve her kullanıcı grubu için farklı bir dizin tablosu alanı oluşturabilir ve atayabilirsiniz. Dizin tablosu alanı veri dosyaları, kullanıcı veri dosyalarından veya sistem veri dosyalarından ayrı bir depolama biriminde saklanmalıdır.
  - *Geçici tablo alanı*. Bu alan birleştirme, sıralama veya toplama işlemleri için geçici bir depolama alanı olarak kullanılır. Her uygulama ve her kullanıcı grubu için farklı bir geçici tablo alanı oluşturabilir ve atayabilirsiniz.
  - *Geri alma segmenti tablo alanı*. Bu, işlem kurtarma amaçları için kullanılır.
- Yüksek kullanımlı tabloları kendi tablo alanlarına yerleştirin, böylece veritabanı diğer tablolarla çakışmayı en aza indirir.

- Dizinler, sistem ve yüksek kullanımlı tablolar için ayrı depolama birimlerinde ayrı veri dosyaları atayın. Bu, dizin işlemlerinin son kullanıcı verileri veya veri sözlüğü tablo erişimi işlemleriyle çakışmamasını sağlar. Bu yaklaşımın bir diğer avantajı da farklı birimlerde farklı disk bloğu boyutları kullanabilmenizdir. Örneğin, veri birimi 16 K blok boyutu kullanırken, dizin birimi 8 K blok boyutu kullanabilir. Dizin kayıt boyutunun genellikle daha küçük olduğunu ve blok boyutunu değiştirerek çekişmeyi azaltacağınızı ve G/Ç işlemlerini en aza indireceğinizi unutmayın. Bu çok önemlidir; birçok veritabanı yöneticisi bir çekişme kaynağı olarak dizinleri göz ardı eder. Ayrı depolama birimleri ve farklı blok boyutları kullanarak, veriler ve dizinler üzerindeki G/Ç işlemleri eşzamansız olarak (farklı zamanlarda) gerçekleşir; daha da önemlisi, sayfa kilitleri daha az kaydı kilitleme eğiliminde olduğundan, yazma işlemlerinin okuma işlemlerini engelleme olasılığı azalır.
- Veritabanında mevcut olan çeşitli tablo depolama organizasyonlarından yararlanın. Örneğin, Oracle'da dizinle düzenlenmiş tabloların (IOT) kullanımını düşünün; SQL Server'da kümelenmiş dizin tablolarını düşünün. **İndeksle organize edilmiş tablo** (veya **kümelenmiş indeks tablosu)**, son kullanıcı verilerini ve indeks verilerini daimi depolama alanında ardışık konumlarda depolayan bir tablodur. Bu tür bir depolama organizasyonu, belirli bir dizin sırası üzerinden yaygın olarak erişilen tablolara performans avantajı sağlar çünkü dizin, veri satırlarının yanı sıra dizin anahtarını da içerir. Bu nedenle, DBMS daha az I/O işlemi gerçekleştirme eğilimindedir.
- Tabloları kullanıma göre bölümleme. Bazı RDBMS'ler tabloların niteliklere göre yatay olarak bölümlenmesini destekler. (Bkz. Bölüm 12, Dağıtılmış Veritabanı Yönetim Sistemleri.) Bu sayede tek bir SQL isteği birden fazla veri işlemcisi tarafından işlenebilir. Tablo bölümlerini en çok kullanıldıkları yere en yakın yere yerleştirin.
- Uygun olan yerlerde denormalize edilmiş tablolar kullanın. Başka bir deyişle, bir tabloyu daha yüksek bir normal formdan daha düşük bir normal forma (genellikle üçüncü normal formdan ikinci normal forma) alarak performansı artırabilirsiniz. Bu teknik veri yinelemesini artırır, ancak birleştirme işlemlerini en aza indirir. (Denormalizasyon, Bölüm 6, Veritabanı Tablolarının Normalizasyonu'nda ele alınmıştır).
- Hesaplanmış ve toplu öznitelikleri tablolarda depolayın. Kısacası, tablolarınızda türetilmiş öznitelikler kullanın. Örneğin, INVOICE tablosunda fatura alt toplamını, vergi tutarını ve toplamı ekleyebilirsiniz. Türetilmiş özniteliklerin kullanılması, özellikle toplu sorguların yürütülmesi sırasında sorgulardaki ve birleştirme işlemlerindeki hesaplamaları en aza indirir.

Dizinle düzenlenmiş tablo Bir DBMS'de, son kullanıcı verilerini ve dizin verilerini kalıcı depolama alanındaki ardışık konumlarda depolayan bir tür tablo depolama organizasyonu. *Küme-indeksli tablo* olarak da bilinir.

kümelenmiş dizin tablosu
İndeks organize tablosuna bakın.

11-8 Sorgu Optimizasyonu Örneği

Artık sorgu optimizasyonunun temelini öğrendiğinize göre, yeni bilgilerinizi test etmeye hazırsınız. Basit bir örnek, sorgu iyileştiricinin nasıl çalıştığını ve çalışmasına nasıl yardımcı olabileceğinizi göstermektedir. Örnek, önceki bölümlerde kullandığınız tablolara benzeyen QOVENDOR ve QOPRODUCT tablolarını temel almaktadır. Ancak, önceki tabloların üzerine yazmadığınızdan emin olmak için tablo adı için QO öneki kullanılır.

Bu sorgu optimizasyon örneğini gerçekleştirmek için Oracle `SQL*Plus` ara yüzünü kullanacaksınız. Sorgu optimizasyonunu test etmeye başlamadan önce, aşağıdaki adımlarda açıklandığı gibi bazı ön çalışmalar yapılmalıdır:

1. Eğitmeniniz tarafından sağlanan kullanıcı adı ve parolayı kullanarak Oracle SQL*Plus'ta oturum açın.
2. QRYOPTDATA.SQL kod dosyasını (www. .cengage.com adresinde bulunabilir) kullanarak yeni bir tablo kümesi oluşturun Bu adım, Oracle'ın yeni bir tablo kümesine sahip olması ve yeni tabloların istatistik içermemesi için gereklidir. SQL> komut istemine şunu yazın:

   @path\QRYOPTDATA.SQL

   Burada *yol,* dosyanın bilgisayarınızdaki konumudur.

