

46309

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

**DAĞITIK VERİTABANLARINDA
SORGU OPTİMİZASYONU**

YÜKSEK LİSANS TEZİ

Müh. Banu TEZEL

Tezin Enstitüye Verildiği Tarih : 6 Ekim 1995

Tezin Savunulduğu Tarih : 16 Ekim 1995

Tez Danışmanı : Prof. Dr. Mithat UYSAL

Diğer Jüri Üyeleri : Prof. Dr. Metin DEMİRALP

: Doç. Gazanfer ÜNAL

**Y.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ**

EKİM - 1995

ÖNSÖZ

Tez çalışmamdaki yardımlarından dolayı tez danışmanım sayın Prof. Dr. Mithat UYSAL ve arkadaşlarıma teşekkür ederim.

Banu TEZEL

İÇİNDEKİLER

ÖNSÖZ.....	ii
İÇİNDEKİLER.....	iii
ŞEKİL LİSTESİ.....	iv
ÖZET.....	v
SUMMARY.....	vi
BÖLÜM 1. GİRİŞ.....	1
BÖLÜM 2. MERKEZİLEŞMİŞ SORGU OPTİMİZASYONU ALGORİTMALARI.....	4
2.1. Sorgu Optimizasyonuna Girdiler.....	4
2.1.1. Maliyet Modeli.....	4
2.1.2. Veritabanı İstatistikleri.....	7
2.1.3. Ara Sonuçların Kardinaliteleri.....	8
2.2. Merkezileşmiş Sorgu Optimizasyonu.....	11
2.2.1. Ingres Algoritması.....	12
2.2.2. Sistem R Algoritması.....	17
2.3. Kısmi Sorgularda Birleşme Algoritması.....	23
2.3.1. Birleşme Sorgulaması.....	23
2.3.2. Yarı Birleşme Tabanlı Algoritmalar.....	26
2.3.3. Tam Ve Yarı Birleşimlerin Karşılaştırılması.....	30
BÖLÜM 3. DAĞITIK SORGU OPTİMİZASYON ALGORİTMALARI.....	32
3.1. Dağıtık Ingres Algoritması.....	33
3.2. R* Algoritması.....	38
3.3. Sdd-1 Algoritması.....	43
3.4. Ahy (Apers-Hevner-Yao) Algoritmaları.....	48
SONUÇLAR VE ÖNERİLER.....	58
EKLER.....	61
KAYNAKLAR.....	81
ÖZGEÇMİŞ.....	82

ŞEKİL LİSTESİ

Şekil	2.1.	Bir Sorgu İçin Veri Transferi Örneği.....	7
Şekil	2.2.	Sorgu q_i 'in Birleşim Grafiği	24
Şekil	2.3.	Alternatif Birleşim Sıralamaları.....	25
Şekil	2.4.	Binary Operasyonlarda Operand'ların Transferi.....	28
Şekil	2.5.	Dağıtık Sorgunun Birleşme Grafiği.....	28
Şekil	2.6.	Dairesel Sorgunun Dönüştürülmesi.....	33
Şekil	2.7.	Tam ve Yarıbirleşimlerin Karşılaştırılması.....	35
Şekil	3.1.	Sorgu Optimizasyon Algoritmalarının Karşılaştırılması.....	37
Şekil	3.2.	Plan Örneği.....	56
Şekil	3.3.	Optimal Plan.....	58
Şekil	3.4.	Sorgu Örneğinin Uygun Başlangıç Planı.....	59
Şekil	3.5.	Sorgu Örneğinin Genişlemiş Planı.....	60
Şekil	3.6.	Sorgu Örneğinin Optimal Planı	60

ÖZET

Bu çalışmada, Dağıtık Veritabanı Sistemlerinin Yönetimi' nde önemli problemlerden biri olarak yer alan “Sorgu Optimizasyonu” konusu incelenmiştir.

Merkezi bir veritabanı sistemi sözkonusu olduğunda bile veritabanı üzerinde sorgulamayı optimize yapmak zor bir problemdir.Farklı veritabanları,veritabanı yönetim sistemleri, işletim sistemleri, ağ protokolleri vs. bileşenlerinden oluşan dağıtık bir veritabanı yönetim sisteminde bu problemin daha zor ve kompleks bir hal alacağı aşıkardır.

Tez konu itibari ile iki ana bölümden oluşmaktadır.İlk bölümde,dağıtık sorgu optimizasyonu ile ilgili temel kavram ve teknikler üzerinde durulmuştur.Daha sonra bu Merkezileşmiş Sorgu Optimizasyon algoritmalarından INGRES,System R incelenmiştir.İkinci bölümde ise Dağıtık INGRES,R*,SDD-1 ve AHY algoritmaları incelenmiş ve karşılaştırmaları yapılmıştır.Bölümler içinde konularla ilgili örnekler bulunmaktadır.

SUMMARY

QUERY OPTIMIZATION OF DISTRIBUTED DATABASE SYSTEMS

Query optimization represents both a challenge and an opportunity for relational systems;

Incidentally, the advantage of optimization is not merely that users do not have to worry about how best to state their queries (i.e., how to phrase requests in order to get the best performance out of the system). On the contrary, there is a real possibility that the optimizer might actually do better than a human programmer. There are several reasons for this state of affairs.

1. A good optimizer will have a wealth of information available to it that human programmers typically do not have. To be specific, it should have certain statistical information, such as the cardinality of each domain, the cardinality of each table, the number of values in each column, the number of times each different value occurs in each column, and so on. (This information will be kept in the system catalog, of course.) As a result, the optimizer should be able to make a more accurate assessment of the efficiency of any given strategy for implementing a particular request, and thus should be more likely to choose the most efficient implementation.
2. Furthermore, if the database statistics change significantly (e.g., if the database is physically reorganized), then a different choice of strategy may be desirable; in other words, reoptimization may be required. In a relational system, reoptimization is trivial - it simply involves a reprocessing of the original relational request by the system optimizer. In a nonrelational system, by contrast, reoptimization involves rewriting the program, and will probably therefore not be done at all.
3. Third, the optimizer is a program, and therefore by definition is much more patient than a typical application programmer. The optimizer is quite capable of considering literally hundreds of different implementation strategies for a given request, whereas it is extremely unlikely that a human programmer would ever consider more than three or four.
4. Fourth, the optimizer can be regarded in a sense as embodying the skills and services of "the best" human programmers. As a consequence, it has the effect of making those skills and services available to everybody which means, of course, that it

is making an otherwise scarce set of resources available to a wide range of users, in an efficient and cost effective manner.

All of the above should serve as evidence in support of the claim, made at the beginning of this 2nd section that optimizability -i.e., the fact that relational requests are optimizable- is in fact a strength of relational systems.

The overall purpose of the optimizer, then, is to choose an efficient strategy for evaluating a given relational expression.

In the second section two of the most popular query optimization techniques for centralized systems are presented. This presentation is a prerequisite to understanding distributed query optimization for three reasons. First, a distributed query is translated into local queries, each of which is processed in a centralized way. Second, distributed query optimization techniques are often extensions of the techniques for centralized systems. Finally, centralized query optimization is a simpler problem; the minimization of communication costs makes distributed query optimization more complex. In this section some common techniques for query decomposition, two popular relational database systems are compared: INGRES and System R. Furthermore, both systems have distributed versions whose optimization algorithms are extensions of the centralized version.

The optimization techniques of these systems differ significantly. INGRES employs a dynamic optimization algorithm and System R uses a static optimization algorithm based on exhaustive search using statistics about the database. We note that most commercial relational DBMSs use an exhaustive search approach for its efficiency and compatibility with query compilation.

In the third section the basic concepts and techniques for distributed query optimization are presented. Their application in the basic query optimization algorithms of Distributed INGRES, R*, SDD-1, and AHY are reviewed. With the availability of faster communication networks (e.g., local area networks), more recent distributed DBMSs consider local processing costs as well. Performance evaluation of R* has shown significant contributions of local costs, even for general networks, which have become faster.

The main factor affecting the performance of an execution strategy is the size of the intermediate relations that are produced during the execution. When a subsequent operation is located at a different site, the intermediate relation must be

transmitted over the network. Therefore, it is of prime interest to estimate the size of the intermediate results of relational algebra operations in order to minimize the size of data transfers. This estimation is based on statistical information about the base relations and formulas to predict the cardinalities of the results of the relational operations.

Important inputs to the query optimization problem are the database statistics and the formulas used to estimate the size of intermediate results. There is a direct trade-off between performance and the accuracy (and maintenance cost) of statistics. The critical operation is generally the join of distributed relations. For most frequent joins that are not on foreign keys, join selectivity factors should be of great benefit. The use of statistics can be avoided by applying a simple algorithm based on heuristics to transform a query. However, it has been recognized that exhaustive search of the solution space, based on statistics, significantly outperforms the heuristic approaches. When done statically (at compile time), the overhead incurred by exhaustive search is rapidly amortized if the query becomes complex or is executed frequently.

As a prerequisite to understanding distributed query optimization, we have introduced centralized query optimization. The two most popular algorithms used by INGRES and System R were presented. These algorithms form the basis of a distributed optimization algorithm for distributed INGRES and R*, respectively.

Two approaches to solve distributed join queries, which are usually considered the most important type of queries have been seen. The first illustrated by Distributed INGRES and R*, considers join ordering. Semijoins are beneficial only when a join has good selectivity, in which case the semijoins act as powerful size reducers. The first system that made extensive use of semijoins assumed a slow wide area networks, the local processing cost. However, with fast local area networks, the local processing cost and sometimes even more important. Therefore, semijoins should be employed carefully and not systematically since they tend to increase the local processing cost. Join and semijoin techniques should be both employed as shown in, because each technique is better than the other, depending on database-dependent parameters. Semijoins implemented by hashed bit arrays are shown in to be very beneficial.

The query optimization algorithm of Distributed INGRES (Epstein et al. 1978) is derived from the algorithm used in centralized INGRES (see Section 9.2.1.). Therefore, it consists of dynamically optimizing the processing strategy of a given

query. The objective function of the algorithm is to minimize a combination of both the communication cost and the response time. Because these two objectives are conflicting (increasing communication cost may well decrease response time), the function can give a greater weight to one or the other. Note that this query optimization algorithm ignores the cost of transmitting the data to the result site. The algorithm also takes advantage of fragmentation, but only horizontal fragmentation is handled for simplicity.

The distributed query optimization algorithm of R* (Selinger and Adiba, 1980) and (Lohman et al., 1985) is a substantial extention of the techniques developed for System R's optimizer (see Section 9.2.2). Therefore, it uses a compilation approach where an exhaustive search of all alternative strategies is performed in order to choose the one with the least cost. Although predicting and enumerating these strategies is costly, the overhead of exhaustive search is rapidly amortized if the query is executed frequently. Although the algorithm described in (Selinger and Adiba, 1980) deals with fragmentation, the implemented version of R* supports neither fragmentation nor replication. Therefore, the R* query processing algorithm deals only with relations as basic units. Query compilation is a distributed task in R*, coordinated by a master site, where the query is initiated. The optimizer of the master site makes all intersite decisions, such as the selection of the execution sites and the fragments as well as the method for transferring data.

Another promising approach to solving join queries is through the use of auxiliary and small data structures, called join indices which have the potential of minimizing both local processing cost and communication cost. The performance analyses reported in already exhibit the value of join indices in a centralized context for optimizing both complex relational queries and recursive queries. Join indices can be useful in a distributed context as well.

In this section, mostly on join queries for two reasons: join queries are the most frequent queries in the relational framework and they have been studied extensively. Furthermore, the number of joins involved in queries expressed in languages of higher expressive power than relational calculus can be extremely large, making the join ordering more crucial. However, the optimization of general queries containing joins, unions, and aggregate functions is a harder problem. Distributing unions over joins is a simple and good approach since the query can be reduced as a union of join subqueries, which are optimized individually. Note also that the unions are more frequent in distributed DBMSs than in centralized DBMSs because they permit the localisation of horizontally fragmented relations.

The concepts and solutions presented in this section have been illustrated by their use in four query processing algorithms, chosen for their generality and practical interest (all of which were implemented). However, other methods can be derived from these concepts. A good approach is to specialize the optimization algorithm to take advantage of the network topology, such as star networks, satellite networks and broadcast networks.

Query processing in a distributed system requires the transmission of data between computers in a network. The arrangement of data transmissions and local data processing is known as a distribution strategy for a query. Two cost measures, response time and total time are used to judge the quality of a distribution strategy. Simple algorithms are presented that derive distribution strategies which have minimal response time minimal total time, for a special class of queries. These optimal algorithms are used as a basis to develop a general query processing algorithm. Distributed query examples are presented and the complexity of the general algorithm is analyzed. The integration of a query processing subsystem into a distributed database management system is discussed.

BÖLÜM 1. GİRİŞ

Veritabanı merkezileşmiş olduğu zaman bile optimal durumda veritabanını sorgulamak zor bir problemdir.Örneğin;ilişkisel bir sistemde, bir sorgu kısıt, projeksiyon ve birleştirme oprasyonları serilerine ayrılabilir. Fakat, direk sorgulamalar için, bu operasyonların çalıştırılabileceği farklı yolların sayısı çok fazladır. [1]

Bu nedenle, bu tip sistemlerin çoğunluğunda, mükemmel veya optimal sırada çalışmayı bulmak için, durum uzayını taramaya yarayan bütünleyici bölüm optimize edicidir.Verilen bir sorgu için optimum işlem sırasını bulmak sorgu optimizasyoncusu, yada kısaca “ optimizer” in ana rolüdür.

Yapılması istenen bir sorgulamanın performansı, sorgulamanın istendiği yerden bağımsız olmalıdır.İlişkisel bir sistemdeki sorgulamalar son derece yüksek bir dil düzeyinde ifade edildiğinden, normalde yürütölmelerine ilişkin pek çok olası strateji vardır.Doğallıkla farklı stratejilerin farklı başarımlar nitelikleri olacaktır.

İyi bir strateji seçmek temel önem taşır.”Optimizer” in ilişkisel sistemlerde bu kadar kritik bir bileşen olmasının nedeni de budur.Bir sorgunun optimal sıralandırılması hesapsal olarak güç olduğu için “Optimizer” in esas hedefi optimala yaklaşmak için uygun bir strateji bulmak, daha da önemlisi kötü stratejilerden kaçınmaktır.Bu tür bir sistemdeki başarımlar bu nedenle temel düzeyde sistem optimizasyon kalitesine bağlıdır.

Bunun tersine ilişkisel olmayan bir sistemdeki başarımlar, temelde uygulamanın programcısına bağlıdır. Uygulamanın programcısı kötü bir strateji seçtiyse sistemin bunu iyi bir stratejiye dönüştürebileceği bir yol yoktur. Bu ilişkisel olmayan sistemlerin başlıca dezavantajlarından biridir.[2]

Yukarıdaki noktaların tümü dağıtık sistemlerde daha çok geçerlidir. Dağıtık sistemlerde iyi bir strateji ile kötü bir strateji arasındaki başarımların farkı katlanarak logaritmik boyutlara varabilir. Dağıtık bir sistemde sorgu yürütmenin kendisi de dağıtımlı bir süreç olur. Bu etkinlik birden fazla ayrı sistemde hem yerel MIB (CPU) hem giriş/çıkış etkinliğinin yanı sıra, bu sistemler arasında bir miktar veri iletişimi gerektirecektir. Sonuncusu, en azından yavaş, uzun bağlantılı bir ağda, tüm diğer etkenlerin önüne geçecek bir başarımların bileşenidir.

Bu nedenle optimizasyon için başlıca hedef bu etkeni en aza indirmektir. “An Introduction to Database Systems” [3] adlı kitapta bu konuda çarpıcı bir örnek yer almaktadır: Bir sorgu örneğinin yürütülmesine ilişkin altı değişik strateji çözümlenmiş ve gerçekleşen yanıt süresi bir saniye ile 2 1/3 gün arasında oynamıştır.

Üç yerel sistem var olsun X, Y, Z. Q sorgulama prosesi X yerel sistemine ait olsun (X sisteminde yaratılsın) ve bu prosesi Y sistemindeki İy ilişkisindeki yüzlerce kayıtlarla, Z sistemindeki İz ilişkisinden milyonlarca kayıtların bileşimi alınmak istensin; X sistemindeki optimizasyon global bir strateji belirlemek zorundadır. Bu stratejinin İy’ den Z’ ye hareket şeklinde belirlenmesi önemlidir. İz’ den Y’ ye veya hele hele İy ve İz’ nin her ikisinden X’ e olması yanlış bir yönlendirme olacaktır. Bu hareketin yönü İy’ den Z’ ye şeklinde belirlendikten sonra Z sistemindeki gerçek birleşme stratejisine yerel optimizasyon tarafından karar verilmelidir.

Dağıtık bir sistemde sorgu işlemi, bir networkte bilgisayarlar arasında veri iletişimi gerektirir. Veri iletimlerinin düzenlenmesi ve lokal veri işlemi, bir sorgu için dağıtım stratejisi olarak bilinir. İki maliyet unsuru , cevaplama zamanı ve toplam zaman , bir dağıtım stratejisinin kalitesini tayin etmekte kullanılır. Basit algoritmalar, Özel olarak sınıflandırılan sorgular için minimal cevap zamanı ve minimal toplam zaman olan dağıtım stratejilerinin kökeni olarak tanıtılmaktadır. Bu optimal algoritmalar, genel bir sorgu işlemi algoritması geliştirmek için temel olarak kullanılırlar. Dağıtık sorgu örnekleri sunulmakta ve genel algoritmanın karmaşıklığı analiz edilmektedir. Bir sorgu işlem alt sisteminin dağıtık veri tabanı yönetimi sistemine entegre edilmesi tartışılmaktadır.

Bir networkte veya merkezi olmayan bir bilgisayar sisteminde veri dağıtımı, bir tek bilgisayarda veri merkezileştirilmesi üzerine bir çok çekici avantaj sunmaktadır. Bu avantajlar ,arttırılmış veri güvenirligi olan veriye daha hızlı ve lokalize edilmiş ulaşımı ile veri kapnodsinin daha yüksek kademelendirilmesi için potansiyeli de içermektedir. Önemli problemlerden biride, dağıtık bir sistemde sorguların etkin işlemidir . Değişik bilgisayarlarda saklanan veriye erişim merkezileştirilmiş bir bilgisayardaki veriye erişimden iki önemli yönden ayrılır. Haberleşme yolları üzerinden gerekli veri iletimleri önemli vakit gecikmelerine yol açar. Dağıtık sistemin bir avantajı da networkteki farklı noktalarda verinin paralel olarak işlem ve iletim kabiliyetidir. Veritabanı yönetim sistemi bu gerçekleri göz önünde bulundurmalı ve dağıtık sorgu işlemi için veri iletimi ile lokal veri işleminin etkin olarak düzenlenmesini oluşturmalıdır. Bu veri işlemi ve veri iletimlerinin düzenlenmesi bir sorgu için “dağıtım stratejisi” olarak bilinmektedir. Belirli bir sorgu için olan uygulanabilir dağıtım stratejileri arasında zaman maliyetinde büyük farklar bulunmaktadır. Bu durumda , dağıtık sistemlerde etkin sorgu işlem metodları kullanımının önemi vurgulanmaktadır.

Bu alanda önceki araştırmalarda , etkin dağıtım stratejilerini bulmak için klasik optimizasyon araştırma teknikleri kullanılmıştır. Wong , SDD-1 sisteminde halen uygulanan bir algoritma önermiştir. Önce uygulanabilir bir strateji seçilmiştir. Daha sonra , standart bir “tepeye tırmanma “ optimizasyon tekniği daha fazla maliyet gelişimi bulunamayana kadar daha düşük maliyet stratejilerini tekrarlayarak arama yapar. Sonuçtaki dağıtım stratejisi , arama uzayındaki lokal minimum maliyet çözümüdür.

Diğer veri erişim metodları, networkteki uygulanabilir dağıtım stratejilerini bulabilmek için dağıtık ortama merkezileştirilmiş sorgu taktikleri sunar. Basit bir sorgu sınıfı için , bazı maliyet unsurunun minimize edilmesi anlamında optimal dağıtım stratejileri türetecek optimizasyon algoritması geliştirilmiştir. Diğer uygulanabilir dağıtım stratejileri ile elde edilen dağıtım stratejisi karşılaştırmaya tabi tutulur. Sorgu optimizasyon algoritmasının dağıtık veritabanı yönetim sistemi tasarımına entegrasyon kabiliyeti tartışılmaktadır.

BÖLÜM 2. MERKEZİLEŞMİŞ SORGU OPTİMİZASYONU ALGORİTMALARI

Verilen bir sorgu için optimum işlem sırasını bulmak sorgu optimizasyoncusu, ya da kısaca “Optimizer” ‘in ana rolüdür. Optimizerin esas hedefi optimale yaklaşmak için uygun bir strateji bulmaktır. Optimizerin oluşturduğu stratejiye optimal strateji diyoruz. Optimizerin çıktısı parçalar üzerinde tanımlanmış cebirsel sorgulardan ve sorgunun üzerinde çalışmasını destekleyen iletişim işlemlerinden oluşan optimize edilmiş bir cetveldir.

Optimal stratejinin seçimi gerçekten çalıştırmadan önce alternatif aday sıralamalarının veri işleme masraflarının tahmin edilmesini gerektirir. Veri işleme masrafları I/O, CPU ve iletişim masraflarının ağırlıklı bir kombinasyonu olarak ifade edilir. Bununla beraber tipik bir basitleştirme; iletişim masraflarının daha baskın olduğunu varsayarak yerel veri işleme masraflarının (I/O ve CPU masrafları) ihmalidir. Veri işleme masraflarının tahmini için optimize gerekliliği; parça istatistikleri ve kişisel işlemlerin sonuçlarının bu kardinalitelerinin tahmini için gerekli formüllerdir.[4]

2.1. Sorgu Optimizasyonuna Girdiler

2.1.1. Maliyet Modeli

Bir dağıtık veri işleme stratejisinin masrafı, ya toplam masraflara ya da cevap zamanına göre belirtilir. Toplam masraf tüm masraf bileşenlerinin toplamıdır, cevap süresi ise sorgunun başlangıcından tamamlanmasına kadar geçen zamandır. Toplam masrafları belirlemek için genel bir formül aşağıdaki gibi gösterilebilir;

$$\text{Toplam Maliyet} = C_{CPU} * \# \text{ insts} + C_{I/O} * \# \text{ I/Os} + C_{MSG} * \# \text{ msgs} + C_{TR} * \# \text{ bytes}$$

İlk iki bileşen yerel veri işleme zamanını ölçer; C_{CPU} bir CPU talimatının masrafı, $C_{I/O}$ bir disk I/O' nun masrafıdır. İletişim masrafları son iki bileşenle tanımlanmıştır. C_{MSG} bir mesajı başlatmak ve almanın sabit masrafıdır. C_{TR} ise bilgi ünitesini bir yerden başka bir yere iletmnin data masrafıdır. Burada ünite byte biriminde verilmiştir, fakat farklı birimlerde de olabilirdi (Örn. paketler). Tipik bir kabul C_{TR} ' ın sabit olmasıdır. Bu bazı noktaların diğerlerinden çok uzak olduğu geniş alan şebekelerinde doğru olmayabilir. Fakat bu kabul sorgu optimizasyonunu bir hayli basitleştirir. Böylece # bytelık datanın bir noktadan diğerine transferinin masrafı byteın bir lineer fonksiyonu varsayılır.

$$CC(\# \text{ bytes}) = C_{MSG} + C_{TR} * \# \text{ bytes}$$

Masraflar genellikle, daha sonra başka birimlere (Örn. para) dönüştürülebilir bir birim olan zamanla ölçülür.