3. Belirli bir sorgu için erişim planı bilgilerini saklamak üzere Oracle tarafından kullanılan özel bir tablo olan PLAN_TABLE'ı oluşturun. Son kullanıcılar daha sonra Oracle'ın sorguyu nasıl yürüteceğini görmek için PLAN_TABLE'ı sorgulayabilir. PLAN_TABLE'ı oluşturmak için, RDBMS'deki UTLXPLAN. SQL komut dosyasını Oracle RDBMS kurulumunuzun RDBMS\ADMIN klasöründe çalıştırın. UTLXPLAN.SQL komut dosyası www.cengage.com adresinde de mevcuttur. SQL komut istemine şunu yazın:

   @path\UTLXPLAN.SQL

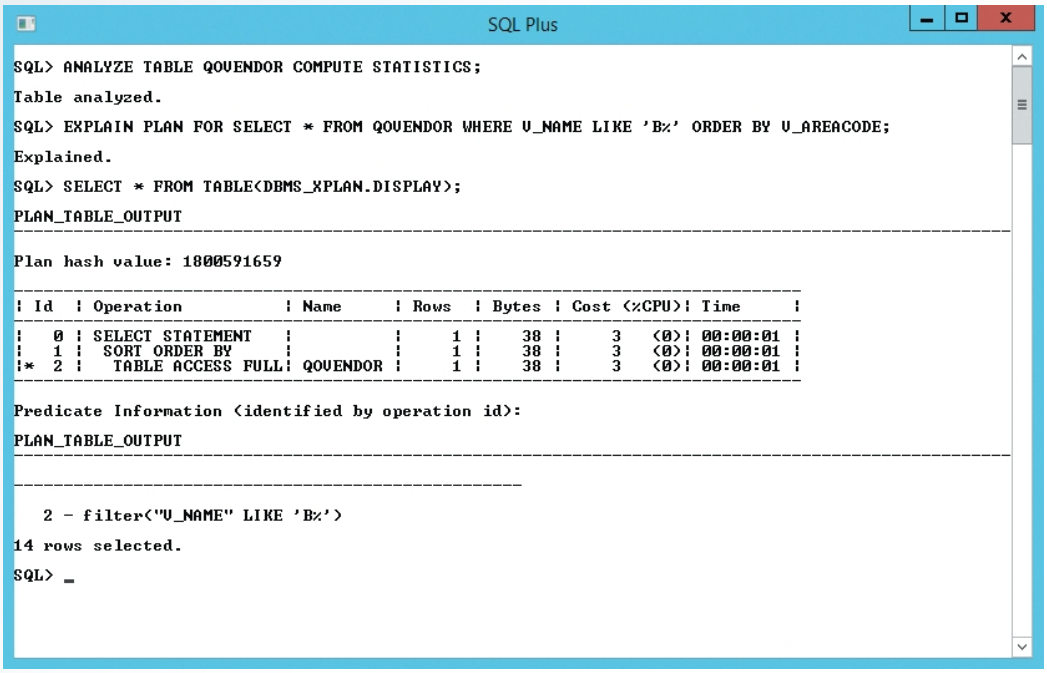
Bir SQL sorgusunun yürütme planını PLAN_TABLE'da saklamak için EXPLAIN PLAN komutunu kullanırsınız. Ardından, SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY) komutunu kullanarak belirli bir SQL deyimi için erişim planını görüntüleyebilirsiniz.

Not

Oracle, MySQL ve SQL Server'ın tümü varsayılan olarak maliyet tabanlı optimizasyona sahiptir. Oracle'da, tablo istatistikleri mevcut değilse, DBMS kural tabanlı bir iyileştiriciye geri dönecektir.

DBMS tarafından sorgunuzu yürütmek için kullanılan erişim planını görmek için, Şekil 11.5'te gösterildiği gibi EXPLAIN PLAN ve SELECT deyimlerini kullanın. İlk SQL deyiminin QOVENDOR tablosu için istatistikler oluşturduğuna dikkat edin. Ayrıca, Şekil 11.5'teki ilk erişim planı QOVENDOR tablosunda tam bir tablo taraması kullanır ve planın maliyeti 3'tür.

Şekil 11.5 İlk Açıklama Planı

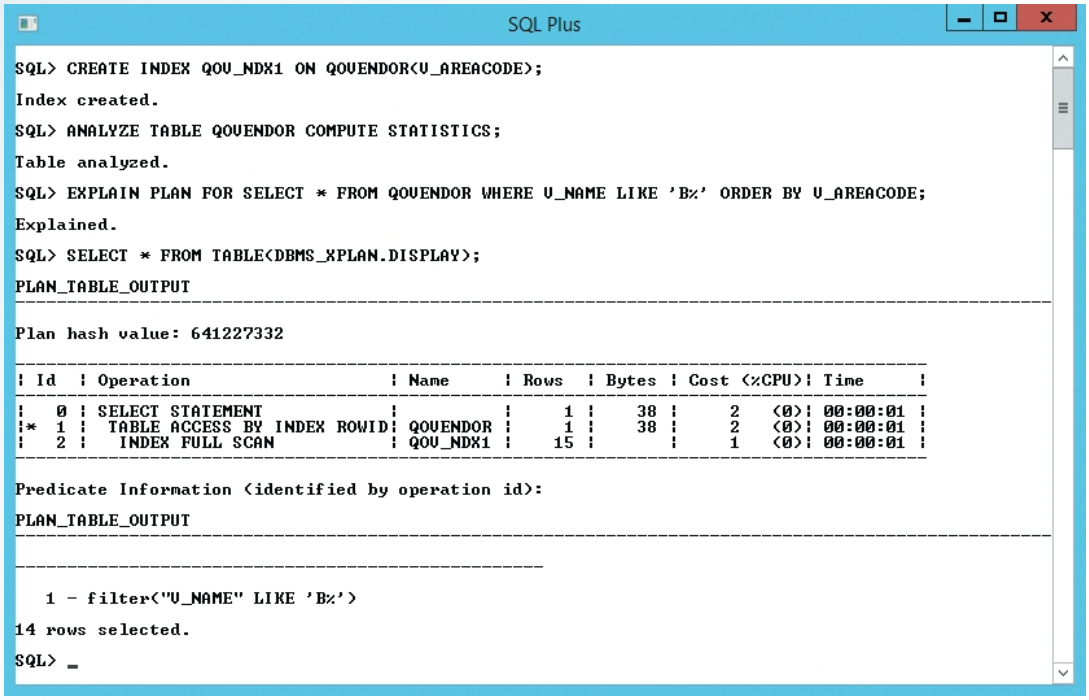


Şimdi V_AREACODE üzerinde bir dizin oluşturun (ORDER BY cümlesinde V_AREACODE kullanıldığına dikkat edin) ve bunun maliyet tabanlı iyileştirici tarafından oluşturulan erişim planını nasıl etkilediğini görün. Sonuçlar Şekil 11.6'da gösterilmektedir.