Masraf katsayılarının ilişkili değerleri dağınık veri tabanı ortamını karakterize eder. Şebekenin topolojisi masraf bileşenlerinin oranını önemli derecede etkiler. “ARPANET” gibi geniş alanlı bir şebekede (10 kbps mertebesinde); iletişim yerel işlemeden en az bir mertebe yavaş olduğu için, baskın olan faktör iletişim masrafıdır. Bir sayfa için tipik bir iletişim masrafı I/O masrafı oranı 20:1.

Bu yüzden geniş alanlar için tasarlanmış VTYS' lerinin çoğu yerel işleme masraflarını ihmal edip iletişim masraflarını minimize etmek üzerinde yoğunlaşır. “Ethernet” (10 Mbps) gibi bir yerel alan şebekesinde yerel işleme masraflarının toplam masrafa katkısı oldukça kayda değerdir. Tipik bir iletişim masrafı / I/O masrafı oranı 1:1.6' dır. Ayrıca FDDI (100 Mbps) gibi daha yüksek hızlı yerel alan şebekelerine doğru gidildikçe bu oran iletişim masrafları yönünde gelişir. Bu yüzden yerel alan şebekeleri için tasarlanmış VTYS' leri her üç masraf bileşenini de değerlendirmeye alır.

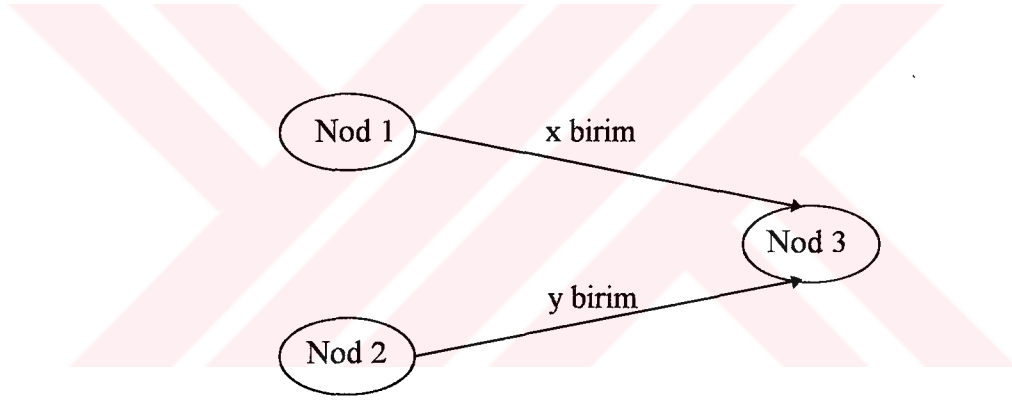
Optimizelerin ana görevi sorgunun cevap süresi ise, paralel iletişim gözönüne alınmalıdır. Cevap zamanı için genel bir formül

$$\begin{aligned} \text{Cevaplama zamanı} = & C_{CPU} * seq\text{-}\# \text{ insts} + C_{I/O} * seq\text{-}\# \text{ I/Os} \\ & + C_{MSG} * seq\text{-}\# \text{ msgs} + C_{TR} * seq\text{-}\# \text{ bytes} \end{aligned}$$

dir. Burada $seq\text{-}\#x$, sorgunun işletilmesi için ardarda yapılması gereken x ' lerin sayısıdır. x , bir talimat, I/O, masraf ya da bytes olabilir. Böylece paralel yapılan tüm işlemler ihmal edilir.

Örnek: 2.1.

Toplam masraf ile cevap zamanı arasındaki farkı Şekil 2.1'de bir soruya 1. ve 2. nodlardaki datalardan verilen cevabı hesaplayan örnekle gösterelim. Basitlik için sadece iletişim masraflarının gözönüne alındığını kabul ediyoruz.



Şekil 2.1. Bir Sorgu İçin Veri Transferi Örneği

C_{MSG} ve C_{TR} nin zaman biriminde ifade edildiğini kabul edelim. 1. Nod' dan 3. Nod' a X birim verinin 2. Nod' dan 3. Nod' a Y birim verinin iletiminin toplam masrafı

$$\text{Toplam zaman} = 2 C_{MSG} + C_{TR} * (x + y)$$

İletimler paralel yapılabileceğine göre, aynı sorgunun cevap süresi aşağıdaki gibi tahmin edilebilir.

$$\text{Cevaplama zamanı} = \max (C_{MSG} + C_{TR} * x, C_{MSG} + C_{TR} * y)$$

Cevap süresini minimize etmek paralel işletimin derecesini arttırmakla mümkündür. Ama bu toplam masrafında minimize edildiği anlamına gelmez. Tam tersine söz gelişi daha fazla yerel işlemler ve iletme sebep olarak toplam masrafı artırabilir. Toplam masrafın minimize edilmesi kaynak kullanımının gelişmesi anlamına gelir. Bu da sistem veriminin artması demektir. Pratikte bu ikisi arasında bir denge bulunması istenir. Bölüm 2.4 te toplam masraf ve cevap süresi kombinasyonunu herhangi biri daha ağırlıklı olarak optimizasyonuna ilişkin algoritmalar bulunmaktadır.

2.1.2. Veri Tabanı İstatistikleri

Bir veri işlem stratejisinin performansını etkileyen ana faktör işletim boyunca üretilen aracı ilişkilerin büyüklüğüdür. Ardışık bir işlem farklı bir noda yerleştirildiğinde, aracı ilişki, şebeke üzerinden iletilmelidir. Bu yüzden data transferlerinin boyutunu minimize etmek için ilişkisel cebirsel işlemlerin ara sonuçlarının boyutlarını tahmin etmek çok önemlidir. Tahminler, ilişkisel işlemlerin sonuçlarının kardinalitelerini tahmin eden formüller ve temel ilişkiler hakkındaki istatistiksel bilgilere dayanır. İstatistiklerin hassasiyeti ile bunları kullanma maliyeti arasında direkt bir çelişki vardır, istatistikler ne kadar hassassa maliyeti o kadar fazladır. $A = (A_1, A_2, \dots, A_n)$ simgeleriyle tanımlanmış ve $R_1, R_2, \dots, R_r$ olarak parçalanmış bir R ilişkisi için tipik bir istatistiksel datalar aşağıdaki gibidir;

- 1- Her A_i sembolü için $\text{lenght}(A_i)$ ile byte cinsinden boyu $\text{card}(\pi_{A_i}(R_j))$ ile R_j parçasının A_i sembolü için R_j parçasının A_i üzerine yansımalarının kardinalitesi ile A_i 'nin farklı değerlerinin sayısı gösterilir.
- 2- Sıralandırılabilen bir değerler kümesi (örn. tam sayılar, gerçel sayılar) üzerinde tanımlanan her A_i sembolünün domain' i için maksimum ve minimum değerler $\min(A_i)$ ve $\max(A_i)$ ile gösterilir.
- 3- Her A_i sembolünün domain' i için kardinalite $\text{card}(\text{dom}[A_i])$ ile gösterilir. Bu değer $\text{dom}[A_i]$ deki tekil değerlerin sayısını verir.

4- Her R_j parçasındaki kayıtların sayısı $card (R_j)$ ile gösterilir.

Bazen istatistiksel data bazı ilişki çiftleri için birleşme ayırıcı faktörleri, yani birleşmeye katılan kayıtların oranını da verir. R ve S ilişkilerinin SF_j ile gösterilen “birleşme seçici faktörü” 0 ve 1 arasında gerçel bir değerdir.

$$SF_j (R, S) = \frac{card (R \bowtie S)}{card (R) * card (S)}$$

Örneğin birleşme seçici faktörünün 0,5 olması çok geniş bir birleşmiş ilişkiye karşılık gelirken 0,0001 küçük birine karşı gelir. Birinci durumda birleşmenin seçiciliği kötü, ikincisinde iyidir deriz. Bu istatistikler ara ilişkilerin büyüklüğünü tahminde faydalıdırlar. Bir R ara bağlantısının büyüklüğünü aşağıdaki gibi tanımlanabilir.

$$büyüklük (R) = card (R) * length (R)$$

Burada $length (R)$, R ’ nin sembollerinin uzunluklarından hesaplanmış kayıtların uzunluğudur. $card (R)$ ’ nin ve R ’ deki kayıtların tahmini aşağıdaki kısımda verilen formüllerin kullanılmasını gerektirir.

2.1.3 Ara Sonuçların Kardinaliteleri

Sorguların ara sonuçlarının kardinalitelerinin değerlendirilmesinde veritabanı istatistikleri faydalıdır. Veritabanı hakkında iki tane basitleştirici kabul genellikle yapılır. Bir bağıntıdaki değişkenlerin dağılımı düzenli sayılır ve tüm değişkenlerin bağımsız olduğu yani bir değişkenin değerinin bir başkasını etkilemediği varsayılır. Bu iki kabul uygulamada genellikle yanlıştır, ancak problemi çözülebilir kılarlar. Aşağıda basit ilişkisel cebirsel işlemlerin (seçme, yansıma, kartezyen çarpım, birleşme, yarıbirleşme, toplanma ve fark) sonuçlarının kardinalitelerini tahmin edici formüller verilmektedir. İşlemsel bağıntılar R ve S ile gösterilmiştir. O işlemin sonuçlarına katılan bir işlem bağıntısının tuplelerinin oranı olan seçicilik faktörü SF_{OP} ile gösterilir, burada OP işlemi gösterilir.

Seçim

Seçimin kardinalitesi

$$Card(\sigma_F(R)) = SF_s(F) * card(R)$$

$SF_s(F)$ seçim formülüne bağlıdır ve aşağıdaki gibi hesaplanır;

$P(A_i)$ ve $\bar{I}(A_j)$ A_i ve A_j değişkenleri üzerindeki predicatesleri gösterir.

$$SF_2(A = value) = \frac{1}{card(\Pi_A(R))}$$

$$SF_s(A > value) = \frac{max(A) - value}{max(A) - min(A)}$$

$$SF_s(A < value) = \frac{value - min(A)}{max(A) - min(A)}$$

$$SF_s(p(A_i)(p(A_j))) = SF_s(p(A_i)) * SF_s(p(A_j))$$

$$SF_s(p(A_i)(p(A_j))) = SF_s(p(A_i)) + SF_s(p(A_j)) - (SF_s(p(A_i)) * SF_s(p(A_j)))$$

$$SF_s(A(\text{values})) = SF_s(A = value) * card(\text{values})$$

Yansıma

Yansıma çiftleme eliminasyonu ile de çiftleme eliminasyonsuz da olabilir. Biz çiftleme eliminasyonlu yansımayı gözönüne alıyoruz. Yansıtılmış değerler arasındaki korolasyonlar genellikle bilinmediği için, rasgele bir yansımanın hassas olarak değerlendirilmesi / incelenmesi zordur. Bununla birlikte bunun çok basit olduğu iki durum vardır. Eğer R bağıntısının yansıması tek bir A (değişkenine dayanmışsa)/ değişkeni üzerindeyse kardinalite, yansıma gerçekleştirildiğinde tuple lerin sayısıdır. Eğer yansıtılan değerlerden biri R'nin anahtarı ise

$$card(\Pi_A(R)) = card(R)$$

Kartezyen Çarpım

R ve S' nin kartezyen çarpımının kardinalitesi basitce

$$card(R \times S) = Card(R) * Card(S) \text{ dir.}$$

Birleşme (Join)

İlave bilgi olmaksızın bir birleşimin kardinalitesini bulmanın genel bir yolu yoktur. Birleşim kardinalitesinin üst sınırı kartezyen çarpımın kardinalitesidir. Dağıtık İNGRES gibi bazı sistemler oldukça kötümser olan bu üst sınırı kullanırlar. R^* (Selinger ve Adiba); bu üst sınırı, birleşim sonucunun kartezyen çarpımın sonucundan daha küçük olacağı gerçeğini yansıtmak bir katsayıya bölerek kullanır. Bununla birlikte tahminin basit olduğu sıkça bir durum vardır. Eğer R bağıntısı S bağıntısı ile R den A değişkeni , S'den B değişkeni üzerinde eşit birleşim yaparsa ve A R bağıntısının bir anahtarı ve B, S bağıntısının yabancıysa sonucun kardinalitesi aşağıdaki gibi tahmin edilebilir;

$$Card(R \bowtie_{A=B} S) = card(R)$$

Çünkü S'nin her tuple'li R'nin en fazla bir tuple'li ile eşleşebilir. Tabii ki B, S'nin bir anahtarı , A'da R'nin yabancıysa da aynı şey sözkonusudur. Ancak, R'nin tüm tuple' lerinin birleşime katıldığını varsaydığından bu tahmin sadece bir üst sınırdır. Diğer önemli birleşimler için istatistiksel bilgilerin parçası olarak birleşime seçicilik faktörünü kullanmak faydalıdır. Bu durumda sonuç kardinalitesi

$$Card(R \bowtie S) = SF_j * card(R) * card(S)$$

Yarıbirleşme (Semijoin)

R'nin S ile yarıbirleşiminin seçicilik faktörü S'nin tuple'lerinin yüzdesini verir. Yarıbirleşme seçicilik faktörü için bir yaklaşım (Heuner and Yao, 1979) da aşağıdaki gibi verilmiştir.

$$SF_{sj}(R \ltimes S) = \frac{card(\Pi_A(S))}{card(dom(A))}$$

Bu formül yalnızca S'nin A değişkenine bağlıdır. Bu yüzden S'nin A değişkeninin ($SF_{sj}(S.A)$ diye gösterilen) seçicilik faktörü denir ve S,A'nın herhangi bir diğer birleşebilir değişkeninin seçicilik faktörüdür. Bu yüzden yarıbirleşmenin kardinalitesi

$Card(R \bowtie S) = SF_{sj}(S.A) * card(R)$ şeklinde verilmiştir.

Birleşme (Union)

R ve S'nin birleşmenin kardinalitesini tahmin etmek , R ve S arasındaki çiftlemeler birleşmeyle kaldırıldığı için zordur. Sadece üst ve alt sınırların belirlenmesinde kullanılan formülleri veriyoruz.

$Card(R) + Card(S)$	ÜST
$Max. Card(R), Card(S)$	ALT

Dikkat edilmelidir ki R ve S'nin çiftlenen eleman içermedikleri varsayılmıştır.

Fark

Birleşme gibi burada da sadece alt ve üst sınırları veriyoruz. $Card(R-S)$ in üst sınırı $card(R)$, alt sınırı 0 dır.

2.2. Merkezileşmiş Sorgu Optimizasyonu

Bu bölümde merkezlenmiş sorgular için en çok tanınan iki sorgu optimizasyon yöntemini anlatacağız. Üç nedenden dolayı bu konunun anlatımı dağıtık sorgu optimizasyonu için ön koşuldur. Birincisi ; Bir dağıtık sorgu herbiri merkezlenmiş olarak çalışan yerel sorgulara dönüşür. İkincisi; dağıtık sorgu optiminasyon teknikleri genellikle merkezlenmiş sorgu optimizasyon tekniklerinin uzantılarıdır. Son olarak merkezlenmiş sorgu optimizasyonu daha basit bir problemdir; iletişim masraflarının minimize edilmesi dağıtık sorgu optimizasyonunu daha karışık bir hale getirir.

INGRES ve R optimizasyon algoritmaları merkezlenmiş versiyonlarının uzantısı olan dağıtık versiyonları vardır.

Bu sistemlerin optimizasyon teknikleri belirgin şekilde farklıdır. INGRES dinamik bir optimizasyon algoritması kullanılır, sistem R ise veritabanı istatistikleri kullanarak yoğun aramaya dayanır. Statik bir optimizasyon algoritması kullanılır. Belirtmek isterizki ticari Veri Tabanı Yönetim Sistem lerinin çoğu , etkinliği ve sorgu derlemesiyle uyumu nedeniyle yoğun araştırma yaklaşımını seçerler.

2.2.1 Ingres Algoritması

INGRES algoritması bir cebirsel sorguyu ardarda küçük paralara bölen dinamik bir sorgu optimizasyon algoritması kullanılır. Hesapsal cebirin iki fazını (ayırma, optimizasyon) kullanılır. Birçok değişkenli sorgu, ilk önce tek bir ortak değişken içeren sorgulara indirgenir. Sonra her tek değişkenli sorgu bir Tek Değişken Sorgu İşlemcisi ile çalıştırılır.

TDSİ o bağıntıya en uygun yaklaşım metodunu, ifadeye bağlı olarak (örn:index, sıralı tarama) seçerek, tek değişkenli bir sorgudaki bağıntıya yaklaşımı optimize eder.

Örneğin; İfade $\langle A=value \rangle$ şeklindeyse A değişkeninde mevcut bir index kullanılır. ancak eğer ifade $\langle A \neq value \rangle$ şeklindeyse , A'nın bir index i ise yaramayacaktır, sıralı tarama daha uygun olacaktır.

Anlatımımız QUEL dilinin "retrieve " komutu INGRES tarafından SQL 'nın "select" komutuna benzer şekilde kullanılan ana sorgu tipinde yoğunlaşacaktır. Bununla birlikte kitabın bütününde uygunluğu sağlamak için örnekleri anlatmak için SQL 'li kullanacağız. Algoritma ilk önce çok değişkenli işlemlerin ara sonuçlarının büyüklüklerini minimize etmek için çalışır ve ilk önce tek değişkenli işlemleri yapar.

$q(i-1) \dots q(i)$; $q(i-1)$ ve $q(i)$ olarak iki alt sorguya indirgenmiş bir sorgu olsun , $q(i-1)$ ilk işlenen alt sorgu , $q(i)$ ise onun sonuçlarını tüketen altsorgu olsun. n Değişkenli bir sorgu verildiğinde INGRES sorgu işlemcisi q' 'yu $q_1, q_2, q_3, \dots, q_n$ olarak alt sorgulara ayırır.

Bu ayırma iki temel teknik kullanılır; ayırma (koparma) ve yerine koyma. Bu teknikler bölümün geri kalan kısmında anlatılmaktadır. Ayırma (koparma) sorgu işlemcisiince uygulanan ilk tekniktir. q' nün sonucu olan ortak bir değişkene göre q sorgusunu q' (q'' şeklinde sorgulara ayırır. Eğer sorgu SQL ' de ifade edilirse aşağıdaki yapıda olur.

```

SELECT       $V_2.A_2, V_3.A_3, \dots, V_n.A_n$ 
FROM         $R_1 V_1, R_2 V_2, \dots, R_n V_n$ 
WHERE        $P_1(V_1.A_1)$ 
AND          $P_2(V_1.A_1, V_2.A_2, \dots, V_n.A_n)$ 

```

Burada FROM cümlesi bağıntılar (R_1) üzerinde tuple değişkenleri tanımlar; A_i ve A_i , R_1 bağıntısının değişkenlerinin listeleridir, P_1 R_1 den değişkenler içeren içeren bir tek değişkenli ifadedir; ve P_2 , $R_1, R_2, \dots, R_n$ bağıntılarının attribute' lerini içeren çok değerli bir ifadedir. Böyle bir sorgu ortak V_1 değişkeninin ayrılmasıyla q' ve ardından q'' olarak iki alt sorguya dönüştürülebilir.

```

q':  SELECT       $V_1.A_1. INTO R'_1$ 
FROM         $R_1 V_1$ 
WHERE        $P_1(V_1.A_1)$ 

```

Burada R' sorgunun devam etmesi için gerekli bilgiyi içeren geçici bir bağıntıdır.

```

q'': SELECT       $V_2.A_2, \dots, V_n.A_n$ 
FROM         $R_1 V_1, R_2 V_2, \dots, R_n V_n$ 
WHERE        $P_2(V_1.A_1, \dots, V_n.A_n)$ 

```

Bu basamak q ' sorgusunun V_i değişkeninin tanımlandığı bağıntının büyüklüğünü azaltır. Ayrıca yaratılan R_i bağıntısı , takip eden sorguları hızlandıracak özel bir yapıda tutulabilir.Örneğin R_i 'nin q ' nin birleşme değişkenleri üzerinde dilenmiş bir dosyada saklanması birleşmenin işlenmesini daha verimli yapar.

Ayırma olan seçme operasyonlarını çıkarır.Bu yüzden mümkün olduğu zaman ayırma sistematik olarak yapılır. Unutulmamalıdır ki eğer seçimin seçiciliği kötüyse performansı kçtü yönde etkiler.

Örnek 2.2

Ayırma tekniğini göstermek için aşağıdaki sorguyu uyguluyoruz .

"CAD/CAM Projesinde çalışan memurların adları "

Bu sorgu SQL 'de aşağıdaki gibi ifade edilebilir.

```
q1:  SELECT    E.ENAME
      FROM      E, G, J
      WHERE     E.ENO=G.END
      AND       G.JNO=J.JNO
      AND       JNAME= 'CAD/CAM'
```

Seçimlerin ayrılmasından sonra , q_1 ' in yerine q_{11} geçer ve onu q takip eder JVAR bir ara bağıntıdır.

```
q11:  SELECT    J.INO INTO JVAR
      FROM      J
      WHERE     JNAME= 'CAD/CAM'

q1:   SELECT    E.ENAME
      FROM      E, G, JVAR
      WHERE     E.ENO=G.END
      AND       G.JNO=JVAR.JNO
```

q₁₂: SELECT *G.ENO INTO GVAR*
 FROM *G.JVAR*
 WHERE *G.JNO=JVAR.JNO*

q₁₃ SELECT *E.ENAME*
 FROM *E, GVAR*
 WHERE *E.ENO=GVAR.END*
 AND *G.ENO=GVAR.ENO*

Başka alt sorguların olası olduğunu göz önüne alırız.

Böylece q₁ sorgusu q₁₁-q₁₂-q₁₃ sorgularına indirgenmiştir. q₁₁ sorgusu tek değişkenli olduğundan OVQP ile işletilebilir. Ancak q₁₂ ve q₁₃ tek değişkenli olmadığından ayırma ile indirgenemez. Daha fazla ayıramayan çok değişkenli sorgular (örnek 9.2 deki q₁₂ ve q₁₃ gibi) "tuple substitution" ile tek değişkenli sorgulara dönüştürülürler. n değişkenli q sorgusunu ele alırsak , bir değişkenin "tuples" değeriyle yer değiştirir, böylece (n-1) değişkenli bir sorgu oluşur. "Tuple " yer değiştirmesi aşağıdaki şekilde olur. İlk önce q sorgusundan birtek değişken tuple yer değişimi için seçilir. R1 ilişkisinde (ranging over relation) V1 değişken olsun. Sonra R1 içindeki her tli tuple için , q içindeki V1 tarafından olan atamalar tli deki gerçek değeriyle yer değiştirir; böylece (n-1) değişkenli q' sorgusu oluşur. Bu yüzden toplam q' sorgu sayısını R1 kartında görürüz. ? Tuple yer değiştirmesi aşağıdaki gibi belirtilebilir:

$$q'(t_{12}, V_2, V_3, \dots, V_n), t_{12} \in R_1 \quad q_1(V_1, V_2, \dots, V_n) \text{ yerini alır.}$$

Alt sorgu daha indirgenemez durumunda değilse , elde edilen her tuple için işlem tekrarlanır.

Örnek:2.3

q₁₃ sorgusu şöyle olsun:

SELECT *E.ENAME*
 FROM *E, GVAR*
 WHERE *E.ENO=GVAR.ENO*

GVAR değişkeniyle belirlenmiş ilişki tek özellik(alan) içermektedir. Sadece 2 tuple içerdiğini varsayalım ; <E1> ve <E2>.GVAR ın değiştirilmesi sonucu 2 adet tek değişkenli alt sorgu oluşur.

q₁₃₁ SELECT *E.ENAME*
 FROM *E*
 WHERE *E.ENO= 'E1'*

q₁₃₂ SELECT *E.ENAME*
 FROM *E*
 WHERE *E.ENO= 'E2'*

Bu sorgu daha sonra OVQP kullanılabilir.

INGRES 'in sorgu iyileştirme algoritması (INGRES-QOA) 2.1 Algoritmasında gösterilmiştir. Algoritma işlenecek tek değişkenli sorgu kalmayana kadar işlevini sürdürür. Çalışma şekli , ayırma metoduyla en kısa zamanda seçim ve projeksiyonları uyarlamaktır. Tek değişkenli sorguların sonuçları bilgi yapılarında (data structures)saklanır. Bu bilgi yapılarının daha sonraki sorguları iyileştirebilme ve OVQP ile kullanılabilme imkanı vardır. Ayırma işleminden sonra kalan indirgenemez durumdaki sorgular tuple substitution yapılmalıdır. MVQ' de belirtilen indirgenemez sorgular için , bir önceki sorgudan önemi bilinen en küçük ilişkiyle değişken yerdeğiştirme için seçilir. Bu basit metotla en az sayıda alt sorgu elde edebiliriz.

İndirgeme algoritmasıyla işlenen tek değişkenli sorgular daha sonra OVQP ile işlenir; OVQP sorgu kalifikasyonuna göre , ilişki için olası en iyi patikayı seçer.

Algoritma 2.1 INGRES QOA

input: MVQ: multivariable query with n variables

```

output: output:      result of execution
begin
output ( (
if n=1then
output ←run (MVQ)                {execute the one variable query}
else begin
{detach MVQ into m one-variable queries and one multivariable query}
OVQ1....OVQm, MVQ' ←MVQ
for i ←1 to m do
begin
output' ←run (OVQi)                {execute OVQi}
output ← output Uoutput'          {merge all results}
end-for
V ← CHOOSE-VARIABLE (MVQ') (V chosen for tuple substitution)
for each tuple t∈V do
begin
MVQ'' ← substitute values for t in MVQ'
output' ← INGRES -QOA (MVQ'')    {recursive call}
output ← output U output'        {merge all results}
end -for
end-if
end. {INGRES-QOA}

```

2.2.2 Sistem R Algoritması

Sistem R, çözüm alanının (“solution space”) yoğun aranmasını temel alan statik sorgulama optimizasyonu yapar (Selinger et al.,1979). Sistem R’nin optimizör girişini, bir SQL sorgusunun çözülmesiyle oluşan ilişkisel bir cebir ağacı (“algebra tree”) oluşturur. Çıkış ise “optimal” ilişkisel cebir ağacını kullanan bir işlem planıdır.

INGRES’de olduğu gibi birleşmelerden (“joins”) önce seçme operasyonlarının sistematik olarak işlemek yerine, Sistem R, ancak daha iyi bir strateji oluşturacaksa bu işlemi yapar. Optimizör aday ağaçlardan her birine bir fiyat atar ve en düşük fiyatlıyı tutar. Aday ağaçlar, sorgulamanın n ilişkisinin birleşik üslerinin kümülatif ve “associative” kurallarla permütasyonu ile elde edilir. Optimizasyon fiyatını düşürmek amacıyla alternatif ağaç sayısı dinamik programlamayla azaltılır.

Alternatif stratejiler seti dinamik olarak oluşturulup, iki birleşim kümülatif olarak eşit çıkarsa, en ucuzu muhafaza edilmektedir. Ek olarak, Kartezyan çarpımları içeren stratejiler mümkün olduğunca ortadan kaldırılır.

Aday bir stratejinin fiyatı I/O ve CPU fiyatlarının ağırlıklı birleşimiyle ortaya çıkartılır. Bu tip fiyatların kestirimi (derleme sırasında) bir fiyat modeli üzerine oluşturulur. Bu model düşük seviyeli her operasyon için bir fiyat formülü sağlar (örneğin, bir aralık yüklemiyse (“ranqe prediale”) B-ağacı indisi kullanımı seçimi.)

Çoğu operasyon için (tümüyle uyuma seçimi hariç), bu fiyat formülleri işlem elemanlarının kardinallik özelliği üzerine oturtulur. Veri tabanında saklanan ilişkilerin kardinallik bilgileri veri tabanı istatistiklerinde bulunur. Bunlar, Sistem R tarafından otomatik olarak idare edilir. Ara sonuçların kardinallikleri, operasyon seçim faktörlerine bağlı olarak kestirilir.

Optimizasyon algoritması iki ana adımdan oluşur. Önce, seçilen bir yükleme göre her ilişkiye en iyi erişim metodu kestirilir (bu en az fiyatlı olandır.) İkinci olarak, her R ilişkisi için, en iyi birleşim dizisi kestirilir. Burada R’ye önce, en iyi tek-ilişki erişim metoduyla erişilir. En ucuz diziliş, en iyi işlem planının temelini oluşturur.

Birleşmeler konusunda iki algoritma bulunmaktadır. bunlardan sadece birisi optimal olmaktadır. İki ilişkinin birleşmesi için, “tuple”leri ince okunan ilişkiye “harici” denir. “Tuple”leri harici ilişkiden elde edilen değerlerle bulunan diğerine ise “iç ilişki” denir. Her iki birleşim metoduyla da verilecek önemli bir karar, iç ilişkiye giden en ucuz erişim yolunu tanımlamaktır.

“İç içe geçmiş çevrimler” (“nested loops”) adı verilen ilk metod, iki ilişkinin çarpımından oluşur. Harici ilişkinin her “tuple”si için, birleşik yüklemi tatmin eden iç ilişkinin “tuple”leri, sonuç ilişkiden birer birer seçilir. Birleşme özelliği üzerinde bir indis, iç ilişkiye ulaşmak için çok etkili bir erişim yoludur. Bi indis olmadığında, n_1 ve n_2 sayfalarından oluşan ilişkilerde, bu algoritmanın fiyatı $n_1 * n_2$ olur ki, n_1 ve n_2 yüksek olduğunda fiyat engelleyici olabilir.

“Birbiri içine karışarak birleşme” adı verilen ikinci yöntem, iki ayrı sınıfa dahil ilişkilerin müşterek niteliklere sahip olmak üzere birleştirilmesinden ibarettir. Müşterek niteliklere dayalı indeksler erişim yolu olarak kullanılabilir. Eğer birleşim kriteri bir eşitlik ise, n_1 ve n_2 sayfalık iki ilişkinin birleştirilme maliyeti, sırasıyla $n_1 + n_2$ ile orantılıdır. Dolayısıyla, bir eşit birleşim olduğunda ve ilişkiler önceden sınıflandırılmış olduğu takdirde, daima bu yöntem seçilir. Eğer bu ilişkilerin yalnızca biri sınıflandırılmış ya da hiçbiri sınıflandırılmamışsa, iç içe geçmiş halkalardan oluşan algoritmanın maliyetinin, birleştirme ve sınıflandırma maliyeti ile karşılaştırılması gerekir n sayfa sınıflandırılmasının maliyeti, $n \log n$ ile orantılıdır. Genelde, geniş kapsamlı ilişkiler söz konusu olduğunda, karıştırarak birleştirme algoritmasını sınıflandırarak uygulamak yararlıdır.

Seçilmiş bir proje için birleşim sorguine ait R sistemi optimizasyon algoritmasının basitleştirilmiş versiyonu Algoritma 2.2’de gösterilmiştir. Bu sistem, iç içe iki halkadan ibaret olup; birincisi sorguteki herbir ilişkiye en iyi tek ilişkili geçiş

yöntemini seçer; ikincisi ise mümkün olabilecek tüm birleşim sırası permütasyonlarını inceler (n ilişkili $n!$ permütasyon vardır ve sorgu için en iyi geçiş stratejisini seçer. Permütasyonlar, dinamik yapıda olan bir alternatif stratejiler ağacı tarafından meydana getirilir. Öncelikle, her bir ilişkinin bir diğer ilişki ile birleşimi ve sonra da üç ilişkinin birleşimi dikkate alınır. n ilişkinin birleşimleri optimum noktaya getirilinceye kadar bu böyle devam edip gider. Aslında, bazıları faydasız olduğu için algoritma mümkün olabilecek permütasyonların tümünü ortaya çıkarmaz. Daha önce incelediğimiz gibi, Kartezyen ürünlerini içeren permütasyonlar en yüksek maliyet düzeyinde birbirlerine eşdeğer stratejiler olduğundan yok edilmiştir. Konunun anlaşılmasına yarayan bu iki unsur vasıtasıyla, incelenen stratejilerin sayısı, $n!$ yerine 2^n bağıntısı ile ifade edilebilen bir üst sınıra sahiptir.

Algoritma 2.2. RQOA

input: QT: query tree with n relations

output: the result of execution

begin

for each relation $R_i \in QT$ **do**

begin

for each access path A_{Pij} to R_i **do**

 determine cost (A_{Pij})

end-for

 best_ $AP_i \leftarrow A_{Pij}$ with minimum cost

end-for

for each order $(R_{i1}, R_{i2}, \dots, R_{in})$ with $i = 1, \dots, n!$ **do**

begin

 build strategy $(\dots((\text{best } AP_{i1} \Join R_{i2}) \Join R_{i3}) \Join \dots \Join R_{in})$

 compute the cost of strategy

end-for

output $\leftarrow$ strategy with minimum cost

end {R.QOA}

Örnek 2.4

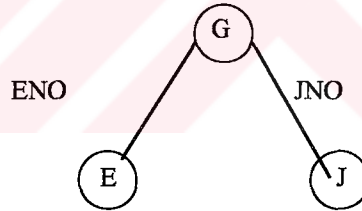
Bu algoritmayı mühendislik veri tabanında q_1 sorgui (bkz. örnek 2.2) ile açıklayalım, q_1 'in birleşim grafiği şekil 2.2'de verilmiştir. Kısaca ifade etmek gerekirse, E-G kenarı üzerindeki ENO işareti E.ENO=G.ENO ifadesini, G-J kenarı üzerindeki JNO işareti ise G.JNO=J.JNO ifadesini belirtir.

Aşağıdaki indeksleri varsayalım:

E, ENO üzerinde bir indekse sahiptir

G, INO üzerinde bir indekse sahiptir

J, INO ve JNAME üzerinde bir indekse sahiptir.



Şekil 2.2 q_1 Sorgunun Birleşim Grafiği

Algoritmanın ilk halkasının aşağıdaki en iyi tek ilişkili geçiş yolunu seçtiğini varsayalım:

E: Sıralı tarama (E üzerinde hiçbir seçim olmadığından)

G: Sıralı tarama (G üzerinde hiçbir seçim olmadığından)

J: JNAME üzerindeki indeks (JNAME'e dayalı olarak J üzerinde bir seçim olduğundan) Alternatif stratejiler ağacının dinamik yapısı Şekil 2.3'de gösterilmektedir. Maksimum birleşim derecesi sayısının 3!; olduğuna ve dinamik

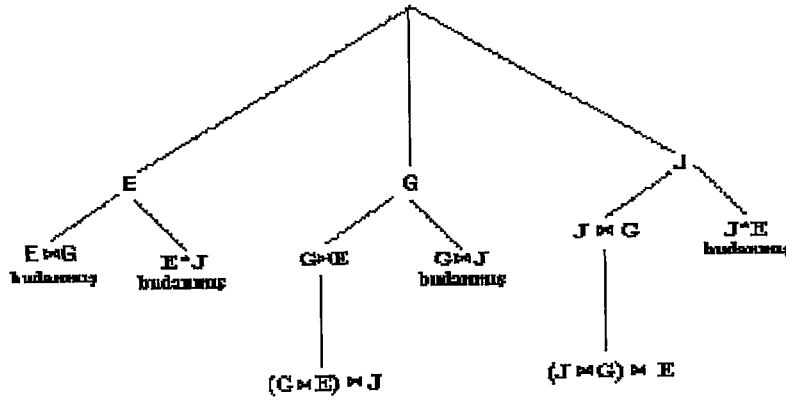
araştırmanın Şekil 2.3’de görüleceği gibi daha az alternatifi dikkate aldığına dikkat ediniz.

“Pruned” (budanmış) işaretli işlemler dinamik olarak ortadan kaldırılmıştır. Ağacın ilk düzeyi en iyi tek ilişkili geçiş yöntemini belirtmekte; ikinci düzey ise, bunların herbiri için herhangi bir diğer ilişki ile en iyi birleşim yöntemini göstermektedir ($E \bowtie J$) ve ($J \bowtie E$) stratejileri, (diğer stratejiler tarafından) engellenebilen Kartezyen ürünleri olduğundan budanmıştır. ($E \bowtie G$)’nin ($G \bowtie E$)’den, ($G \bowtie J$) nin ise ($J \bowtie G$)’den daha yüksek bir maliyete sahip olduğunu varsayalım. Karşılıklı yer değişim suretiyle eşdeğer nitelikte daha iyi birleşim dereceleri mevcut olduğundan, bunlar budanmıştır. Geriye kalan iki olasılık, ağacın üçüncü düzeyinde verilmiştir. En iyi toplam birleşim derecesi ($((G \bowtie E) \bowtie J)$ ile $((J \bowtie G) \bowtie E)$)’nin en az maliyetli olanıdır. Seçmeli nitelik üzerinde faydalı indekse ve G ve E ‘nin birleşim noktalarına direkt geçiş imkanına sahip olan sadece ikincisi olup; bu nedenle, aşağıdaki erişim yöntemleriyle seçilir:

JNAME üzerindeki indeksi kullanarak J’yi seçin.

JNO üzerindeki indeksi kullanarak G ile birleştirin.

ENO üzerindeki indeksi kullanarak E ile birleştirin.



Şekil 2.3 Alternatif Birleşim Sıralamaları

R Sistemi'nin performans ölçümü (Mackert ve Lokman, 1986), CPU maliyetinin sorgu toplam maliyeti içindeki önemli katkısını kanıtlamaktadır. İlişkilerin ana bellek tamponları içine alınabildiği durumlarda, optimizör tahminlerinin doğruluk derecesi genellikle iyi olmakla beraber; ilişkiler boyut olarak arttıkça ve disk üzerine yazıldıkça bu doğruluk derecesinin kalitesi düşer. Daha iyi biçimde önceden kestirim için dikkate alınması gereken önemli bir parametre de tampon kullanımıdır.

2.3 Kısmi Sorgulamalarda Birleşme Algoritması

Bölüm 2.2'de görmüş olduğumuz gibi, birleşimlerin sıraya konulması merkezi sorgu optimizasyonunun önemli bir unsurudur. Kısımlar arasındaki birleşimler iletişim maliyetini artırabileceğinden, bir dağınık yapı içindeki birleşim sıralaması daha da çok önemlidir. Parçalı sorgulardaki birleşimlerin sıraya konulması için iki temel yaklaşım mevcut olup; bunlardan birincisi birleşimlerin sıraya konulmasını direkt olarak yapmaya çalışır, diğeri ise iletişim maliyetlerini en aza indirmek için birleşimlerin yerine yarı birleşim kombinasyonları kullanır.

2.3.1 Birleşme Sorgulaması

Bazı algoritmalar, yarı birleşimler kullanmaksızın birleşimlerin derecelendirilmesini direkt olarak optimize eder. Dağınık INGRES ve R algoritmaları, yarı birleşimlerden ziyade birleşimlerin kullanıldığı algoritmaları temsil eder. Bu bölümün amacı, birleşim sıralamasının ortaya çıkardığı güçlüğü vurgulamak ve birleşim sorgularının optimizasyonu için yarı birleşimlerin kullanılmasını inceleyen bir sonraki bölümü motive etmektir.

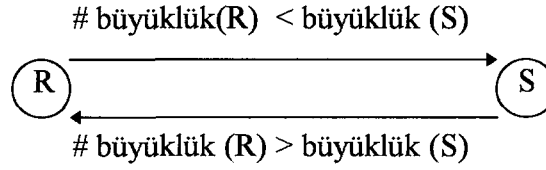
İlgi odağını ana konular üzerine çekmek için bir dizi varsayım gereklidir. Sorgu parçalara dayalı olduğu ve bu şekilde ifade edildiği için, aynı ilişkiye sahip

parçalar ile farklı ilişkiler içeren parçalar arasında herhangi bir ayırım yapmamız gerekmez. Notasyonu basitleştirmek bakımından, “ilişki” terimini belirli bir yerde depolanmış bir parçayı belirtmek için kullanacağız. Keza, ilgiyi birleşim sıralaması konusu üzerinde yoğunlaştırmak için, (seçme, projeksiyon gibi) redaksiyon işlemlerinin birleşim öncesi ya da birleşim süresince yerel olarak yapılmış olduğu varsayımı ile yerel işleme maliyetini ihmal edeceğiz (seçme işleminin ilk önce yapılmasının her zaman verimli olduğu anlamına gelmediğini unutmayınız). Dolayısıyla sadece işlemsel ilişkileri farklı yerlerde depolanmış olan birleşim sorgularını dikkate alacağız. İlişki aktarımlarının birleştirme modundan ziyade ayarlanmış modda yapıldığını varsayacak ve son olarak da, bir sonuç yerinde veri üretimine ait transfer mahiyetini dikkate almayacağız.

İlk olarak, daha basit bir problem olan, tek bir birleşimde bilgi aktarımını ele alalım. R ve S farklı yerlerde depolanmış ilişkileri ifade etmek üzere, sorgu RXS’dir. Aktarılabilecek ilişkinin seçimi, daha küçük olan ilişkiyi büyük olanın bulunduğu yere göndermektir ki bu da, şekil 2.4’de gösterildiği gibi iki olasılığı artırır. Bu seçimi yapmak için, R ve S’nin büyüklüğünü değerlendirmemiz gerekir. Şimdi de, birleştirilecek ikiden fazla ilişkinin mevcut olduğu bir durumu düşünelim. Bir tek birleşim durumunda olduğu gibi, birleşim derecelendirmesinin amacı, daha küçük işlem verilerini iletmektir. Buradaki güçlük, birleştirme işlemlerinin ara sonuçların büyüklüğünü azaltabilmesinden veya artırabilmesinden kaynaklanır. O nedenle, birleşim sonuçları büyüklüğünün önceden tahmin edilmesi zorunlu, fakat o nispette de zordur. Bu probleme getirilebilecek bir çözüm, tüm alternatif stratejilerin iletişim maliyetlerin tahmin edip en iyisini seçmektir. Ancak, stratejilerin sayısı ilişkilerin sayısı ile orantılı olarak hızlı biçimde artır. R sistemi tarafından kullanılan bu yöntem, sorgunun sık olarak uygulanması halinde ilgili genel giderlerin çabuk biçimde amortismanını sağlamasına rağmen optimizasyonu yüksek mahiyetli kılar.

(R) büyüklüğü < (S) büyüklüğü ise

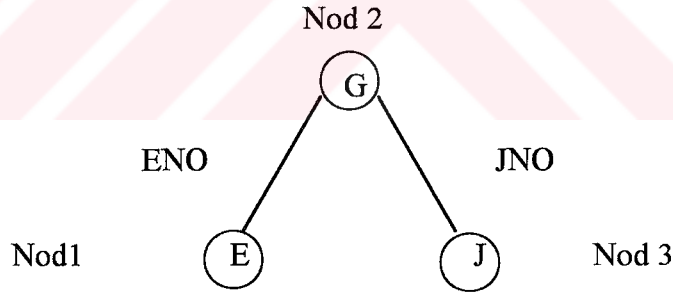
(R) büyüklüğü > (S) büyüklüğü ise



Şekil 2.4 Binary Operasyonlarda Operand'ların Transferi

Örnek 2.5

Birleşim grafiği Şekil 2.5'de verilen ve: ilişkisel cebir cinsinden ifade edilen aşağıdaki sorgu ele alalım. Üç ilişkinin yerleri hakkında belli varsayımlar yapmış olduğumuzu da dikkatimizden kaçırmayalım. Bu sorgu en az beş ayrı biçimde uygulanabilir. Bu stratejileri, (RXJ yeri)'nin R ilişkisi J mahalline aktarılmıştır" anlamına gelmek üzere, aşağıdaki programlarla tarif edebiliriz.



Şekil 2.5 Dağıtık Sorgunun Birleşim Grafiği

1. E→2. yer
2. yer E' = E ⋈ G'yi hesaplar
- E' →3. Yer
3. Yer E' ⋈ J'yi hesaplar
2. G→1 Yer
- 1.Yer E' = E ⋈ G'yi hesaplar

$E' \rightarrow 3.$ Yer

3.Yer $E' \bowtie J'$ 'yi hesaplar

3. $G \rightarrow 3.$ Yer

3.Yer $G' = G \bowtie J'$ 'yi hesaplar

$G' \rightarrow 1.$ Yer

1.Yer $G' \bowtie E'$ 'yi hesaplar

4. $J \rightarrow 2.$ Yer

2.Yer $J' = J \bowtie G'$ 'yi hesaplar

$J' \rightarrow 1.$ Yer

1.Yer $J' \bowtie E'$ 'yi hesaplar

5. $E \rightarrow 2.$ Yer

$J \rightarrow 2.$ Yer

1.Yer $E \bowtie J \bowtie G$ hesaplar

Bu programların birini seçmek için, şu büyüklüklerin bilinmesi veya önceden kestirilmesi gereklidir, (E), (G), (J), $(E \bowtie G)$ ve $(G \bowtie J)$. Ayrıca, eğer ele alınan konu yanıt süresi ise,

2.3.2. Yarı Birleşme Tabanlı Algoritmalar

Bu bölümde, yarı katılımlı operasyonların birleşik querilerinin toplam maliyetini düşürmek için nasıl kullanıldığını göstereceğiz. Yarı katılım teorisi 1981 yılında Brenstein ve Chiu'da tarif edildi. Biz aynı tahminleri yapıyoruz, bölüm 2.3.1'deki gibi. Önceki bölümde tarif edilen birleşik yaklaşımın ana özeti bütün operand ilişkilerinin nodler arasında transfer edilme zorunluluğuydu. Yarı katılım bir ilişki için seçimin yaptığı kadar boyut düşürücü rol oynar.