Şekil 11.6'da, yeni erişim planının sorguyu yürütme maliyetini yüzde 30 azalttığına dikkat edin! Ayrıca bu yeni planın QOV_NDX1 dizinini taradığına ve dizin satır kimliğini kullanarak QOVENDOR satırlarına eriştiğine dikkat edin. (Satır kimliğine göre erişimin en hızlı erişim yöntemlerinden biri olduğunu unutmayın.) Bu durumda, QOV_NDX1 dizininin oluşturulmasının genel sorgu optimizasyon sonuçları üzerinde olumlu bir etkisi olmuştur.

Diğer zamanlarda, dizinler sorgu optimizasyonuna yardımcı olmak zorunda değildir; örneğin küçük tablolarda dizinleriniz olduğunda veya sorgu tablo satırlarının büyük bir yüzdesine eriştiğinde

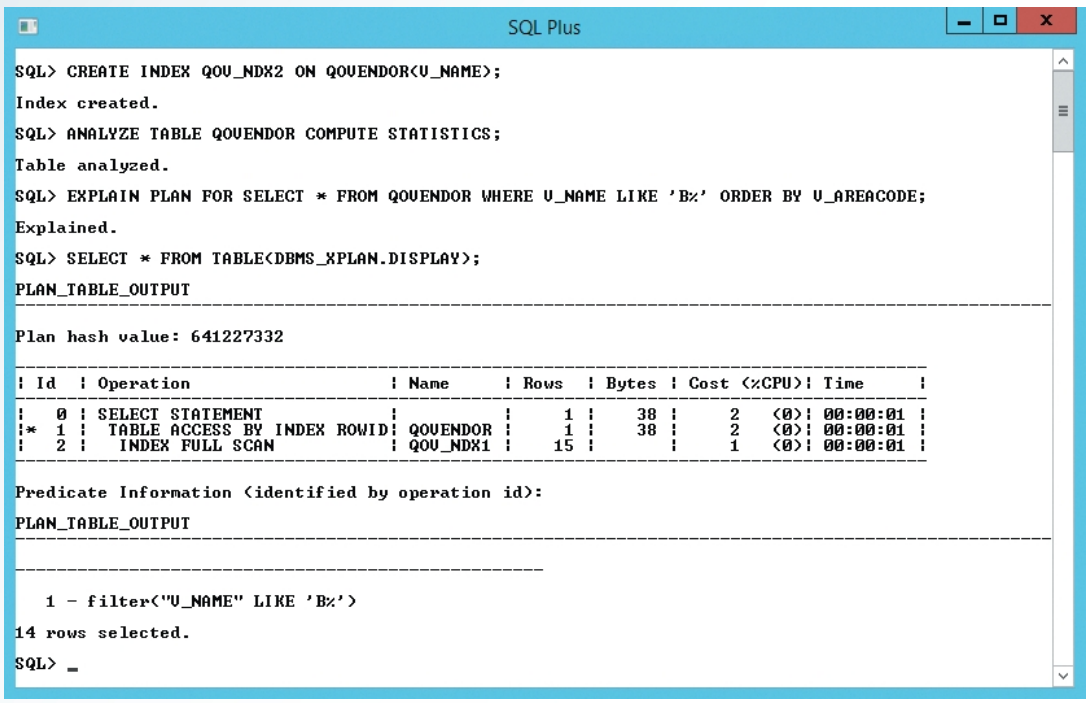
Şekil 11.6 V_AREACODE üzerinde Dizin Sonrası Planı Açıkla



Her neyse. V_NAME üzerinde bir dizin oluşturduğunuzda ne olduğuna dikkat edin. Yeni erişim planı Şekil 11.7'de gösterilmektedir. (V_NAME öğesinin WHERE cümlesinde koşullu ifade işleneni olarak kullanıldığına dikkat edin).

Şekil 11.7'de görebileceğiniz gibi, ikinci dizinin oluşturulması sorgu optimizasyonuna yardımcı olmamıştır. Ancak, bazı durumlarda bir dizin optimize edici tarafından kullanılabilir, ancak sorgunun yazılma şekli nedeniyle çalıştırılmaz. Örneğin, Şekil 11.8 V_NAME sütununu kullanan farklı bir sorgunun erişim planını göstermektedir.

Şekil 11.7 V_NAME Dizininden Sonra Planı Açıkla



Şekil 11.8 V_NAME üzerinde Dizin Kullanarak Erişim Planı

SQL Plus

```
SQL> EXPLAIN PLAN FOR SELECT V_NAME, P_CODE FROM QOVENDOR V, QOPRODUCT P
  2                  WHERE  V.V_CODE = P.V_CODE AND V_NAME = 'ORDVA, Inc.';

Explained.

SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------

Plan hash value: 4146133223

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |            |     2 |    62 |     5   (0)| 00:00:01 |
|*  1 |  HASH JOIN                            |            |     2 |    62 |     5   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED | QOVENDOR   |     1 |    17 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                   | QOV_NDX2   |     1 |       |     1   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL                   | QOPRODUCT  |    14 |   196 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------


Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("V"."V_CODE"="P"."V_CODE")
   3 - access("V_NAME"='ORDVA, Inc.')
   4 - filter("P"."V_CODE" IS NOT NULL)

18 rows selected.

SQL> _
```

Şekil 11.8'de, bu yeni sorgu için erişim planının V_NAME sütununda QOV_NDX2 dizinini kullandığına dikkat edin. Aynı sorguyu V_NAME üzerinde UPPER fonksiyonunu kullanarak yazsaydınız ne olurdu? Sonuçlar Şekil 11.9'da gösterilmektedir.