R ve S gibi iki ilişkinin birleşimi A sonucunu doğurur, bunlar 1 ve 2 nodlerinde depolanır, aşağıdaki kurallar kullanılarak 1 veya tüm operand ilişkilerin yerine yarıkatılım ile diğer ilişkiler kullanılarak hesaplanır.

Bu üç yarıkatılımdan birinin tercih edilmesi için onların her birinin maliyet tahminine ihtiyaç vardır.Eğer bütün operand ilişkilerinin diğer nodye gönderme maliyetinden yarı-katılımın üretim ve gönderme maliyeti daha düşük ise kullanımı yararlıdır.Potansiyel yararlarını hayal etmek için yarıkatılımın,iki alternatifin maliyetlerini karşılaştıralım.

$$\begin{aligned} R \bowtie_A S &\Leftrightarrow (R \bowtie_A S) \bowtie_A S \\ &\Leftrightarrow R \bowtie_A (S \bowtie_A R) \\ &\Leftrightarrow (R \bowtie_A S) \bowtie_A (S \bowtie_A R) \end{aligned}$$

Bir sonraki program bölüm 2.3.1 notasyonlarını kullanarak yarıkatılım işlemlerinin kullanılmasındır.

1. $\Pi_A(S) \rightarrow \text{Nod 1}$
2. Nod 1 hesaplar $R' = R \bowtie_A S$
3. $R' \rightarrow \text{Nod 2}$
4. Nod 2 hesaplar $R' \bowtie_A S$

$C_{TR} * \text{büyüklük}(R)$ ölçeğinde; basitliği sağlamak için iletişimde maliyet tahmininde kullanılan C_{MSG} sabitini yok varsayarsak daha büyüktür.Böylece iki seçenek arasında aktarılan bilgiyi karşılaştırabiliriz.Ortak bazlı alagoritmanın taşınmasının fiyatı Nod 2 nin R'siyle ilişkilendirilir.Yarı ortak bazlı alagoritmanın maliyet basamakları yukarıdaki 1 ve 3 aşamadır. Böylece yarı ortak bazlı görüş aşağıdaki şartı yerine getirirse daha iyidir.

$$\text{büyüklük}(\Pi_A(S)) + \text{büyüklük}(R \bowtie_A S) < \text{büyüklük}(R)$$

Yarı ortak bazlı görüş yeterli olarak indirgen şekli ile yapıldığında çok daha olumludur. Bu anlamda R kendisinin bağı katılımı mümkün olmaktadır.Çokca kabul edilen bir görüşe göre bütün kayıtların R'ye ortak katılımı daha iyidir.Çünkü yarı ortak katılımın görüşü ortak özellik yansıtmada ek bir işlem gerektirmektedir.Projenin

kodlanma maliyeti küçük diziler biçiminde yapılırsa minimize edilebilir. Böylece ortak özelliklerin transfer maliyetleri düşürülebilir. Şu da önemle belirtilmelidir ki bu sistemlerden ne biri, nede diğeri en iyisidir, bunlar birbirini tamamlayıcı mahiyettedir.

Genelde yarı ortak bazlı görüş, işlem giren elemanlarda çoklu ortak sorgulu ilişkilerinde ölçeğini düşürmek için daha yararlı olabilir. Bununla beraber sorgulama optimizasyonu (en iyilemesi) bundan genelde çok daha karışık bir işlemdir. Şekil 2-5 e tekrar başvurarak $E \bowtie G \bowtie J$ is $E' \bowtie G' \bowtie J$, where $E' = E \bowtie G$ and $G' = G \bowtie J$ arasındaki ortak ilişkiyi tartışalım. Bir önceki baş vurduğumuz ortak algoritmayı bireysel ortaklıklarda kullanabiliriz.

Buna örnek olarak $E'' = E \bowtie (G \bowtie J)$ 'dır.

Bununla beraber birden fazla yan ortak bazla görüş kullanarak işlem gören operandların ilişkileri hacmini daha da ileri biçimde düşünebiliriz. örnek olarak E' bir önceki programda E'' olarak yerine kullanılırsa $E \bowtie (G \bowtie J)$ biçiminde ifade edilebilir. Bu yoldan E yarıortak bazlı sistemde şu biçimde indirgenebilir. $E \bowtie (G \bowtie J)$. Yarı ortak bazlı sıralama, yarı ortak program biçiminde de algılanabilir. Benzer olarak yarı ortak programlar herhangi bir sorgulama tanımlı olabilir. Mesela J ortak bazlı programda $J \bowtie (G \bowtie E)$ biçiminde indirgenebilir. Özelde bütün ilişkiler sorgular indirgenecek zorunluluğu olmamasına rağmen bu ilişkileri sonuç istemlerinde kaale almayabiliriz.

Verilen örnekte potansiyel çeşitli yarı ortak bazlı ilişkiler vardır. Olasılıkların sayısı üstel olarak , ilişkilerin sayısına bağlıdır. Her ilişkide R indirgeyicisinden çok daha iyi indirgeyen tam indirgeyici denen optimal yarı ortak bazlı bir program vardır. Esas problem tam indirgeyiciyi bulabilmektir. Bütün mümkün olan yarı ortak bazlı programların indirgeyici boyutunun pekiştirilmesinde seçilen en iyi basit bir yöntem vardır. Problemi tanımlayabilmekte iki yön vardır.

1. Dairesel sorgulama denilebilecek bir sınıf sorgulama vardır ki bunların kendi ortak şekli vardır ve bunların tam indirgeyecileri yoktur.

2. Diğer sorgulama ağaç sorgulaması olarak tanımlanabilir. Tam indirgeyecileri vardır, fakat bilinen yarı ortak bazlı programların sayısı ilişkilerin üssü olarak

tanımlayıcı yaklaşımda zordur. Bir sonraki bahiste bu problemin çözüm yollarını tartışacağız.

Örnek 2.6

İzleyen ilişkiye ek olarak CITY niteliğine E(ET olarak yeniden adlandırarak) ve J(yeniden adlandırarak JT) mühendislik uygulaması

ET(ENO, ENAME, TITLE, CITY)

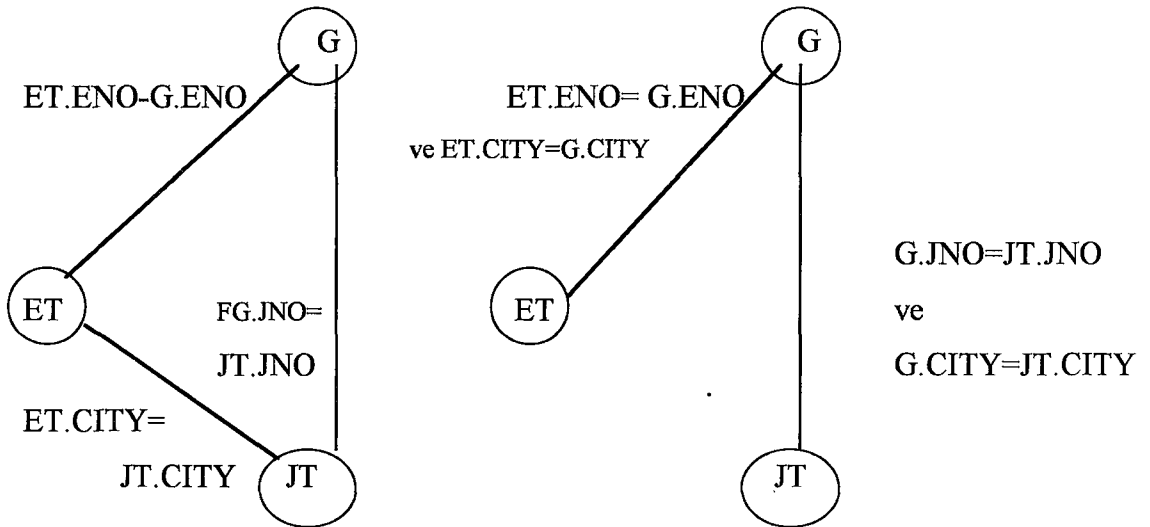
G (ENO, JNO, RESP, DUR)

JT (JNO, JNAME, BUDGET, CITY) oluşturulursa

İzleyen SQL uygulamasında şehirde yaşayan bütün çalışanların isimleri projeye yerleştirilmiştir.

```
SELECT    ET NAME
FROM      ET, G, JT
WHERE     ET.ENO=G.ENO
AND       G.ENO=JT.ENO
AND       ET.CITY = JT.CITY
```

Şekil 2.6'da bu sorgulama gösterilmektedir.



Şekil 2.6. Dairesel sorgunun dönüştürülmesi

2.6'daki örnekte tüm indirgeyici oluşmamıştır. Gerçekte yarı ortak sorgulayıcı program indirgemeyi içersede her ilişkinin yarattığı kayıt sayısının çarpımla artması sebebiyle bu görüş etkin değildir. Dairesel şekilden ağaç şekline geçiş için değişim bir çözüm önerisi olarak gözardı edilen değişikliklerin korunması diğer uygun tahmin acılarıyla giderilebilir.

Şekil 2.6 da örnek olarak ET, JT açısı ek ön yüklenmeleri $ET.CITY = G.CITY$ ve $G.CITY = JT.CITY$ yi dönüştürerek $ET.CITY = JT.CITY$ yi ifade eder. Böylece dairesel olmayan sorgulama dairesel sorgulamaya eşit. Ön tahminlerin eklenmesi G ilişkisinde CITY niteliği tanımlar. Meğer ki CITY nin değerini ET veya G den yollanmış olmasın.

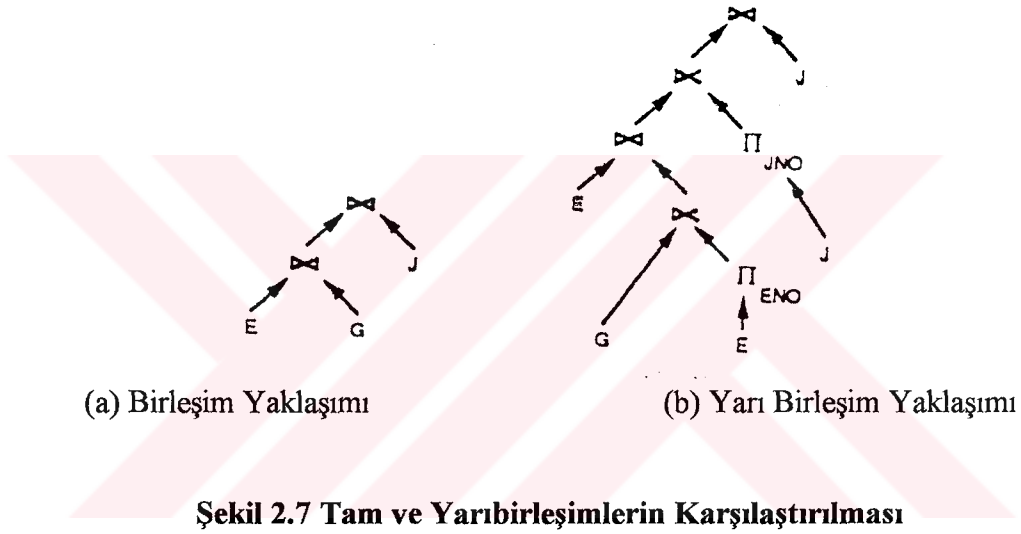
Ağaç sorgulamasında tam indirgeyiciler olsada problem NP de bulmak zordur. Bununla beraber zincir sorgulama ile adlandırılan önemli bir sorgulama sınıfı vardır ki polinomial algoritmalarda kullanılır. Zincirleme sorgulama ortak bir şekilde düzenlenebilir ve her ilişki sadece sonraki ilişki ile irtibatlandırılır. Daha da ötesinde sorgulamanın sonucu zincirin sonundadır. Örnek olarak şekil 2.5 zincirleri sorgulama örneğidir. Çoğu sistem tek yarı ortak bazlı indirgemeyi seçerler ilişki büyüdüğünde çünkü tüm indirgeme algoritmasını kurmak zordur.

2.3.3. Tam Ve Yarı Birleşimlerin Karşılaştırılması

Yarı ortak bazlı indirgeme ortak bazlı anlayışa göre daha fazla işlem gerektirir, olasılıkla daha hiçbir operanda ihtiyaç gösterir. Şekil 2.7 de yarıortak E ve G ikilisinin arasında farkları gösterir. Şekil 2.5 de E ve G ilişkisinde diğer tarafa bir tek ilişki ile gönderme yapmaktadır. E, o hal olarak tamamlanmaktadır. Yarıortak kullanıldığında G nin dönüşüm ilişkisi saf dışı edilir. Bunun yerine E nin ilişkisini nitelik değerlerinin dönüşümü G nin ilişkinin yerine geçer. Bunu takip ederek G nin karşılaştırılan kayıtları yerine E nin ilişkisi ilave edilir, böylece zincir tamamlanmış olur.

Eğer ortak niteliklerin uzunluğu bütün kayıtlarından ve yarı ortak bazlı seçilmiş kayıtlardan daha küçükse, yarı ortak bazlı görüşün sonucu iletişim maliyetlerinde belirgin bir tasarruf sağlayabilir.

Yarıortak bazlı görüşün kullanılması yerel işlem maliyetlerinde iyi bir düşüş sağlayabilir. Eğer iki ortak ilişkiye iki kere ulaşılması durumuna düşülürse, örnek olarak E ve J Şekil 2.7 de iki kere ulaşıldığını gösterir. Daha da ötesinde iki orta düzey ilişkiyi yaratan yarıortak bazlı görüş indisini temel bazlı ilişkilerde düşer. İletişim maliyeti temel alınan baskın karar faktörü değilse yerel bölgesel ağlarda yarıortak bazlı görüş çok iyi bir fikir olmayabilir .



Yarıortak bazlı görüş hızlı network lerde hala yararlıdır. Eğer çok iyi bir bit dizisi kurulursa ve çok iyi bir seçim yapılır. Bir bit dizisi BA(1:n) ortak niteliklere haiz değerlerin kodlanmasında mevcut tek bir ilişkinin değerlemesinde yararlı olabilir.

BÖLÜM 3. DAĞITIK SORGU OPTİMİZASYON ALGORİTMALARI

Bu bölümde daha önceki bölümlerde bahsi geçen 4 sorgulama optimizasyon algoritmasından bahsedeceğiz. Dağıtılmış INGRES' in İndirgeme Algoritması, Sistem R* Algoritması, SDD-1 Algoritması ve AHY olarak da adlandıracağımız Apers, Hevner ve Yao Algoritmaları. Bunları burada açıklayacak olmamızın nedeni başka algoritmalara örnek teşkil etmeleridir. Aralarındaki farklar;

1- Dağıtılmış INGRES' de zamanlama dinamik, diğerlerinde ise zamanlama statiktir.

2- SDD-1 ve R* algoritmalarının esas amaçları toplam harcamayı azaltmak iken, dağıtılmış INGRES ve AHY hem işlem zamanını ve hem de toplam harcamayı azaltmayı amaçlar.

3- SDD-1 için harcama fonksiyonu optimizasyon faktörleri mesaj uzunluğudur. AHY mesaj uzunluğunu ve mesaj sayısını kullanır. Lokal işlem harcamalarını da gözönüne alan sistem R*, optimizasyon faktörü olarak mesaj uzunluğunu, mesaj sayısını, I / O ve CPU harcamalarını kullanır. Dağıtık INGRES hem I / O + CPU harcamalarını hem de mesaj uzunluğunu kullanır.

4- SDD-1 ve AHY' nin network yapısı wide area noktadan noktaya' dır. INGRES ve R* algoritmaları hem local hem de wide area networklerde çalışabilir.

5- SDD-1 ve AHY optimizasyon tekniklerinde yarı birleştirmeleri (semi-joins) kullanır. Dağıtık INGRES ve R* algoritmaları kendi benzerleri INGRES ve Sistem R algoritmalarının da kullandığı gibi merkezi sorgulama optimizasyonu algoritmalarının kullandıkları birleştirmelere benzer birleştirmeler kullanırlar.

6- Her algoritma, bilgiler hakkında istatistikler tutar ve kullanır. Şekil 3.1' de de gösterildiği gibi yarı birleştirme (semi-join) algoritmaları daha çok bilgi kullanırlar.

7- INGRES parçalanma problemini çözümleyebilir.

Bu 4 algoritma arasındaki farklar şekil 3.1' de gösterilmiştir. Diğer kısımlarda bu algoritmaları inceleyeceğiz.

Algoritmalar	Opt.zaman-laması	Obj. fonksiyonlar	Opt. faktörler	Ağ Topolojisi	Yarıbir-letirmeler	Staist	Fragman
Dağıtık INGRES	Dinamik	Cevaplama zamanı ve toplam maliyet	Mesaj uzunluğu işleme maliyeti	Genel veya yaygın	Yok	1	Yatay
R*	Statik	Toplam maliyet	Mesaj sayısı mesaj uzunluğu ID, CPU	Genel veya lokal	Var	1,2	Yok
SDD-1	Statik	Toplam maliyet	Mesaj uzunluğu	Genel	Var	1,3,4,5	Yok
AHY	Statik	Cevaplama zamanı toplam maliyet	Mesaj sayısı mesaj uzunluğu	Genel	Var	1,3,5	Yok

Şekil 3.1. Sorgu Optimizasyon Algoritmalarının Karşılaştırılması

3.1. Dağıtık Ingres Algoritması

Dağıtık INGRES' de kullanılan sorgulama optimizasyonu algoritması Merkezi INGRES' den çıkarılmıştır.Bu nedenle verilen bir sorgulamanın optimizasyonunu dinamik olarak gerçekleştirir.Bu algoritmanın esas fonksiyonu haberleşme harcamalarını ve zamanını azaltmaktır.Fakat bu iki amaç birbirleriyle uyuşmadıklarından (Haberleşme harcamalarını arttırmak pekala kullanılan zamanı azaltabilir), algoritma bu iki amaçtan birine daha fazla öncelik sağlayabilir.Algoritma aynı zamanda fragmentasyon tekniklerinden de yararlanabilir, fakat basitliğinden dolayı sadece horizontal fragmentasyon tekniğini kullanır.

Hem genel hem de yayın ağları düşünüldüğünden, sorgulama optimizasyonu ağ yapısını da hesaba katar.Yayın ağlarında bir bilgi ünitesi tek bir transfer ile bir alandan diğer tüm alanlara dağıtılabilir ve algoritma bunun avantajlarını kullanabilir.Mesela yayın ağlarından fragmanları çoğaltmak ve sonra da paralelliği maksimize etmekte yararlanılabilir.

Sorgulama Optimizasyon Algoritması için gerekli olan girişler, ilişkisel tuple şeklinde ifade edilmiş bir sorgulama (birleştirici normal formda) ve şema bilgisidir (ağ yapısı, fragmanların yeri ve büyüklükleri). Merkezileştirilmiş versiyonda olduğu gibi, dağıtık sorgulama optimizasyonunu bir bilgiyi açığa çıkarma olayında açıklıyoruz. Bu algoritma, algoritmanın başlatıldığı ve adına 'master nod ' da denilen alanda gerçekleştirilmektedir. D-INGRES-QOA olarak adlandırılan algoritma, Algoritma 3.1' te verilmektedir.

Algoritma 3.1 D-INGRES-QOA

input: MVQ: multivariable query

output: result of the last multivariable query

begin

for each detachable OVQ_i in MVQ **do** (run all one-variable queries)

run (OVQ_i):

end-for

MVQ' list ← REDUCE (MVQ) {replace MVQ by n
irreducible queries} (2)

while n > 0 do {n is the number of irreducible queries} (3)

begin

{choose next irreducible query involving the smallest fragments}

MVQ' ← SELECT_QUERY (MVQ' _list) (3.1)

{determine fragments to transfer and processing nod for MVQ'}

Fragment-nod-list ← SELECT_STRATEGY (MVQ') (3.2)

{move the selected fragments to the selected nods}

for each pair (F, S) in Fragment-nod-list do (3.3)

move fragment F to nod S

end-for

run (MVQ') (3.4)

n ← n - 1

end-while {output of the algorithm is the result of the last MVQ'}

end {D-INGRES-QOA}

Tüm tek değişkenli (seçme ve kısıtlama) ve birbirinden ayrılabilir sorgular local olarak gerçekleştirilir.Sonra bir indirgeme algoritması (Wong ve Youssefi,1976) orjinal sorguya uygulanır.İndirgeme, tüm ayrıştırılamaz ve tek değişkenli alt sorgulamaların ayrıştırma ile izole edildikleri safhadır.Tek değişkenli alt sorgular dikkate alınmaz, çünkü ilk adımda işlem görmüşlerdir.Adım 2’ deki indirgenemez sorgulamalar ve her fragmanın büyüklüğü baz alınarak,en az iki değişkene sahip olan bir sonraki sorgulama MVQ’ na adım 3.3 ve 3.4 uygulanır.Adım (3.2) , MVQ’ yu işlemek için en iyi stratejiyi seçer.Bu strateji (F,S) listesi ile açıklanır.Burada F,S alanına transfer edilecek fragmanı gösterir.Adım (3.3), tüm fragmanları işlem görecekleri alanlara transfer eder.Son olarak adım (3.4), MVQ sorgusunu çalıştırır.Eğer arta kalan alt sorgular varsa algoritma 3. adıma döner ve bir sonraki iterasyonu gerçekleştirir.Aksi takdirde algoritma son bulur.

Optimizasyon (3.1) ve (3.2) adımlarında gerçekleşir.Algoritma alt sorgular ve bunlar arasındaki bağımlılıklar ve ilişkileri oluşturmuştur.(İlişkisel cebir ağacı örneğine benzer şekilde).Adım (3.1)’ de, bir sonraki alt sorgulama için en basit seçenek, bir öncesi olmayan ve daha küçük fragmanlarla ilgilenen bir sonraki sorguyu almaktır.Bu, bundan sonra oluşacak sonuçları minimize eder..Mesela, q diye bir sorgunun alt kümeleri q_1 (q_3 , q_2 (q_3 olsun ve q_1 tarafından referans edilen fragmanlar, q_2 tarafından referans edilenlerden daha küçükse q_1 seçilsin.Network’ ün yapısına bağlı olarak, bu seçim birbirine bağlı-ilişkili fragmanlara sahip olanlar tarafından etkilenebilir.Epstein bu seçim hakkında bazı detaylar verir.

Daha sonra, seçilen n-değişkenli alt sorgu çalıştırılmalıdır.Ayrılma ile indirgenemeyen böyle bir sorgu şekil 3.2’ de gösterilmiştir.Bir ilişki fragmanlanmış olabileceğinden ve değişik alanlarda tutulabileceğinden, alt sorgunun daha alt parçalara ayrılması imkansızdır.

Örnek 3.1.

Nod 1	Nod 2
E ₁	E ₂
G	J

1. Execute the entire query $(E \bowtie G \bowtie J)$ by moving E_1 and G to node 2
2. Execute $(E \bowtie G) \bowtie J$ by moving $(E_1 \bowtie G)$ and G to node 2, and so on.

Adım (3.2)'de oluşan bir diğer optimizasyon problemi, hareketi sağlanacak fragmanların seçimiyle gerçekleştirilecek sorgulamanın nasıl yürütüleceği ve işlemin yapılacağı alanın seçimidir. N -değişkenli bir ilişki için, $n-1$ ilişkinin fragmanları geri kalan ilişkinin olduğu, mesela adına R_p diyelim, alana transfer edilmeli ve orada çoğaltılmalıdır. Ayrıca geriye kalan ilişki k eşit parçaya bölünmelidir ki paralellik artsın. Bu metod *fragmanla ve çoğalt* adıyla anılır ve fragman yedeklerini-benzerlerini kullanır.

Kalan ilişkinin seçimi ve partitionlara ayırmanın olacağı k işlem alanlarının sayısı, esas fonksiyon ve ağ yapısı ile ilgilidir. Unutmamak gerekir ki çoğaltma yayın ağlarında, noktadan noktaya (point-to-point) ağlardakinden daha ucuzdur. Ek olarak, işlem alanlarının sayısının seçimi zaman ve harcamalar arasında karar vermeyi gerektirir. Alan sayısının büyümesi işlem zamanını azaltır, fakat toplam harcamayı artırır, özellikle haberleşmenin fiyatını yükseltir.

Epstein' da verilen formüller ya harcamaları ya da işlem zamanını azaltmak içindir. Bu formüller giriş olarak fragman büyüklüğünü, yerini ve ağ tipini kullanırlar. Her iki harcamayı da azaltabilirler ama birine daha fazla ağırlık vererek. Bu formülleri açıklamak için, haberleşme harcamalarını azaltmanın kurallarını veriyoruz. Bu kurallar çok komplekstir. Biz p eden assumption' ları kullanıyoruz; n -değişkenli sorgulamada $R_1, R_2, \dots, R_n$ gibi n ilişki vardır. R_i , node j 'de saklanan R_i 'nin fragmanını gösterir. Ağ üzerinde m alan vardır. Sonuç olarak, $CC_k (\# \text{ bytes})$, $1 < k < m$ ile $\# \text{ bytes}$ ' in k alanına transferinin haberleşme harcamalarını gösterir.

Haberleşme harcamalarını azaltmanın kuralı, ağ yapısını ayrı olarak ele alır. Öncelikle yayın ağını ele alalım. Bu durumda

$$CC_k (\# \text{ bytes}) = CC_1 (\# \text{ bytes})' \text{ dir.}$$

Kural şu şekilde oluşturulabilir:

if $\max_{j=1,m} (\sum_{i=1}^n \text{büyüklük} (R_i^j)) > \max_{i=1,n} (\text{büyüklük} (R_i))$
then işlem alanı daha fazla bilgiye sahip olan j
else işlem alanı daha fazla ilişkiye sahip olan R_p

Eğer eşitsizlik sağlanırsa, bir alan sorgu için gerekli daha fazla ve en büyük ilişkiden daha büyük bilgiye sahip olduğundan işlem alanı olacaktır. Aksi takdirde bilgiden daha büyük ilişkiye sahip olan alan işlem alanı olacaktır. Dolayısıyla bu ilişki R_p ve işlem alanları da bunun fragmanlarına sahip olanlar olacaktır.

Şimdi noktadan-noktaya ağırları ele alalım. Bu durumda

$$CC_k (\# \text{ bytes}) = k * CC_1 (\# \text{ bytes})$$

Haberleşmeyi azaltan R_p açık olarak en büyük ilişkidir. Alanları sorgulama için gerekli sahip oldukları bilgi uzunluklarına göre büyükten küçüğe doğru sıralandıklarını gözönüne alırsak

$$\sum_{i=1}^n \text{size}(R_i^j) > \sum_{i=1}^n \text{size}(R_i^{j+1})$$

if $\sum_{i \neq p} (\text{size}(R_i) - \text{size}(R_i^j)) > \text{size}(R_p^j)$

then

$k=1$

else

$$\sum_{i \neq p} (\text{size}(R_i) - \text{size}(R_i^j)) \leq \text{size}(R_p^j)$$

Bu kural işlem alanının alması gereken bilgi, o alan işlem alanı olmasaydı gönderecek olacağı bilgiden daha küçükse o alanı işlem alanı olarak seçer. Kuralın **then** kısmı alan 1' in R_p fragmanını sakladığını kabul eder.

Dağıtık INGRES çözüm alanının kısıtlanmış bir çözümü ile karakterize edilebilir. Global optimizasyon kararlarına bakılmadan her adım için ayrı optimizasyon teknikleri seçilir. Buna rağmen hatalı local bir kararı düzeltebilir. Kısıtlanmış çözüme alternatif çözüm tüm olası stratejileri gözönüne alarak en iyi stratejiyi seçmektir. (R* algoritması tarafından kullanılır.) Epstein ve Stonebraker' da iki yaklaşım simüle edilir ve transfer edilen bilgi bazında karşılaştırılır. Bu çalışmanın önemli bir sonucu, sorgunun 3' ten fazla ilişkiye sahip olması durumunda exhaustive aramanın diğerine büyük üstünlük sağladığıdır. Başka bir sonuç da, dinamik aramanın, sonuçların gerçek büyüklüklerini verdiğinden yararlı olduğudur.

3.2. R* Algoritması

R* dağıtık sorgu optimizasyonu, sistem R' nin optimize edicisine yapılmış bir ektir. Bu nedenle tüm olası stratejilerin araştırılıp en ucuz harcamanın bulunduğu derleme bir metod kullanır. Bu stratejilerin önceden bilinmesi pahalı olsa da, tümleyici araştırmanın getirdiği karışıklık sorgunun sık icra edilmesi durumunda avantaj sağlayabilir. Selinger ve Adiba tarafından açıklanan algoritma fragmentasyonu kullansa da, Sistem R* ne fragmentasyonu ne de çoğaltmayı destekler. Bu nedenle R* algoritması temel olarak sadece ilişkilerle ilgilenir. Sorgulamanın derlenmesi, sorgulamanın başlatıldığı master alandan koordine edilen dağıtılmış bir görevdir. Master alandaki optimize edici alanlar arası tüm kararları, işlem alanının seçimi-fragmanlar-bilgi transfer metodu gibi, verir. Sorgulamada kullanılan ilişkilere sahip diğer alanlardan olan Apprentice alanları local kararları (alandaki birleştirmelerin sayısı gibi) verir ve sorgu için local erişim planları üretirler. Sistem R* algoritmasının esas amacı lokal işlem ve haberleşme harcamalarını da hesaba katan genel harcamaları hesaplamaktır.

Şimdi Sistem R* sorgu optimizasyonu algoritmasını açıklayalım. Algoritmaya yapılan giriş, ilişkisel cebir ağacı (sorgu ağacı), ilişkilerin lokalizasyonu ve onların istatistikleri olarak expressed lokalize edilmiş sorgudur. Algoritma, R*-QOA prosedüründe Algoritma 3.2' de tanımlanmaktadır.

Algoritma 3.2. R*-QOA

input : QT : sorgu ağacı

output : strat : minimum fiyat stratejisi

begin

for each relation $R_i \in QT$ **do** (tüm ilişkiler için)

begin

for each access path AP_{ij} to R_i **do** (tüm ilişki yolları için)

determine $cost(AP_{ij})$ (AP_{ij} harcamalarını hesapla)

end-for

$best_AP_i \leftarrow AP_{ij}$ minimum harcama ile

end

for each order $(R_{i1}, R_{i2}, \dots, R_{in})$ with $i = 1, \dots, n!$ **do** (tüm sıralar için)

begin

build strategy $((best_AP_{i1} \bowtie R_{i2}) \bowtie R_{i3}) \bowtie \dots \bowtie R_{in}$

(strateji oluştur)

compute the cost of strategy (strateji harcamasını oluştur)

end-for

$strat \leftarrow$ strategy with minimum cost (minimum harcamalı strateji)

for each nod k storing a relation involved in QT **do** (QT içinde yer

alan

ilişkilere sahip her k alanı için)

begin

$LS_k \leftarrow$ local strategy (strategy, k)

send (LS_k , nod k) (each local strategy is optimized at nod k)

(gönder (LS_k , alan k) (her lokal strateji k alanında optimize edilmiştir }

end-for

end. (R*-QOA }

Merkezileşmiş yapıda olduğu gibi, optimize edici birleştirme sırasını, birleştirme algoritmasının (içiçe döngü ya da merge ederek birleştirme) ve her bir fragman için erişim yolunu (örneğin lineer arama) seçmelidir. Bu kararlar istatistikleri ve sonuçların büyüklüğünü saptayan formülleri temel alırlar. Ek olarak optimize edici birleştirme alanlarını ve alanlar arası bilgi transfer metodunu seçmelidirler. İki ilişkiyi birleştirmek için üç aday alan vardır: İlk ilişkinin alanı, 2. ilişkinin alanı, 3. bir alan (3. bir ilişkinin de birleşebileceği). R* algoritmasında alanlar arası data transferi için iki metod desteklenir.

1. *Ship-whole*. Tüm ilişki, ilişki alanına taşınır ve orada geçici bir ilişkide saklanır. Eğer birleştirme algoritması merge birleştirmeyi kullanıyorsa, ilişki saklanmak zorunda değildir.
2. *Fetch-as-needed*. Dış ilişki lineer olarak taranır ve her bir ünite için birleştirme değeri iç ilişkiye gönderilir. İç ilişkiden uyuyan üniteler seçilir ve dış ilişkiye gönderilir.

Bu iki metod arasında yapılacak seçim son derece açıktır. Hepsini taşımak daha fazla bilgiyi transfer eder ama mesajlar azalır. Bu nedenle az sayıdaki ilişkilerin hepsini taşımak daha iyidir. Aksine eğer ilişki büyükse bu ilişkiye uyan az eleman varsa, gerekli elemanlar gerekli görüldüğünde fetch edilebilirler. R* algoritması tüm birleştirme ve transfer metodlarını dikkate almaz çünkü bunlardan bazıları değersizdir.

Attribute'u A olan S adlı bir iç ilişkiye sahip R dış ilişkisi verilsin. Bu ilişki için 4 birleştirme stratejisi vardır. Aşağıda LC lokal işlem fiyatını (I/O+ CPU zamanı) ve CC kominikasyon fiyatını gösterir. Banod indirgemek amacıyla sonucu oluşturmanın maliyetini gözönüne almıyoruz.s, bir kayıda R denk düşen ortalama kayıtlara S verilen ad olmak üzere;

$$s = \frac{\text{card}(S \bowtie_A R)}{\text{card}(R)}$$

1. Strateji : Dış ilişkinin tümünü iç ilişkinin olduğu yere taşı. Bu durumda dış kayıtlar, ulaştıkları anda S ile birleşebilirler. Böylece

$$\begin{aligned} \text{Toplam maliyet} = & LC(\text{retrieve card } (R) \text{ tuples from } R) \\ & + CC(\text{büyüklük } (R)) \\ & + LC(\text{retrieve } s \text{ tuples from } S) * \text{card } (R) \end{aligned}$$

2. Strateji : İç ilişkinin tümünü dış ilişkinin olduğu yere taşı. Bu durumda iç kayıtlar, ulaştıkları anda birleşemezler ve T adlı geçici bir ilişkide konumlandırılırlar. Böylece

$$\begin{aligned} \text{Toplam maliyet} = & LC(\text{retrieve card } (S) \text{ tuples from } S) \\ & + CC(\text{büyüklük } (S)) \\ & + LC(\text{store card } (S) \text{ tuples in } T) \\ & + LC(\text{retrieve card } (R) \text{ tuples from } R) \\ & + LC(\text{retrieve } s \text{ tuples from } T) * \text{card } (R) \end{aligned}$$

3. Strateji : Dış ilişki için gerekli iç ilişkinin kayıtlarını ihtiyaç duyulduğunda fetch et. Bu durumda R' daki her kayıt için, S alanına bir birleştirme attribute'u gönderilir. Sonrada o değere uygun düşen S kayıtları s' I, vardıkları anda birleşmek üzere R alanına gönderilirler. Böylece

$$\begin{aligned} \text{Toplam maliyet} = & LC(\text{retrieve card } (R) \text{ tuples from } R) \\ & + CC(\text{length } (A)) * \text{card } (R) \\ & + LC(\text{retrieve } s \text{ tuples from } S) * \text{card } (R) \\ & + CC(s * \text{length } (S)) * \text{card } (R) \end{aligned}$$

4. Strateji : İlişkilerin ikisinide üçüncü bir alana gönder ve birleşmeyi orada yap. Bu durumda, iç ilişkinin kayıtları 3. bir alana gönderilir ve T geçici ilişkisinde tutulur. Daha sonra dış ilişkinin kayıtları da vardıkları anda birleşmek üzere bu üçüncü alana gönderilirler. Böylece

$$\begin{aligned}
\text{Toplam maliyet} = & LC (\text{retrieve card } (S) \text{ tuples from } S) \\
& + CC (\text{büyüklük } (S)) \\
& + LC (\text{store card } (S) \text{ tuples in } T) \\
& + LC (\text{retrieve card } (R) \text{ tuples from } R) \\
& + CC (\text{büyüklük } (R)) \\
& + LC (\text{retrieve } s \text{ tuples from } T) * \text{card } (R)
\end{aligned}$$

Örnek 3.2

JNO attribute' u üzerinde bir j dış ve G iç ilişkilerine sahip olduğumuzu düşünelim. Ayrıca J ve G' nin ayrı alanlarda olduğunu ve JNO attribute' nda G için bir indeks olduğunu da varsayalım. Olası sorgulama gerçekleştirme stratejileri aşağıdaki gibi olur.

1. Tüm J ilişkisini G' nin alanına aktar.
2. Tüm G ilişkisini J' nin alanına aktar.
3. Her J kaydına uygun düşen G kayıtlarını gerekli oldukları anda fetch et.
4. G ve J' yi üçüncü bir alana aktar.

R* algoritması her strateji için toplam maliyeti hesaplar ve en ucuzunu seçer. J(G birleşmesini izleyen hiç bir operasyon olmadığı verildiğinden, strateji 4 açıkça en pahalı maliyete sahip olan stratejidir, çünkü iki ilişkinin de 3. bir alana aktarılmasını ister. Eğer J' nin büyüklüğü G' nin büyüklüğünden çok daha fazla ise, 2. strateji haberleşme maliyetini azaltır ve lokal işlem maliyeti 1. ve 3. stratejilere oranla daha ucuz ise en iyi strateji olarak kabul edilebilir. Fakat, 1. ve 3. stratejilerin indeks kullanması, lokal işlem maliyetinin 2. stratejiden daha ucuz olması sonucunu doğurur.

Eğer 2. strateji en iyi değilse, seçim 1. ve 3. arasından yapılmalıdır. Bu iki stratejide de lokal işlem maliyeti birbirine eşittir. Eğer J büyükse ve G' nin sadece küçük bir parçası J ile uyuyorsa 3 en iyi stratejidir. Aksi takdirde, J küçük ve G' nin bir çok kaydı uyuyorsa, 1. strateji en iyidir.

Konu başlığı olarak algoritma, ilişki birleşme sırası, birleşme metodları (seçme ve birleşme algoritmalarını da kapsar), sonucun alındığı alan, iç ilişkiye ulaşım yolu ve alanlar arası transfer modu özelliklerinin permutasyonlarıyla en iyi stratejinin aranması olarak kabul edilebilir. Böyle bir algoritmanın karmaşıklığı içindeki ilişki sayısıdır. Gerçekten, R^* algoritması, sistem R^* optimize edicisi gibi, dinamik programlamayı kullanarak alternatiflerin sayılarını azaltır. Dinamik programlama ile, alternatiflerin ağacı dinamik olarak oluşturulur ve yararsız seçimlerin elimine edilmesi metodu ile ağaç indirgenir.

Lohman ve Mackert sistem R^* optimize edicisinin performans değerini lokal ağ benzeri yüksek hızlı ağlar bazında ve orta hızlı wide area network' lerde gösterdiler. Testler, lokal işlemin maliyetinin, wide area network' ler de bile, performans üzerindeki azımsanmayacak bir katkısını gösterdi. Özel olarak, dağıtık birleşmede, tüm iç ilişkinin aktarılmasının gerekli olduğunda fetch etmeye olan üstünlüğü gösterildi.

3.3 SDD-1 Logaritması

SDD-1 sorgulama optimizasyon algoritması, kendisinden önce oluşturulan 'hill-climbing' (tepeye tırmanma) adlı ilk dağıtık sorgu optimizasyon algoritmasından türemiştir. Bu algoritmada, bir ilk uygun çözümü, maliyette ilerleme sağlanamayınca kadar kendi içinde recursively olarak hesaplanır. Algoritma yarı birleşmeleri, fragmentasyon ya da replikasyonları kullanmaz. Geniş alandaki noktadan noktaya ağlar için geliştirilmiştir. Sonuçların, son alana taşıma maliyetleri gözardı edilir. Bu algoritma genel olarak herhangi bir objektif fonksiyonu, işlem zaman ve maliyeti de içine alarak minimize edebilir.

Örnek 3.3.

Mühendislik veri tabanlı E, S, J ve G ilişkilerini içeren aşağıdaki sorgui kullanarak, tepeye tırmanma algoritmasını inceleyelim.

“CAD/CAM projesinde çalışan mühendislerin maaşlarını bulunuz”.

İlişkisel cebir cinsinden sorgu:

$\Pi_{SAL} (S \bowtie_{TITLE} (E \bowtie_{ENO} (G \bowtie_{JNO} (\sigma_{JNAME=*CAD/CAM}(J))))))$ şeklindedir.

$C_{MSG} = 0$ ve $C_{TR} = 1$ olduğunu varsayalım. Ayrıca yerel bilgi işlemeyi ihmal edelim.

Buna göre, veri tabanı:

İlişki	Büyükük	Yer
E	8	1
S	4	2
J	1	3
G	10	4

olur.

Bu örneği basitleştirmek için, her ilişkinin birleşim yeri uzunluğunun 1 olduğunu, yani herhangi bir ilişkinin büyüklüğünün o ilişkin kardinalitesine eşit olduğunu varsayalım. Ayrıca, ilişkinin yerleşiminin keyfi olarak sayıldığını kabul edelim. Birleşim seçilebilirliğine dayalı olarak, $büyükük (EMS) = büyükük (E)$, $büyükük (J \bowtie G) = 2 * büyükük (J)$ ve $büyükük (G \bowtie E) = büyükük (G)$ olduğunu biliyoruz.

Sadece veri transferleri dakkate alındığında, ilk rantabl çözüm, sonuç yeri olarak 4.Yeri seçmektir ve bu da şu programı üretir:

$ESo: E \rightarrow 4.Yer$

$S \rightarrow 4.Yer$

$J \rightarrow 4.Yer$

$Toplam maliyet (ESo) = 4 + 8 + 1 = 13$

Herhangi bir başka çözümün maliyeti yukarıdaki alternatiften daha büyük olacağı için, bu çözüm doğrudur. Örneğin eğer 2.Yer sonuç yeri olarak seçilip, tüm ilişkiler bu yere iletilirse, toplam maliyet

$$\begin{aligned} \text{Toplam maliyet} &= \text{maliyet } (E \rightarrow \text{nod } 2) + \text{maliyet } (G \rightarrow \text{nod } 2) + \\ &\text{maliyet}(J \rightarrow \text{nod } 4) = 19 \end{aligned}$$

olacaktır. Benzer biçimde, 1.Yerin ya da 3.Yeri sonuç yeri olarak seçilmesi halinde toplam maliyet, sırasıyla 15 ve 22 olur.

Bu programın (ES'diyelim) kısımlara ayrıştırılmasının bir yolu şöyledir:

$$ES_1: E \rightarrow \text{nod } 2$$

$$ES_2: (E \bowtie S) \rightarrow \text{nod } 4$$

$$ES_3: J \rightarrow \text{nod } 4$$

$$\text{Toplam maliyet } (ES') = 8 + 8 + 1 = 17$$

İkinci bir ayrıştırma alternatifi (ES'') de aşağıdaki gibidir:

$$ES_1: S \rightarrow \text{nod } 1$$

$$ES_2: (S \bowtie E) \rightarrow \text{nod } 4$$

$$ES_3: J \rightarrow \text{nod } 4$$

$$\text{Toplam maliyet } (ES'') = 4 + 8 + 1 = 13$$

Yukarıdaki alternatiflerin her ikisi de Eso'nın maliyetinden daha büyük ya da buna eşit olduğu için, Eso nihai çözüm olarak saklanır. Algoritma tarafından gözardı edilen daha iyi bir çözüm:

$$B: J \rightarrow \text{Nod } 4$$

$$G' = (J \bowtie G) \rightarrow \text{Nod } 1$$

$$(G \bowtie E) \rightarrow \text{Nod } 2$$

$$\text{Toplam maliyet } (B) = 1 + 2 + 2 = 5$$

‘Tepeye tırmanma’ algoritması şöyledir: Algoritmanın girişleri, sorgulamanın grafiği, ilişkilerin yerleri ve ilişki istatistikleridir. İlk lokal işlemin bitmesinden hemen sonra, alanlar arası transferleri de gözönüne alan global işleme planı olan uygun bir çözüm seçilir. Bu çözümde tüm ilişkileri aday bir sonuç alanına aktarma işlemini gerçekleştirecek stratejilerin maliyetleri hesaplanır ve en ucuzu seçilir. İlk stratejimiz ES_0 olsun. Optimize edici, ES_0 , ES_1 ve ES_2 adlı iki parçaya ayrılır. ES_1 birleşmede rol alacak olanlar ilişkiyi diğer alana gönderecek olan strateji parçasıdır. İlişkilerin ikisinde lokal olarak birleştirilirler ve sonuçta oluşan ilişki seçilmiş olan sonuç alanına gönderilirler. (Bu durumda ES_2) eğer ES_1 ve ES_2 ’nin gerçekleştirilme maliyetleri artı lokal işlem maliyeti ES_0 ’dan az ise, ES_0 , ES_1 ve ES_2 ile replace edilir. Aynı işlem sırayla ES_1 ve ES_2 ’ye artık hiçbir yararın sağlanamayacağı ana kadar uygulanır. Eğer n yollu birleşmeler kullanılırsa ES_0 iki değil n alt plana bölünecektir.

$$\text{Maliyet } (R \bowtie_A S) = C_{MSG} + C_{TR} * \text{büyüklük } (II_A(S))$$

$$\text{Kar } (R \bowtie_A S) = (1 - SF_{SJ}(S.A)) * \text{büyüklük}(R) . C_{TR}$$

‘Tepe tırmanma’ algoritması, initial bir uygun çözüm yola çıkarak iteratif olarak bunu geliştirdiğinden hızlı bir algoritmadır.