Şekil 11.9 Dizinlenmiş Sütunlarda İşlevler Kullanarak Erişim Planı

SQL Plus

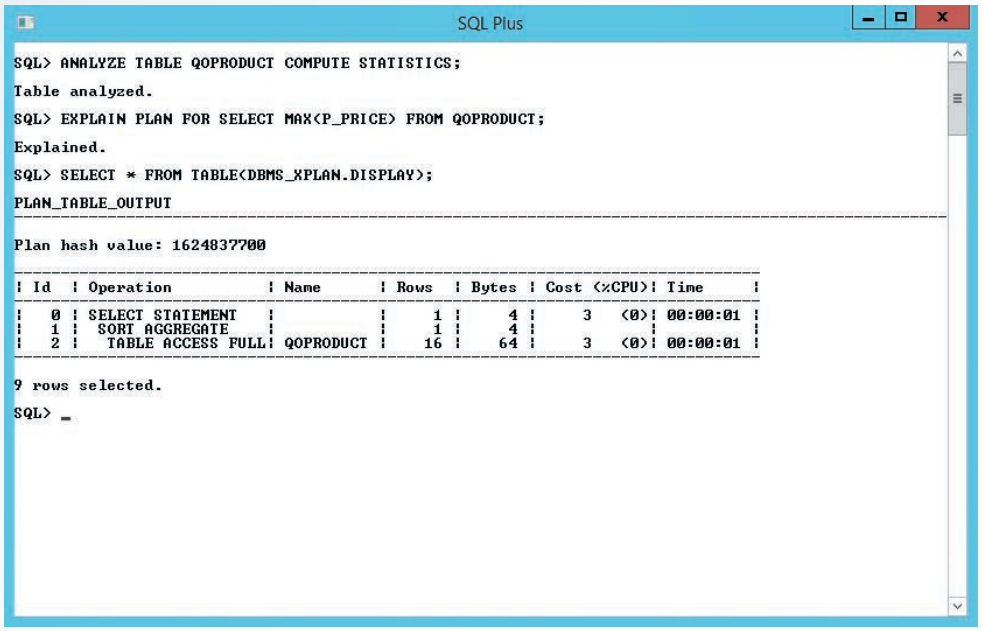
```
SQL> EXPLAIN PLAN FOR SELECT V_NAME, P_CODE FROM QOVENDOR V, QOPRODUCT P
  2                  WHERE  V.V_CODE = P.V_CODE AND UPPER(V_NAME) = 'ORDVA, INC.';

Explained.

SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------

Plan hash value: 1347990970

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                    |     1 |    31 |     5   (0)| 00:00:01 |
|*  1 |  HASH JOIN               |                    |     1 |    31 |     5   (0)| 00:00:01 |
|   2 |   VIEW                   | index$_join$_001   |     1 |    17 |     2   (0)| 00:00:01 |
|*  3 |    HASH JOIN             |                    |       |       |            |          |
|*  4 |     INDEX FAST FULL SCAN | QOV_NDX2           |     1 |    17 |     1   (0)| 00:00:01 |
|   5 |     INDEX FAST FULL SCAN | SYS_C0058569       |     1 |    17 |     1   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------

|*  6 |   TABLE ACCESS FULL      | QOPRODUCT          |    14 |   196 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("V"."V_CODE"="P"."V_CODE")
   3 - access(ROWID=ROWID)
   4 - filter(UPPER("V_NAME")='ORDVA, INC.')
   6 - filter("P"."V_CODE" IS NOT NULL)

21 rows selected.

SQL> _
```

Şekil 11.9'da görüldüğü gibi, dizinlenmiş bir sütunda bir işlevin kullanılması, DBMS'nin sorgunun maliyetini potansiyel olarak artırabilecek ek işlemler gerçekleştirmesine neden olmuştur. Tablolarınız çok daha fazla satır içeriyorsa ve dizin seyrekliği farklıysa, aynı sorgu farklı maliyetler üretebilir.

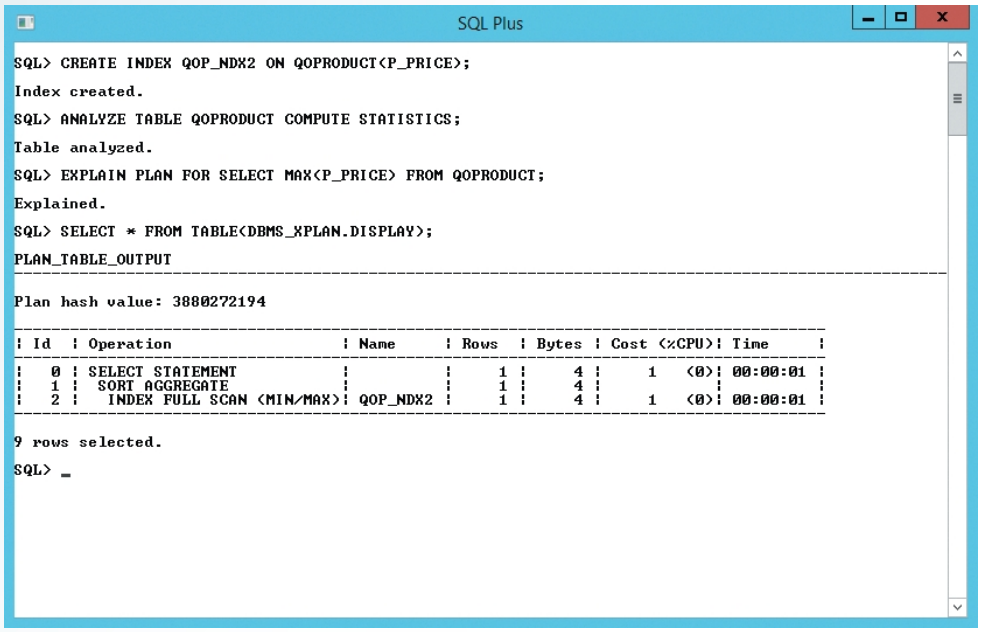
Şimdi QOPRODUCT tablosunu kullanarak, toplama işlevi sorguları çalıştırılırken bir dizinin nasıl yardımcı olabileceğini gösterin. Örneğin, Şekil 11.10 MAX(P_PRICE) toplama işlevini kullanan bir SELECT deyiminin erişim planını göstermektedir. Bu planda, toplam maliyeti 3 olan bir tam tablo taraması kullanılır.

3'lük bir maliyet zaten çok düşüktür, ancak P_PRICE üzerinde bir dizin oluşturarak önceki sorgu performansını artırabilirsiniz. Şekil 11.11, dizin oluşturulduktan ve QOPRODUCT tablosu analiz edildikten sonra plan maliyetinin nasıl üçte iki oranında azaldığını göstermektedir. Ayrıca erişim planının ikinci versiyonunun sorguyu yanıtlamak için yalnızca QOP_NDX2 dizinini kullandığına dikkat edin; *QOPRODUCT tablosuna asla erişilmez*.

Şekil 11.10 Önce Planı Açıkla: İndekslenmemiş Sütun Üzerinde Toplama Fonksiyonu

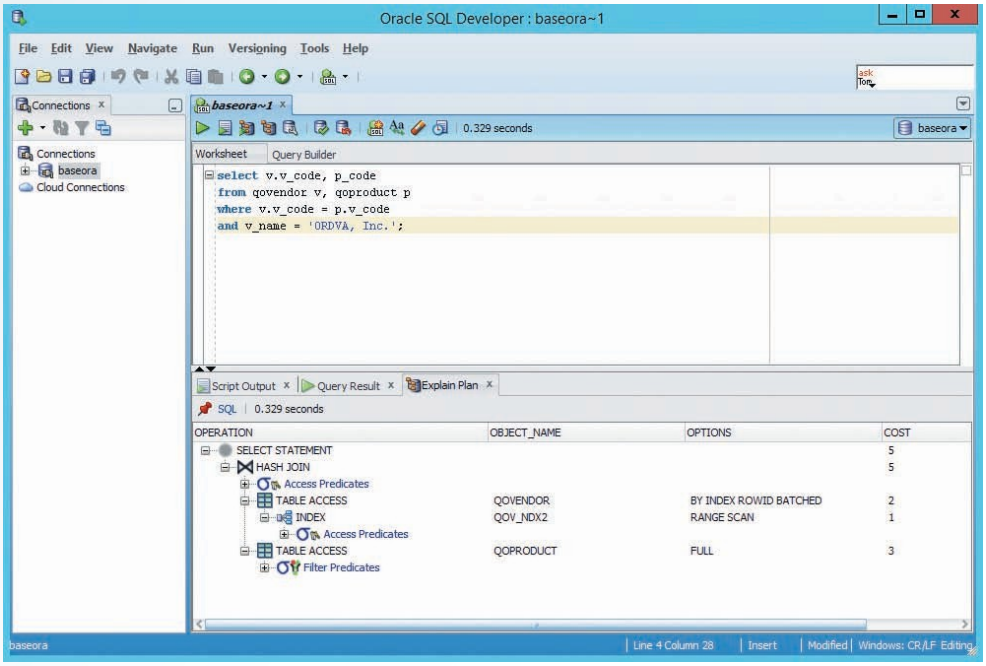


Şekil 11.11 İkinci Açıklama Planı: İndeksli Sütun Üzerinde Toplama Fonksiyonu

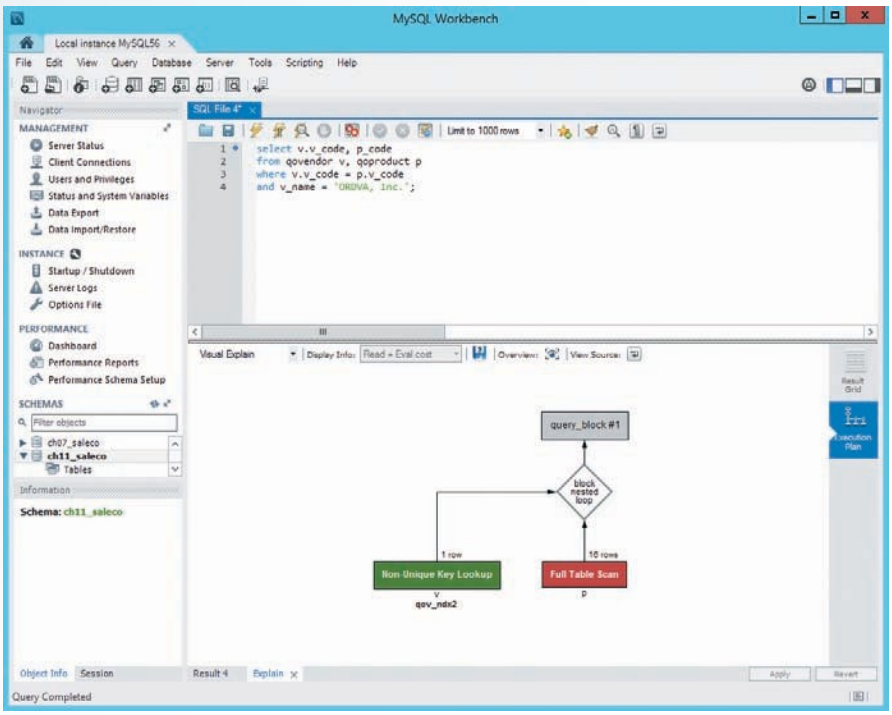


Bu bölümdeki birkaç örnek, sorgu optimizasyonu için uygun dizin seçiminin önemini gösterse de, dizin oluşturmanın sorgu performansını artırmadığı örnekleri de gördünüz. Bir DBA olarak, ana hedefin genel veritabanı performansını optimize etmek olduğunun farkında olmalısınız - sadece tek bir sorgu için değil, tüm istekler ve sorgu türleri için. Çoğu veritabanı sistemi, performans izleme ve test etme için gelişmiş grafiksel araçlar sağlar. Örneğin, Şekil 11.12, 11.13 ve 11.14'te Oracle, MySQL ve Microsoft SQL Server araçları kullanılarak erişim planının grafiksel gösterimi gösterilmektedir.

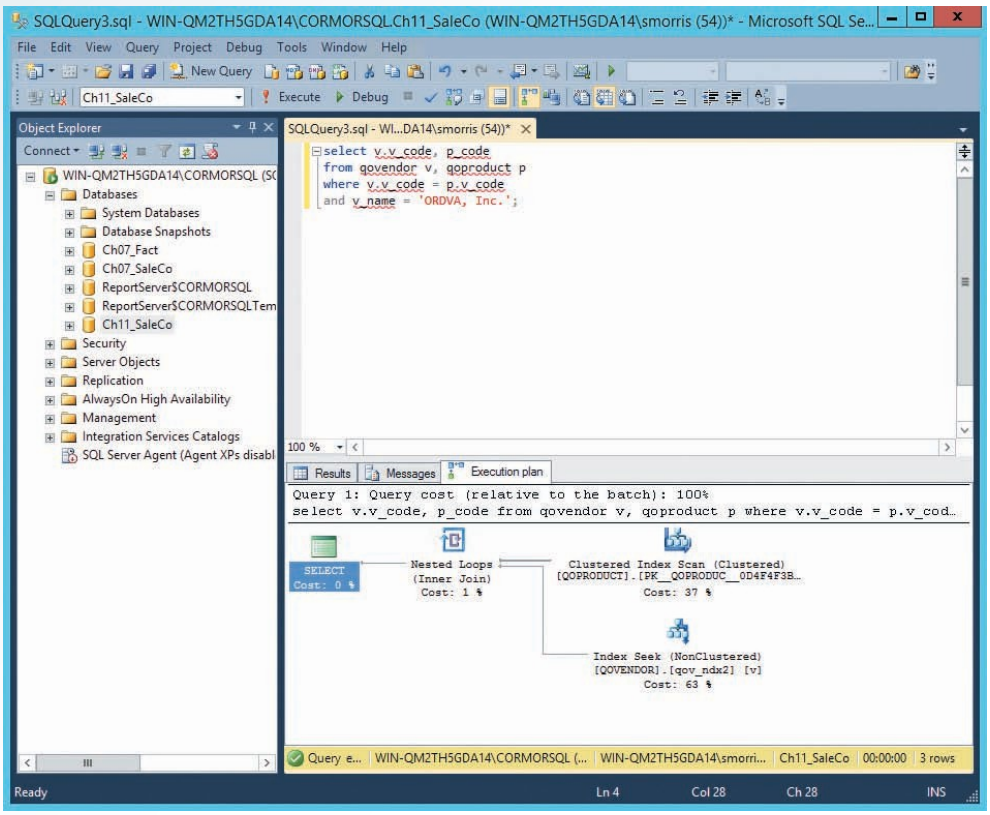
Şekil 11.12 Sorgu Optimizasyonu için Oracle Araçları



Şekil 11.13 Sorgu Optimizasyonu için MySQL Araçları



Şekil 11.14 Sorgu Optimizasyonu için Microsoft SQL Server Araçları



Özet

- Veritabanı performans ayarlaması, bir son kullanıcı sorgusunun VTYS tarafından en kısa sürede işlenmesini sağlamak için tasarlanmış bir dizi faaliyet ve prosedürü ifade eder. SQL performans ayarı, sunucu ucunda minimum miktarda kaynak kullanarak en kısa sürede doğru yanıtı veren SQL kodunu oluşturmak için tasarlanmış istemci tarafındaki faaliyetleri ifade eder. DBMS performans ayarı, DBMS'nin mevcut kaynakları en iyi şekilde kullanırken istemcilerin isteklerine mümkün olan en hızlı şekilde yanıt verecek şekilde yapılandırılması için sunucu tarafındaki faaliyetleri ifade eder.
- Veritabanı istatistikleri, DBMS tarafından toplanan ve veritabanı nesnelerinin özelliklerinin anlık görüntüsünü tanımlayan bir dizi ölçümü ifade eder. DBMS, tablolar, dizinler ve kullanılan işlemci sayısı, işlemci hızı ve mevcut geçici alan gibi kullanılabilir kaynaklar gibi nesneler hakkında istatistikler toplar. DBMS, sorgu işleme verimliliğini artırma konusunda kritik kararlar almak için istatistikleri kullanır.
- VTYS'ler sorguları üç aşamada işler. Ayrıştırma aşamasında, DBMS SQL sorgusunu ayrıştırır ve en verimli erişim/çalıştırma planını seçer. Yürütme aşamasında, DBMS seçilen yürütme planını kullanarak SQL sorgusunu yürütür. Getirme aşamasında, VTYS verileri getirir ve sonuç kümesini istemciye geri gönderir.
- İndeksler, veri erişimini hızlandıran süreçte çok önemlidir. İndeksler aramayı, sıralamayı, toplama işlevlerini ve birleştirme işlemlerini kullanmayı kolaylaştırır. Veri erişim hızındaki iyileşme, bir dizinin dizin anahtarını ve işaretçileri içeren sıralı bir değerler kümesi olması nedeniyle gerçekleşir. Veri seyrekliği, bir sütunun sahip olabileceği farklı değerlerin sayısını ifade eder. İndeksler, arama koşullarında kullanılan yüksek seyreklikli sütunlarda tavsiye edilir.
- Sorgu optimizasyonu sırasında, DBMS hangi indekslerin kullanılacağını, birleştirme işlemlerinin nasıl gerçekleştirileceğini, ilk olarak hangi tablonun kullanılacağını vb. seçmelidir. Her DBMS'nin en verimli yolu belirlemek için kendi algoritmaları vardır

verilere erişim. En yaygın iki yaklaşım kural tabanlı ve maliyet tabanlı optimizasyondur.

- Kural tabanlı bir iyileştirici, bir sorguyu yürütmek için en iyi yaklaşımı belirlemek üzere önceden belirlenmiş kuralları ve noktaları kullanır. Maliyet tabanlı bir iyileştirici, bir sorguyu yürütmek için en iyi yaklaşımı belirlemek üzere erişilen nesneler hakkındaki istatistikleri temel alan gelişmiş algoritmalar kullanır. Bu durumda, en iyi duruma getirme işlemi, belirli bir yürütme planının toplam maliyetini belirlemek için işlem maliyetini, G/Ç maliyetlerini ve kaynak maliyetlerini (RAM ve geçici alan) toplar.
- SQL performans ayarlaması, istatistikleri iyi kullanan sorgular yazmakla ilgilenir. Özellikle, sorgular indeksleri iyi kullanmalıdır. İndeksler, bir koşula bağlı olarak büyük bir tablodan küçük bir satır alt kümesi seçmek istediğinizde çok kullanışlıdır.
- Sorgu formülasyonu, gerekli sonuçları üretmek için iş sorularının belirli SQL koduna nasıl çevrileceği ile ilgilenir. Bunu yapmak için, istenen çıktıyı oluşturmak için hangi sütunların, tabloların ve hesaplamaların gerekli olduğunu dikkatlice değerlendirmelisiniz.
- DBMS performans ayarlaması, birincil bellekteki DBMS süreçlerinin yönetimi (önbellekleme amacıyla bellek ayırma) ve fiziksel depolamadaki yapıların yönetimi (veri dosyaları için alan ayırma) gibi görevleri içerir.

Anahtar Terimler

erişim planı
otomatik sorgu optimizasyonu
bitmap dizini
B-ağacı
dizin tampon
önbelleği
kümelenmiş dizin
tablosu maliyet
tabanlı iyileştirici
veritabanı performans
ayarlama veritabanı
istatistikleri
veri bloğu
veri
önbelleği
veri
dosyaları
veri
seyrekliği
DBMS performans ayarlama
dinamik sorgu optimizasyonu dinamik
istatistiksel üretim modu kapsamları
dosya grubu
işlev tabanlı dizin
karma dizin
bellek içi veritabanı
dizin-organize tablo
dizin seçiciliği
giriş/çıkış (G/Ç) isteği G/Ç
hızlandırıcı
manuel sorgu optimizasyonu
manuel istatistiksel üretim modu
optimize edici ipuçları
prosedür
önbelleği sorgu
iyileştirici
sorgu işleme darboğazı RAID
kural tabanlı optimize edici
kural tabanlı sorgu optimizasyon algoritması
statik sorgu optimizasyonu
istatistiksel tabanlı sorgu optimizasyon algoritması
SQL önbelleği
SQL performans ayarlama
tablo alanı

İnceleme Soruları

1. SQL performans ayarlaması nedir?
2. Veritabanı performans ayarlaması nedir?
3. Performans ayarlama faaliyetlerinin çoğunun odak noktası nedir ve bu odak noktası neden vardır?
4. Veritabanı istatistikleri nedir ve neden önemlidir?
5. Veritabanı istatistikleri nasıl elde edilir?
6. Tablolar, dizinler ve kaynaklar için tipik veritabanı istatistik ölçümleri nelerdir?
7. SQL DDL deyimlerinin (CREATE TABLE gibi) işlenmesi DML deyimlerinin gerektirdiği işlemden nasıl farklıdır?
8. Basit bir ifadeyle, DBMS bir sorguyu üç aşamada işler. Aşamalar nelerdir ve her aşamada ne gerçekleştirilir?

9. İndeksler bu kadar önemliyse, neden her tablodaki her sütunu indekslemiyoruz? (Veri seyrekliğinin oynadığı rol hakkında kısa bir tartışma ekleyin).
10. Kural tabanlı bir optimize edici ile maliyet tabanlı bir optimize edici arasındaki fark nedir?
11. Optimize edici ipuçları nedir ve nasıl kullanılır?
12. Dizinlerin oluşturulması ve kullanılmasına ilişkin bazı genel kurallar nelerdir?
13. Çoğu sorgu optimizasyon tekniği, optimize edicinin işini kolaylaştırmak için tasarlanmıştır. SQL kodunda koşullu ifadeler yazmayı düşünüyorsanız hangi faktörleri aklınızda tutmalısınız?
14. Çok sayıda tablo ve dizin içeren bir VTYS'de veri dosyalarını yönetmek için ne gibi önerilerde bulunursunuz?
15. RAID ne anlama gelir ve yaygın olarak kullanılan bazı RAID seviyeleri nelerdir?

Problemler

Problem 1 ve 2 aşağıdaki sorguya dayanmaktadır:

```
SEÇİNİZ      EMP_LNAME, EMP_FNAME, EMP_AREACODE, EMP_SEX
FROM         ÇALIŞAN
NEREDE       EMP_SEX 5 'F' VE EMP_AREACODE 5 '615'
ORDER BY     EMP_LNAME, EMP_FNAME;
```

1. EMP_SEX sütununun olası veri seyrekliği nedir?
2. Hangi indeksleri oluşturmalısınız? Gerekli SQL komutlarını yazın.
3. Tablo 11.4'ü örnek olarak kullanarak iki alternatif erişim planı oluşturun. Aşağıdaki varsayımları kullanın:
   a. 8.000 çalışan bulunmaktadır.
   b. 4.150 kadın çalışan bulunmaktadır.
   c. Alan kodu 615'te 370 çalışan bulunmaktadır.
   d. Alan kodu 615'te 190 kadın çalışan bulunmaktadır.

Problem 4-6 aşağıdaki sorguya dayanmaktadır:

```
SEÇİNİZ      EMP_LNAME, EMP_FNAME, EMP_DOB, YEAR(EMP_DOB) AS YEAR
FROM         ÇALIŞAN
NEREDE       YIL(EMP_DOB) 5 1976;
```

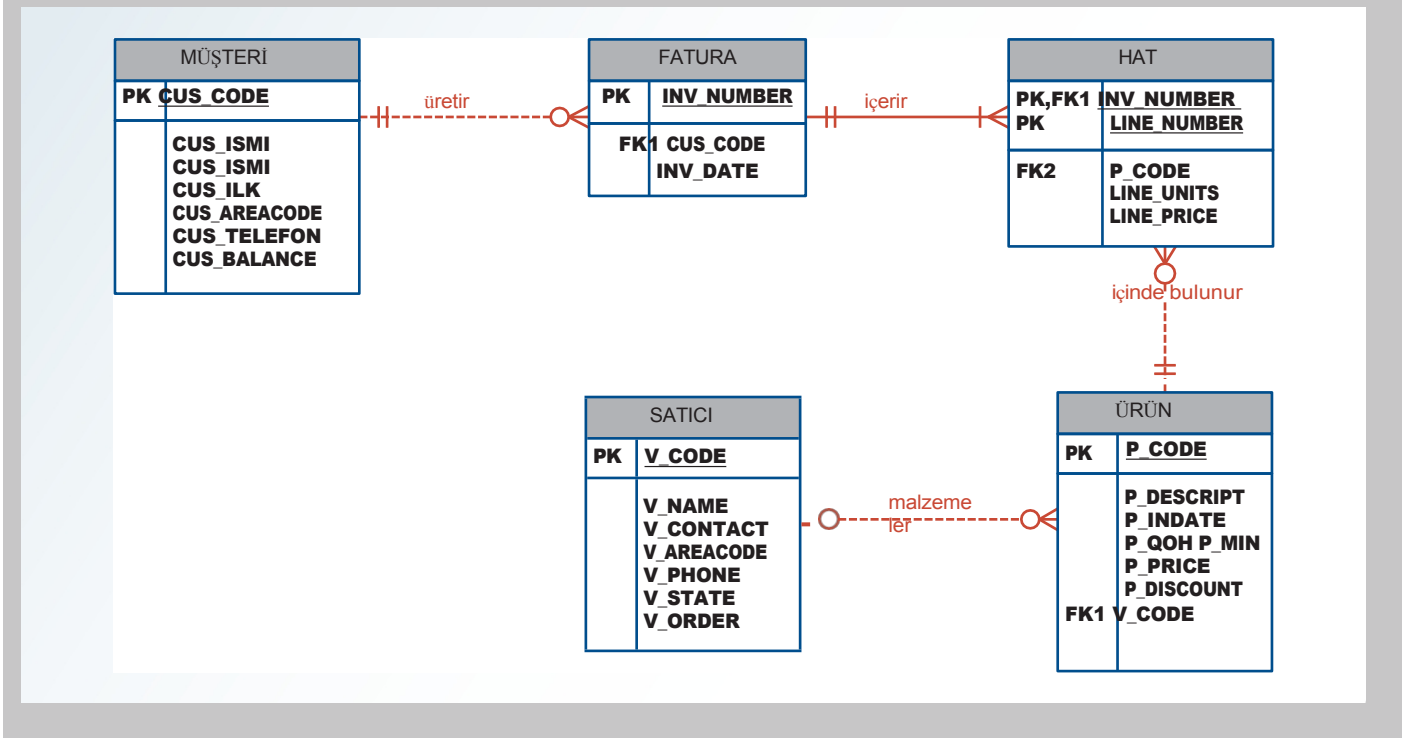
4. EMP_DOB sütununun olası veri seyrekliği nedir?
5. EMP_DOB üzerinde bir dizin oluşturmalı mısınız? Neden ya da neden olmasın?
6. Sorgu tarafından muhtemelen ne tür veritabanı G/Ç işlemleri kullanılacaktır? (Bkz. Tablo 11.3.)

7-29 numaralı problemler Şekil P11.7'de gösterilen ER modeline dayanmaktadır. Problemler 7-10 aşağıdaki sorguya dayanmaktadı

```
SEÇİNİZ      P_CODE, P_PRICE
FROM         ÜRÜN
NEREDE       P_PRICE >5 (SELECT AVG(P_PRICE) FROM PRODUCT);
```

7. Tablo istatistikleri olmadığını varsayarsak, VTYS ne tür bir optimizasyon kullanacaktır?
8. Sorgu tarafından muhtemelen ne tür veritabanı G/Ç işlemleri kullanılacaktır? (Bkz. Tablo 11.3.)
9. P_PRICE sütununun olası veri seyrekliği nedir?
10. Bir dizin oluşturmalı mısınız? Neden ya da neden olmasın?

Şekil P11.7 Sorunlar 7-29 için Ch11_SaleCo ER Modeli



11-14 numaralı problemler aşağıdaki sorguya dayanmaktadır:

```
SEÇİNİZ          P_CODE, SUM(LINE_UNITS)
FROM             HAT
GROUP BY         P_CODE
SAHİP OLMAK      SUM(LINE_UNITS)> (SELECT MAX(LINE_UNITS) FROM LINE);
```

11. LINE_UNITS sütununun olası veri seyrekliği nedir?
12. Bir dizin oluşturmalı mısınız? Eğer öyleyse, dizin sütun(lar)ı ne olurdu ve dizini neden oluştururdunuz? Oluşturmayacaksanız, gerekçenizi açıklayın.
13. P_CODE üzerinde bir dizin oluşturmalı mısınız? Eğer öyleyse, dizini oluşturmak için SQL komutunu yazın. Değilse, gerekçenizi açıklayın.
14. Bu tablo için istatistik oluşturacak komutu yazınız.

Problem 15 ve 16 aşağıdaki sorguya dayanmaktadır:

```
SEÇİNİZ          P_CODE, P_QOH *  P_PRICE
FROM             ÜRÜN
NEREDE           P_QOH *  P_PRICE> (SELECT AVG(P_QOH *  P_PRICE) FROM PRODUCT);
```

15. P_QOH ve P_PRICE sütunlarının olası veri seyrekliği nedir?
16. Bir dizin oluşturmalı mısınız? Eğer öyleyse, dizin sütun(lar)ı ne olacaktır ve dizini neden oluşturmalısınız? Problem

17-20 aşağıdaki sorguya dayanmaktadır:

```
SEÇİNİZ          V_CODE, V_NAME, V_CONTACT, V_STATE
FROM             SATICI
NEREDE           V_STATE 5 'TN'
ORDER BY         V_NAME;
```

17. Hangi indeksleri oluşturmalısınız ve neden? İndeksleri oluşturmak için SQL komutunu yazın.

18. 10.000 satıcının Tablo P11.18'de gösterildiği gibi dağıtıldığını varsayalım. Sorgu tarafından satırların yüzde kaçı döndürülecektir?

Tablo P11.18

Eyalet	Satıcı Sayısı
AK	15
AL	55
AZ	100
CA	3244
CO	345
FL	995
GA	75
HI	68
IL	89
İÇİNDE	12
KS	19
KY	45
LA	29
MD	208
MI	745
MO	35
MS	47
NC	358
NH	25
NJ	645
NV	16
OH	821
TAMAM.	62
PA	425
RI	12
SC	65
SD	74
TN	113
TX	589
UT	36
VA	375
WA	258

19. Sorguyu yürütmek için büyük olasılıkla ne tür I/O veritabanı işlemleri kullanılır?

20. Tablo 11.4'ü örnek olarak kullanarak iki alternatif erişim planı oluşturun.

Problem 21-23 aşağıdaki sorguya dayanmaktadır:

```
SEÇİNİZ      P_CODE, P_DESCRIPT, P_PRICE, .V_CODE, V_STATE
FROM         ÜRÜN P, SATICI V
NEREDE       P.V_CODE 5 V.V_CODE
             VE V_STATE 5 'NY'
             VE V_AREACODE 5 '212'
ORDER BY     P_PRICE;
```

21. Hangi endeksleri önerirsiniz?

22. Problem 21'de önerdiğiniz dizinleri oluşturmak için gereken komutları yazın.

23. ÜRÜN ve SATICI tablolarının istatistiklerini oluşturmak için kullanılan komut(lar)ı yazın.

24. Aşağıdaki sorguya dayalı olarak hangi dizini önerirsiniz ve bunu oluşturmak için hangi komutu kullanırsınız?

```
SEÇİNİZ      P_CODE, P_DESCRIPT, P_QOH, P_PRICE, V_CODE
FROM         ÜRÜN
NEREDE       V_CODE 5 '21344'
ORDER BY     P_CODE;
```

Problem 25 ve 26 aşağıdaki sorguya dayanmaktadır:

```
SEÇİNİZ      P_CODE, P_DESCRIPT, P_QOH, P_PRICE, V_CODE
FROM         ÜRÜN
NEREDE       P_QOH< P_MIN
             VE P_MIN 5 P_REORDER VE
             P_REORDER 5 50
ORDER BY     P_QOH;
```

25. Sorguyu yeniden yazmak ve gerekli sonuçları daha verimli bir şekilde üretmek için Bölüm 11-5b'de verilen önerileri kullanın.

26. Hangi indeksleri önerirsiniz? Bu indeksleri oluşturmak için gerekli komutları yazınız.

Problem 27-29 aşağıdaki sorguya dayanmaktadır:

```
SEÇİNİZ      CUS_CODE, MAX(LINE_UNITS * LINE_PRICE)
FROM         MÜŞTERI DOĞAL BIRLEŞTIRME FATURA DOĞAL
BIRLEŞTIRME HATTI NEREDE CUS_AREACODE 5 '615'
GROUP BY     CUS_CODE;
```

27. Ayda 15.000 fatura ürettiğinizi varsayarsak, tasarımcıya türetilmiş niteliklerin kullanımı hakkında ne önerirsiniz?

28. Problem 27'de verdiğiniz tavsiyelere uyduğunuzu varsayarsak, sorguyu nasıl yeniden yazardınız?

29. Problem 28'de yazdığınız sorgu için hangi indeksleri önerirsiniz ve hangi SQL komutlarını kullanırsınız?