Algoritma 3.3. SDD-1 QOA

input: QG: query graph with n relations; statistics for each relation

output: ES: execution strategy

begin

ES $\leftarrow$ local-operations (QG);

```

modify statistics to reflect the effect of local processing
 $BS \leftarrow \phi$  { set of beneficial semijoins }
for each semijoin  $SJ$  in  $QG$  do
    if  $cost(SJ) < benefit(SJ)$  then
         $BS \leftarrow BS \cup SJ$ 
    end-if
end-for
while  $BS \neq \phi$  do { selection of beneficial semijoins }
begin
     $SJ \leftarrow$  most-beneficial ( $BS$ ) (  $SJ$  : semijoin with max (benefit - cost) )
     $BS \leftarrow BS - SJ$  { remove  $SJ$  from  $BS$  }
     $ES \leftarrow ES + SJ$  { append  $SJ$  to execution strategy }
    modify statistics to reflect the effect of incorporating  $SJ$ 
     $BS \leftarrow BS -$  nonbeneficial semijoins
     $BS \leftarrow BS \cup$  new beneficial semijoins
end-while
( assembly node selection )
 $AS(ES) \leftarrow$  select node  $i$  such that  $i$  stores the largest amount of data after all
    local operations
 $ES \leftarrow ES \cup$  transfers of intermediate relations to  $AS(ES)$ 
( postoptimization )
for each relation  $R_i$  at  $AS(ES)$  do
    for each semijoin  $SJ$  of  $R_i$  by  $R_i$  do
        if  $cost(ES) > cost(ES - SJ)$  then
             $ES \leftarrow ES - SJ$ 
        end-if
    end-for
end-for
end. (SDD-1- QOA)

```

Initialization fazı, yararlı yarı birleşmelerin bir kümesini $BS = (SJ_1, SJ_2, \dots, SJ_k)$ ve lokal işlemi içeren bir ES işlem stratejisini oluşturur. Bir sonraki faz en yararlı yarı

birleşme SJ_i 'yi iteratif olarak seçerek, database istatistiklerini ve BS 'i değiştirerek, yararlı yarı birleşmeleri BS ' den seçer. Değişme, SJ ' de bulunan R ilişkisinin istatistiklerini ve bu ilişkiyi kullanan BS ' de kalan yarı birleşmeleri etkiler. İteratif faz, BS ' deki tüm yarı birleşmeler işlem stratejisine eklenince sona erer. Yarı birleşmelerin ES ' e eklenme sıraları, yarı birleşmelerin işlem sıralarıdır.

Sonraki faz, gerekli tüm bilgilerin transferinin maliyetini gözönüne alarak bir birleşme alanı saptar. Bu alan, bilginin transferinin maliyeti en düşük olan alan olacaktır. Son olarak, optimizasyon sonrası fazı birleşme alanında saklanan ilişkileri etkileyen yarı birleşmelerin işlem stratejilerinden ayrılmasına müsaade eder. Bu faz gereklidir, çünkü birleşmenin yeri tüm yarı birleşmeler sıralandığında seçilir. SDD-1 optimize edicisi ilişkilerin başka bir alana transfer edilebileceğini baz alır. Bu kural, yararlı yarı birleşmelerin kabul edilmesinden sonra yapılan ve birleşme sahasında tutulan seçimler için geçerli değildir. Bu nedenle bazı yarı birleşmeler yanlışlıkla yararlı olarak kabul edilebilirler. İşte bunları işlem stratejisinden ayırmak optimizasyon sonrası fazının rolüdür.

Kendisinin atası olan 'tepe-tırmanma' algoritması gibi SDD-1 algoritması da lokal olarak optimal stratejileri seçer. Bu nedenle yararlanmayı arttırıp diğer yarı birleşmelerin maliyetini azaltan yüksek maliyetli yarı birleşmeleri dikkate almaz. Bu yüzden algoritma global manada bir minimum maliyet seçimi yapamayabilir.

3.4 AHY (Apers-Hevner-Yao) Algoritmaları

Apers, Hevner ve Yao ya işlem zamanını ya da maliyeti azaltan ve yarı birleşmeleri kullanan bir algoritma ailesi önerdiler. Metod, basit sorgulara uygulanabilen optimal algoritmalar kümesini baz alır. İlk lokal işleminden sonra, her bir ilişki kendi içinde sadece tek bir ortak attribute' u bulunduruyorsa ve bu sorgunun da çıkışı ise, basit sorgular olarak adlandırılır.

Basit sorgular, optimal olarak çözümlenebilirler. Çoklu birleşim özelliği taşıyan genel bir sorgu, her birine optimal algoritmaların uygulanabileceği basit

sorgulara ayrılır.Genel sorgu, lokal işlem maliyetini dikkate almaz, yani esas amacı SDD-1' de de olduğu gibi haberleşme maliyetidir.Bu doğrulanır çünkü, algoritma ağ olarak noktadan-noktaya wide area network' leri kabul eder.

AHY algoritması bir ilişkiyi önce lokal olarak işler, sonra sorguyu her birine yarı birleşmeler yoluyla optimal stratejilerin uygulanabileceği basit sorgulara ayırır, en son olarak da bu stratejileri tek bir işlem stratejisine birleştirir.Bu nedenle, tüm sorgular yarı birleşmeler yoluyla indirgenirler ve sonra tek bir sonuç sahasına gönderilirler.Orada sonuçlar hesaplanır.Algoritma sorgudaki tüm ilişkilerin ayrı yerlerde tutulduğunu varsayar, fakat algoritma mantığı,n tek bir alandaki çoklu ilişkileri de çözümlemek için genişletilebilir.

Öncelikle basit sorgular için 2 algoritma sunuyoruz: Toplam maliyet optimizasyonu için SERİ algoritma ve işlem zamanı optimizasyonu için PARALEL algoritma.Sonra SERİ ve PARALEL algoritmaları kullanan genel bir algoitma sunuyoruz.Bu sorgu optimizasyon işlem stratejisi, planlar ile grafiksel olarak sunulabilir.Plan (schedule), bilgi transferini n transfer büyüklüğüne oranlanmış çizgiler ile gösterildiği ağaçtır.Her düğüm noktası bir yarı birleşmeye (basit sorgulardaki kesişme gibi) karşılık gelir ve bunlar kendilerini izleyen dağıtımdan önce gerçekleştirilirler.

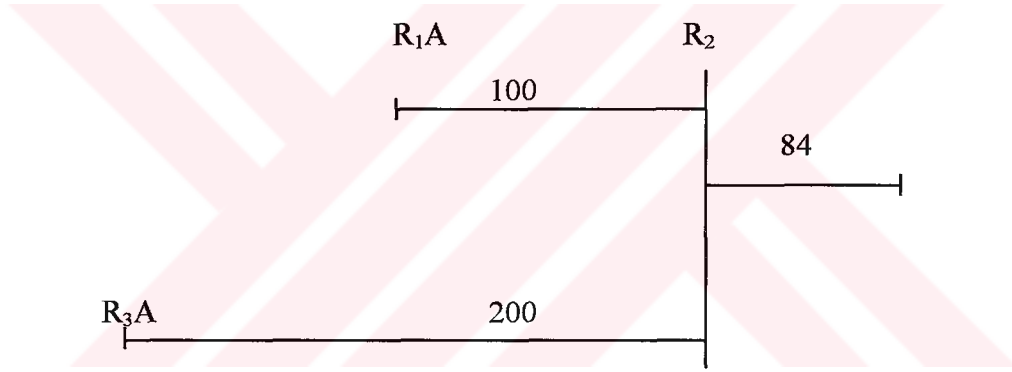
Planları kullanmamızın nedeni, AHY algoritmalarını grafiksel olarak cebir ağaçlarından daha iyi açıklamalarıdır.Düğüm adlarını, transfer edilmiş ilişkilerin attribute' larından alırlar.Kenarlar adlarını transmit eilen bilgilerin büyüklüğünden alırlar.Bir yarı birleşmenin sonucunun büyüklüğü, ilişki büyüklüğü çarpı seçimlik yarı birleşmelerin sayısına eşittir.Attribute' ların birbirinden bağımsız olduğu kabul edilirse, bir çok yarı birleşmenin seçimi, her birinin seçiminin çarpımı olarak düşünülebilir.

Basit Sorguların Minimizasyonu. Optimizasyon, yollanan datanın minimizasyonu bazında çalışır.Algoritma lokal işlemlerden sonra ilişkilerin büyüklüklerini küçükten büyüğe sıralayarak küçük ilişkileri büyüklere

yollar. Algoritmanın ilk adımı ilişkileri küçükten büyüğe sıralamaktır. Aşağıdaki sıralamanın oluştuğunu varsayalım.

$$Büyüklik (R_1) \leq büyüklik (R_2) \leq \dots \leq büyüklik (R_n)$$

Anlaşılması kolay olsun diye ilişkilerin indislerinin, ilişkilerin yerlerini de gösterdiğini de varsayalım; R_i i alanında saklanıyor demektir. Algoritma iki olası gönderme stratejisini kabul eder. Eğer sonuç alanı R_r ilişkisini içeriyorsa, stratejilerden biri şudur: Her ilişki, r alanını da kapsayan daha geniş bir sonraki ilişkinin alanına transfer edilmelidir. Eğer R_i ve R_{i-1} yarı birleşmelerinden oluşan geçici ilişki T_i olarak adlandırılırsa, schedule aşağıdaki gibi oluşur:



Şekil 3.2. Plan örneği

$$R_1 \rightarrow \text{nod } 2$$

$$T^2 = (R_2 \times R_1) \rightarrow \text{nod } 3$$

$$T^{r-1} = (R_{r-1} \times T^{r-2}) \rightarrow \text{nod } r$$

$$T^r = (R_r \times T^{r-1}) \rightarrow \text{nod } (r+1)$$

....

$$(R_n \times T^{n-1}) \rightarrow \text{nod } r \text{ (} r \text{ is the result nod)}$$

İkinci plan (schedule) birleşme operasyonu sırasında sonuç alan r' yi pas geçer ve R_r ile olan birleşmeyi en sonda yapar.

$$R_1 \rightarrow \text{nod } 2$$

$$T^2 = (R_2 \times R_1) \rightarrow \text{nod } 3$$

$$T^{r-1} = (R_{r-1} \times T^{r-2}) \rightarrow \text{nod } (r+1)$$

$$T^{r+1} = (R_{r+1} \times T^{r-1}) \rightarrow \text{nod } (r+2)$$

....

$$T^n = (R_n \times T^{n-1}) \rightarrow \text{nod } r \text{ (} r \text{ is the result nod)}$$

$$(R_r \times T^n)$$

Görmemiz gereken önceki algoritmanın sonuç alanından 2 kere geçmesidir.

SERİ (SERIAL) algoritma, Algoritma 3.4' de ayrıntılarıyla anlatılmıştır.

Algoritma 3.4. SERIAL

input: SQG: simple query graph with n relations

output: ES: execution schedule

begin

order relations in SQG such that büyüklük (R_1) < büyüklük (R_2) <.....< büyüklük (R_n)

if there is no relation R_i at result nod **then**

ES $\leftarrow R_1 \rightarrow R_2 \rightarrow \dots \rightarrow R_n \rightarrow \text{Result nod}$

else begin { select least cost strategy between ES_1 and ES_2 }

$ES_1 \leftarrow (R_1 \rightarrow R_2 \rightarrow \dots \rightarrow R_i \rightarrow \dots \rightarrow R_n \rightarrow R_i)$

$ES_2 \leftarrow (R_1 \rightarrow R_2 \rightarrow \dots \rightarrow R_{i-1} \rightarrow R_{i+1} \rightarrow \dots \rightarrow R_n \rightarrow R_i)$

if cost (ES_1) > cost (ES_2) **then**

ES₁ $\leftarrow$ ES₂

else

ES $\leftarrow$ ES₁

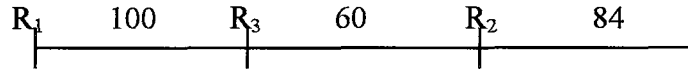
end-if

end-if

end {SERIAL}

Örnek 3.4

$C_{MSG} = 0$, $CTR = 1$ ve sonuç alanı sorguda yer almayan bir alandır. Seçimlerde her zaman basit sorgulardaki ilişkilerle orantılıdır. Önce en küçük ilişki R_1 seçilir. R_2 'nin R_1 ile yarı birleşmeden sonraki büyüklüğü $400 * 0.3 = 120$, R_3 'ün R_1 'den sonraki büyüklüğü $200 * 0.3 = 60$ 'dır. Bu nedenle R_3 planın 2. ilişkisidir ve R_1 , R_3 'ün alanına taşınır. $R_3 \times R_1$ sonucu daha sonra R_2 'nin alanına aktarılır. R_2 'nin yarı birleşme sonrası uzunluğu $400 * 0.7 * 0.3 = 84$. ($0.7 * 0.3$ revize edilmiş seçici faktörüdür). Optimum schedule figür 9.12' de verilmiştir.



Şekil 3.3. Optimal Plan

Basit Sorgularda İşlem Minimizasyonu. Optimizasyon, seri olarak gönderilmek zorunda olan bilgi hacmini paralel işlemleri kullanarak minimize etmektir. İlişkilerin küçükten büyüğe sıralandığını kabul ediyoruz. İlk bir uygun çözüm seçilir ve tüm ilişkiler tek bir sonuç altına paralel olarak aktarılır. Sonra, optimize edici bazı ilişkileri başka bir yere göndererek sonucu daha da geliştirmeye çalışır. Optimize edici verilen bir R_1 ilişkisi için üretilebilecek her ilişkiyi göz önüne almak zorunda değildir. Bu nedenle, R_1 'den büyük ilişkiler R_1 'in orijinal planını geliştiremezler ve kısaltma ve birleşme attribute'larından sonraki sıralamada ihmal edilirler. Algoritma 3.5 PARALEL algoritmayı gösterir.

Algoritma 3.5. *PARALLEL*

input: SQG: simple query graph with n relations

output: ES: execution schedule

begin

order relations such that büyüklük (R_1) < büyüklük (R_2) < < büyüklük (R_n)

$ES_i \leftarrow (R_i \rightarrow \text{Resultnod } \forall R_i)$ (initial solution for each relation)

$ES_0 \leftarrow ES_1 U ES_2 U \dots U ES_n$ (initial feasible solution)

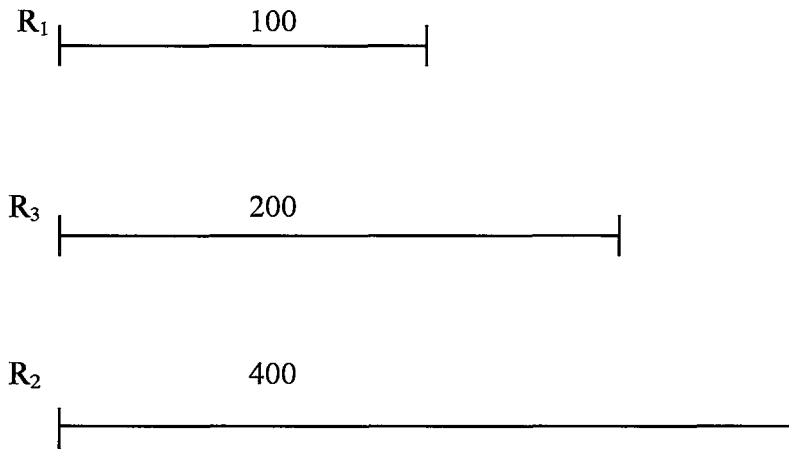
```

for each relation  $R_i$  starting with  $I=1$  do
  begin (find an alternative strategy for each relation}
  for each  $R_j (\forall < i)$  do
     $ES_{ij} \leftarrow (R_j \rightarrow R_i \text{ and } R_k \rightarrow R_i, \forall k < j)$ 
  end-for
   $ES_i \leftarrow \text{minimum cost } (ES_{ij})$ 
  if  $\text{cost } (ES_i) \leq \text{cost } (ES_i)$  then
    begin (replace the strategy for  $R_i$  in  $ES_0$  with new strategy
       $ES_0 \leftarrow ES_0 - ES_i$ 
       $ES_0 \leftarrow ES_0 \cup ES_i$ 
    end-if
  end-for
  eliminate redundant strategies
end (PARALLEL)

```

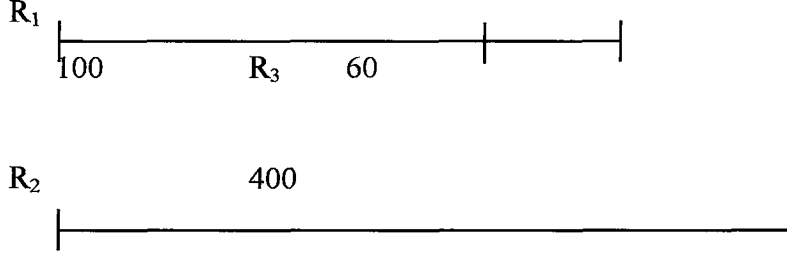
Örnek 3.5.

Bu algoritmayı 3.3' de örnek sorguda açıklıyoruz. Şekil 3.4'te ilk uygun çözüm gösterilmiştir.



Şekil 3.4. Sorgu Örneğinin Uygun Başlangıç Planı

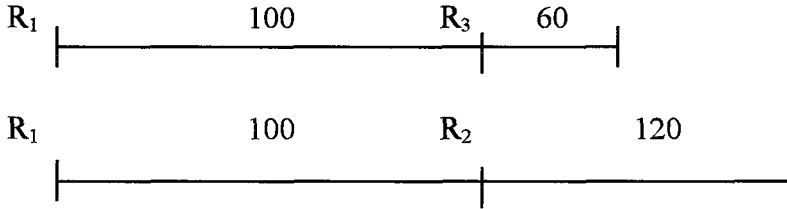
R_1 en küçük ilişki olduğundan schedule' ü geliştirilemez. R_3 ' ün planı R_1 ' i içine alacak şekilde daha iyi bir planla yer değiştirebilir (replace). Bunu R_3 ' ün sonuç alınma transferi izler. R_3 ' ün yarı birleşme sonrası büyüklüğü $200 * 0.3 = 60$. Yeni çözüm figür 2.14' te gösterilmektedir.



Şekil 3.5. Sorgu Örneğinin Genişlemiş Planı

Sonuçta R_2 ve R_1 birleşirler. R_2 ' nin R_1 ile birleşmeden sonraki uzunluğu $400 * 0.33 = 120$.

Bu algorithmada işlem zamanı 220' dir. (Seri algoritma da 244 idi). Fakat toplam maliyeti 380' dir. Sonuçta toplam maliyet ve işlem zamanı arasında karar verilmelidir.



Şekil 3.6. Sorgu Örneğinin Optimal Planı

Genel Algoritma. Genel algoritma işlem zamanının ya da toplam maliyetin minimize edildiği genel bir algoritma olarak özetlenebilir. Algoritma 3.6'da genel sorguları optimize etmek için kullanılan AHY algoritması gösterilmiştir. Algoritma, sorgudaki her R_i ilişkisi için, R_i ' yi yarı birleşmeler yoluyla optimize edilmiş

birleştirilmiş stratejiler üretir ve sonuçta R_i , sorgunun işlenmesi için sonuç alanına gönderilir.

Algoritma 3.6 GENEL

```

input: QG: query graph with n relations
output: ES-set: set of strategies for each relation
begin
perform local operations
CS  $\leftarrow \emptyset$            { candidate strategies for simple queries }
  for each join attribute  $A_j$  describing a simple query SQGi do
    begin
      if objective is response time then
         $ES_j \leftarrow \text{PARALLEL}(SQG)$ 
      else                                     { objective is total cost }
         $ES_j \leftarrow \text{SERIAL}(SQG)$ 
      end-if
       $CS \leftarrow CS \cup ES_j$            { save ESJ as a candidate strategy }
    end-for
    if objective is response time then
      for  $I \leftarrow 1$  to n do
        begin
           $CS \leftarrow CS \cup (R_i \rightarrow \text{result nod})$ 
          order strategies for  $R_i$  on  $A_{ij}$  in increasing order of response time
          for each strategy  $ES_{ij}$  in CS.in ascending order do
            integrate  $ES_{ij}$  in strategy  $IS_i$ 
          end-for
          ES-set  $\leftarrow$  ES-set  $\cup IS_i$  with minimum response time
        end-for
      else                                     { objective is total cost }
        for  $I \leftarrow 1$  to n do
          begin

```

```

CS ← CSU ( $R_i \rightarrow$  result nod)
order strategies for  $R_i$  on  $A_{ij}$  in increasing order to total time
for each strategy  $ES_{ij}$  in CS in ascending order do
    integrate  $ES_{ij}$  in strategy  $IS_i$ 
end-for
ES-set ← ES-set  $\cup$   $ES_{ij}$  with minimum total time
end-for
end-if
for  $i \leftarrow 1$  to  $n$  do                                     {remove redundant strategies}
    if  $ES_i$  is already contained in  $ES_i$  ( $j \neq i$ ) then
        ES-set ← ES-set -  $ES_i$ 
    end-if
end-for
end (GENERAL)

```

Algoritma şöyle işler; Her alanda, ilk bir lokal işlem yapılır. Sorgudaki her birleşme özelliği için, karşılık gelen basit sorgu izole edilir ve esas amaç gözönünde bulundurularak PARALEL ya da SERİ algoritma sorguya uygulanır ve aday stratejiler kümesi üretilir. (Her sorgu için bir strateji) Her bir basit sorgu için oluşan strateji, içinde her R_i ilişkisinin indirgendiği global bir strateji ile birleştirilir. Bu entegrasyonun gerçekleştirilme biçimi hedefe göre değişir.

Entegrasyonun ilk adımı hem işlem zamanı hem de toplam zaman minimizasyonları için özdeştir. Birleşme özelliğindeki her sorgunun aday planı sıralanır. Her R_i ilişkisinin her A_j özelliğinin stratejileri büyükten küçüğe sıralanmıştır. $ES_i = (ES_{i1}, ES_{i2}, \dots, ES_{is})$ R_i için her aday planın artan sıradaki sırasını gösterebilir.

Eğer amaç işlem zamanını azaltmak ise, her ES_{ij} , işlem zamanının artan sırası ile kabul edilir ve IS_i ile gösterilen ona entegre strateji oluşturulur. Strateji, her bir R_i ilişkisi için ES_{ij} her bir aday strateji ES_{ik} 'nın, $k < j$, R_i 'nin alanına paralel olarak gönderilmesini ve buna bağlı olarak indirgenmiş her R_i ilişkisinin sonuç alanına

gönderilmesini kapsar. Sonra R_i ' yi sonuç alanına gönderen stratejilerden işlem zamanı en düşük olan strateji ve entegre stratejiler seçilir.

Eğer hedef, toplam zamanı azaltmak olsaydı, her strateji toplam zaman olarak artan sırada kabul edilecekti ve yukarıdakine benzer şekilde her ilişki için entegre stratejiler oluşturulacaktı. R_i ' yi sonuç alanına gönderen stratejilerden toplam zamanı en düşük olan strateji ve entegre stratejiler seçilir.

Algoritmanın final adımı, bir ilişkinin hem kendi planı olan ve aynı zamanda bir başka planında bir parçası olduğu durumlarda ortaya çıkan plan indirimlerini elimine eder.



SONUÇLAR VE ÖNERİLER

Bu tezde dağıtık sorgu optimizasyonu için temel kavramlar ve teknikler üzerinde durulmuştur. Dağıtık INGRES, R*, SDD-1 ve AHY (Apers, Hevner ve Yao)'nin temel sorgu optimizasyon algoritmalarında uygulamaları gösterilmiştir. Eski dağıtık veri tabanı yönetim sistemleri iletim masraflarını mesaj büyüklüğüne göre değerlendiren objektif bir fonksiyon kullanırlar. Daha hızlı network'lerin bulunabilir olmasıyla daha yeni dağıtık VTYS'leri yerel işleme masrafları da göz önüne alınmaya başlamıştır. R*'ın performans değerlendirmesi ve daha hızlanan genel network'ler için bile yerel masraflara önemli katkılar göstermiştir.

Sorgu optimizasyonu problemine önemli girdiler veri tabanı istatistikleri ve ara sonuçların boyutlarını tahmin etmek için kullanılan formüllerdir. İstatistiklerin performansı ve doğruluğu (ve bakım masrafları) arasında direkt bir ilişki vardır. Genellikle kritik işler dağıtık ilişkilerin birleşimidir. Sorgunun birleşme transformasyonu için heuristic kullanımına dayanan basit bir algoritmanın uygulanmasıyla istatistik kullanımından kaçınılabilir. Bununla beraber çözüm uzayının istatistiklere dayalı zorlu araştırması heuristic yaklaşımlardan belirgin şekilde üstündür. İstatistiksel olarak yapıldığı zaman (derleme sırasında) zorlu araştırmanın oluşturduğu masraf eğer sorgu karışık yada çok sık kullanılıyor ise kısa sürede kendini amorti eder.

Dağıtık sorgu optimizasyonunu anlamanın bir ön koşulu olarak merkezleştirilmiş sorgu optimizasyonu anlatılmıştır. INGRES ve R sistemi tarafından en popüler iki algoritma sunulmuştur.

En önemli sorgu tipleri olduğuna inanılan dağıtık birleşme sorgularını çözen iki yaklaşım gösterilmiştir. Dağıtık INGRES ve R* da gösterilen ilki birleşme sırasıyla ilgilidir. SDDI ve AHY ile örneklenen ikincisi ise birleşmeleri yarı birleşmelerle

hesaplar. Yarı birleşmeler eğer bir birleşmenin seçiciliği iyi ise faydalıdır ki bu durumda yarı birleşmeler kuvvetli birer büyüklük düşürücülerdir. Yarı birleşmelerden aşırı derecede faydalanan ilk sistemler yavaş bir geniş alan şebekesi kabulü yaptıklarından bölgesel işleme masraflarının artması pahasına sadece iletişim masraflarının minimize edilmesi üzerinde yoğunlaşırlardı. Artık hızlı local network'lerle yerel işleme masrafları iletişim masrafları kadar, hatta bazen daha da fazla önemli hale gelmiştir. Bu yüzden yerel işleme masraflarını arttırdıkları için yarı birleşimler dikkatli ve sistematik olmayan bir şekilde kullanılmalıdır. Veri tabanına bağlı parametrelere bağlı olarak tüm teknikler birbirinden üstün olduğu için birleşme ve yarı birleşme teknikleri de gösterildiği gibi kullanılmalıdır. [4]

Birleşme sorgularını çözmede bir diğer gelecek vadeden önemli yöntem de hem yerel işleme hem de iletişim masraflarını minimize etme potansiyeli olan birleşme indisleri denen küçük ve yardımcı veri yapıları kullanmaktır [4] da rapor edilen performans analizleri; şimdiden merkezlenmiş bir yapıdaki birleşme indislerinin, hem karışık ilişkisel sorguların hem de tekrarlı sorguların optimizasyonunda değerli olduğunu ortaya koymuştur. Birleşme indisleri dağıtık bir yapıda da kullanışlı olabilirler.

Bu teze iki nedenden dolayı genellikle birleşme sorgularında yoğunlaşılmıştır: birleşme sorguları ilişkisel ana yapılardaki en sık karşılaşılan sorgulardır ve geniş olarak incelenmişlerdir. Ayrıca ilişkisel cebirinden daha yüksek ifade gücü olan dillerdeki sorgulardaki birleşmelerin sayısı çok fazla olabilir ki, buda birleşme sıralamasını daha da önemli yapar. Bununla beraber, birleşmeleri, bağlanmaları ve toplama fonksiyonlarını içeren genel sorguların optimizasyonu daha zor bir problemdir. Toplamaları birleşmeler üzerine dağıtmak sorgu bağımsız olarak optimize edilen birleşme alt sorgularına indirgenebileceği için basit ve güzel bir yaklaşımdır. Yatay parçalanmış ilişkileri mümkün kıldığı için toplanmalar dağıtık VTYS' lerinde merkezlenmiş VTYS' lerden daha sıktır.

Bu bölümde anlatılan kavramlar ve çözümler genelliğe ve pratiğe yakınlığı nedeniyle seçilmiş 4 sorgu işlemi algoritmalarında örneklenmiştir. Bununla beraber bu

konseptlerden diğ er metotlar da t uretilebilir. İyi bir yaklařım yıldız network'ler, uydu network'ler, yayın network'leri gibi network'ler topolojisinden faydalanmak i in optimizasyon algoritmasının  zelleřtirilmesidir.

Merkezleřmiř sistemlerde sorgu optimizasyonu i in iki temel algoritma; INGRES ve Sistem R* dır. INGRES dinamik bir optimizasyon y ntemi kullanırken, Sistem R* en iyi sonu  i in statik arařtırma yapar. Birleřme indeksi denen hem merkezlenmiř hem de paralel ortamlarda birleřme sorgularını optimize eden genel ve basit bir veri yapısı tanımlanmıřtır.



EKLER

Güçlü bir client-server yapısına database trigger 'ların ve stored procedure'ların kalkısı büyük olmaktadır. Oracle altında da bu özellikler kullanılarak hız ve maliyet açısından güçlü bir client-server yapısı kurulabilir.

Database trigger'ler otomatik veri işlemi ve sorgu yapmak için ideal bir araçlardır. Veri tabanı üzerindeki kayıt ve tablolar üzerinde silme, güncelleme ve kayıt ekleme olaylarından önce veya sonra otomatik olarak tetiklenerek sorgu ve veri işleme zaman ve maliyeti minimize edilebilir.

Uygulamada bulunan mağazacılık sistemi ile ilgili bir örnek verilirse; farklı lokasyonlarda yer alan mağazalarda satılan mağazalara ait miktar ve satış fiyatlarını genel merkezde yer alan bir database server'a bağlı client üzerinden gerçekleştirmek istendiğinde farklı clientlerinin bu isteklerinin yerine getirilmesi için database trigger kullanılabilir. Aylık bazı satışların yer aldığı tabloya her kayıt girildiğinde, tetiklenen bir database trigger vasıtasıyla, karşı sistem üzerinden koşturulan bir stored procedure vasıtasıyla ilgili mal kaydına ait o aya ait satışlar ve miktarlar genel merkezdeki veri tabanına sorgulanıp işlenebilir.

Burada database trigger genel merkezdeki yerel ağ optimizasyonunu sağlamaktadır. Bütün clientlerin isteği yerel veri tabanı server'ı tarafından otomatik olarak tetiklenmekte ve bu isteklerin hepsi client kademesi atlanarak direkt diğer database server'a iletilmektedir. Bu isteğin direkt yerel database server üzerinden idare edilmesi, bu isteklerin clientlerden sağlanmaya çalışılıp bunların sonuçlarını database server'a bindirilmesi ve network trafiğinin artması dikkate alınırsa büyük bir avantaj sağlanmaktadır.

Database trigger'larda olduğu gibi stored procedure'ların da sorgu optimizasyondaki faydaları son derece büyüktür. Farklı lokasyonlarda yer alan

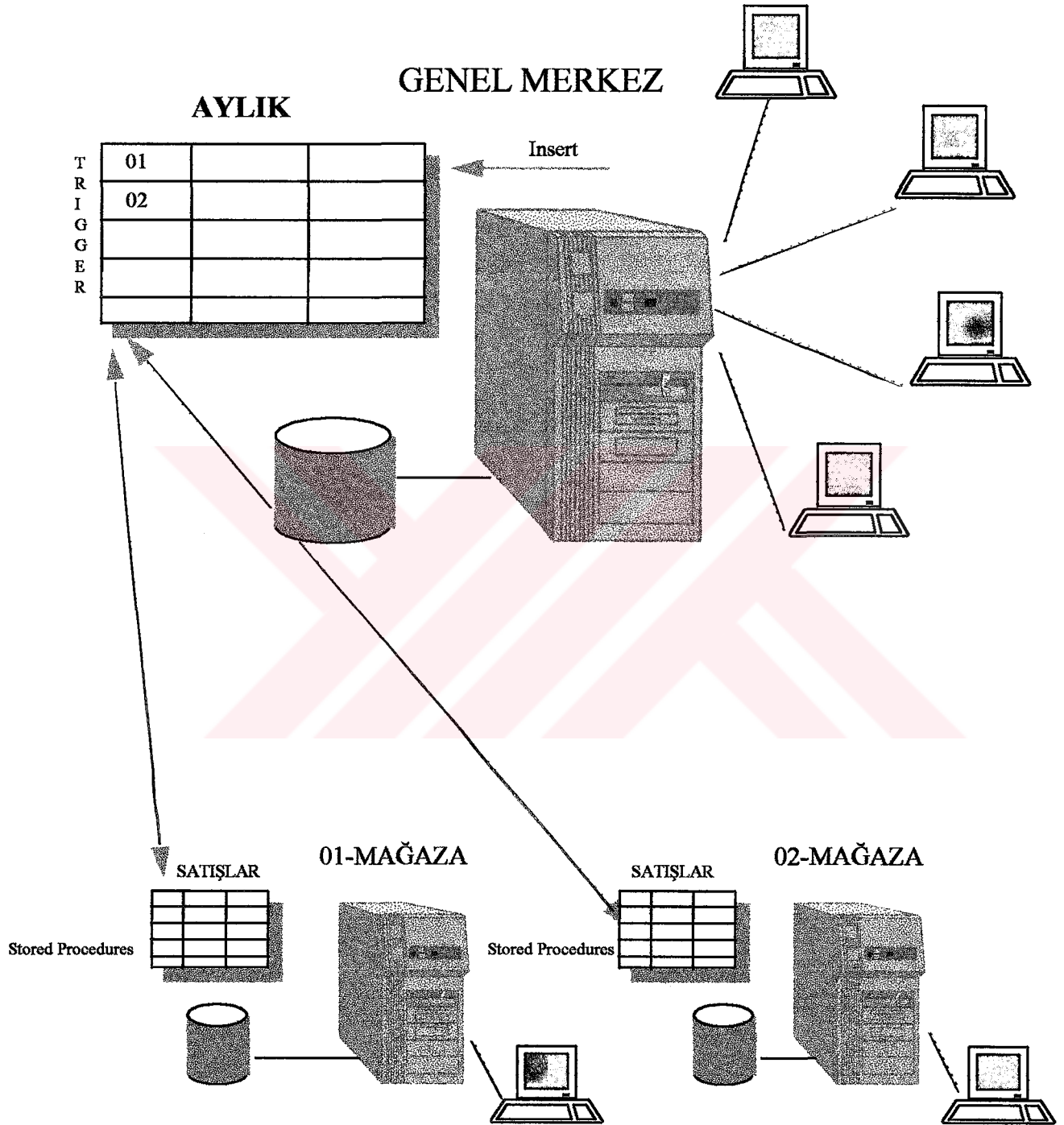
database serverler arasındaki sorgunun sorgu isteğinin bulunduğu lokasyondaki database server üzerinden gerçekleştirmek ile sorgu isteğinin bulunduğu database server üzerinden (RPC-Remote Procedure Call) gerçekleştirmek arasında çok büyük farklılıklar olabilmektedir. Yukarıda verilen belli bir mala ait satış aylık satış ve miktarların toplamının hesaplanması örneğini ele alarak konuya biraz daha açıklık kazandırılabilir.

Belli bir mal bir ay içerisinde 100.000 adet satış işlemi görmüş olsun. Her mala ait mal kodu+fiyat+miktar bilgileri 100 byte tutuyor olsun. 100 byte bu geniş alanlı network üzerinde 0.1 sn de aktarılsın. İki farklı senaryo söz konusu olabilir.

1. Aylık satışların yerel client üzerinden hesaplanması
2. Aylık satışlar karşı database server üzerinden hesaplatılıp, yerel database'e sadece hesap sonucunun gelip işlenmesi.

Birinci senaryoya göre her malın bilgisi network üzerinden yerel sisteme gelecek $100.000 \times 100 \times 0.1 = 10.000.00$ sn. byte hem veri transferi olacak hemde bu veriler üzerinde satış tutar ve miktarları üzerinde toplam yapılacak, buda X sn sürsün. Bu şekildeki toplam sorgu zamanı $10.000.00$ sn. + X + yapılan işlemlerin onayı için geçen süre şeklinde belirlenecektir.

İkinci senaryoda ise karşı database server üzerinden hangi procedure'un çağrılacağı bilgisi sadece karşı sisteme bildirilecek ve 100.000 datanın toplam satış ve tutarı Y sn gibi ($X < Y$) sürede hesaplanacak ve sadece bu sonucun değeri 100 byte sorgu isteğinin bulunduğu yerel sisteme gönderilecektir. Toplam süre $0,1$ sn + Y ($Y < X$) yapılan işlemin onayı bilgisi olacaktır. Görüldüğü gibi senaryo iki daha avantajlıdır. Hem süre çok daha kısadır, hem de daha az veri transferi yapıldığı için ağ maliyeti daha azdır.



PROCEDURE FISNOBUL IS

IT_IDB6 ITEM;

cursor c1 is select fisno from HAREKET where tarih = :tarih1 and tip = :tip1
order by fisno;

cursor c2 is select fisno from HAREKET where tarih = :tarih1 and
(tip = 'A' or tip = 'B' OR TIP = 'AF' OR TIP = 'BF')
order by fisno;

cursor c3 is select fisno from HAREKET where tarih = :tarih1 and
(tip = 'DN' OR tip = 'DO' OR tip = 'DF') order by fisno;

oldnum hareket.fisno%type;

bmagazano node.magazano%type;

BEGIN

select magazano into bmagazano from node;

--IT_IDB6 := FIND_ITEM('FIS_BUTT');

--SET_ITEM_PROPERTY(IT_IDB6,DISPLAYED,PROPERTY_ON);

:FISNO1 := NULL;

IF :TIP1 = 'B' OR :TIP1 = 'A' THEN

select max(fisno) into :fisno1 from hareket

where

(hareket.tip = 'A' OR
HAREKET.TIP = 'B'OR
HAREKET.TIP = 'AF' OR
HAREKET.TIP = 'BF');

if nvl(:fisno1,0) = 0 then

:fisno1 := substr(to_char(bmagazano),4,2)||'0000';

end if;

:fisno1 := NVL(:fisno1,0) + 1;

select max(fisno) into oldnum from hareket

where hareket.tarih = (:tarih1 - 1) and

(hareket.tip = 'A' OR
HAREKET.TIP = 'B'OR
HAREKET.TIP = 'AF' OR
HAREKET.TIP = 'BF');

for c2rec in c2 loop

if (c2rec.fisno - oldnum) > 1 then

:fisno1 := oldnum + 1;

end if;

oldnum := c2rec.fisno;

end loop;

elsif :tip1 = 'DO' OR :TIP1 = 'DF' OR :TIP1 = 'DN' THEN

```

select max(fisno) into :fisno1 from hareket
      where
        (hareket.tip = 'DO' OR
         HAREKET.TIP = 'DF' OR
         HAREKET.TIP = 'DN');
if nvl(:fisno1,0) = 0 then
  :fisno1 := substr(to_char(bmagazano),4,2)||'0000';
end if;
:fisno1 := NVL(:fisno1,0) + 1;
  select max(fisno) into oldnum from hareket
      where hareket.tarih = (:tarih1 - 1) and
        (hareket.tip = 'DO' OR
         HAREKET.TIP = 'DF' OR
         HAREKET.TIP = 'DN');

  for c3rec in c3 loop
    if (c3rec.fisno - oldnum) > 1 then
      :fisno1 := oldnum + 1;
    end if;
    oldnum := c3rec.fisno;
  end loop;
ELSE
  select max(fisno) into :fisno1 from hareket
      where
        (hareket.tip = :TIP1);
  select max(fisno) into oldnum from hareket
      where hareket.tarih = (:tarih1 - 1) and
        hareket.tip = :TIP1;
if nvl(:fisno1,0) = 0 then
  :fisno1 := substr(to_char(bmagazano),4,2)||'0000';
end if;
:fisno1 := NVL(:fisno1,0) + 1;

  for c1rec in c1 loop
    oldnum := c1rec.fisno;
    if (c1rec.fisno - oldnum) > 1 then
      :fisno1 := oldnum + 1;
    end if;
  end loop;
END IF;

END;

PROCEDURE HARISLE IS
CURSOR C1 IS SELECT * FROM MVTRNS WHERE TARIH = :TARIH1
AND

```

```

FISNUM = :FISNO1 AND NVL(MIKTAR,0) != 0 AND NVL(SFIYAT,0) != 0
AND TIP = :TIP1;
SIPMIK1 SIPARIS.BRMMIK%TYPE;
SAYI NUMBER;
BSIM HAREKET.SIMH%TYPE;
BEGIN

DO_KEY('COMMIT_FORM');

FOR C1REC IN C1 LOOP
    SELECT COUNT(*) INTO SAYI FROM HAREKET WHERE
    HAREKET.TARIH = C1REC.TARIH AND
    HAREKET.FISNO = C1REC.FISNUM AND
    HAREKET.TIP = C1REC.TIP AND
    HAREKET.MSIRANO = C1REC.MSIRANO;

    SELECT SIM INTO BSIM FROM MAL_TANIM WHERE
    MAL_TANIM.MSIRANO = C1REC.MSIRANO;

    IF (:TIP1 = 'TR') THEN
    IF SAYI = 0 THEN

    INSERT INTO
    HAREKET(HARNO,MAGAZANO,MVNO,NEDENNO,BELGETUR,BELGEN
    O,IRSTARIH,FISNO,TARIH,MSIRANO,TIP,MIKTAR,SFIYAT,SIMH)
    VALUES(HAR_SEQ.NEXTVAL,:NEREYE,:MVNO1,'X','T',TO_CHAR(:IRSAL
    IYENO),:IRSTARIH1,:FISNO1,
    :TARIH1,C1REC.MSIRANO,:TIP1,C1REC.MIKTAR,C1REC.SFIYAT,NVL(B
    SIM,1));

    END IF;
    /*ELSE

    UPDATE HAREKET SET
    MIKTAR = NVL(C1REC.MIKTAR,0),
    SIMH = NVL(BSIM,0) WHERE
    HAREKET.TARIH = C1REC.TARIH AND
    HAREKET.FISNO = C1REC.FISNUM AND
    HAREKET.TIP = C1REC.TIP AND
    HAREKET.MSIRANO = C1REC.MSIRANO;

    END IF;*/

    Select SUM(brmmik) into sipmik1 from siparis where msirano =
    C1REC.MSIRANO and status = 'AD' AND

```

```
MVNO = :MVNO1;
IF SIPMIK1 <> 0 THEN
```

```
UPDATE siparis SET  GELMIK = C1REC.MIKTAR,
                    STATUS = 'K',
                    KALMIK = SIPMIK1 - C1REC.MIKTAR,
                    IRSALNO = :IRSALIYENO,
                    FISNO = :FISNO1,
                    FISTARIH = :TARIH1,
                   IRSTARIH = :IRSTARIH1
```

```
where msirano = C1REC.MSIRANO and status = 'AD' AND MVNO =
:MVNO1;
```

```
END IF;
```

```
ELSIF (:TIP1 = 'TI') THEN
```

```
IF SAYI = 0 THEN
```

```
INSERT INTO
```

```
HAREKET(HARNO,MAGAZANO,MVNO,NEDENNO,FISNO,TARIH,MSIRA
NO,TIP,MIKTAR,SFIYAT,SIMH)
```

```
VALUES(HAR_SEQ.NEXTVAL,:NEREDEN,:MVNO1,'V',:FISNO1,:TARIH1,
C1REC.MSIRANO,:TIP1,C1REC.MIKTAR,
C1REC.SFIYAT,NVL(BSIM,1));
```

```
ELSE
```

```
UPDATE HAREKET SET
```

```
MIKTAR = C1REC.MIKTAR,
```

```
SFIYAT = C1REC.SFIYAT,
```

```
SIMH = BSIM
```

```
WHERE
```

```
HAREKET.TARIH = C1REC.TARIH AND
```

```
HAREKET.FISNO = C1REC.FISNUM AND
```

```
HAREKET.TIP = C1REC.TIP AND
```

```
HAREKET.MSIRANO = C1REC.MSIRANO;
```

```
END IF;
```

```
END IF; /*SAYI*/
```

```
COMMIT;
```

```
END LOOP;
```

```
END;
```

```
PROCEDURE ODTESYUK IS
```

```
CURSOR C2 IS SELECT * FROM UL_LOPI,KASACEK WHERE
```

```
TO_CHAR(UL_LOPI.DATATIME,'DD-MON-YY') =
```

```
TO_CHAR(KASACEK.TARIH,'DD-MON-YY');
```



```

if sayi = 0 then
insert into HESABAT (ckrno,total_id,total_amt,datetime)
      values (c2rec.ckr_no,9300,c2rec.amount,btarih);
else

      update hesabat set total_amt = nvl(total_amt,0) + nvl(c2rec.amount,0)
      where hesabat.datetime = btarih and
            hesabat.ckrno  = c2rec.ckr_no and
            hesabat.total_id = 9300;
end if;

elsif c2REC.tend_id = 10 THEN
      select count(*) into sayi from hesabat where hesabat.ckrno  =
to_char(c2rec.ckr_no) and
            hesabat.datetime = btarih and
            hesabat.total_id = 9700;

if sayi = 0 then
insert into HESABAT (ckrno,total_id,total_amt,datetime)
      values (c2rec.ckr_no,9700,c2rec.amount,btarih);
else

      update hesabat set total_amt = nvl(total_amt,0) + nvl(c2rec.amount,0)
      where hesabat.datetime = btarih and
            hesabat.ckrno  = c2rec.ckr_no and
            hesabat.total_id = 9700;
end if;

elsif c2REC.tend_id = 9 THEN
      select count(*) into sayi from hesabat where hesabat.ckrno  =
to_char(c2rec.ckr_no) and
            hesabat.datetime = btarih and
            hesabat.total_id = 9500;

if sayi = 0 then
insert into HESABAT (ckrno,total_id,total_amt,datetime)
      values (c2rec.ckr_no,9500,c2rec.amount,btarih);
else

      update hesabat set total_amt = nvl(total_amt,0) + nvl(c2rec.amount,0)
      where hesabat.datetime = btarih and
            hesabat.ckrno  = c2rec.ckr_no and
            hesabat.total_id = 9500;
end if;

```

```

    elsif c2REC.tend_id = 11 THEN
        select count(*) into sayi from hesabat where hesabat.ckrno =
to_char(c2rec.ckr_no) and
                                hesabat.datatime = btarih and
                                hesabat.total_id = 9900;

        if sayi = 0 then
            insert into HESABAT (ckrno,total_id,total_amt,datatime)
                values (c2rec.ckr_no,9900,c2rec.amount,btarih);
        else

            update hesabat set total_amt = nvl(total_amt,0) + nvl(c2rec.amount,0)
            where hesabat.datatime = btarih and
                hesabat.ckrno = c2rec.ckr_no and
                hesabat.total_id = 9900;
        end if;

    elsif c2REC.tend_id = 12 THEN
        select count(*) into sayi from hesabat where hesabat.ckrno =
to_char(c2rec.ckr_no) and
                                hesabat.datatime = btarih and
                                hesabat.total_id = 10100;

        if sayi = 0 then
            insert into HESABAT (ckrno,total_id,total_amt,datatime)
                values (c2rec.ckr_no,10100,c2rec.amount,btarih);
        else

            update hesabat set total_amt = nvl(total_amt,0) + nvl(c2rec.amount,0)
            where hesabat.datatime = btarih and
                hesabat.ckrno = c2rec.ckr_no and
                hesabat.total_id = 10100;
        end if;
    end if;

END LOOP;

END;

PROCEDURE PLUAL IS
CURSOR C1 IS SELECT * FROM ITEMDC WHERE TUTAR <> 0;
BHARNO HAREKET.HARNO%TYPE;
SAYI NUMBER;

```

```

BEGIN
DELETE FROM HAREKET1;
COMMIT;
FOR C1REC IN C1 LOOP
SELECT COUNT(*) INTO SAYI FROM HAREKET1,MALBAR
      WHERE MALBAR.BARCODE = C1REC.BARCODE AND
      HAREKET1.TARIH = C1REC.TARIH AND
      HAREKET1.MSIRANO = MALBAR.MSIRANO AND
      HAREKET1.TIP = 'S';
IF SAYI = 0 THEN
IF C1REC.FLAG1 = '1' THEN
insert into hareket1 (msirano,tip,miktar,belgetur,magazano,sfiyat,harno,TARIH)
      select malBAR.msirano, 'S' , C1REC.AGIRLIK, 'P'
,node.magazano,(C1REC.TUTAR/C1REC.AGIRLIK),
har_seq.nextval, C1REC.TARIH from malBAR,NODE
      where malBAR.barcode = C1REC.BARCODE ;
ELSE
insert into hareket1 (msirano,tip,miktar,belgetur,magazano,sfiyat,harno,TARIH)
      select malBAR.msirano, 'S' , C1REC.MIKTAR, 'P' ,node.magazano,
(C1REC.TUTAR/C1REC.MIKTAR), har_seq.nextval,C1REC.TARIH from
malBAR, NODE where malBAR.barcode = C1REC.BARCODE ;
END IF;
ELSE
SELECT HARNO INTO BHARNO FROM HAREKET1,MALBAR
      WHERE MALBAR.BARCODE = C1REC.BARCODE AND
      HAREKET1.TARIH = C1REC.TARIH AND
      HAREKET1.MSIRANO = MALBAR.MSIRANO AND
      HAREKET1.TIP = 'S';
IF C1REC.FLAG1 = '1' THEN
UPDATE HAREKET1 SET HAREKET1.MIKTAR =
NVL(HAREKET1.MIKTAR,0) + NVL(C1REC.AGIRLIK,0)
WHERE HAREKET1.HARNO = BHARNO;
ELSE
UPDATE HAREKET1 SET HAREKET1.MIKTAR =
NVL(HAREKET1.MIKTAR,0) + NVL(C1REC.MIKTAR,0)
WHERE HAREKET1.HARNO = BHARNO;
END IF;
END IF;
END LOOP;
INSERT INTO HAREKET SELECT * FROM HAREKET1 WHERE TIP = 'S';
END PLUAL;
PROCEDURE SORGULA IS
BEGIN
IF NVL(:FISNO1,0) = 0 THEN
MESSAGE('FIS NUMARASI GIRILMEMIS!');

```

```

MESSAGE(' ');
GO_ITEM('FISNO1');
ELSIF (:TIP1 = 'TF') AND (NVL(:NEREYE,0) = 0 OR LENGTH(:NEREYE_D)
= 0 OR NVL(:NEREDEN,0) = 0 OR
LENGTH(:NEREDEN_D) = 0)
THEN MESSAGE('EKSİK MAGAZA BİLGİSİ GİRİLMİŞ!');
MESSAGE(' ');
ELSE
GO_BLOCK('BMAL');
set_block_property('bmal',default_where,'tip = :tip1 AND tarih = :tarih1 and
fisno = :fisno1 AND TIP != "T" AND STNSKL != "ÇV"AND DURUM != "4" ');
:GLOBAL.KABL := '1';
execute_query;
END IF;
END;

```

```

create trigger TAMIM_TRIG
BEFORE insert on MIGROS.TAMIM
for each row
DECLARE SAYI NUMBER;
bfiyat number(13,3);
MAGAZ NODE.MAGAZANO%TYPE;
begin
SELECT MAGAZANO INTO MAGAZ FROM NODE;
IF :NEW.TAMIMKODU = '2068' THEN
select count(*) into SAYI from mv_tanim where mvno =
to_number(SUBSTR(:NEW.BILGI,1,4));
if SAYI = 0 then
INSERT INTO MV_TANIM (MVNO, MVADI)
VALUES
(to_number(SUBSTR(:NEW.BILGI,1,4)),SUBSTR(:NEW.BILGI,5,30));
ELSE UPDATE MV_TANIM SET MVADI = SUBSTR(:NEW.BILGI,5,30)
WHERE MVNO = to_number(SUBSTR(:NEW.BILGI,1,4));
end if;
END IF;
IF :NEW.TAMIMKODU = '1068' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI = 0 then
INSERT INTO MAL_TANIM (kayitno,msirano,etik_stat,satis)
VALUES (to_number(SUBSTR(:NEW.BILGI,1,5)),:NEW.msirano,'X','S');
end if;
select count(*) into SAYI from mv_tanim where mvno =
to_number(SUBSTR(:NEW.BILGI,6,4));
if SAYI = 0 then
INSERT INTO MV_TANIM (MVNO)

```

```

VALUES (to_number(SUBSTR(:NEW.BILGI,6,4)));
end if;
select count(*) into SAYI from malver_mal where mvno =
to_number(SUBSTR(:NEW.BILGI,6,4))
and msirano = :NEW.msirano;
if SAYI = 0 then
INSERT INTO MALVER_MAL (MSIRANO,MVNO)
VALUES (:NEW.MSIRANO,to_number(SUBSTR(:NEW.BILGI,6,4)));
end if;
/* select count(*) into SAYI from malbar where
barcode = :NEW.barcode;
if SAYI = 0 then
INSERT INTO MALBAR (MSIRANO,BARCODE)
VALUES (:NEW.MSIRANO,:NEW.BARCODE);
end if;*/
END IF;
/* YENI EKLENDI.....*/
IF :NEW.TAMIMKODU = '1069' THEN
select count(*) into SAYI from mv_tanim where mvno =
to_number(SUBSTR(:NEW.BILGI,7,4));
if SAYI = 0 then
INSERT INTO MV_TANIM (MVNO,MVSNO)
VALUES
(to_number(SUBSTR(:NEW.BILGI,7,4)),to_number(SUBSTR(:NEW.BILGI,1,6
)));
else
UPDATE MV_TANIM SET MVSNO = to_number(SUBSTR(:NEW.BILGI,1,6))
WHERE mvno = to_number(SUBSTR(:NEW.BILGI,7,4));
end if;
select count(*) into SAYI from malver_mal where mvno =
to_number(SUBSTR(:NEW.BILGI,7,4))
and msirano = :NEW.msirano;
if SAYI = 0 then
INSERT INTO MALVER_MAL (MSIRANO,MVNO)
VALUES (:NEW.MSIRANO,to_number(SUBSTR(:NEW.BILGI,7,4)));
END IF;
END IF;
IF :NEW.TAMIMKODU = '0110' THEN
select count(*) into SAYI from malbar where
barcode =
TO_NUMBER(SUBSTR(:NEW.BILGI,1,13));
if SAYI = 0 then
INSERT INTO MALBAR (MSIRANO,BARCODE)
VALUES
(:NEW.MSIRANO,TO_NUMBER(SUBSTR(:NEW.BILGI,1,13)));

```

```

    end if;
END IF;
IF :NEW.TAMIMKODU = '0111' THEN
select count(*) into SAYI from malbar where
                                barcode =
TO_NUMBER(SUBSTR(:NEW.BILGI,14,13));
    if SAYI = 0 then
        INSERT INTO MALBAR (MSIRANO,BARCODE)
        VALUES
        (:NEW.MSIRANO,TO_NUMBER(SUBSTR(:NEW.BILGI,14,13)));
    end if;
    select count(*) into SAYI from malbar where
                                barcode =
TO_NUMBER(SUBSTR(:NEW.BILGI,1,13));
    if SAYI != 0 then
        UPDATE MALBAR SET PLU_STAT = 'D' WHERE MALBAR.BARCODE
        =
        TO_NUMBER(SUBSTR(:NEW.BILGI,1,13)) ;
    end if;
END IF;
IF :NEW.TAMIMKODU = '0119' THEN
    select count(*) into SAYI from malbar where
                                barcode =
TO_NUMBER(SUBSTR(:NEW.BILGI,1,13));
    if SAYI != 0 then
        UPDATE MALBAR SET PLU_STAT = 'D' WHERE MALBAR.BARCODE =
        TO_NUMBER(SUBSTR(:NEW.BILGI,1,13)) ;
    end if;
END IF;
/* EKLEME SONU ../
    IF :NEW.TAMIMKODU = '2073' THEN
        select count(*) into SAYI from mv_tanim where mvno =
to_number(SUBSTR(:NEW.BILGI,1,4));
    if SAYI <> 0 then
        UPDATE MV_TANIM SET POSTANO =
TO_NUMBER(SUBSTR(:NEW.BILGI,5,5)),
        HESAPNO = TO_NUMBER(SUBSTR(:NEW.BILGI,10,9)),
        mvsno = TO_NUMBER(SUBSTR(:NEW.BILGI,14,5))
        WHERE MVNO = TO_NUMBER(SUBSTR(:NEW.BILGI,1,4));
    end if;
    END IF;
IF :NEW.TAMIMKODU = '2069' THEN
    select count(*) into SAYI from mv_tanim where mvno =
to_number(SUBSTR(:NEW.BILGI,1,4));
    if SAYI <> 0 then

```

```

UPDATE MV_TANIM SET ADRES1 = SUBSTR(:NEW.BILGI,5,30)
WHERE MVNO = TO_NUMBER(SUBSTR(:NEW.BILGI,1,4));
end if;
END IF;
IF :NEW.TAMIMKODU = '2070' THEN
  select count(*) into SAYI from mv_tanim where mvno =
to_number(SUBSTR(:NEW.BILGI,1,4));
if SAYI <> 0 then
  UPDATE MV_TANIM SET ADRES2 = SUBSTR(:NEW.BILGI,5,30)
  WHERE MVNO = TO_NUMBER(SUBSTR(:NEW.BILGI,1,4));
end if;
END IF;
IF :NEW.TAMIMKODU = '2071' THEN
  select count(*) into SAYI from mv_tanim where mvno =
to_number(SUBSTR(:NEW.BILGI,1,4));
if SAYI <> 0 then
  UPDATE MV_TANIM SET ADRES3 = SUBSTR(:NEW.BILGI,5,17),
  IL = SUBSTR(:NEW.BILGI,22,14)
  WHERE MVNO = TO_NUMBER(SUBSTR(:NEW.BILGI,1,4));
end if;
END IF;
IF :NEW.TAMIMKODU = '2072' THEN
  select count(*) into SAYI from mv_tanim where mvno =
to_number(SUBSTR(:NEW.BILGI,1,4));
if SAYI <> 0 then
  UPDATE MV_TANIM SET VERGID = SUBSTR(:NEW.BILGI,5,19),
  VERGINO = SUBSTR(:NEW.BILGI,24,12)
  WHERE MVNO = TO_NUMBER(SUBSTR(:NEW.BILGI,1,4));
end if;
END IF;
IF :NEW.TAMIMKODU = '1513' THEN
  select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
  UPDATE MAL_TANIM SET PAZKOD =
TO_NUMBER(SUBSTR(:NEW.BILGI,1,1))
  WHERE MSIRANO = :NEW.MSIRANO;
end if;
END IF;
IF :NEW.TAMIMKODU = '1607' THEN
  select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
  UPDATE MAL_TANIM SET MADI = SUBSTR(:NEW.BILGI,1,35),
  ETIK_STAT = 'X'
  WHERE MSIRANO = :NEW.MSIRANO;
end if;

```

```

END IF;
IF :NEW.TAMIMKODU = '1507' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
    UPDATE MAL_TANIM SET MKADI = SUBSTR(:NEW.BILGI,1,20),
        ETIK_STAT = 'X'
        WHERE MSIRANO = :NEW.MSIRANO;
end if;
END IF;
IF :NEW.TAMIMKODU = '1530' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
    UPDATE MAL_TANIM SET DURUM =
TO_NUMBER(SUBSTR(:NEW.BILGI,1,1))
        WHERE MSIRANO = :NEW.MSIRANO;
end if;
END IF;
/*IF :NEW.TAMIMKODU = '1532' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
    UPDATE MAL_TANIM SET DURUM =
TO_NUMBER(SUBSTR(:NEW.BILGI,1,1))
        WHERE MSIRANO = :NEW.MSIRANO;
end if;
END IF;
IF :NEW.TAMIMKODU = '1534' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
    UPDATE MAL_TANIM SET DURUM =
TO_NUMBER(SUBSTR(:NEW.BILGI,1,1)),
        SATIS = 'T'
        WHERE MSIRANO = :NEW.MSIRANO;
end if;
END IF;
IF :NEW.TAMIMKODU = '1536' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
    UPDATE MAL_TANIM SET DURUM =
TO_NUMBER(SUBSTR(:NEW.BILGI,1,1)),
        SATIS = 'T'
        WHERE MSIRANO = :NEW.MSIRANO;
end if;
END IF;
IF :NEW.TAMIMKODU = '1538' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;

```

```

if SAYI <> 0 then
    UPDATE MAL_TANIM SET DURUM =
    TO_NUMBER(SUBSTR(:NEW.BILGI,1,1)),
        SATIS = 'T'
        WHERE MSIRANO = :NEW.MSIRANO;
end if;
END IF;*/
IF :NEW.TAMIMKODU = '1613' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
    if substr(:NEW.bilgi,1,1) = 'S' then
UPDATE MAL_TANIM SET TIPI = 'D'
        WHERE MSIRANO = :NEW.MSIRANO;
        elsif substr(:new.bilgi,1,1) = '' or substr(:new.bilgi,1,1) = null /*yeni eklendi
12ekim*/
        then
            UPDATE MAL_TANIM SET TIPI = 'D'
                WHERE MSIRANO = :NEW.MSIRANO;
        else
            UPDATE MAL_TANIM SET TIPI = SUBSTR(:NEW.BILGI,1,1)
                WHERE MSIRANO = :NEW.MSIRANO;
        end if;
    end if;
END IF;
IF :NEW.TAMIMKODU = '0127' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
    UPDATE MAL_TANIM SET KDV =
    TO_NUMBER(SUBSTR(:NEW.BILGI,1,2))
        WHERE MSIRANO = :NEW.MSIRANO;
end if;
END IF;
IF :NEW.TAMIMKODU = '1542' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
    UPDATE MAL_TANIM SET STNSKL = SUBSTR(:NEW.BILGI,1,2)
        WHERE MSIRANO = :NEW.MSIRANO;
end if;
END IF;
IF :NEW.TAMIMKODU = '1742' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
    IF (TO_NUMBER(SUBSTR(:NEW.BILGI,1,6)) = 0) THEN
UPDATE MAL_TANIM SET SIM = 1.0
        WHERE MSIRANO = :NEW.MSIRANO;

```

```

ELSE
UPDATE MAL_TANIM SET SIM =
TO_NUMBER(SUBSTR(:NEW.BILGI,1,6),'99999.9')
WHERE MSIRANO = :NEW.MSIRANO;

END IF;
end if;
END IF;
IF :NEW.TAMIMKODU = '1642' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
UPDATE MAL_TANIM SET BIRIM = SUBSTR(:NEW.BILGI,1,2)
WHERE MSIRANO = :NEW.MSIRANO;

end if;
END IF;
IF :NEW.TAMIMKODU = '0064' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
select NVL(mal_tanim.sfiyat,0) into bfiyat from mal_tanim where msirano =
:new.msirano;
UPDATE MAL_TANIM SET ETIK_STAT = 'X' WHERE MSIRANO =
:NEW.MSIRANO;
if NVL(bfiyat,0) != TO_NUMBER(SUBSTR(:NEW.BILGI,1,10)) then
UPDATE MAL_TANIM SET SFIYAT =
TO_NUMBER(SUBSTR(:NEW.BILGI,1,10))
WHERE MSIRANO = :NEW.MSIRANO;
INSERT INTO HAREKET
(HARNO,MSIRANO,BELGETUR,SFIYAT,TIP,TARIH,MAGAZANO)
VALUES
(HAR_SEQ.NEXTVAL,:NEW.MSIRANO,'T',TO_NUMBER(SUBSTR(:NEW.
BILGI,1,10)), 'F',:NEW.TARIH,MAGAZ);
end if;
END IF;
END IF;
IF :NEW.TAMIMKODU = '1030' THEN
select count(*) into SAYI from mal_tanim where msirano = :NEW.msirano;
if SAYI <> 0 then
UPDATE MAL_TANIM SET AKBASTAR =
TO_DATE(SUBSTR(:NEW.BILGI,1,8)),
AKSONTAR = TO_DATE(SUBSTR(:NEW.BILGI,9,8))
WHERE MSIRANO = :NEW.MSIRANO;

end if;
END IF;
END;

```

PROCEDURE YUKLE IS

```
CURSOR C1 IS SELECT
PLU_NUM,PLU_DESC,PLU_PRICE,PLU_STATUS,PLU_DEPT FROM
UL1_PLU;
YENINO MALBAR.MSIRANO%TYPE;
PRICE UL1_PLU.PLU_PRICE%TYPE;
KDV MALYENI.KDV%TYPE;
BARCTIPI MALBAR.BARCTIPI%TYPE;
TERKOD MALBAR.TERKOD%TYPE;
SAYI NUMBER;
BEGIN
```

```
FOR C1REC IN C1 LOOP
  IF SUBSTR(C1REC.PLU_STATUS,20,1) = '1' THEN
    PRICE :=C1REC.PLU_PRICE*10;
    IF (LENGTH(C1REC.PLU_NUM) = 13 AND
SUBSTR(TO_CHAR(C1REC.PLU_NUM),1,2) = '24') THEN
      TERKOD
:=TO_NUMBER(SUBSTR(TO_CHAR(C1REC.PLU_NUM),3,4));
      BARCTIPI :='05';
    ELSE
      TERKOD
:=TO_NUMBER(SUBSTR(TO_CHAR(C1REC.PLU_NUM),1,4));
      BARCTIPI :='03';
    END IF;
  ELSE
    PRICE := C1REC.PLU_PRICE;
    IF (SUBSTR(TO_CHAR(C1REC.PLU_NUM),1,3) = '200') THEN
      BARCTIPI :='02';
      TERKOD := NULL;
    ELSE
      BARCTIPI :='01';
      TERKOD := NULL;
    END IF;
  END IF;
  IF SUBSTR(C1REC.PLU_STATUS,24,1) = '1' THEN KDV :=1;
  ELSIF SUBSTR(C1REC.PLU_STATUS,25,1) = '1' THEN KDV :=8;
  ELSIF SUBSTR(C1REC.PLU_STATUS,26,1) = '1' THEN KDV :=15;
  ELSIF SUBSTR(C1REC.PLU_STATUS,27,1) = '1' THEN KDV :=23;
  ELSIF SUBSTR(C1REC.PLU_STATUS,28,1) = '1' THEN KDV :=0;
```

```
ELSE KDV :=0;
END IF;
```

```
SELECT COUNT(*) INTO SAYI FROM MALBAR
      WHERE MALBAR.BARCODE = C1REC.PLU_NUM;
```

```
IF SAYI = 0 THEN
```

```
  insert into Y
```

```
  (Y_NUM,Y_DESC,Y_PRICE,Y_DEPT,Y_KDV,Y_TERKOD,Y_BARCTIPI,Y
   _MBSTAT) VALUES
```

```
  (C1REC.PLU_NUM,C1REC.PLU_DESC,PRICE,C1REC.PLU_DEPT,KDV,TE
   RKOD,BARCTIPI,C1REC.PLU_STATUS);
```

```
  COMMIT;
```

```
ELSE
```

```
SELECT MSIRANO INTO YENINO FROM MALBAR WHERE
      MALBAR.BARCODE = C1REC.PLU_NUM;
```

```
UPDATE MALYENI SET MALYENI.KDV=KDV,
```

```
      MALYENI.SFIYAT = PRICE,
```

```
      MALYENI.MKADI = C1REC.PLU_DESC,
```

```
      MALYENI.ETIK_STAT = 'C'
```

```
      WHERE MALYENI.MSIRANO = YENINO;
```

```
COMMIT;
```

```
END IF;
```

```
END LOOP;
```

```
END;
```

KAYNAKLAR

- [1] E.LARSON, Robert,John G.O'REILLY,Paul L.Mc ENTIRE Distributed Computing Concepts and Implementations
- [2] DATE, C.J.,Database Systems,Volume 1 Addison-Wesley Publishing Company,(1984).
- [3] R.HEVNER,Alan-YAO,S.Bing Query Processing in Distributed Database Systems
- [4] ÖZSU, M.Tamer-University Of Alberta Edmonton, VALDURIEZ,Patrick-INRIA Paris France, Principles Of Distributed Database Systems, Printence Hall,(1991).

Y.C. YÜSEKÖKÜREKİN KURBAN
DOKÜMANİYASININ MÜHÜRÜ

ÖZGEÇMİŞ:

08.02.1969 tarihinde İstanbul'da doğmuştur. 1986 yılında Yıldız Koleji'nden mezun olup 1987'de İstanbul Teknik Üniversodsi Fen Edebiyat Faköltesi Matematik Mühendisliğı Bölümü' ne girmiştir. Aynı bölümden 1991 yılında mezun olmuştur. Yine 1991 yılında İstanbul Teknik Üniversodsi Fen Bilimleri Enstitüsü Mühendislik Bilimleri Anabilim Dalı Sistem Analizi Programı'nda Yüksek Lisans eğitime başlamıştır.

Bir Koç kuruluşu olan OTOKAR Otobüs Karoseri ve Sanayi A.Ş.' de 1989-1993 yılları arasında part-time, 1993 yılından beri ise Cad Sistem Şefi olarak çalışmaktadır.

**E.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ**