İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DAĞITIK VERİTABANI MANTIĞINA DAYALI DEĞİŞKEN TARİFELİ FATURALAMA OTOMASYONU

YÜKSEK LİSANS TEZİ

Matematik Müh. Bülent DAL

Tezin Enstitüye Verildiği Tarih : 13 Haziran 1994

Tezin Savunulduğu Tarih : 29 Haziran 1994

Tez Danışmanı : Yrd. Doç. Dr. Bedri ŞEFİK

Diğer Jüri Üyeleri : Prof. Dr. Galip TEPEHAN

Yrd. Doç. Dr. Gazanfer ÜNAL

HAZİRAN 1994

ÖNSÖZ

Tez çalışmamdaki yardımlarından dolayı Sayın hocam Prof. Dr. Mithat UYSAL, tez danışmanım Sayın Doç Dr. Bedri ŞEFİK ve HAVAŞ çalışanlarına teşekür ederim.

Bülent DAL

İÇİNDEKİLER

BAZI KISALTMA VE KELİMELER:

VTYS	Veritabanı Yönetim Sistemi
İVTYS	İlişkisel Veritabanı Yönetim Sistemi
DVTYS	Dağıtık Veritabanı Yönetim Sistemi
DVTS	Dağıtık Veritabanı Sistemi
IATA	International Air Transport Association
GHCN	Ground Handling Charge Note
Network	Ağ
Client	İstemci
Server	Sunumcu
Distrubuted	Dağıtık
Processing	İşlem, İşleme
Host	Üye, Makine ismi
Service	Servis
OSI	Open Systems Interconnection
TCP	Transfer Control Protocol
IP	Internet Protocol
Layer	Katman
Session	Oturum

ŞEKİL LİSTESİ

ÖZET

Bu çalışmamda havacılık sektöründe, birden fazla coğrafi yerel yapı üzerinde yer hizmeti veren bir şirket üzerindeki faturalama sisteminin ihtiyaçlarını karşılayan bir uygulama geliştirdim.

Uygulama Konusu, şirketin organizasyonel yapısı ve konuyla ilgili işlemlerin yerel yapılar itibarıyla farklılık göstermesinden dolayı dağıtık veritabanı yönetim sistemi mantığıyla tasarlanmıştır. Şirket üzerinde bir bilgisayar ağına bağlı farklı coğrafi yerlerde altı adet Unix makine mevcutttur. Bu makineler birbirleri ile X.25 ağı ile bağlıdır. Her makinenin kendi yerel veritabanı mevcuttur.Her veritabanını kullanan uygulamalar yerel yapılar itibarıyla birbirinden bağımsız çalışmaktadır. Fakat gerektiği zaman kullanılan veritabanı yönetim sisteminin sağladığı Client-Server mantığı ile veritabanları arasındaki bilgi akışı veya güncellemeler sağlanmaktadır. Bu da veritabanları arasındaki mantıksal korelasyonu mümkün kılmaktadır. Mümkün olduğu kadar bütün yerel uygulamalar bir başka makinedeki veritabanınına bağımlı olmadan kesintisiz çalışmaktadır.

Tez konu itibarı ile üç ana bölüm bazında hazırlanmıştır. ilk bölümde; dağıtık veritabanı yönetim sistemleri ile ilgili temel tanımlar, problemler, avantaj ve dezavantajlar, dağıtık veritabanı olma kuralları ve TCP/IP - OSI referans modeli hakkında bilgi verilmektedir. İkinci bölümde; Uygulamanın geliştirildiği Progress Veri Tabanı Yönetim Sistemi hakkında dağıtıklık ve veritabanı değerlendirme kriterleri baz alınarak değerlendirme yapılmaktadır. Üçüncü bölümde; uygulama: amacı , geliştirilme yöntemi, bilgi akışı, programlar ve çıktılar vs. yer almaktadır.

SUMMARY

Invoicing Automation with Changeable Price List, On the Idea Of Distrubuted Database System

It's well known that during the 1970's computers were extensively used for building powerful, integrated database systems. The database systems has become the basic element of yhe information systems. One of the most important problems of database systems whch they were slow, is becoming not a problem with the development of the computer technology. Because of this development a lot of data processsing center in an action from software's which is coded with by a third generatiom language to software's which is coded by any fourth generation language on a database management system.

After this development of database systems, computer networks have been extensively developed, allowing the connection of different computers and the exchange of data and other resources between them. In recent years the availability of databases and computer networks has given rise to a new field: distrubuted databases . A distrubuted database is, in brief, an integrated database which is built on top of a computer network rather than on a single computer. The data which constitute the database are stored at the different sites of the computer network, and the application programs which are run by the computer access data of different sites.

In this study, it is developed an application which is built on the idea of distrubured processing and distrubuted database system, on the Airports Ground Handling Company called HAVAS.

HAVAS is a sub company of the State Airport Authority founded in March 1987. The basic facility of Havas is to give every kind of Handling Services to scheduled and non-scheculed aircrafts and their passengers, baggages and cargo. Havas is performing services at five international Turkish Airports. These airports are Istanbul - Ataturk, Ankara - Esenboga, Izmir - Adnan Menderes, Antalya and Mugla - Dalaman .

Havas is giving Handling Services to sixteen national and one hundered and four foreign, total one-hundered and twenty air transporting firms. There world top airlines among these firms such as Air France, British Airways, KLM, Aerolloyd, Delta, Singapore.

Havas's personnel number is 2375, 222 out of this qualified personnel are employed at the General Management and the rest 2135 personnel are working at the airports mentioned above.

The services given by Havas are generally :

i.) Representation and Communications
ii.) Load Control and Communications
iii.) Unit Load Device Control
iv.) Passenger and Baggage
v.) Cargo and Mail
vi.) Ramp
vii.) Aircraft Servicing
viii.) Fuel and Oil
ix.) Flight Operations and Crew Administration
x.) Surface Transport
xi.) Catering Services (Catering Ramp Handling)
xii.) Supervision and Administration

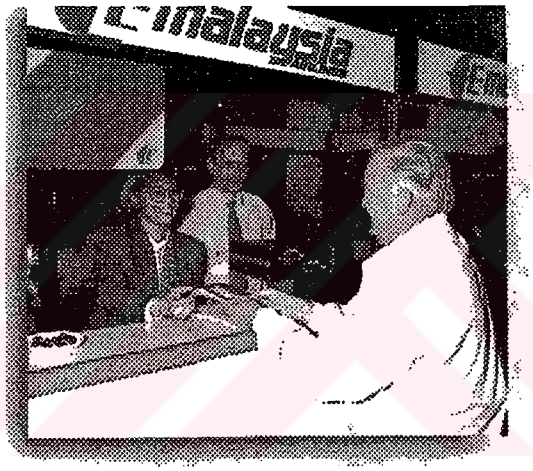


Figure 1. Passenger and Baggage

Havas as a company which has a very big effect in the development of Turkish tourism and as company which is very important in Turkish Air Transportation aims high quality in Ground Handling Services more than profit.

The Havas computer automation project has been in progress since 1988. Until 1992 there were only PC's, at the stations in the use for the Fligths connected the SITA network (A Common Airline Companies Network) and at General Management for some management applications. Since the last of 1992, all the application on PC's and the new applications which would be used also at the stations which would be related with General Management has been moved the new multi user systems (UNIX) connected by the X.25 (TURPAK) network.

One of these projects is Ground Handling Charge Notes and Invoicing automation project. It is about the charging and entry of the services which is given at the stations and the invoicing of these services to the interested firms.

Ground Handling Charge Notes (shortly GHCN) represents the services and the prices without discount which is given for a flight. The real prices for services are being represented in the invoices in the invoices.

There are two kinds of invoices: Cash Invoices , for the firms which are contracted as Cash and Credit invoices, for the firms which are contracted as Credit.

For the services, the source for invoices are GHCN's. For the Cash firms every GHCN corresponds to an invoice and for the Credit firms all GHCN's from individual stations which in the period of time credit, correspond to only one invoice.

Cash firms arebeing invoiced at the stations which give the services. Credit firms are being invoiced at the General Management at Istanbul for ever station, every period of time credit and every kind of speciality (flight type, flight place etc.)

As it is clearly seen that some works are being done at some individual local places and some works are not being done at some local places:

i.) Every station manages only the services which they give.

ii.) GHCN's are being formed only at the stations.

ii.) Cash firms are only being invoiced at the stations which the services are given.

iv.) Credit firms are only being invoiced for the services given at the stations in the General Management.

v.) The firms are being contracted at the General Management and the stations have to obey these contracts.

In this position for the reasons which are examined in this study, distrubuted database system are the most reliable with the advantages of distrubuted processing and storage capabities.

Distrubuted Processing uses more than one processor to divide the processing for a set of related jobs. Distrubuted processing reduces the processing load on a single processor by allowing diffrent processors to concentrate on a subset of related tasks, thus improving the performance and capabities of the system as a whole.

A database system should easily take advantage of distrubuted processing by using its client-server architecture. In this architecture, the database system, is divided in two parts: a front-end or a client portion and a back-end or server portion.

The Client portion is the front-end database application and interacts with a user through the keyboard, display, and the pointining device such as a mouse.The client portion has no data access responsibilities; it concentrates on requesting, processing and presenting data managed by the server portion. The client workstation can be optimized for its job. For example it might not need a large disk capacity or it might benefit from graphic capabilities.

The server portion runs the RDBMS software and hendles the functions required for concurrent, shared data access. The server portion receives and processes SQL or any other 4GL statements originating from client applications. The computer that manages the server portion can be optimized for its duties. For example , it can have large disk capacity and fast processors.

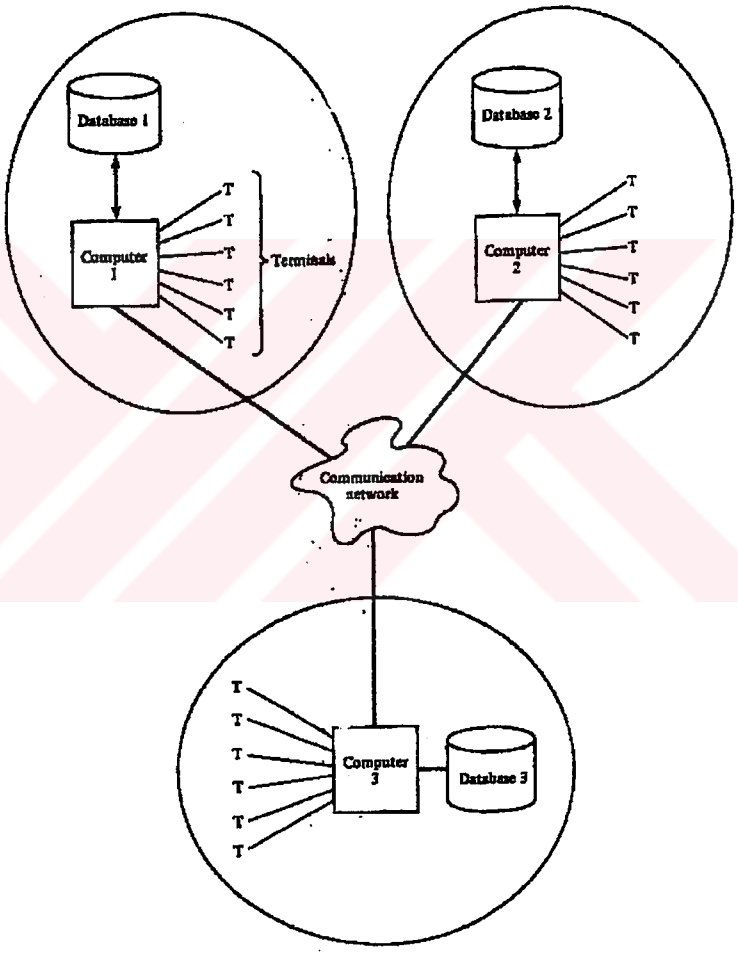


Figure 2. A distrubuted database system on a geographical dispersed net-

A distrubuted database is a collection af data which belong logically to the same system but are spread over the sites of a computer network. This definitions emphasizes two equally important aspects of distrubuted database:

i.) Distribution: i.e. The fact, that the data are not resident at the same site (processor) , so that we can distinguish a distrubuted database from a single, centralized database .

ii.) Logical Correlation: i.e. the fact that the data have some properties which tie them together, so that we can distinguish a distrubuted database from e set of local databases or files which are resident at different sites of a computer network.

All of the application is developed by Progress RDBMS. In recent years Progress has made a big development in Database Management Systems Area. It is a new product in this area. With the advantages and easines in developing an application Progress 4GL is very helpful. It has an individual 4GL very different from SQL, which is the base of the other database systems such as Oracle, Informix, Sybase. But it also supports the first level of ANSI SQL fully and also the second level partly. In Progress :

```
select * from customer      and
For each customer:
        Display customer.
end.
```

produces the same report.

In Progress 4GL ,with the abilities of block structures which is very close to 3GL's ,makes the programmer stronger to produce complex parametric applications. In the study I used only the Progress 4GL to develop applications.

General Management has full compiler, run time end the other Progress components. The other machines have only Progress Run-Time module. So the applications are developed on the General Management . The other machines only runs the generated Progress run-time code.

For the distrubuted scenario on the project, The Client - Server Architecture and the database connection properties of Progress were used. TCP/IP protocol was used to make database connections between different machines.

There are six Unix Machine at different local places (General Management and Istanbul - Atatürk , Ankara - Esenboga, Izmir - Adnan Menderes, Antalya - Antalya, Mugla - Dalaman Airports). All sites have their own local database. All Machines are connected each other with packet switching X.25 (Turpak) network. Database connections on TCP/IP is being over X.25. For file transfers between systems ftp or uucp are used. And for remote logins "pad" over X.25 and telnet over TCP/IP are used.

The main definitions on distrubuted processing and distrubuted databases, on overview of the characteristics the problems , the advantages and disadvantages of distrubuted databases, the compilaion of 12 rules by the author C. J. DATE are being represented in the first section.

In the second section, Progress RDBMS, which is used to manage this distrubuted system and to develop application, is examined by depending on the specialities which characterizes a Database Manageement System and Distributed Database Management Systems .

In the next section the information about the application , the system and also some explanations on the programmes about distrubution from the apllication with the chosen examples is given.

At the last section some important programmes and outputs are placed.

BÖLÜM 1. GİRİŞ

HAVAŞ HAVAALANLARI YER HİZMETLERİ A.Ş.

FAALİYET ALANI:

Şirket uluslararası hava trafiğine açık Türk havaalanlarında tarifeli ve tarifesiz seferler yapan Türk ve yabancı uçaklara ve bu uçaklarla taşınan yolcu ve kargoya her türlü yer hizmetini (handling) vermektedir.

Trafik, harekat ve teknik olarak verilen yer hizmetleri kapsamında;

a. Hizmet verilen şirketlere ait uçakların uçuş programlarının hazırlanması, uçakla ve uçağın hareket ettiği (geldiği-gideceği) meydanlarla haberleşme sağlanması,

b. Açık alana veya körüğe park etmiş uçakların Push-Back araçlarıyla itme ve çekme işlemlerinin yapılması, merdiven ve körüğün yanaştırılması, yolcu kapılarının açtırılarak gelen ve giden yolcuların uçak terminal arasında taşınması,

c. Uçağın ambar kapılarının açtırılarak yolcu bagajı ve her nevi ağırlıktaki kargoların konveyör belt ve high loader cihazlarıyla yükleme ve boşaltmalarının yapılması, bagajların yolcu terminaline, kargoların ise kargo terminaline teslim işlemlerinin yapılması,

d. Uçakla gelen yolcuların bilet pasaport ve bagaj (check-in) işlemlerinin yapılması, bu işlemlerle ilgili diğer yetkili kurumlarla (Polis, Gümrük, DHMİ vb.) işbirliği yapılarak gerekli form ve dökümanların düzenlenmesi, bilgilerin ilgili ünitelere aktarılması,

e. Uçakların teknik bakımı ve kontrolünün yapılması neticesinde ilgili uçak kaptanına veya yetkilisine bilgi verilmesi,

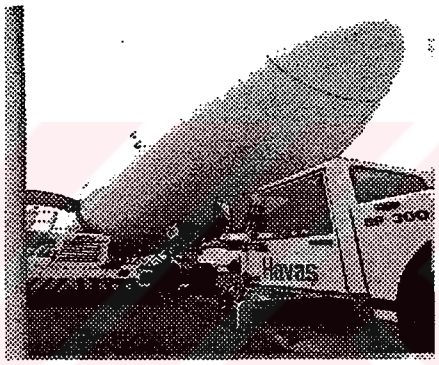
f. Uçaklara gerekli olan her türlü uçak içi ısıtıcısı soğutucuları uçağa enerji yükleyen özel jeneratörler uçak motoru için sıcak ve tazyikli hava püskürtücü araçlar, hareketli yolcu merdiveni, yük ve kargo yükleme-boşaltma araçlar, buz çözücü araç, uçak çekici traktörler, uçak foseptik araçları vb. teknik araç ve

techizatla işletme ihtiyaçları'nın giderilmesi, teknik cihazlarla uçakların her türlü iç ve dış temizliği'nin yapılması,

g. Uçağın yolcu yakıt ve kargo figürlerinin ilgili formlara işlenip gerekli teknik hesapların yapılarak yükleme dağıtım planlarının hazırlanması ve bu planlar doğrultusunda uçakların emniyetli uçuşlarının temini,

h. Uçakların uçuş harekat hizmetleri çerçevesinde hareketinden önce uçuş güzergahındaki ve yedek meydanlardaki meteorolojik raporların incelenmesi, uçuş planlarının ve slotların takibi, uçuş esnasında ortaya çıkabilecek yer hizmetleriyle ilgili ihtiyaçların temininde hava-yer arası haberleşmenin sağlanması olup,

i. Ayrıca havaalanları şehir arası yolcu taşınması, otopark işletmeciliği vb. hizmetler Havaş tarafından sağlanmaktadır..



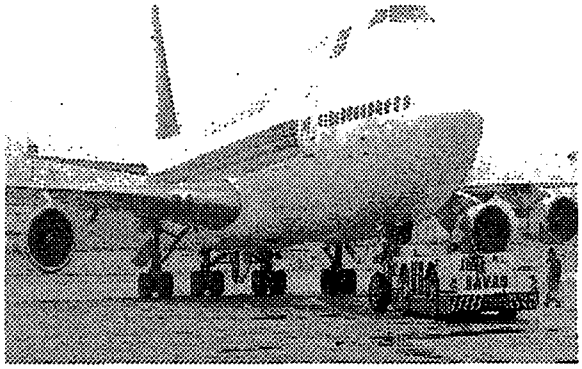
Şekil 1.1. Hizmet verilirken

Şirket Yönetim Kurulu ile Genel Müdürlüğü'nün yer aldığı merkezi İstanbul'da, halen hizmet verdiği beş İstasyon ise, İstanbul, Ankara, İzmir, Antalya ve Dalaman Havaalanlarındadır. Ayrıca bu sene içerisinde hizmete açılan Adana, Trabzon istasyonlarında da hizmet verilmektedir.

Havaş şu anda Türkiye'de havaalanlarında yer veren iki büyük şirkettten biridir ve halihazırda Türkiye"de hizmet alan havayolu şirketlerinin %65 'ine hizmet vermektedir. Bu hizmet verilen firmaların çoğu yabancı havayolu şirketlerinden oluşmaktadır. Verilen hizmetler uçağın tonajına göre değişmektedir.

Havaş yukarıda bahsedilen hizmetleri daha sağlıklı verebilmek, Şirket yönetimine ve hizmetlerin daha verimli planlanmasına faydada bulunmak ve kendi bilişim sistemini kurmak üzere çalışmalara başlamıştır. Daha çok şirket yönetimi bazında var olan bilgisayar uygulamaları, 1993 yılından itibaren bağlanılan X.25

(Turpak) ağı ve Genel Müdürlük ve İstanbul, Ankara, İzmir, Antalya , Dalaman istasyonlarında kurulan Unix işletim sistemli makinelerle hizmete dayalı



Şekil 1.2. Push Back Hizmeti

uygulamaların geliştirilmesiyle devam etmiştir. Bu çalışmalar halen devam etmektedir.

Unix sistemleri üzerindeki uygulamalar İlişkisel Progress Veritabanı Yönetim Sistemi ile geliştirilmektedir. Gerekli uygulamalarda veritabanı bağlantıları kullanılarak Dağıtıklık özelliği sağlanabilmektedir. Veritabanları arasındaki bağlantılar X.25 üzerinden, TCP/IP network protokolü kullanılarak sağlanmaktadır.

Benim tez konusu olan uygulama ise, yukarıda bahsedilen hizmetlerin takibinin yapılabilmesi, verilen hizmetlerin şirketlere ilgili birimler tarafından vaktinde ve hatasız olarak fatura edilmesini, hizmet verilen firmalarla ilgili bilgi takibinin yapılması ve bu hizmetlerin muhasebeleşmesi görevlerini üstlenmektedir.

Sistemin Kurulması ve Uygulamanın Gerçekleştirilmesine Etki Eden Sebepler:

20. yüzyılın sonlarına yaklaşırken, bilişim dünyasındaki gelişmeler tabiri caizse ışık hızıyla devam etmektedir. Amaç bir işletme yönetiminde olduğu gibi mümkün olduğu kadar minimum maliyetle , maksimum faydayı elde etmektir. Maliyeti belirleyen faktörleri genellikle donanım , yazılım, iletişim , bakım-onarım, Bilgi işlem kadrosu ve bunların birçok alt unsurları belirler. En çok fayda görülmesi beklenen unsurların başında özellikle donanım açısından tabii ki hız gelmektedir; fakat hızla beraber güvenlik, modülerlik, özellikle açık bir sistemin gereklerinin yerine getirilmesi gibi unsurlar büyük önem kazanmaktadır. Donanımların hızları açısından olaya baktığımızda; teknolojik gelişmelerin bilgisayar dünyasına da yansımasıyla

artık çok güçlü-hızlı makinelerin eskiye göre çok daha ucuza mal olduğu görülmektedir. Hızla beraber bazen daha da önemli olarak donanım ve yazılımların açıklığı, yazılımların güvenilirliği , hataya töleranslı olması, bakımlarının kolay olması vb. gibi faktörler önem kazanmaktadır.

Bahsedilen maliyetleri azaltıp daha fazla performans sağlamanın bir yöntemi olarak işlem yükünü merkezi bir yapıda tutmaktansa , farklı sistemlere dağıtmayı hedef alan "dağıtık işleme" (Distrubuted Processing) popüler bir hale gelmiştir. Konuyla ilgili olarak Donanımdaki seyire bakıldığında, artık büyük merkezi bilgisayarların (mainframelerin) yerini daha küçük çaplı fakat güçlü server'ların aldığı; Büyük Bilgisayar ağlarında: haberleşme hızındaki artışlar ve maliyetin azalmasıyla beraber işlem yükünü tek bir makine üzerinde yoğunlaştırmaktansa farklı yerlere farklı makineler koyarak işlemi makineler arası paylaştırarak daha iyi bir performans ve daha az haberleşme maliyeti sağlandığı , Yerel bilgisayar ağlarında: paralel programlama teknikleriyle beraber birden fazla bilgisayarı eşanlı olarak çalıştırıp işlem yükünü dağıtırak, Bellek paylaşımlı çoklu işlemcili güçlü bilgisayarlarda: işlem yükünü farklı işlemcilere dağıtarak daha çok verim alındığı görülmektedir. Bütün bu gelişmelerle beraber açık sistem kavramının donanımda da her geçen gün yerleşmesiyle beraber farklı bilgisayar markalarının, işletim sistemlerininin, farklı haberleşme protokollerinin desteklediği bilgisayar ağlarının beraber etkileşimli olarak çalışabilmesi büyük önem kazandığı görülmektedir.

Yazılımda ise donanımla paralel olarak, dağıtılmış organizasyonel yapıya cevap veren, paralel programlamayı destekleyen yazılım geliştirme araçları ön plana çıkmıştır. Bununla beraber bilgisayarların daha güçlü ve hızlı bir hale gelmesiyle, vaktiyle hızlarındaki yavaşlıklardan dolayı eleştirilen Veri Tabanı Yönetim Sistemleri, veri güvenliği,eş anlılık konrolü, veri bütünlüğü, veri entegrasyonu, daha modüler program yazma konusunda getirdiği avantajlarla uygulama yazılımları geliştirmek ve bunların bakımını sağlamak açısından üçüncü kuşak programlama dillerine göre daha çok kullanılır olmuşlardır. Tabii ki dağıtık organizasyonel yapılar için de dağıtık veri tabanı kurallarına daha uygun veritabanı yönetim sistemleri daha kullanılır hale gelmişlerdir.

Ben de tez çalışmamda "GHCN ve Faturalama" Projesini şirket organizasyonel yapısını göz önünde bulundurarak, bilgisayar ağının tanıdığı imkanlar ölçüsünde dağıtık işlem ve dağıtık veritabanı mantığı ile gerçekleştirdim.

BÖLÜM 2. DAĞITIK VERİTABANI YÖNETİM SİSTEMLERİ

2.1. Dağıtık Veritabanları ve Dağıtık Bilgi İşleme

2.1.1. Dağıtık Bilgi İşleme (DISTRUBUTED PROCESSING)

Dağıtık Bilgi İşleme son yıllarda gittikçe popüler olan ve her geçen gün gelişmesini sürdüren bir kavramdır. Fakat daha hala gelişmesini sürdürdüğünden dolayı gerçekten birçok tanımla ifade edilmekte hatta bazen çok yanlış yakıştırmalara uğramaktadır. Örneğin çok işlemcili sistem, bilgisayar ağı, dağıtık fonksiyon, uydu bilgisayarları/uydu bilgi işlemesi vs. gibi kavramlarla yeri geldiğinde yanlış bir şekilde özdeş tutulmaktadır. Bu konu üzerinde birçok kişi tarafından değişik tanımlamalar yapıldığından dolayı gerçekten tam bir kavram kargaşası yaşanmaktadır.[1]

Kavram Kargaşasına yol açmayacak bir Dağıtık İşleme tanımı yapmak istenirse: Bir bilgisayar networkü ile birbirlerine bağlı , birden fazla, özerk işlem gerçekleştiren elemanların kendilerine verilen görevi beraberce yerine getirmeleri durumudur şeklinde ifade edilebilir. Burada eleman tanımından kendi başına program icra eden araç (bilgisayar) kastedilmektedir. Dağıtık Bilgi işlemede dağıtılan ögeler: İşlem mantığı, fonksiyon, data ve kontroldür.

Dağıtık Bilgisayar Sistemleri birkaç kritere göre sınıflandırılabilir. Bunlar bağlılık(coupled) derecesi, arabağlantı yapısı, bileşenlerin karşılıklı dayanışması ve bileşenler arasındaki senkronizasyondur. Bağlılık derecesi, işlem gerçekleştiren elemanların biribirlerine ne kadar yakın bağlı olduğunu ifade etmektedir. Bu yakınlık ölçüsü değişen veri miktarının, gerçekleştirilen işlem miktarına olan oranıyla saptanır. Eğer haberleşme bir bilgisayar ağı üzerinde ise, ilşlem gerçekleştiren elemanlar arasında zayıf bağlılık vardır. Fakat bileşenler paylaşılıyorsa kuvvetli bir bağlılık söz konusu demektir. Paylaşılan bileşenler ana bellek veya ikincil depolama araçlarının her ikisi de olabilir. Ara bağlantı yapısı, işlem gören elemanların birbirlerine noktadan-noktaya şeklinde bağlı olduklarını veya bütün elemanlara açık genel bir kanalla bağlı olduklarını; yani arabağlantının hangi şekilde gerçekleştiğini ifade etmektedir. Bileşenlerin karşılıklı dayanışması ,bileşenlerin görevlerini yerine getirirken birbirlerine aşırı bağımlı çalıştıklarını ya da minimum düzeyde bağımlı olarak, işlemin başında veya sonunda gönderilen mesajlar

düzeyinde bağımlılık gösterdiğini ifade etmektedir. Senkronizasyondan kasıt ise işlemin senkron veya asenkron modda gerçekleştiğidir. Dikkat edilmesi gereken bu kriterler ayırt etmede kendi başlarına bağımsız olarak rol oynamamaktadırlar. Örneğin işlemler arasında senkron bir ilişki varsa işlem elemanlarının birbirlerine kuvvetli bir şekilde bağlı oldukları (tightly-coupled) ve birbirlerine çok yakın oldukları söylenebilir.

Günümüzde Bilgi İşlem dünyasındaki organizasyonel yapılara bakıldığında, daha çok geniş alanlara dağıtılmış sistemlere dayalı yatırımlar göze çarpmaktadır. Bu sistemler homojen ya da heterojen yapıda olabilmektedir. Veriler genelde girildikleri yerlerde, fiziksel bir harekete uğramadan kalabilmekte ve yeri geldiğinde bu veriler diğer taraflardaki verilerle beraber işlenebilmektedirler. Bununla beraber bellek ve işlem elemanı maliyetleri de sürekli azaldığından, dağıtık bir sistem kurmak ekonomik açıdan da rahatlık getirmektedir. Daha fazla genişleme şansı meydana gelmekte, veri işleme fonksiyonları arasındaki değişimlerin etkisi azaltılmakta ve daha evvelden paylaşılamayan farklı yerlere dağılmış verilerin paylaşılabilmesi imkanı doğmaktadır. Bu sebeplerden dolayı dağtık işleme daha çok tercih edilir olmaktadır.

Daha global bir perspektif içerisinden bakıldığında dağtık işlemenin tercih edilmesindeki ana sebep olarak , karşılaşılan büyük ve karmaşık problemleri iyi tasarlanmış ufak parçalara bölüp çözme yöntemini kullanmakta getirdiği kolaylıklar göze çarpmaktadır. Dağıtık işleme için gerekli yazılım desteği bulunduğunda, bu tür büyük karmaşık problemler ufak parçalara bölünür ve her parça için farklı bilgisayarlarda çalışan yazılımlar geliştirilerek çok işlemcili elemanlardan oluşan , fakat verilen bir görevi daha etkin bir şekilde yapan bir sistem kurulabilir.

Ekonomik açıdan bakıldığında bu şekilde bir yaklaşım iki temel avantaj sağlamaktadır. Birincisi; tek bir işlem elemanı için, işlem hızında ulaşılacak limit noktaya daha çabuk varılmaktadır. Hedef daha fazla işlem gücü elde etmektir; dolayısıyla çok işlemcili bir yapı optimal düzeyde kullanılrak maksimum düzeyde verim alınabilecektir. İkinci sebep, problem daha az veya daha fazla özerk çalışan küçük gruplara bölündüğünden dolayı, yazılım maliyeti belli bir disiplin altına alınabilir. Bilindiği gibi yazılım maliyeti, donanım eğilimlerinin yarattığı maliyetle orantılı olarak artmaktadır.

Dağıtık veritabanı sistemleri de bu bakış açısı içerisinde incelenebilir. Dağıtık işlemeyi daha hızlı ve etkin kullanmada yardımcı olan araçlar olarak da nitelendirilebilirler. Dağıtık veritabanlarının bilgi işleme dünyasına önerdikleri ve veritabanı teknolojisinin halihazırda sağladıkları arasında belli bir analoji (benzeşim) görmek

mümkündür. Bu nedenle hiç şüphe yoktur ki, dağıtık veritabanı sistemlerinin genel amaçlarında, adapte olabilirliğinde, etkinliğndeki gelişmeler dağıtık mantığa dayalı yazılımın gelişmesinde yönlendirme ve görev verme açısından büyük ölçüde katkıda bulunacaktır.

2.1.2. Dağıtık Veritabanları ve İlgili Tanımlar

Son yıllarda dağıtık veritabanları, bilgi işlemede önemli bir alan olmuşlardır ve bu önem artışı günümüzde de büyük bir hızla devam etmektedir.

Bu önem artışının hem organizasyonel hem de teknik sebepleri vardır: Dağıtık veritabanları, merkezi veritabanlarındaki kusurları yok etmekte ve farklı organizasyonel yapılarda, merkezi bir yere bağlı olamayan doğal verileri tutma şansı vermektedir. Farklı şekillerde tanımlar yapılmaktadır. Bunların bazıları şöyledir:

Tanım1: Mantıksal olarak aynı sisteme ait fakat bilgisayar ağı üzerinde farklı sistemlere yayılmış veri koleksiyonudur.[2]

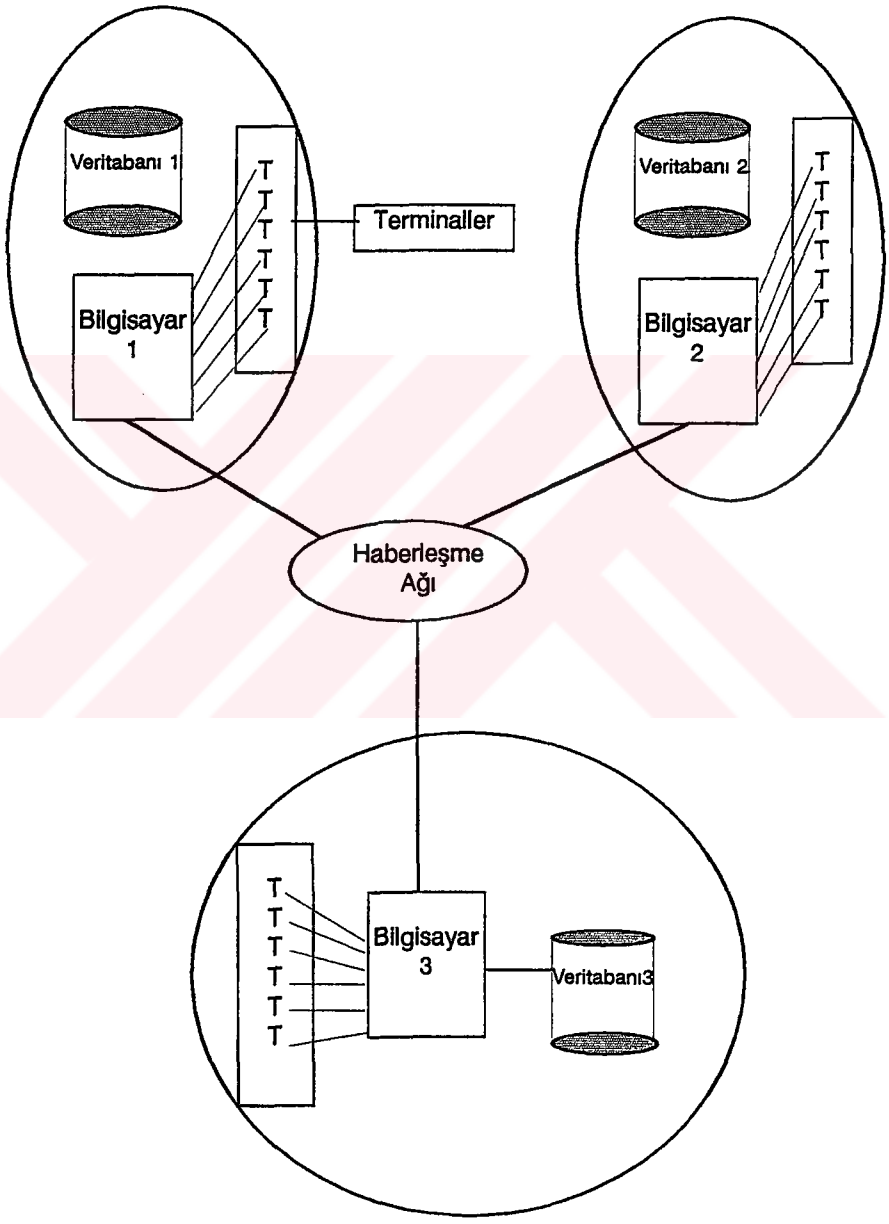
Bu tanımdan dağıtık veritabanlarının iki görünüşü ortaya çıkar. a) Dağıtıklık: Veri tek bir sistemde değildir. Dolayısıyla dağıtık veritabanını tek merkezi bir veritabanınından ayırabiliriz.

b) Mantıksal Korelasyon: Veri kendini birbirlerine bağlayan birtakım mantıksal özelliklere sahiptir. Dolayısıyla dağıtık veritabanını, bilgisayar ağı üzerinde farklı sistemlerde yer alan yerel veritabanları ve dosyalrdan ayırt edebiliriz.

Örnek1:

Farklı coğrafi yerlerde yerlerde üç şubesi olan bir banka olsun: Her şubede bir bilgisayar, konuşan terminallerden gelen isteklerini kontrol etmekte ve hesap verilerini kendi üzerindeki bir veritabanında tutmaktadır. Her şubedeki bilgisayar kendi yerel hesap veritabanı ve belli bir haberleşme ağı ile birbirlerine bağlı bilgisayarların oluşturduğu dağıtık veritabanı sisteminin bir tarafını - alt sistemini oluşturmaktadır. Belli bir şube içerisindeki normal uygulama esnasında terminallerden gelen istekler, o şubede yer alan veritabanına erişilerek karşılanmaktadır. Bu tür uygulamalar tamamıyla yer aldıkları bilgisayar tarafından yürütüldüğü için "Yerel Uygulamalar" olarak adlandırılmaktadır.

Birden fazla şube veritabanı tarafından karşılanan isteklere ilişkin uygulamalar ise "Global Uygulamalar" olarak adlandırılmaktadır. Bir dağıtık veritabanı üzerinde her iki türden uygulamalarda olmalıdır. Global bir uygulamaya örnek olarak; bir



Şekil 2.1.1. Coğrafi Sistem Üzerindeki Dağıtık Veritabanı

şubedeki banka hesabından başka bir şubedeki banka hesabına yapılan havale işlemi verilebilir. Bu uygulama iki şubedeki veritabanlarının güncellenmesini gerektirmektedir. Yalnız , burada dikkat edilmesi gereken bu işlemin basit anlamda iki veritabanının güncellenmesi işlemi olmadığıdır. İşlemin başarı ile sonuçlanması için iki güncellenmenin de başarı ile gerçekleştiğinin onayının gelmesi gereklidir. Bu tür coğrafi bir sistem üzerinde yayılı bilgisayarlar arası işlem yapmak yerel işlemlere göre daha güçtür.

Örnek2:

Aynı banka uygulamasının şekil 2.1.2. deki konfigürasyon üzerinde gerçekleştiği varsayılsın. örnek1'deki aynı işlemcilere sahip bilgisayarlar, veritabnlarıyla aynı bina üzerinde üç farklı şubeye hizmet verecek şekilde ayrılmışlardır. Bu sefer çok daha hızlı bir yerel ağ ile birbirlerine bağlıdırlar. Her bir işlemci ve veritabanı yerel bilgisayar ağının belli bir tarafını-alt sistemini teşkil etmektedir.

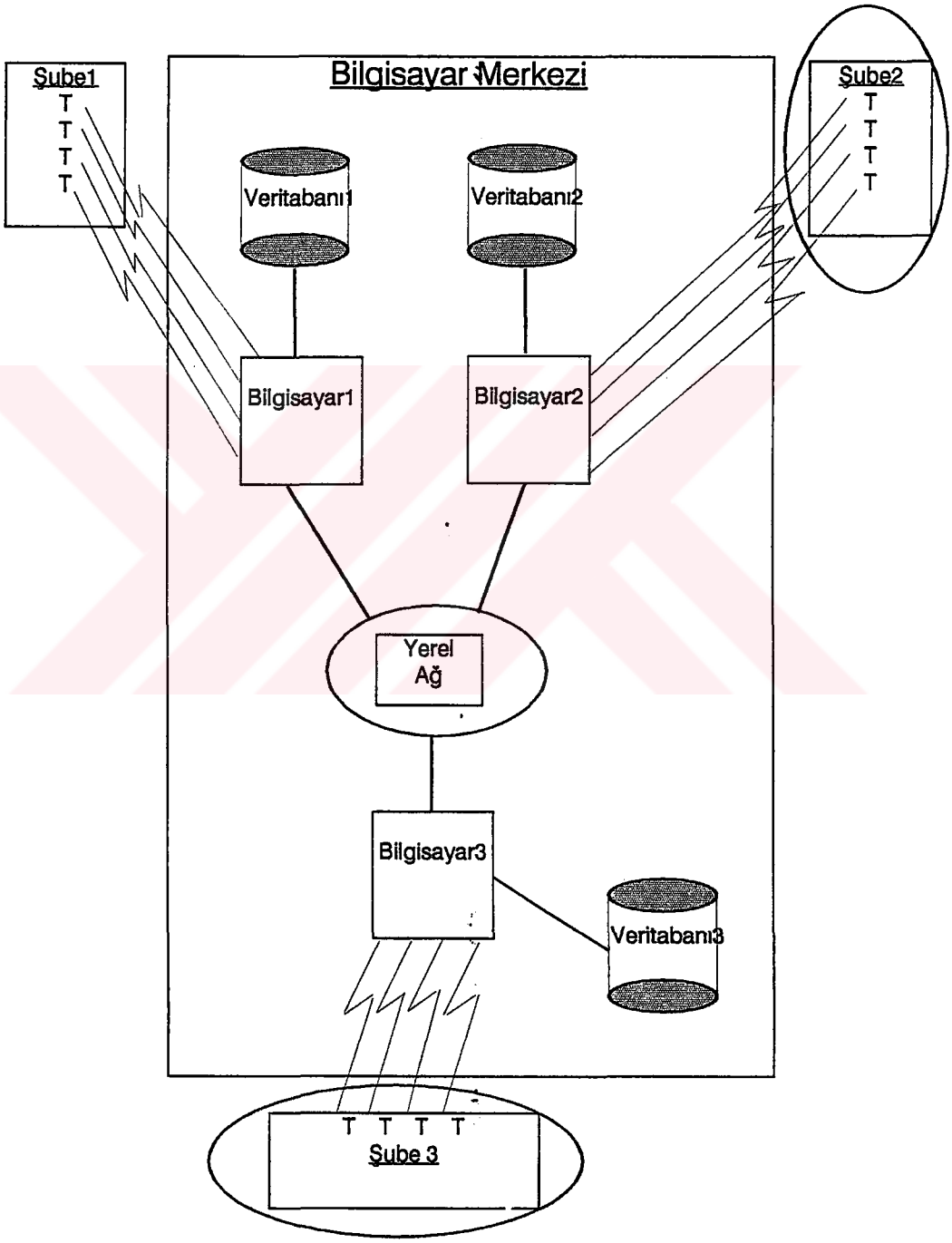
Örnek1'deki bütün uygulamalar, aynı veritabanları ve bilgisayarlar ile gerçekleşmektedir. Aradaki tek fark tarafların bir coğrafi yapı üzerinde yayılmamış olmasıdır. Yerel bir ağ yapısı, coğrafi ağ yapısına göre daha fazla uygunluk ve daha iyi bir performans göstermektedir. Coğrafi sistemi biribirine bağlayan ağı on-line hareketlere cevap verebilecek düzeyde olması gerekmektedir.

Örnek3:

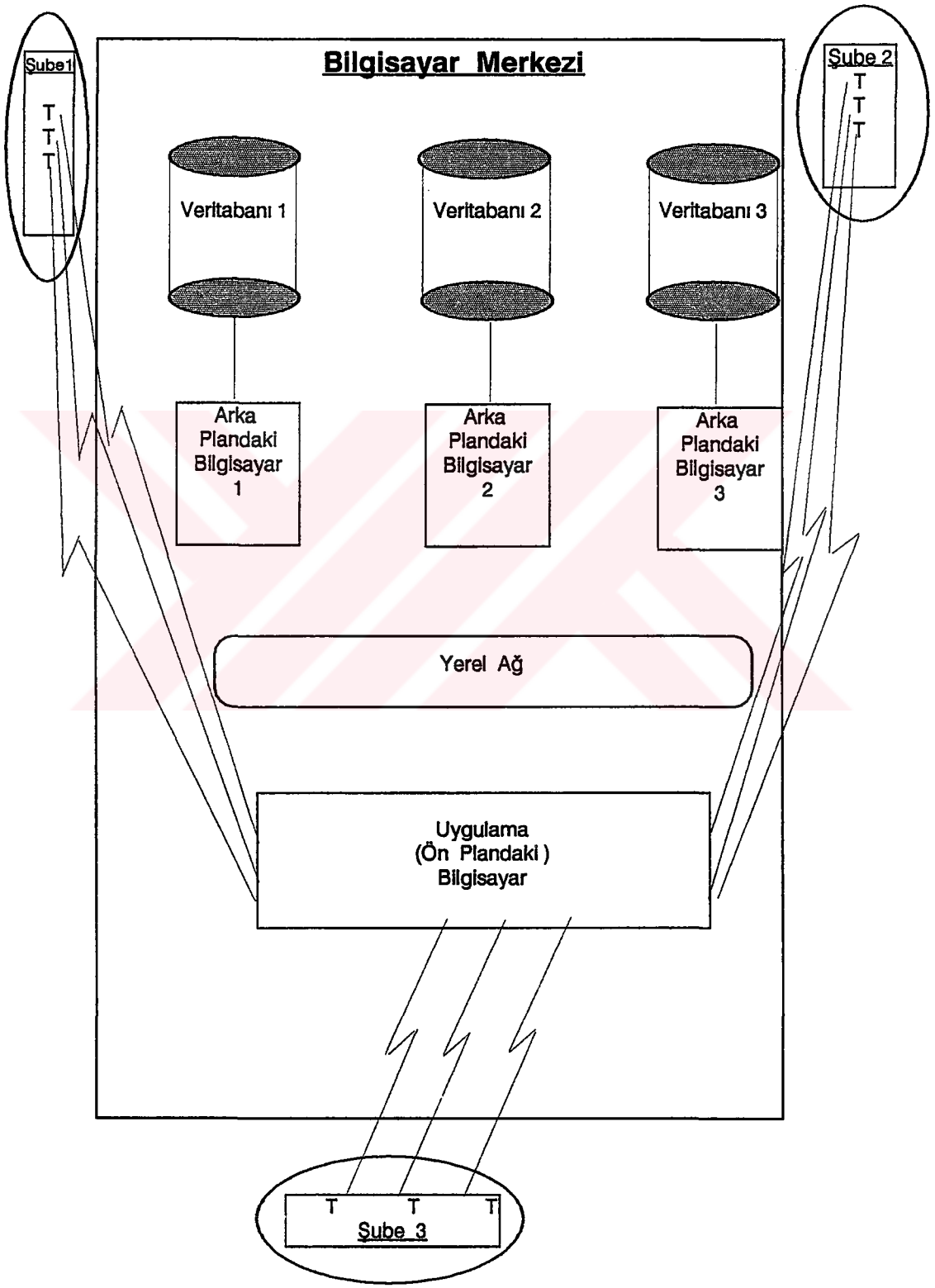
Yine aynı bankanın aşağıdaki şekilde görüldüğü gibi bir bilgisayar sistemine sahip olduğu düşünülsün: Farklı şubelerin verileri, veritabanı yönetim fonksiyonlarını yöneten arka plandaki üç bilgisayara dağıtılmıştır. Uygulama programları ön plandaki farklı bir bilgisayar tarafından icra edilmektedir; bu bilgisayar şubelerden gelen isteklere göre arka plandaki bilgisayarlardaki veritabanı servislerine ulaşmaktadır.

Bu şekildeki bir sistem dağıtık bir veritabanı olarak nitelendirilemez. Veriler fiziksel olarak farklı işlemcilere dağıtılmasına karşılık, aynı dağıtım uygulama programalrının işleyişinde yapılmamaktadır. Burada dağıtıklık için, yerel uygulamalrın eksikliği görülmektedir. Ön plandaki bilgisayarda bir sorun çıktığı zaman tüm sistem çalışamaz durumda atıl kalmaktadır.

Bu örneklerden sonra tanımı şu şekilde daha ayrıntılı yapabiliriz.



Şekil 2.1.2. Yerel Bir Ağdaki Dağıtık Veritabanı



Şekil 2.1.3. Çoklu İşlemcili Bir Sistem

Dağıtık bir veritabanı, bir bilgisayar ağı üzerinde farklı bilgisayarlarda dağıtılmış veri kolleksiyonudur. Ağ üzerindeki her sistem özerk proses yeteneğine sahiptir ve yerel uygulamaları icra edebilir. Her sistem an azından tek bir global uygulamanın işleyişine katılır ve bu global uygulama haberleşme alt sistemini kullanarak birden fazla sistemdeki veriye erişme ihtiyacı duyar.

Tanım 2: (C.J. DATE)

Bir dağıtık veritabanı yönetim sistemi , belli haberleşme ağlarına bağlı öyle sistemlerin kolleksiyonundan ibarettir ki

- Her sistem kendi haklarına sahip bir veri tabanı sistemidir, Fakat

- Sistemler -gerekliyse birlikte çalışmayı kabul etmişlerdir, öyle ki herhangi bir sistemdeki kullanıcı, ağ üzerindeki herhangi yerdeki bir veriye sanki veri kendi sistemindeymiş gibi erişebilir.[3]

Şekil 2.1.1 ve şekil 2.1.2 'ye balkılırsa; yukarıdaki tanımdan yola çıkarak : Her sistem kendi gerçek veritabanına sahiptir. Öyle ki her sistemin kendi yerel kullanıcıları, kendi yerel VTYS ve hareket yönetici yazılımı ve kendi veri haberleşme yöneticisi vardır. Özel olarak; belli bir kullanıcı o veri üzerindeki işlemleri o sistem sanki dağıtılmış bir sistem değilmiş gibi icra edebilmelidir.

Tanım 3 (CLIENT- SERVER YAKLAŞIMI) :

Client - Server Mimarisi (Dağıtık İşlem): Dağıtık işlem, birtakım bağlantılı işleri işlemeyi bölmek için birden fazla işlemci kullanır. Dağıtık işlem, farklı işlemcilere alt gruplardaki ilişkili işleri toplatarak ve böylece tüm sistemin performans ve yeteneklerini artırarak, tek bir işlemciye yüklenen işlem yükünü azaltır.

Bir veri tabanı sistemi kendi client-server mimarisini kullanarak dağıtık işlemenin avatajlarına sahip olmalıdır. [4]

Client:

Bu kısım front-end veri tabanı uygulamasıdır ve klavye, ekran veya mouse başındaki kullanıcı ile birbirini etkiler vaziyettedir. Client kısmının veriye erişim sorumluluğu yoktur; Server tarafından idare edilen istekleri, işlemleri ve veri sunumunu toplar. Client iş istasyonu, yaptığı iş için optimize edilebilir. Örneğin çok fazla disk kapasitesine ihtiyaç duymayabilir veya daha çok grafik yeteneklerden faydalanabilir şekilde konfigüre edilebilir.

Server:

Bu kısım İVTYS (İlişkisel VTYS) yazılımını çalıştırır ve eşanlı ve paylaşımlı veri erişimi için ihtiyaç duyulan fonksiyonları idare eder. Server kısmı, Client uygulamalarından kaynaklanan SQL veya 4GL programlarını alır ve işletir. Server kısmını idare eden bilgisayar görevleri için optimize edilebilir. Örneğin büyük disk kapasitesine ve hızlı işlemcilere sahip olmalıdır.

Dağıtık Veri Tabanı:

Kullanıcıya mantıksal tek bir veritabanı gibi görünen, çoklu veritabanı server'ları tarafından idare edilen veri tabanları ağıdır. Bütün veritabanlarının datası eşzamanlı olarak erişilebilmeli ve modifiye edilebilmelidir. Dağıtık veri tabanının en önemli faydası fiziksel olarak ayrık veri tabanlarının , mantıksal olarak tak bir parça gibi birleşmesi ve ağ bilgisayar ağı üzerindeki bütün kullanıcılar tarafından erişilebilir olmasıdır.

Dağıtık bir veri tabanında, bir veritabanını idare eden her bigisayara bir düğüm denir. Kullanıcının direkt olarak bağlı olduğu veri tabanına "Yerel Veritabanı" denir. Aynı kullanıcı tarafından diğer erişilen veritabanlarına uzak (remote) veritabanları denir. Yerel bir veritabanı bilgi için uzak bir veritabanına bağlandığında, yerel veri tabanı uzak server'ın Client'ıdır.

Dağıtık veri tabanı ağ üzerinde büyük artan veri hacmine erişimi sağlarken aynı zamanda verinin saklandığı yeri ve erişirken izlenen kompleks yapıyı kullanıcıya hissettirmemelidir. Dağıtık İVTYS her yerel veri tabanını sanki dağıtık değilmiş gibi idare etmenin avantajlarını da sağlamalıdır.

İki Aşamalı Taahhüt

İVTYS dağıtık olmayan bir veri tabanının sahip olduğu veri tutarlılığını aynı güvenle sağlamalıdır. Bu güven de hareket modeli ve iki aşamalı taahhüt mekanizmasının kullanılmasıyla sağlanır. Dağıtık olmayan veri tabanlarında 4GL veya SQL ile program parçacıklarıyla belirlenen hareketin tamamını kabul etmek veya birim olarak geri çevirmek (iptal etmek) mantığıy- la dikkatli bir şekilde planlanmalıdır. İki aşamalı taahhüt mekanizması ağ üzerinde hangi çeşit aksamalar olursa olsun, dağıtık bir hareket bütün çapraşık düğümlerde işlenir ya da bütün çapraşık düğümlerde geri çevrilir. Bu da bütün dağıtık veritabanı üzerindeki veri tutarlılığını sağlar.

Veri Tabanı Kullanıcılarına Tam Bir Şeffaflık:

İki aşamalı taahhüt mekanizması, dağıtık hareketleri oluştu- ran kullanıcılara tamamıyla şeffaf olmalıdır. Veritabanı uygulama (yazılım) yapısı içinde SQL veya 4GL ile extra rutinlere ihtiyaç duyulmaksızın, taahhüt veya kabul komutları veya doğal yapı sayesinde bu mekanizma sağlanmalıdır.

Sistem veya Ağ Sorunlarına Karşı Otomatik Muhafaza:

Taahhüt işlemi sırasında sistem veya ağın herhangi bir yerinde meydana gelen sorunlar karşısında müdahale gören şüpheli hareketler, sorun giderildikten sistem komple çalışır hale geçtikten sonra bütün düğümlerde ya tamamıyla kabul edilmeli, ya da tamamıyla reddedilmelidir.

Opsiyonel Manuel Geri Çevirme Yeteneği:

Sistem veya ağ üzerindeki uzun sürel, sorunlar karşısında yerel veritabanlarının yöneticileri manüel olarak şüpheli hareketleri tamamıyla kabul etme veya tamamıyla geri çevirme imkanına sahip olmalıdır. Bu sayede yerel veritabanı yöneticisi şüpheli hareketler tarafından kilitlenen kayıtları kullanıma alarak serbest bırakmış olur. Başka bir yerdeki sornu nedeniyle bütün sistem kilitlenmemiş olur.

Dağıtık Muhafaza (Recovery) İçin Kolaylıklar:

Eğer, bir veritabanı geçmişte belli bir anda onarılmışsa İVTYS'in onarma kolaylıkları diğer taraflardaki veritabanı yö- neticilerine veri tabanlarını geçmişteki o ana döndürme imkanı vermelidir. Bu bütün veritabanı üzerindeki veri tutarlılığını sağlar.

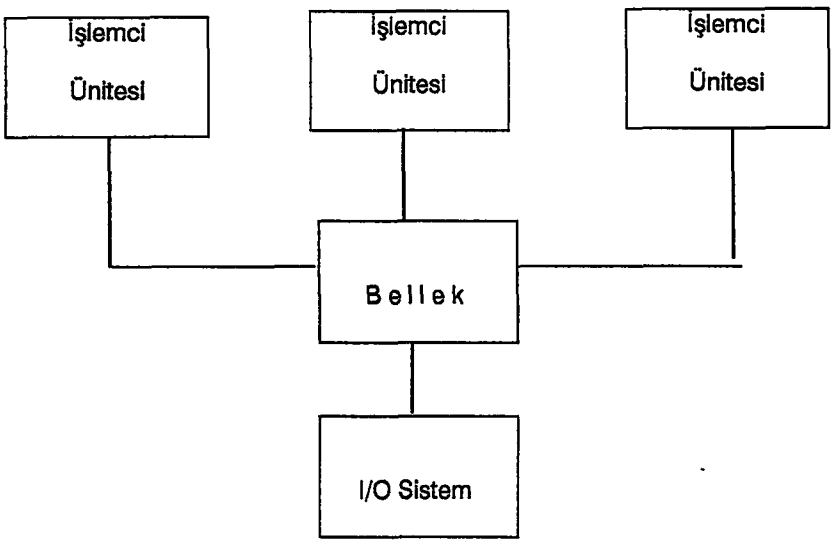
2.1.3. Dağıtık Veritabanı Yönetim Sistemi Nedir; Ne Değildir

Dağıtık veritabanı kabaca bir bilgisayar ağı üzerind birbirleriyle mantıksal olarak ilişkili birden fazla veritabanının koleksiyonudur. Bu tanımdan yola çıkarak Dağıtık Veritabanı Yönetim Sistemi; farklı dağıtık veritabanı sistemlerinin yönetimine izin veren ve dağıtıklığı kullanıcılara şeffaf (heryerden erişime açık) yapan bir yazılım sistemi olarak tanımlanabilir.[3]

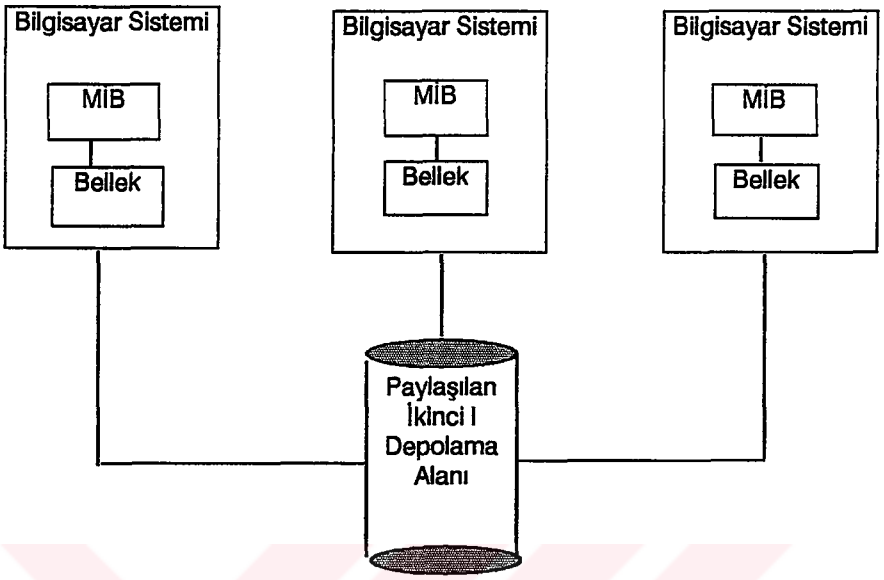
Çoğu yerde dağtık veritabanı sistemi ile ilgili kavram kargaşası görülmekte, farklı sistemler dağtık veritabanı sistemi olarak adlandırılmaktadır. Bu kısımda hangi sistemlerin DVTS olduğu hangilerinin olmadığı incelenecektir. Bu incelemede yukarıdaki tanımlamalardan çıkarılan iki kriter baz alınacaktır. Bu kriterler "Mantıksal İlişkililik" ve "Bir bilgisayar ağı üzerinde dağıtılmış olmak " terimleridir.

Öncelikle DVTS , bir bilgisayar ağı üzerindeki her düğümde yer alan dosyaların koleksiyonu değildir. Dosyaların birbirleriyle sadece mantıksal olarak ilişkili olması da yeterli değildir. Dosyalar arasında yapısal bir bağ olmalı ve bütün dosyalara erişim, genel herkese açık bir arayüzden sağlanmalıdır. Bazen verinin fiziksel ayrılığının en göze çarpıcı unsur olmadığı varsayılır. Bu bakış açısını savunan kimseler aynı bilgisayar sistemi içerisindeki birbirleriyle ilişkili iki veritabanını rahatlıkla dağıtık olarak nitelendirebilmektedir. Halbuki dağıtıklık için verilerin fiziksel olarak ayrılığı büyük önem taşımaktadır. Veritabanları aynı bilgisayar üzerinde bulunduğunda hiç rastalanmayan problemler çıkmaktadır. Dikkat edilmesi gereken bir nokta da fiziksel dağıtıklıktan kasıt , bilgisayar sistemlerinin birbirinden uzak olarak coğrafi bir sistem üzerinde yayılmış olması değldir; aynı oda içinde bile yer alabilirler. Burada önemli olan veritabanları arasındaki haberleşmenin paylaşımlı bellek tarafından değil, bir network tarafından sağlanması ve paylaşılan tek kaynağının network olmasıdır.

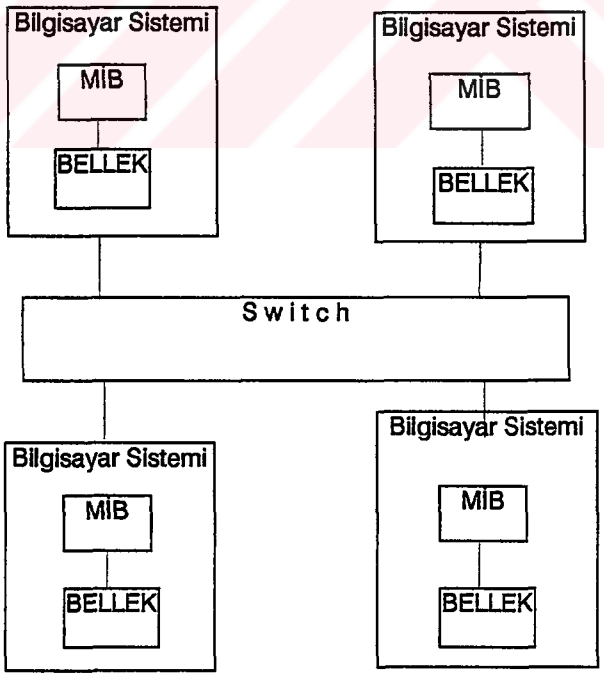
Yukarıda söylenenlerden başka bir sonuç daha çıkarılabilir: Çok işlemcili sistemler dağıtık veritabanı sistemi olarak nitelendirilemezler. Çok işlemcili bir sistem bir veya daha fazla işlemcinin ortak bir belleği; ya birincil bellek (Bunlara sıkı bağlı - tightly coupled çok işlemcili denir. (Şekil 2.1.4) , ya da ikincil bellek - (Bunlara gevşek bağlı - Loosely Coupled çokişlemcili denir.(Şekil 2.1.5.) kullandığı bir



Şekil 2.1.4. Tightly Coupled Çok İşlemcili



Şekil 2.1.5. Loosely Coupled Çoklu İşlemci

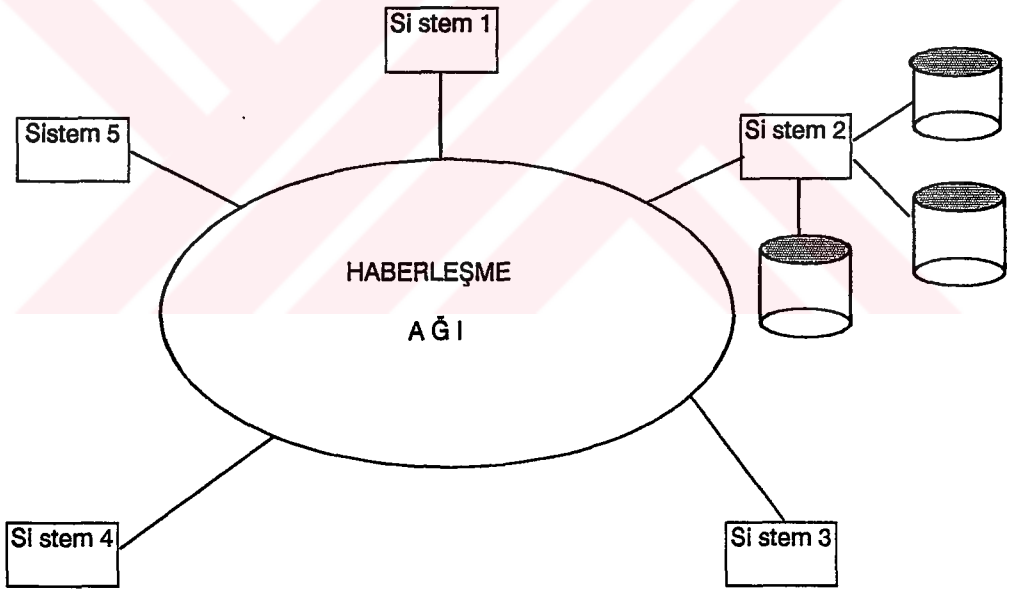


Şekil 2.1.6. Switch bazlı Çok işlemcili Sistem

sistemdir. Paylaşılan bellek işlemcilerin birbirlerine mesaj göndermeden heberleşmesini sağlamaktadır.

Mikroişlemci ve VLSI teknolojisinde meydana gelen gelişmeler doğrultusunda tek bir switch ile birbirine bağlı mikroişlemcilerden oluşan çoklu işlemciler ortaya çıkmıştır. (Şekil 2.1.6.)

Burada sözü gelmişken, genelde yapılan başka bir ayırımdan; herşeyin paylaşıldığı ve hiçbirşeyin paylaşılmadığı mimarilerden bahsetmekte fayda vardır. Yukarıdaki sitemlerdeki mimarilerde , işlemciler herşeye (birincil ve ikincil belleklere ve diğer çevre birimlere) erişebilmektedir. Hiçbirşeye erişilemeyen mimaride; her işlemci kendi birincil , ikincil belliğine ve diğer çevre birimlerine sahiptir; diğer işlemcilerle çok hızlı bir veriyolu (bus) vasıtasıyla haberleşirler. Görüldüğü gibi bu mimari dağıtık mimariye bayağı benzemektedir. Fakat işlemciler arasındaki etkileşimin fazla olduğu çoklu işlemcilerle , etkileşimin az olduğu dağıtık yapı arasında farklılıklar vardır. En temel farklılık işletim yapısındadır. Çoklu İşlemcili bir



Şekil 2.1.7 Bilgisayar Ağı Üzerindeki Merkezi Veritabanı

sistem, herbirinde verilen belli görevlerin icra edildiği ve denetim altında bulunan işlemcilerden oluşan, aynı işletim sisteminin kopyaları tarafından idare edilen , aynı tip işlemci ve bellek bileşenlerinden oluşan simetrik ve homojen bir yapıya sahiptir.

Dağıtık bilgisayar sistemlerinde bu geçerli değildir; işletim sistemi ve donanımlarda genellikle bir heterojenlik söz konusudur.

Bununla beraber bir network üzerinde sadece tek bir düğümde veritabanı varsa bu da dağıtık bir sistem değildir. Bu durum merkezi veritabanı yönetim sistemlerinin sahip olduğu yapıya göre hiçbir farklılık göstermemekte, aynı problemleri taşımaktadır. (Şekil 2.1.7.)Yukarıdaki şekilde de görüldüğü gibi veritabanı tek bir bilgisayar sitemi tarafından yönetilmekte ve ağ üzerindeki diğer sistemlerin katılımı sadece isteklerinin sistem 2'ye gönderilmesi şeklinde gerçekleşmektedir. Burada benzeri sistemlerden tek farklılık, diğer sistemlerden merkezi sisteme olan bilgi akışıdır. Buradan da açıkça görüldüğü gibi bir bilgisayar ağının bulunması ve dosyaların koleksiyonu bir dağıtık veritabanı oluşturmak için yeterli olmamaktadır.

Bu noktada bir dağıtık veritabanı uygulaması örneği verilerek, kavram daha da açıklık kazanabilir. Uluslararası bir üretim firması tasavvur edilsin: Şirket merkezi Newyork olsun, Chicago ve Montreal'de üretim yapan fabrikalar olsun, phoenix ve Edmonton'da stok depoları olsun, Avrupa Sorumlusu Merkez Paris olsun ve Araştırma Geliştirme Bölümü San Francisco'da bulunsun. Böyle bir organizasyon çalışanlar, mühendislik işleri, stok seviyeleri, araştırma projeleri, pazarlama işlemleri ve daha fazlasına ait bilgilere ilişkin kayıtları tutmak isteyecektir. Bu durumda veri ve bilgileri aşağıdaki mantık çerçevesinde tutmakta fayda vardır.

1.) Her lokasyon, kendi yerel yapısı içerisinde kendi çalışanlarının kayıtlarını tutsun.

2.) Araştırma ve geliştirme projelerine ilişkin bilgiler, projelerin kullanılacağı yerlerde tutulsun.

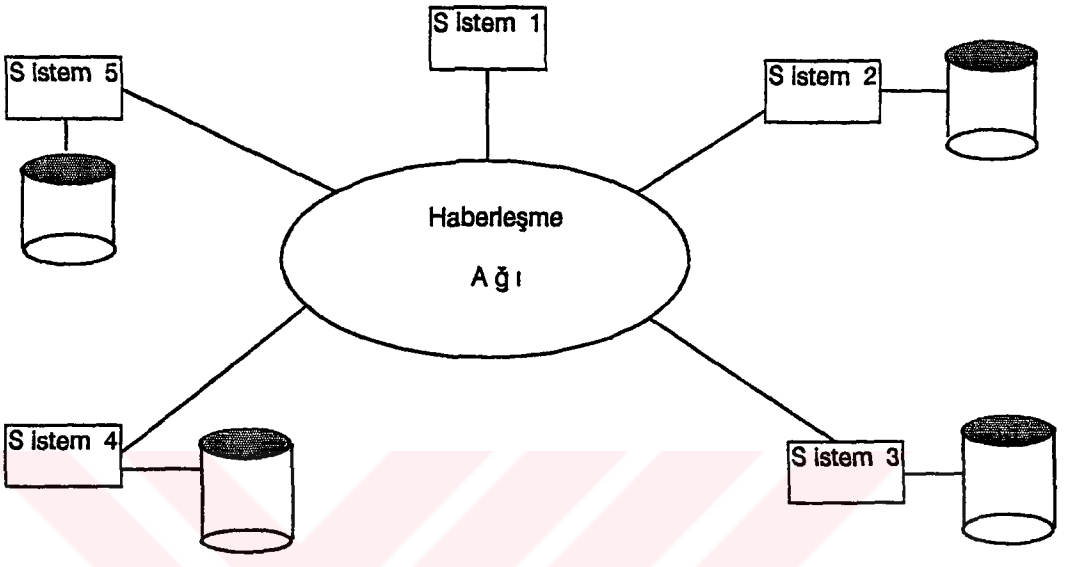
3.) Üretim yerleri (fabrikalar) kendi mühendislik işlerine ait verileri tutsunlar ve bu bilgileri araştırma grubunun anlayacağı şekilde organize etsinler.

4.) Üretim yerleri kendi stok bilgilerini tutsunlar, gerektiğinde stok depolarındaki bilgilere ulaşsınlar.

5.) Stok depoları gerekli kayıtları tutarak kendi stok-kontrol sistemini kursunlar. Üretim yerleri de gerektiğinde stok seviyelerini görmek için buralara erişsinler.

6.) Merkez yerler (Dünya ve Avrupa) bölgelere ait pazarlama ve satış verilerini tutsunlar. Bu veriler diğerleriyle paylaşılabilir olsunlar ve üretim yerleri ve stok depolarındaki bilgilere gerektiğinde ulaşabilsinler.

Bu durumda aşağıdaki şekilde görülen, bütün kayıtların yukarıdaki strateji doğrultusunda bilgisayar ağı üzerinde dağılmış farklı nilgisayarlarda tutulduğu bir dağıtık veritabanı uygulaması gerçekleştirilebilir.



Şekil 2.1.8.Dağıtık Veritabanı Sistemi

2.2. Dağıtık Veritabanı Yönetim Sistemlerinin Merkeziyete Göre Sorun Ve Avantajları

Şu ana kadar dağıtık veritabanları üzerine temel tanımlar ve ilgili kavramlarla ilgili bazı açıklamalardan bahsedilmiştir. Şüphesiz ki dağıtık veritabanlarının merkezi bir veritabanına göre daha sorunlu veya daha avantajlı yanları vardır.

2.2.1. Dağıtıklığın Merkeziyete Göre Sorunlu Yanları

Dağıtık veritabanlarını, merkezi veritabanlarının uygulamala rının dağıtılmış şeklidir diye tanımlamak çok basit olur. Çünkü sistemlerin tasarımı açısından geleneksel merkezi sistemlere göre çok farklı özellikler gösteririler. Geleneksel veritabanınını karakterize eden özellikler merkezi kontrol, veri bağımsızlığı, tekrarın azaltılması, etkin erişim için kompleks fiziksel yapılar, bütünlük, geri alma, eşanlılık kontrolü, gizlilik ve güvenliktir.[2]

Merkezi Kontrol :

Veritabanı yöneticisinin temel görevi veri güvenliğini garanti altına almaktır. Bu görev de tek bir merkezin sorumluluğu altında olmalıdır.

Dağıtık veritabanlarında merkezi kontrol daha az vurgulanmaktadır. Hiyararşik bir kontrol mekanizması söz konusudur. Global bir veritabanı yöneticisi ve yerel sistemlerde yerel veritabanı yöneticileri mevcuttur.

Global veritabanı yöneticisi tüm veritabanından, yerel veritabanı yöneticileri ise kendi yerel veritabanlarından sorumludur. Fakat özellikle vurgulanması gereken yerel veritabanı yöneticilerinin yüksek özerkliğe sahip olmasıdır. Bu nedenle global veritabanı yöneticisi yerel yöneticilerin yaptıklarını tamamıyla izleyemeyebilir. Bu dağıtık veritabanlarının yerel özerklik özelliğnden kaynaklanır.

Veri Bağımsızlığı:

Geleneksel veritabanı anlayışından programlar verinin organizasyonundaki değişiklerden etkilenmezler. Dağıtılmış veritabanlarında da bu "Dağıtımlı effaflık" sayesinde böyledir. Verinin başka sistemlere taşınması programların doğru çalışmasını etkilemez, fakat işletim hızlarını etkiler.

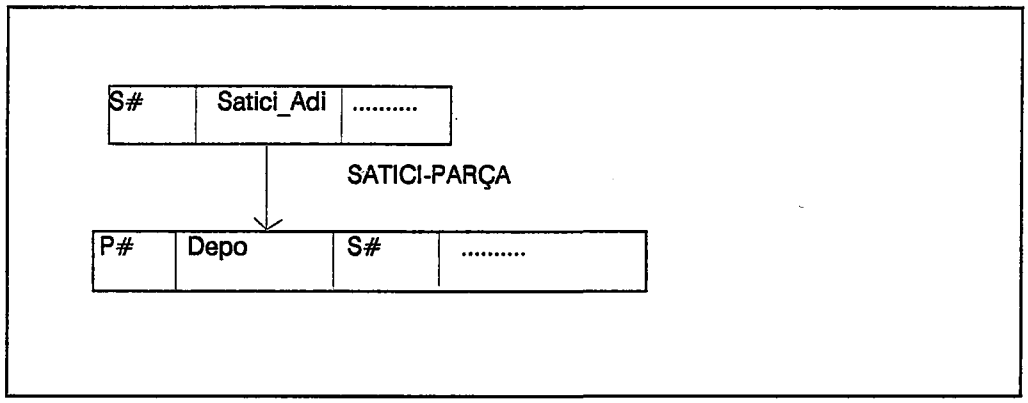
Veri Tekrarının Azaltılması:

Geleneksel veritabanlarında veri tekrarı en az iki sebepten dolayı azaltılma yoluna gidilir.

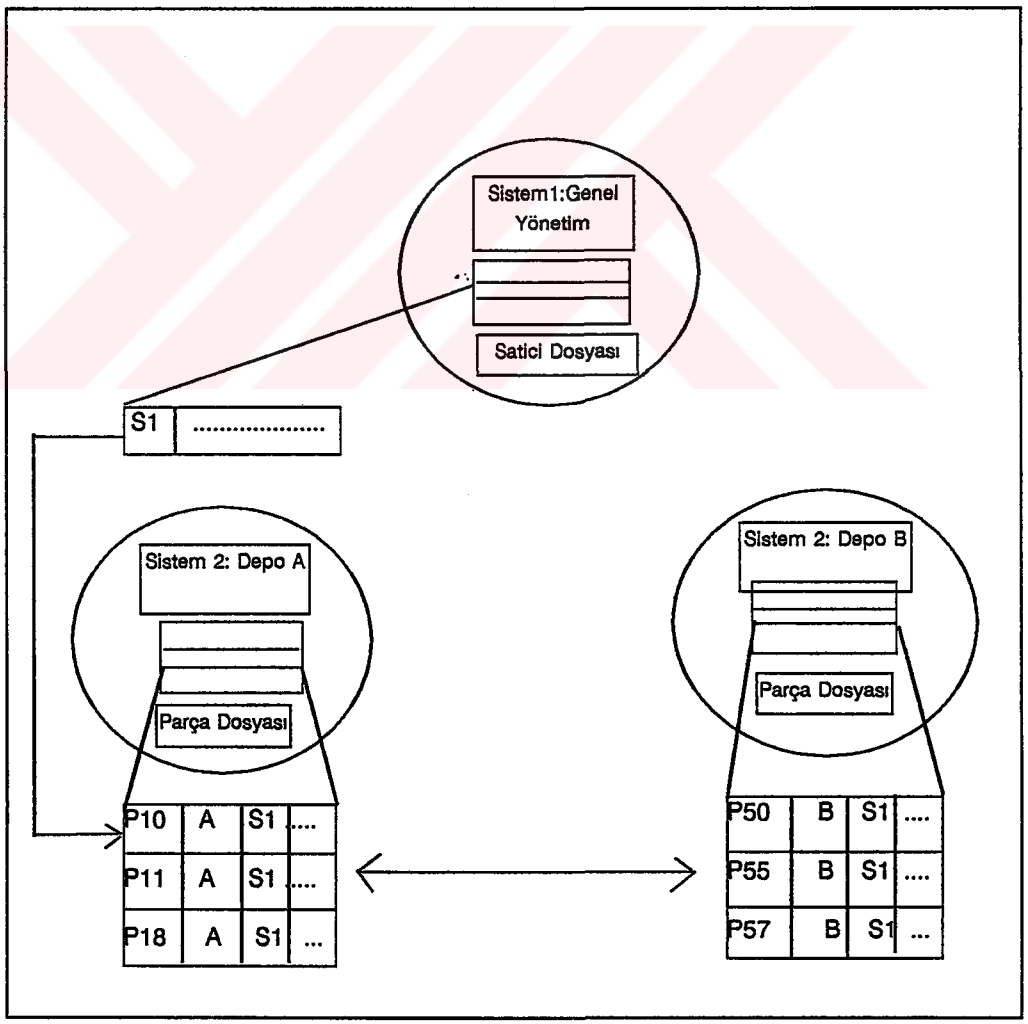
- Aynı mantıksal datanın birden fazla kopyası yüzünden oluşacak veri tutarsızlığından kaçınmak ,

- Veri tekrarı saklanarak bilgi saklama alanının optimize edilmesi için. Veri tekrarının azaltılması veri paylaşımı ile olur. Aynı anda birden fazla uygulamanın aynı tablo ve kayıtlara erişimi sağlanır.

Fakat dağıtık veritabanlarında veri tekrarı arzu edilen bir olaydır. Çünkü birincisi; uygulamaların yerelliği, verilerin kendi sistemlerinde yer almasıyla daha çok artacaktır. Bu nedenle çok kullanılan veriler programın çalıştığı yerel sistemde de saklanacaktır. İkincisi; bu şekilde genel olarak tüm sistemin uygunluğu, her an çalışabilirliği artacaktır. Sistemin herhangi bir yerindeki çökme sırasında aynı veriye sahip başka bir yerel sistem kullanılarak sorun tüm sisteme yansımamış olacaktır. Fakat diğer yandan güncelleme işlemlerinin bütün sistemlere yansıtılması vb. sebeplerden dolayı birtakım extra sıkıntı ve maliyetler gelecektir.



Şekil 2.2.1. Bir Progress Veritabanı Şeması



Şekil 2.2.2. Satıcı-Parça Kümesinin Dağılımı

Karışık (kompleks) Fiziksel Yapılar ve Etkin Erişim :

Kompleks erişim yapıları, ikincil indeksler, ara dosya zincirleri vb. alışılagelmiş veritabanlarının en büyük özelliklerindendir. Bu tür yapılar için destek veritabanlarının en büyük özelliğidir. Kompleks erişim yapıları sağlamaktaki amaç da, veriye etkin erişimi sağlamaktır.

Fakat dağıtık veritabanlarında kompleks erişim yapıları etkin erişim için doğru bir araç değildir. Bu nedenle dağıtık veritabanlarında etkin erişim önemli bir problem olduğundan dolayı, sistemler arası fiziksel yapılar uygun bir teknik değildir.

Bunu Şekil 2.2.1' de ifade edilen bir örnekle verebiliriz.

S1 satıcısının sattığı parçaları bulmak isteyelim. Bunu alışılagelmiş bir veritabanı içerisinde;

```
Find satici where satici.s# = "S1".
repeat:
    find next satici-parca where satici-parca.s# = satici.s#.
     find parca of satici-parca.
    d isplay parca.
 end.
```

İki kayıt tipi var. Bunlar satıcı ve parça. Satici-parca satıcı kaydını parca kaydına bağlayan bir küme tipidir. Böyle bir erişim yöntemini yukarıdaki veritabanı üzerinde uygulandığ düşünülsün:

Eğer yukarıdaki erişim şekli bu sistem üzerinde uygulanırsa satıcı-parça kümesinde uygun bir kayıt bulunmayana kadar her iterasyon için Sistem2 ve Sistem3'e, Sistem1 tarafından birer erişim yapılacak ve her defasında takrarlanacağından bu büyük ölçüde programın erişim hızını yavaşlatacaktır. Aynı zamanda her iterasyon esnasında ,yani Sistem2 ve Sistem3'deki her kayıda erişimde ağ üzerinde bir takım mesajlar gidip gelecektir. Dolayısıyla bu extra bir maliyet ve vakit kaybı getirecektir.

Daha etkin bir uygulama , dağıtık bir erişim planı uygulanarak şu şekilde olabilir:

1) Sistem1'de:
Sistem2 ve Sistem3'e satıcı numarası gönder. (SN)
2) Sistem2 ve 3'de paralel olarak:

Alınan satıcı no'suna göre aşağıdaki programı çalıştır;
For each parca where parca.s# = SN:
export parca.
end.
ve sonucu Sistem1'e gönder.
3) Sistem1'de:
Sistem2 ve Sistem3'den gelen sonuçları birleştir. Neticenin çıktısını al.

Böyle bir plan programcı ya da bir optimizer tarafından gerçeklenmelidir. Görüldüğü gibi prosesler farklı sistemlere dağıtılarak ve gerekirse her yerel sistem içerisinde kompleks erişim yöntemleri kullanarak süre açısından daha iyi bir sonuç alınabilir.

Yukarıdaki gibi otomatik bir erişim planını tasarlayacak optimizere iki problem çözülmek üzere kalmaktadır. Bunlar global optimizasyon ve yerel optimizasyondur.

Global Optimizasyon: Hangi veriye hangi sistemler tarafından erişilecek ve hangi tablolar netice olarak sistemler arasında gönderilecektir. Global optimizasyon için ana optimizasyon parametresi haberleşme maliyetidir.

Yerel Optimizasyon: Her sistemdeki yerel veritabanlarına erişimin hangi şekilde olacağıdır. Burada dağıtık olmayan veritabanlarında kullanılan yöntemler uygulanır.

Bütünlük , Geri Alma ve Eş Anlılık Kontrolü :

Veritabanlarında bütünlüğü sağlamadaki en önemli etken hareketlerin atomik olmasıdır. Atomik hareketten kasıt meydana gelen bütün olayların (değişikliklerin) karşı pozisyona aynen iletilmesi ya da hiçbirşeyin iletilmeden bırakılmasıdır.

Atomik hareketin iki tehlikeli düşmanı vardır: i) Sistemdeki çökme ii) Eş anlılık

Hareketin ortasında gerçekleşen çökmeler atomik hareketi kaçınılmaz kılmaktadır. Farklı hareketlerin eş anlı çalışmaları hareketlerden birinin beklemesine ya da tamamıyla geri alınmasına gereksinim duyabilir. İşte bu tür durumlarda belli bir sistemdeki çöküntüden dolayı ya diğer sistemler de gerçekten hareketler geri alınacak ya da birtakım metadolojiler belirlenecek daha sonra diğer sistemlere de yansıtılmak üzere ana bir sistemde hareket gerçeklenecektir. En uygun politikayı programcı seçecektir.

Gizlilik ve Güvenlik :

Alışılagelmiş veritabanlarında veritabanı yöneticisi merkezi bir kontrole sahiptir ve yalnızca yetki verilen erişimlerin veriyi etkileyeceğinden emindir. Bununla beraber eğer bu otoriteyi sağlayacak kontrol programları yazılmadıysa sonuç özel tablo kullanımı metadolojisinden daha kötü (zor) olabilir.

Dağıtık veritabanlarında yerel yöneticiler hemen hemen merkezi veritabanı yöneticilerinin karşı karşıya olduğu problemlerle karşı karşıyadırlar. Fakat iki özel durumdan söz edilebilir: Birincisi; Yerel özerkliğin de bir sonucu olarak yerel veri sahipleri veriyi daha fazla muhafaza edebilirler, merkezi bir veritabanı yöneticisine bağlı olmak yerine kendi muhafazalarını kendileri tatbik ederler. İkincisi; güvenlik problemleri gerçekten dağıtık veritabanlarında var olan bir sorundur. Bunda da heberleşme ağının korumaya karşı cevap vermedeki zayıflığı sebep olmaktadır.

2.2 .2. Niçin Dağıtık Veritabanları

Dağıtık veritabanlarının belirtilen sorunlara sahip olması açısından bilgi sistemlerinin daha az maliyet ve daha iyi bir performansla çalışmasında çok önemli katkıları olmuştur. Dağıtıklığın kullanılir omasıyla beraber bilgisayarlara dayalı bilgi sistemleri, özellikle coğrafi bir yapı üzerinde dağılmış organizasyonel yapılar için çok büyük avantajlar getirmektedir. Bilgi sistemlerine getirdiği avantajlar, belki de son yıllarda bilgisayar dünyasında toplumsal alandaki en büyük gelişme olmuştur.

Getirdiği avantajlar doğrultusunda bilgisayar dünyası tarafından sürekli destek görmekte ve bu tür yapılara destek veren yazılımlar ihtiyaçları karşılamak için her geçen gün gelişme göstermektedirler. Dağıtık veritabanlarının neden bu kadar gelişmekte olduğunu gösteren sebeplerden önemlileri şunlardır.

i) Organizasyonel ve Ekonomik Sebepler:

Bir çok organizasyon merkeziyetçilikten uzaklaşmaktadır. Dağıtık veritabanı yaklaşımı bu tür organizasyonlar için en uygun yapı teşkil etmektedir. Bilgisayar teknolojisindeki son gelişmelerle birlikte büyük, merkezi bilgisayar sistemlerine sahip olmanın getirdiği ekonomik maliyetler göz önüne alındığında bu tür sistemlerle devam etmek birtakım soru işaretleri doğurmuştur. Dolayısyla yeni yatırımlara dağıtıklığa cevap veren sistemler üzerinde olmuştur.

ii) Mevcut Veritabanlarının Biribirine Bağlı Olması:

Hali hazırda birkaç veritabanının olduğu bir organizasyonda , global bir uygulama geliştirme ihtiyacı doğduğunda; dağıtık veritabanları doğal bir çözüm olarak ortaya çıkmaktadır. Bütün veritabanlarının biribrleriyele mantıksal olarak

bağlantılı çalışabilmesi için yerel veritabanlarından birkaç yapısal değişliklik yeterli olmaktadır.

iii) Gelişme Artışı:

Organizasyon, yeni özerk sistemlerin eklenmesiyle büyümeye başladığında (yeni sistemlerin eklenmesi) dağıtık veritabanı minimum derecede etki ile bu büyümeyi destekler, oysa merkezi veritabanında bu tür büyümeyi gerçekleştirmek için sistemin başlangıç boyutlarının değiştirilmesi ve mevcut yapının büyük ölçüde etkilenmesi olasıdır.

iv) Daha az Haberleşme Maliyeti:

Özellikle coğrafi olarak yayılmış sistemlerde merkezi bir yapı düşünüldüğünde , yapılan her işlemin merkez üzerinden çalıştığı düşünülürse, bunun getireceği haberleşme maliyetinin ne kadar çok olacağı rahatlıkla görülebilir. Oysa dağıtık veritabanlarında yerel işlemler yalnız yerel makineyi meşgul ettiğinden ve sadece global işlemler için haberleşme kullanılacağından ağ trafiği daha az olacak ve maliyet de düşecektir.

v) Performans Düşüncesi:

Nasıl çok işlemcili bir sistemde uygulamalar işlemcilere paralel olarak dağıtıldığında performansta artış gözüküyorsa, Dağıtık veritabanı yönetim sistemlerinde de performans artacaktır. Merkezi bir yapıda, merkez üzerine aşırı yük binmesi bir darboğaz oluşturabilir ve bu sistem perfornasının düşürür. Oysa işlem yükünün farklı sistemlere yayılması bu tür darboğazlardan uzaklaştırır.

vi) Güvenlik ve Uygunluk :

Dağıtık veritabanları veri doygunluğ nedeniyle fazlasıyla güvenilirlik ve uygunluk gösterirler. Dağıtık yapıda, çok fazla bileşen olduğu için merkezi yapıya göre daha fazla çökme gözükür, fakat bu çökmelerin bütün sistemi etkilemesi çok nadir olduğu için sistem daha güvenilir bir şekilde çalışabilir. Merkezi yapıda , merkez sistemdeki sorun yüzünden tüm sistem çöker ve çalışmalar sürdürülemez.

vii) Yerel Özerklik:

Veri dağıldığından dolayı, genelde belli bir veriyi paylaşan kullanıcı grupları bu veriyi kendi yerel yapılarına taşıyabilmekte ve bu sayede yerel bir kontrol mekanizmasına sahip olmaktadırlar. Bu veri kullanımı hakkında yerel düzenleme ve zorunluluklar getirilmesine izin vermektedir. Çoğu zaman bilgi sistemlerinin yönetim ve sorumluluklarıyla ilgili idarenin bölünmesiyle ulaşılan başarının ardında dağıtık bilgi sistemleri yatmaktadır. Bu belki de son yıllarda bilgisayar kullanımımda şahit olduğumuz en önemli toplumsal gelişmedir.

Tabii ki dağıtık bir yapı organizasyonel olarak merkeziyete dayalı bilgi sistemleri için bu kadar çok uygunluk göstermeyebilir. Ancak bilgi sistemi merkeziyetten uzak bir yapı için gerçekten büyük avantajlar getirmektedir.

viii) Paylaşılabilirlik:

Coğrafi bir yapı üzerinde dağıtılmış işlemlere sahip organizasyonlar, verilerini dağıtılmış olarak saklayabilmektedir. Fakat eğer bilgi sistemi dağıtık değilse, bu veri ve kaynakları paylaşmak imkansızdır. Bu nedenle dağıtık veritabanı sistemi verilerin gerektiğinde paylaşılması ile bilgi sistemlerinin daha sağlıklı işlemesini sağlamaktadır.

2.3. Dağıtık Veritabanı Yönetim Sistemi Olma Kuralları

2.3.1. Yerel Özerklik

Bilgisayar Ağı Üzerindeki farklı yerlerdeki sistemler birbirlerinden bağımsız olmalıdırlar. Belli bir taraf (Makine-veritabanı) üzerindeki işlemler o yer tarafından kontrol edilir. Yerel özerkliğin sağlanması aşağıdaki önermelerin geçerli olması anlamına gelir.[3]

- Yerel verilerin iyeliği ve yönetimi yereldir, sorumluluğu yerel sistemdedir. Uzak sistemler tarafından erişilebilir olsalar bile, tüm veriler bir yerel veritabanına aittir. Verilerin güvenliği, bütünlüğü ve saklama hakları, yerel sistemin denetiminde kalır.

- Yerel işlemler saf haliyle yerel kalır. Yalnız yerel verilerle ilişkili işlemler gerçekleştiren kullanıcılar, sistemlerinin dağıtımlı bir sistemdeki katılımından dolayı zarar görmemelidir. Özellikle de,bir sistem dağıtımlı bir sistemin parçası haline gelmeden önce başarılı şekilde o sistemde çalışan uygulamalar, sonrasında da aynı katkıyı sürdürmelidir (verilerin bir kısmı başka sistemlere hareket ediyor olsalar bile) .

- Belirli bir sistemdeki tüm işlemler o sistem tarafından denetlenir: Sistemlerden hiçbiri başarılı şekilde işleyebilmesi için başka birine bağımlı değildir. Tersi

durumda, kandisinde hiçbir hata olmadığı halde herhangi başka bir sistem bakımda olduğu için kullanılamayabilir.

Gerçekte özerklik yukarıdaki önermeler doğrultusunda değerlendirildiğinde tam olarak sağlanamaz. Dağıtık yapı içerisindeki herhangi bir sistemin denetiminin, belli ölçülerde başka bir sistem tarafından sağlandığı durumlar muhakkak olmaktadır. Aşağıdaki madde özerkliğin hedefinin daha iyi anlaşılmasında yardımcı olmaktadır.

2.3.2. Merkezi Bir Sisteme Bağlılık Yoktur.

Yerel Özerkliğin de bir sonucu olarak bütün taraflar, tüm sistemin eşit haklara sahip birer katılımcısıdırlar. Tüm sistem bazında bütün taraflar aynıymış gibidir. Dolayısıyla belli bir taraf diğerlerine göre ayrıcalıklı olamaz, dolayısıyla da tek bir merkez sisteme de bağımlılık söz konusu değildir. Bu kural aslında kısmen birinci kuralın devamıdır. Ancak "bir merkez sistemin dayanak alınmayışı" kendi başına da istenir birşeydir, hem de tam yerel özerklik hedefine ulaşılamasa bile. Bu kuralın gerekliliğini gösteren iki önemli dayanak mevcuttur. Birincisi, eğer merkezi bir sistem varsa bütün yük bu sistem üzerine binecek ve bu da bir darboğaz (bottleneck) yaratacaktır. İkincisi ve daha önemlisi bu merkezi sistemde meydana gelecek bir kapanmadan dolayı diğer sistemler de çalışmalarına merkez sistem çalışır hale gelene kadar ara verecektir. Bu nedenle dağıtımlı bir sistemde işlemler: özellikle, sözlük yönetimi, sorgu işleme, eşzamanlılık denetimi ve kurtarma denetimi gibi.

2.3.3. Sürekli İşletim

Sistem hiçbir zaman planlı bir kapanma ihtiyacı duymamalıdır. Örneğin dağıtık sisteme yeni bir sistem eklemek veya bir versiyon değişikliği yapmak söz konusu ise bu bütün sistemin işleyişini aksatmamalıdır. Ancak istisna olarak dağıtımlı ortamda bir tür kapama ve dolayısıyla da hizmet kesintisinin kaçınılmaz olduğu bazı durumlar vardır. Bu nedenle hedef bu tür kesintileri en alt düzeyde tutmaktır.

2.3.4. Yer Bağımsızlığı

Kullanıcılar verilerin fiziksel anlamda nerede saklandıklarını bilmeye gerek duymaksızın, sanki veriler kendi yerel sistemlerinde saklanıyormuş gibi davranabilmelidirler.

Yer bağımsızlığı, kullanıcı programlarını ve uç birim etkinliklerini basitleştirdiği için istenir bir özelliktir. Özellikle de, verilerin bir sistemden diğerine, o program ya da etkinlikleri gerçekleştirmeksizin göç edebilmesine olanak verir. Böylesi bir göç

yeteneği ise, verilerin değişen başarı gereksinimlerine yanıt verecek şekilde ağ içinde yer değiştirmesine olanak verir.

Örneğin Ankara'da belli bir şirkete verilen hizmetler, İstanbul tarafından sorgulanmak istendiğinde, kullanıcı bu sorgulamayı sanki veriler kendi yerel sistemindeymiş gibi gerçekleştirebilmelidir. Aksi halde bu sorgulama işlemini gerçekleştirmek için programcı tarafından bir çok arayüzlerin yazılması gerekecektir: Ankara'daki makinadan istenen verileri işletim sistemi ortamına atacak , bu verileri İstanbul makinesine gönderecek ,sonra da İstanbul Makinesi üzerinden çalışan bir prosesle bu ham veriyi o yerel veritabanına işleyecek bir arayüz gibi...Oysa dağıtık bir veri tabanı yönetim sistemine sahip olunduğunda basit bir Client-Server mantığıyla İstanbul sisteminin isteği, direkt Ankara veritabanından, hiçbir extra program yazmaksızın, sanki İstanbul veritabanı üzerinde işlem yapıyormuşcasına yazılan programlarla sağlanabilir. Veri sadece Ankara veritabanında kalacak, fakat kullanıcı sanki kendi sistemindeymiş gibi bu veriye erişebilecektir.

2.3.5. Dilimlenme Bağımsızlığı

Belli bir ilişki (relation) fiziksel depolama amacıyla parçalara (dilimlere) ayrılabilmelidir. Bu dilimler bağımsız olmalıdır. Kullanıcılar sanki veri hiç paraçalanmamış gibi davranabilmeli ve bu şekilde veri en çok kullanıldığı yerde saklanacak şekilde dilimlere ayrılabilmelidir. Dilimleme başarım nedenlerinden ötürü istenir. Verilerin en çok kullanıldıkları yerde saklanabilmelerinden dolayı, işlemlerin çoğu yereldir ve ağ trafiği azalır. Örneğin belli bir havayolu şirketine Ankara'da verilen hizmetler Ankara veritabanında, Antalya'da verilen hizmetler ile ilgili bilgiler Antalya yerel veritabanında yer alacak şakilde dilimlenebilmelidir.

Sınırlama ve kesit alma ilişkisel işlemlere karşılık gelen iki temel dilimleme türü vardır: Yatay ve dikey. Daha genel bir ifadeyle bir dilim, sınırlama ve kesit alma işlemleri yoluyla özgün ilişkiden türetilebilecek herhangi keyfi bir alt ilişki olabilir. Dilimlerden özgün ilişkinin yaniden oluşturulması, uygun kaynaştırma (join) ve birleştirme (union) işlemleriyle gerçekleşir. Dilimleme ve yeniden oluşturma kolaylığının, dağıtımlı sistemlerin ilişkisel olmasını gerektiren çok sayıdaki nedenlerden ikisi olduğu akılda bulundurulmalıdır: İlişkisel model tam tamına, bu görevler için gerekli biçimdeki işlemleri sağlar.

şu anda ana noktadan bahsedilirse: Veri dilimlemeyi destekleyen bir sistem, aynı zamanda dilimleme bağımsızlığı da sağlar. Yani kullanıcılar en azından

mantıksal olarak, sanki veriler hiçbir şekilde dilimlenmemiş gibi davranabilmelidirler.

Tıpkı yer bağımsızlığı gibi, dilimleme bağımsızlığı da kullanıcı programlarını ve uç birim etkinliklerini basitleştirdiği için istenir bir özelliktir. Özellikle de değişen başarım gereksinimlerine göre, istendiği zaman, sözkonusu kullanıcı programlarını ya da etkinliklerini geçersizleştirmeksizin verilerin yeniden dilimlenmesine ve dilimlerin yeniden dağıtılmasına olanak verir.

2.3.6. Kopyalama Bağımsızlığı

Belli bir ilişki ya da bir ilişkinin belli bir dilimi tüm sistem üzerinde fiziksel olarak farklı yerlerde birbirinden bağımsız kopyalar halinde saklanabilmelidir. Kopyalama en az iki nedenden ötürü istenir:

-Başarım: Uygulamalar uzak sistemlerle iletişime geçmek zorunda olmaksızın yerel kopyalarla işlenebilir.

-Kullanabilirlik: Kopyalanmış bir nesne, en az bir kopya kullanabilir kaldıkça işleme için kullanılabilir olmaya devam eder.

Kopyalamanın başlıca dezavantajı, kopyalanmış bir nesne (sözgelişi kopyalanmış bir ilişki içindeki bir kayıt) güncellendiğinde, o nesnenin tüm kopyalarının güncellenmesinin gerekmesidir (güncelllemeyi yaygınlaştırma sorunu). Tıpkı dilimleme gibi, kopyalama da kullanıcı için şeffaf olmalıdır. Başka bir deyişle, veri kopyalamasını destekleyen bir sistem, kopyalama bağımsızlığını da desteklemelidir. Kopya bağımsızlığı, yer ve dilim bağımsızlığı gibi, kullanıcı programlarını ve uç birim etkinliklerini basitleştirdiği için istenir bir özelliktir. Özellikle çok uzak sistemler söz konusu ise ve ağ içindeki veri iletişim hızı OLTP için çok yavaşsa kopyalama kaçınılmazdır. Kopya bağımsızlığının önemli bir yararı, kopyaların değişen gereksinimlere yanıt vermek üzere ve söz konusu program ya da etkinliklerden hiçbirini geçersizleştirmeksizin yaratılabilmesine ve yok edilmesine olanak verir.

2.3.7. Dağıtık Sorgu İşleme

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

İyi bir strateji seçmek temel önem taşır. "Optimizer"ın ilşkisel sistemlerde bu kadar kritik bir bileşen olmasının nedeni de budur. Optimizer strateji seçiminden sorumludur. Bu tür bir sistemdeki başarım bu nedenle temel düzeyde sistem optimizasyon kalitesine bağlıdır.

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

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

Bu nedenle optimizasyon için başlıca hedef bu etkeni en aza indirmektir. "An Introductuction to Database Systems" (C.J. Date, II.Cilt, Addision-Wesley Publishing Co., Reading, Massachusets, ABD, 1983) adlı kitapta bu konuda çarpıcı bir örnek yer almaktadır: Bir sorgu örneğinin yürütülmesine ilişkin altı değişik strateji çözümlenmiştir ve gerçekleşen yanıt süresi bir saniye ile 2 1/3 gün arasında oynamıştır. Bu farkı proje için çalıştığım sistem üzerinde de gözlemleyebildim. Örneğin İstanbul dışındaki herhangi uzak bir sistemden (Ankara, Antalya vs.) sekiz bin civarında dört tablodan bilgi alarak yapılan bir update işlemi direk yerel sistemdeki bir sorgulama mantığıyla çalıştırıldığında her iterasyonda her iki sistemdeki farklı tablolardan bilgiler alındığından ve bu esnada ağ üzerinde gidip gelen mesajlar yüzünden ve özellikle ağ iletişim hızının düşüklüğü de göz önüne alınırsa yaklaşık üç-dört saat civarında sürmektedir. Oysa işlem uygun bir şekilde sistemlere yayıldığında bağlanma süreleri ile birlikte birkaç dakikada sona ermektedir.

Bu nedenle dağıtık bir sistemde optimizasyonun genel bir bakış açısıyla gerçekleştirilmesinin önemi büyüktür. Dahası optimizasyon sürecinin kendisinin de dağıtılması önemlidir; hem iletişim trafiğini en az tutacak genel bir optimizasyon adımı içermeli, hem de her sistemdeki yerel optimizasyon adımlarını da birbirinden ayırmalıdır. Örneğin K kuyruğunun X sistemine sunulduğunu ve K'nın başka şeylerin yanısıra, Y sistemindeki 100 kayda ilişkin İy ilişkisinin kaynaştırmasını içerdiğini varsayalsın. İy ilişkisi de Z sistemindeki bir milyon kayda ilişkin İz ilişkisini içersin. X sistemindeki optimizer K'nın yürütülmesi için genel stratejiyi seçecektir:

Burada İz'nin Y'ye taşınmasını değil de (hele İy ve İz'nin her ikisinin X'e taşınması hiç değil) İy'nin Z'ye taşınmasına karar verilmesinin taşıdığı önem açıkça görülüyor.

Optimizer İy'nin Z'ye taşınmasına karar verdikten sonra, Z sistemindeki gerçek kaynaştırma işlemine ilişkin strateji, Z'deki yerel optimizer tarafından kararlaştırılacaktır. Belki de Z'de, X'teki optimizer K'ya ilişkin genel stratejiyi seçtiği sırada onun tarafından bilinmeyen bazı yollara erişim olanağı olacaktır.

2.3.8. Dağıtık Hareket (Transaction) Yönetimi

Sistem üzerindeki hareketler atomik olarak geçekleşmelidir. Yani transactionlar'da "ya hep ya hiç" mantığı geçerli olmalıdır. Hareket yönetiminin, dağıtımlı ortamda geniş olarak ele alınması gereken başlıca iki özelliği vardır: Kurtarma denetimi ve eşzamanlılık denetimi.

Atomik deyimi bir hareketin bir "ya hep ya hiç" önermesi olduğu anlamındadır; bunun özgül anlamı, hareket mantığı veritabanında birden fazla güncelleme gerektiriyorsa, bu güncellemelerin ya hepsi yapılır ya da hiçbiri yapılmaz. Dolayısıyla, bir hareket bazı güncellemeler gerçekleştirdiyse ve daha sonraki bir işlem herhangi bir nedenle başarısız bittiyse, bu güncellemelerin iptal edilmesi ya da geri alınması gerekir.

Öte yandan hareket başarılı bir şekilde sonuçlanırsa, güncellemeler geri alınmak yerine kalıcılaştırmalı ya da "taahhüt edilmeli"dir. Taahhüt edilmemiş güncellemelerin geri alınması, veritabanının önceki ve tercih edilir, doğru bir durumunun kurtarılması işlemini oluşturur; kurtarma denetimi deyimi de bundan dolayıdır.

Dağıtılan bir sistemde tek bir hareket, birden çok sistemde güncelleme içerebilir. Bu nedenle, belirli bir hareketin dağıtımlı ortamda atomik kalmasını güvence altına almak için, sistemin ilgili sistemler grubunun ya hepsinin birden taahhüt etmesini ya da hepsinin birden hareketi geri almasını sağlaması gerekir. Bu etki, iki aşamalı taahhüt (two phase commit) adı verilen bir protokol ya da onun bir türevi yoluyla elde edilebilir.

İki aşamalı taahhütten kısaca bahsedilmek istenirse: Belirli bir hareket için sistemlerden biri (tipik olarak, hareketin başlangıçta sunulduğu yer) bu protokolde koordinatör olarak iş görmeli ve diğerleri de katılımcı olarak davranmalıdır. Değişik hareketler değişik sistemlerde sunulacağından, dağıtımlı hareket yönetiminde, her sistemin bazı hareketler için koordinatör olarak hizmet veririken, başka hareketlere

katılımcı olabilmesi gereklidir. 2. kurala uygun olarak, hiçbir merkez roldeki sistem, hareketlerin tümü için koordinatör işlevi görmemelidir.

Eş zamanlılık denetimine gelinecek olunursa; dağıtımlı bir sistem içinde bu türden bir denetim elbette dağıtımlı olmayan sistemlerde de olduğu gibi kilitlemeye dayalı olacaktır. Araştırma dünyasında eşzamanlılık denetimi konusunda başka yaklaşımlar da araştırılmıştır, ancak henüz kilitleme tekniği en uygun seçim olarak görülmektedir.

Yerel özerklik ilkesine göre her sistem kendi yerel verilerinin kilitleme yönetiminden sorumlu olmalıdır. Bu demektir ki, her sistem kendi kilit yöneticisini içerecektir ve bunun bir sonucu olarak, genel bir sistem kilitlenmesi (deadlock) olanağı vardır. Genel sistem kilitlenmesi, iki ya da daha fazla sistemin aynı anda bekleme durumunda olması, dolayısıyla her birinin işine devam etmeden önce diğerlerinin bir kilidi kaldırmasını beklemesi durumudur.

Genel sistem kilitlenmesine ilişkin ana sorun, hiçbir sistemin bu durumu yalnız o sisteme içsel olan bilgileri kullanarak algılayamayışıdır. Bu nedenle sistemler, diğer sistemlere bazı içsel sistem bilgilerini göndermek durumunda. Dahası, tüm sistemlerin bu türden bilgileri gönderdiği merkezi "kilitlenme algılayıcı" sistem bulunmamalıdır (Kural 2 den dolayı). Bunun yerine sistem, bir tür dağıtımlı kilitlenme algılama mekanizması sağlamalıdır.

Genel kilitlenme algılamanın pahalı olduğu (içerdiği ek ileti trafiği nedeniyle) gözönüne alındığında, herşeyden önce kilitlenmelerin önüne geçmek için bir tür zaman dışı (time- out) mekanizması kullanmak tercih edilebilir.

2.3.9. Donanım Bağımsızlığı

Aynı VTYS farklı donanım sistemlerinde çalışabilmeli ve bu farklı donanım sistemleri dağıtık sisteme eş birer ortak olarak katılabilmelidir. Buradaki amaç farklı tür-marka makinelerden (Unisys,IBM,HP vs.) oluşan sistemlerin herbirindeki verileri bütünleştirmeye ve kullanıcıya tek bir sistem izlenimi sunmaktır.

2.3.10. İşletim Sistemi Bağımsızlığı

Aynı VTYS farklı işletim sistemlerinde çalışabilmelidir. Bu hedef kısmen bir öncekinin devamıdır. Aynı VTYS'yi sadece değişik donanım sistemlerinde değil, aynı zamanda değişik işletim sistemlerinde de ve hatta aynı donanımdaki değişik işletim sistemlerinde (Unix,PC-DOS,VAX/VMS vs.) calıştırabilmelidir.

2.3.11. Ağ Bağımsızlığı

VTYS, farklı sistemleri birbirine bağlayan haberleşme protokollerinin (X.25,TCP/IP gibi) ne olduğuna bakmaksızın ödünsüz olarak çalışabilmelidir.

İdeal durumda sistem hem Ethernet gibi yerel alan ağlarını (LAN), hem de telefon hatları gibi uzun bağlantılı hatları desteklemelidir.

2.3.12. VTYS Bağımsızlığı

Farklı sistemlerdeki VTYS'ler aynı satıcıdan gelsin ya da gelmesin haberleşebilmelidir. Başka bir deyişle farklı VTYS'lerin hepsinin de dağıtımlı siteme bağlanabilmesi (gateway) özelliğidir. Örneğin Dağıtık Sistem altı yerel sistemden oluşuyor ve tüm sistemde PROGRESS VTYS kullanılıyor, bu sistemlerden bir veya (yeni eklenmiş bir sistemde de olabilecek şekilde) birkaçında ORACLE veya INFORMIX veri tabanları mevcut ise PROGRESS , ORACLE veya INFORMIX veri tabanları üzerinde işlem yapabilmek (sorgulama,update vs.) için gerekli gateway'e sahip olmalıdır. Burada gerekli olan şey, değişik sistemlerdeki VTYS'lerin hepsinin aynı arabirimi desteklemesidir.

2. 4. Dağıtık Veritabanlarının Problemleri:

Dağıtık veritabanı kullanırken en önemli amaç gidip gelen mesajların sayısının ve hacminin minimize edilmesidir. Bu doğrultuda temel olarak aşağıda bahsedilen konularda sorunlarla karşılaşılabilir.[3]

1) Sorgu İşleme

2) Katalog Yönetimi

3) Güncelleme Üremesi

4) Güvenlik Kontrolü

5) Eş Anlılık Kontrolü

2.4.1. Sorgu İşleme :

Ağ trafiğini azaltmak demek, sorgu optimizasyon prosesinin kendisinin, sorgu işletim prosesinde olduğu gibi dağıtık olmaya ihtiyaç duymasıdır. Başka bir deyişle proses öyle bir global optimizasyon adımına sahip olmalıdır ki, bu adım kendi tarafını etkileyen yerel optimizasyon adımları tarafından takip edilmelidir. Bunu şöyle bir örnekle açıklanabilir:

Üç yerel sistem var olsun X,Y,Z. Q sorgulama prosesi X yerel sistemine ait olsun (X sistemince yaratılsın). ve bu prosesi Y sistemindeki İy ilişkisindeki yüzlerce kayıtla, Z sistemindeki İz ilişkisinden milyonlarca kayıtın birleşimi alınmak istensin; X sistemindeki optimizer 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 optimizer tarafından karar verilmelidir.

2.4.2. Katalog İdaresi:

Dağıtık bir sistemde, sistem katalogu genelde olduğu gibi yalnızca ilişkiler, indexler,kullanıcılar vs. gibi veriler dışında ayrıca sistemi tasarlanan yerde saklamak için gerekli kontrol, dilimleme ve tekrar bağımsızlığı bilgisini de içerir. Bunu gerçeklemek için şu sorun ortaya çıkar. En uygun katalogun kendisi nerede ve nasıl saklanmalıdır? Genelde aşağıdaki İhtimaller söz konusudur.

a) Merkezi: Bütün katalog tek merkezi bir sistemde saklanır.

b) Tamamıyla Kopyalanmış: Bütün katalogun tamamı her sistemde yer alır.

c) Parçalanmış: Her sistem kendisiyle ilgili kataloga sahiptir. Bütün katalog bu ayrık sistemlerin bileşiminden oluşmaktadır.

d) a ve c'nin kombinasyonu: Her sistem kendi yerel kataloguna sahiptir ve tek bir sistem bütün bu yerel katalogların kopyasına sahiptir.

Bu dört yaklaşım incelendiğinde: a yaklaşımı tek bir merkeze bağımlılık yoktur kuralına tamamıyla aykırıdır. b yaklaşımı özerklik kuralından biraz ödün vermektedir. Her katalog güncellenmesi her sistemde güncelleme anlamına gelmektedir. c yaklaşımı yerel olmayan operasyonları bayağı maliyetli kılmaktadır. Erişmek için uygun nesneyi bulmak sistemlerin yarısından fazlasını dolaşmaya mal olabilir. d yaklaşımı daha etkin bir yaklaşımdır. Çünkü dış bir sistemden bilgi almak tek bir erişimde yeterli olacaktır. Fakat bu yaklaşım "merkezi bir sisteme bağımlılık olmamalıdır" kuralıyla çelişmektedir.

Görüldüğü gibi olası yaklaşımlar tatminkar değildir. öyle bir strateji izlemek en uygun olacaktır. Katalog yapısı şu şekilde olmalıdır: Objenin (tablo) bir yerel ismi bir tüm sistem için global ismi olmalıdır. Bunlar bir eşanlam tablosunda tutulmalıdır.

Bu yapı sağlandıktan sonra, eğer sorgulanmak istenen statu adlı ve mstatu bunun eşanlamlı tablosu ise;
select * from statu veya
select * from mstatu aynı işlevi görecektir.
Yerel isim basit olarak "statu" şeklinde olabilir. Sistem ismi dört kısımdan oluşacaktır.

i) Yaratıcı ismi - Nesneyi yaratan kimse
ii) Yaratıcı Sistem - Yaratma işlemini başlatan sistem
iii) Yerel İsim - Nesnenin tam ismi
iv) Doğduğu Sistem - İlk yüklendiği sistem .

Örneğin bir sistem ismi: Bulent @ Istanbul . Statu @ Ankara . Yerel ismi "statu" olan nesne , İstanbul sisteminde Bulent isminde kullanıcı tarafından yaratılmış ve ilk olarak Ankara sisteminde saklanmıştır. (Ankara ismi hiçbir zaman değişmeyecektir.) Bu söyleneler SQL ile şu şekilde ifade edilebilir:
Create synonym mstatu for Bulent @ Istanbul. statu @ Ankara. Kullanıcı şimdi isterse;
"select * from statu" veya "select * from mstatu" program parçacıkları ile aynı sorgulamayı yapabilirler. Birinci durumda, kullanıcının nesnenin kendi yerel sisteminde olduğunu bildiği varsayılmaktadır.İkinci durumda, Sistem synonym (Eş anlam) tablosundan sistem ismini belirleyip , hangi sistemde olduğunu anlayarak ona göre sorgulama yapıyor. Sorgulamayı yapan kişi nesnenin nerede olduğunu bilmesine gerek kalmıyor. Synonym tablolarıyla beraber her sistemde ;

i) O sistemde doğan her nesne için katalog girdisi.
ii) O sistemde yer alan her nesne için katalog girdisi yer alır.

Eğer kullanıcı "mstatu"dan birşeyler sorgulama talabinde bulunuyorsa: Önce sistem ilgili sistem ismine bakar (Tamamıyla yerel bir erişim). Sonra doğum sisteminin Ankara olduğunu anlar. Uzak bir erişimle Ankara Sistemine ulaşılır. Ankara'daki katalogdaki bir göstrergeye göre nesnenin hala orada olup olmadığı anlaşılır. Eğer gösterge orada olduğunu gösteriyorsa sorun yoktur; tek bir uzak erişimde ilgili neseneye ulaşılmıştır. Fakat nesne Ankara'dan İzmir'e gönderilmişse o zaman ikinci bir göstergeye ihtiyaç vardır. Bu gösterge de nesnenin aktif olarak nerede saklandığını göstermektedir. Yani İzmir'i göstermektedir. Görüldüğü gibi nesne en fazla iki uzak erişimle bulunmaktadır. Nesnenin tekrar, örneğin Antalya'ya gönderilmiş olduğunu varsayalım. Bir Antalya katalog girdisi ekleyip, İzmir katalog girdisi silinecektir ve Ankara'da İzmir'i gösteren katalog girdisi Antalya'yı gösterecek şekilde değiştirelecektir. Bu metodla nesne, kesinlikle en fazla iki

erişimde bulunacaktır. Bu yaklaşım dağıtıklığa tamamen uygundur. Hiçbir merkezi sistem söz konusu değildir. Tek bir sistemdeki aksaklık tüm sistemin çökmesine sebep olmayacaktır. Uygulama veritabanları arasında bu yaklaşımı destekleyen hemen hemen hiç yok gibidir . Ancak programcı kişinin katkılarıyla bu yöntem kısmen uygulanabilir.

2.4.3. Güncelleme Tekrarı:

Veri tekrarındaki en önemli sorun, herhangi mantıksal nesnede (tablo) yapılan değişikliğin o nesnenin bütün kopyalarına yansıtılmasıdır. Burada sorun, bu nesnenin kopyalarının olduğu sistemlerden birinin veya daha fazlasının network çökmesi veya başka bir sorundan dolayı güncelleme işlemi sırasında ulaşılamaz olmasından kaynaklanır. Böyle bir durumda güncelleme işleminin hemen bütün kopyalara yansıması şeklinde olan genel strateji gerçekleşemez. Çünkü kopyalardan bazıları o an için erişilmeye elverişli değildir. Dolayısıyla güncelleme işlemi başarısızlığa uğrayacaktır. Böyle bir durumda da dağıtıklığın bir avantajı olduğu varsayılan kopyalama bağımsızlığının düşünüldüğü gibi fayda sağlamadığı düşüncesi oluşabilir. Veri kopyalamayı avantajlı hale getirmek ve bu problemi çözmek için şu yöntem kullanılabilir:

- Her tekrarlanan nesnenin bir kopyası ana kopya olarak tasarlanır. Diğer kalanlar ikincil kopyalardır.

- Farklı nesnelerin ana kopyaları farklı sistemlerde tutulur (Dağıtıklığa uygun bir yapı).

- Ana kopya yenilendiği zaman güncelleme işlemi başarıyla sonuçlandı sayılır. Bu kopyayı tutan sistem, sonraki sürede değişikliği diğer kopyalara yansıtmakla görevlidir.

Bu çözümün tek eleştirilecek yanı "Yerel Özerklik" ilkesine biraz ters düşmesidir. öyle ki eğer ana kopyaya erişim imkansızsa, değişiklik, diğer ikincil kopyalarda erişimin mümkün olmasına rağmen yansıtılamayacaktır. Dolayısıyla tek bir sisteme bağımlılık doğacaktır.

2.4.4. Düzelteme - Veri Güvenliği:

Dağıtık sistemlerde düzeltme kontrolü, iki safhalı taahhüt protokolündeki mantık baz alınarak gerçeklenir. İki safhalı taahhütte , tek bir hareketin farklı özerk kaynak yöneticileri ile karşılıklı ilişkide bulunduğu zamanlarda ihtiyaç duyulur. Bu özellikle dağıtık sistemlerde çok önemlidir. Çünkü kaynak yöneticileri (Yerel

VTYS'ler) farklı sistemlerde yer almaktadır ve bu yüzden çok özerkdirler. Özellikle bağımsız sorunlardan dolayı zedelenebilirler. Burada şu hususlar önem kazanmaktadır:

1) "Tek bir merkez sisteme güvenmek (bağımlılık) yoktur." kuralı, koordine fonksiyonuna ağ üzerindeki tek bir seçilmiş sistemin sahip olmaması gereğini, farklı hareketlerin farklı sistemler tarafından icre edilmesini ifade eder. Tipik olarak hareket hangi sistem tarafından başlatıldıysa, o sistem tarafından idare edilir.

2) "İki safhalı taahhüt" prosesi koordinatörün diğer bütün katılımcı sistemlerle haberleşme içerisinde olmasına ihtiyaç duyar. Bu da daha fazla mesaj dolayısıyla daha fazla maliyet demektir.

3) Eğer Y, "İki safhalı taahhüt" içerisinde X tarafından yönetilen katılımcı bir sistem ise, Y sistemi X sisteminin istediği herşeyi yapmak zorundadır (commit veya rollback). Bu da yerel özerkliğin başka bir kaybıdır.

4) Tabii ki ideal olarak herhangi bir noktada sistemin veya ağın çökmesi durumunda dahi "İki safhalı taahhüt"ün çalışmasını istenir. Akla gelebilecek her türlü çökmelerde prosesin devamlı sağlıklı bir şekilde gerçeklenmesi beklenir. Fakat ne yazık ki görüldüğü gibi bu prosesin devamlı sağlıklı çalışması garanti değildir. Dolayısıyla problem çözümsüzdür. Yani herhngi bir çökmede bütün sistemlerde hareketin başarısızlığa uğraması veya hareketin başarıya ulaştığını her sisteme haber verip kabul ettirmek garanti değildir. Böyle bir protokol için minimum N mesajın gerekli olduğu , bu N adet mesajın bir kaç tanesinin kaybolduğu varsayılsın; ya bu kayıp mesajlar gereksizdir ki bu da minimum N mesaj gerekliliğiyle çelişmektedir ya da protokol çalışmamaktadır . Her iki durumda da çelişki mevcuttur. Buradan da böyle bir protokol olmadığı sonucu çıkarılır.

2.4.5. Eş Anlılık:

Çoğu dağıtık sistemlerde, diğer dağıtık sistemlerde olduğu gibi eş anlılık kontrolü, klitlemeye dayalıdır. Fakat dağıtık sitemlerde test edilen istekler, kilitlemek ve kilidi çözmek , mesajlar ile sağlanmaktadır. Mesajlar da maliyeti artırmaktadır. Örneğin T hareketinin n tane uzak sistemi güncellemek istediği varsayılsın. Eğer bütün sistemler kendi sistemlerindeki nesneleri kilitlemeye cevap verebilecek durumdaysa, böyle bir uygulama en az 5*n adet mesaja ihtiyaç duyar:

n adet kilit isteği,
n adet kilit teslim etmek,
n adet güncelleme mesajı,

n adet doğrulama,
n adet kilit kakdırma isteği.

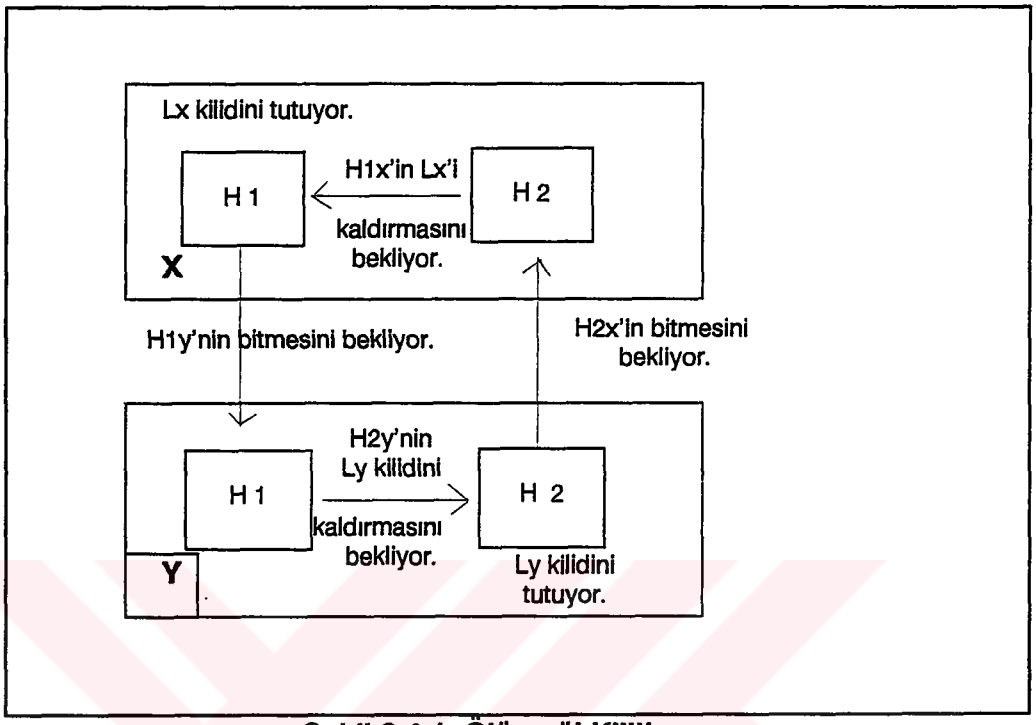
Böylece güncelleme işlemi merkezi bir sisteme göre çok daha uzun sürer ve bu mesajlar extra bir maliyet getirir.

Burada da güncelleme için kulanılan her nesnenin farklı bir sistemde kopyası olma yöntemi kullanılabilir. Yani her nesnenin bir ana kopyası olur. bu şekilde mesaj sayısı 5*n 'den 2*n + 3 'e indirgenebilir. 1 kilit isteği, 1 kilit teslim etmek, n güncelleme, n doğrulama, 1 kilit kaldırma mesajı ile kilit kaldırma kontrolü sağlanır.

Dağıtık sistemlerde (şekil 2.4.1) 'deki durum, deadlock' (Ölümcül kilitlenme) a sebep verebilir. Şekilde de görüldüğü gibi X ve Y sistemlerinde iki farklı hareket icra edilmektedir. Her hareket belli bir kayıdı kilitlemektedir. Fakat şekildeki kilitlemeler aynı kayıt üzerşnde gerçekleştiği ve sistemlerin birbirlerinden tam anlamıyla haberdar olamdığı için ölümcül bir kilitlenmeye sebebiyet vermektedir.

1) H1y hareketi aynı sistemdeki H2y'nin koyduğu Ly kilidinin kaldırılmasını beklemektedir;
2) Diğer yandan H2y hareketi de X sistemindeki H2x hareketinin tamamlanmasını beklemektedir;
3) H2x ise kendi sistemindeki H1x'in koyduğu kilidin kalkmasını, yani H1x 'in tamamlanmasını beklemektedir;
4) H1x ise , H1y hareketinin koyduğu kilidi kaldırmasını beklemekte ve sonuç olarak da ölümcül bir global ölümcül bir kilitlenme meydana gelmektedir.

Buradaki kilitlenmeye sebep hiçbir sistemin yalnız kendi iç bilgilerini kullanarak böyle bir kilitlenmeden kaçamadığıdır. Bu durumdan haberdar olabilmek için akrşı sisteminde yapısından haberdar bir şema bilgisi tutulması gereklidir. Bu da direkt olarak çoğu veritabanı yönetim sistemlerinde sağlanmamaktadır. Veritabanı yöneticisi tarafından kilitlemelerle ilgili birtakım şema bilgileri konması ve bunların kontrolüyle bu durumdan kaçınılabilir.



Şekil 2.4.1. Ölümcül Kilitlenme

2.5 .TCP/IP ve OSI Referans Modeli

Bilgisayar iletişim ağları son 20 yıl içerisinde çok önemli değişikliklere uğramışlardır. Bir zamanlar sadece bazı uzmanların araştırma konularını oluşturan ağ yapıları, günümüzde kişisel bilgisayarlardan süper bilgisayarlara kadar uzanan geniş bir yalpazenin birbirleriyle iletişimini sağlayacak şekilde gelişmiş ve yaygınlaşmış bulunuyor. Hatta farklı ağ yapılarının birbirleriyle iletişim halinde olması söz konusudur. Böyle bir eğilim kaçınılmaz olarak bilgisayar firmalarını bazı standardlar üzerinde anlaşmaya götürmüştür. Söz konusu standardlar uluslararası standardlar organizasyonu tarafından oluşturulan OSI (Open Systems Interconnection) referans modeline uyum sağlamaktadır.

2.5.1. OSI Referans Modeli

OSI referans modeli, verilen sistemler arasında, hatta tek bir sistem içinde ne şekilde yol alacağını belirleyen yedi katmanlı modüler bir yapıyı ifade eder. Bu yapı içinde her katman değişik bir görev üstlenmektedir. TCP/IP dört temel katmandan oluşmuştur.

1'den 4'e kadar olan katmanlar işlemcilerin birbirleri ile bağlantısını sağlayan katmanlar; 5'ten 7'ye kadar yer alan üst katmanlar ise bu işlemciler üzerinde çalışan uygulama yazılımlarının birbirleriyle iletişimini sağlarlar.

A sisteminden B sistemine göndrilen bir mesaj, A sisteminin 7. katmanından 1. katmanına kadar iner ve ağ ortamından geçerek B sisteminin birinci katmanından girip 7. katmanına kadar çıkar.[5]

Bu katmanlardan kısaca bahsedersek:

7. Application Layer (Uygulama katmanı): Ağı kullanan uygulama programları yer alır.

6. Presentation Layer (Takdim Katmanı): Sunulan verilerin uygulamalara göre standardizasyonu yapılır.

5. Session Layer (Oturum Katmanı): Ugulamalar arasındaki oturumların (session) idaresini gerçekleştirir.

4. Transport Layer (Aktarma Katmanı): Gönderen ile alan arasında güvenilir veri akışını sağlamakla yükümlüdür. Hata denetimi ve düzeltmesini sağlar.

3. Network Layer (Şebeke-Ağ Katnmanı): Yönlendirme (routing) ve sıkışıklık-trafik (congestion) kontrolünü yapar.

2. Data Link Layer (Veri İletişim Katmanı): Noktadan noktaya bağlantı esnasındaki verilerin maliyeti düşük-etkin ve hatasız bir dizi halinde gönderilmesini kontrol eder.

1. Physical Layer (Fiziksel Katman): İletişim kanalı üzerindeki ham veri denilen bilgilerin aktarımını sağlamaktır. Sistemin fiziksel karakteristiklerini taşır.

2.5.2.TCP/IP ve Katmanları:

TCP/IP bir yazılım ya da donanım ürünün adı değil; bir standard üretim iletişim topluluğunun adıdır. TCP/IP adı, TCP(Transfer Control Protocol) ve IP (Internet Protocol) adlı iki önemli parçanın birleştirilmesinden oluşmuştur.

4. Application Layer(Uygulama Katmanı): Ağı kullanan uygulama programları yer alır.Transport katmanı ile veri alışverişinde bulunarak bu katmana gerekli bilgiyi ve mesajı geçirirler. Bu uygulama programları sayesinde içlerinde telnet Server veya Ftp Server bulunan bilgisayarlar ile iletişim kurulabilir.

3. Transport Layer: Güvenilir veri akşının kontrolü, hata denetimi ve hata düzeltmesi görevini üstlenir.Bir uygulama programı ile diğer bir uygulama programı arasındaki iletişimi düzenlemektedir. (End to End) Ayrıca bu katman bilginin akışını düzenlemek, hatasız ve doğru sırada gelip gelmediğinden emin olmak, hatalı gelenleri yenideb talep etmek, alınan paketlerle ilgili diğer uca bildirmek (Ack-kowledgement) gibi işlevleri yerine getirir. Bu katmanın aynı anda birçok kullanıcı programından verileri kabul edebilir ve bir alt seviyesine (IP) teslim eder. Bu seviyede eklenen başlık bilgileri TCP başığı şekil 2.5.3. de görüldüğü gibidir.[6]

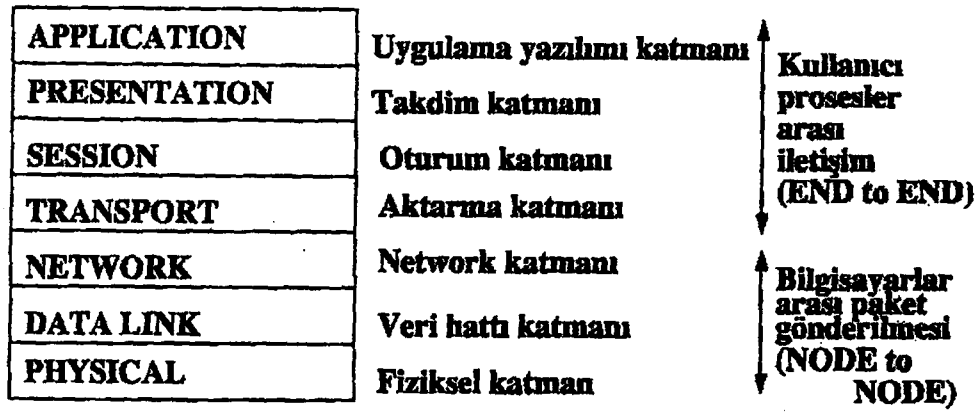
IP katmanı kendisine gelen Datagramların doğru sırada gelip gelmediğini veya gönderilmiş olup da alınmayanları tespit etmez. Sadece gelen bilgiyi Transport katmanına geçirir. Bu nedenle Transport katmanı bütün bu kontrolleri kendisi yapmak zorundadır.

2. Internet Layer: Datagramların kontrolü , yönlendirme ve trafik denetimi idaresini gerçekleştirir. İki bilgisyar arasında olan iletişimi kontrol eder (Node to node). Transport katmanından TCP başlık bilgileri ile gelen bilgiyi alır ve IP datagram haline getirir. Yönlendirme (Routing) algoritması ile bu datagramın network içinde direkt olarak teslim edilip edilmeyeceğini veya teslim için geçit (gateway) e yollanması gerektiğini tespit eder ve uygun arabirimine datagramı aktarır. Internet içinde her datagram bağımsız, kendine yeterli bir birimdir.

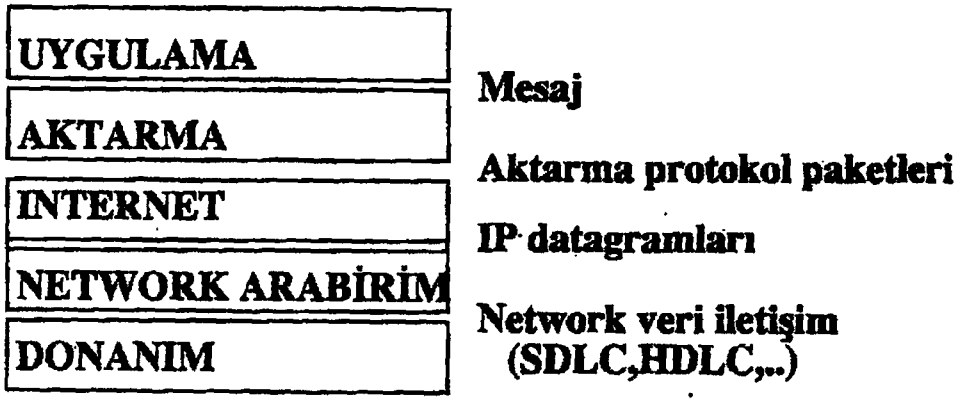
1. Network Access Layer: Noktadan noktaya bağlantı esnasındaki verilerin etkin ve hatasız bir dizi halinde gönderilmesini ve fiziksel ağın erişimini kontrol eder. IP datagramlarını alır ve iletişim ağına geçirir. Bu kademede çalışan yazılım cihaz sürücü olarak (device driver) adlandırılabilir. Fiziksel network içinde bilgiler yine Frame denilen paketler halinde hareket ederler.

2.5.3. Internet

Dünya üzerinde değişik amaçlara hizmet eden irili ufaklı çok sayıda ağ gelişmiştir. Her ağ kendi amacına uygun ve genellikle diğerleri ile uyumsuz bir teknoloji kullanmaktadır. Herkesin birbiri ile haberleşebildiği evrensel bir ağın , kullanıcıları aynı tip yazılım ve donanım kullanmaya zorlayarak oluşturulamayacağı açıktır. Internetworking ya da internetting denilen teknoloji bu sorunun çözümü olarak ortaya atılmış ve başarı ile uygulanmıştır. Günümüzde açık sistem bağlantısı (OSI) üzerindeki çalışmalar belli bir olgunluğa ulaşmış ve yaygın olarak kabul gören pek çok standard ortaya çıkarılmıştır. Burada bahsedilecek konu, ilk Internet girişimlerinden biri olan ve günümüzde dünya üzerinde dağılmış milyonlarca bilgisayarı birbirine bağlayan TCP/IP Internet'dir. TCP/IP ağı 1970 yılında ABD



Şekil 2.5.1. OSI Katmanları



Şekil 2.5.2. TCP/IP Katmanları

IP BAŞLIK YAPISI

0	4	8	16	19	24	31
Vers	B.uzun	Serv.tipi	Toplam uzunluk			
Datagram Kimliği			Flags	Datagram parça yeri		
Varolma süresi		Protokol	Başlık hata kontrol			
Kaynak IP adresi						
Nihai IP adresi						
IP Tercihleri						
BİLGİ						
.....						

TCP BAŞLIK YAPISI

0	4	10	16	24	31
Kaynak Port			Nihai Port		
Sıra numarası					
Onaylama numarası (Ackn. number)					
B.uzun	.....		Pencere		
.....					
.....					
MESAJ					
.....					

Şekil 2.5.3. TCP/IP Başlık Yapısı

Savunma Bakanlığına bağlı DARPA (Defense Advanced Research Projects Agency)nin desteği ile başlatılan bir çalışmanın ürünüdür. Önceleri sadece birkaç üniversite ve araştırma kurumunu bağlayan ağ ticari ve resmi kurumların katılımı ile büyümüş, diğer ülkelerin ulusal ağları ile birleşerek bugünkü halini almıştır.

2.5.4. TCP/IP Numaraları

TCP/IP ağlarında, ağa bağlı her üyenin (PC, Mainframe, Workstation, akıllı yazıcı vs.) bir numarası vardır. Bu numara dört byte (32 bit) uzunluğundadır ve okunmada kolaylık olması için noktalarla ayrılmış dört desimal veya hexadecimal (193.140.195.66 veya 0xC1.8C.C3.42) sayı olarak ifade edilir . Her sayı sekiz bitlik yer tutar. Dolayısıyla her sayı 0 ile 255 arasında değerler almaktadır. Örneğin 142.122.1.1 geçerli bir TCP/IP numarasıdır. Bu ağ numaraları kendi içlerinde değişik boylarda küçük alt ağlar oluşturulacak şekilde düzenlenmiştir. Bir Internet (IP) numarası iki parçadan oluşur ağ numarası ve üye numarası . Toplam dört byte olan Internet numarasının hangi kısmının ağ numarası, hangi kısmının üye numarası olduğunu belirtmek için Internet numaraları alt-ağ büyüklüğüne göre beş sınıfa ayrılmıştır.

- A Sınıfı : 0 adresi ile başlayan, 8 bit ağ numarası, 24 bit üye numarası olmak üzere 1.0.0.0 ile 127.0.0.0 arasında değişen çok büyük ağlardır.

- B Sınıfı : 10 adresi ile başlayan , 16 bit ağ numarası, 16 bit üye numarası ile ifade edilen 128.1.0.0 ile 191.254.0.0 arasında değişir.

- C Sınıfı : 110 adresi ile başlayan, 24 bit ağ numarası, 8 bit üye numarası ile ifade edilen, 192.1.1.0 ile 223.254.254.0 arasında olan ağlardır. Daha küçük ağlardır.

- D Sınıfı : Multicast adresleri içerirler.

- E Sınıfı : İlerideki kullanım için ayrılmış ağlardır.

Adres sınıfları farketmeksizin belli bir fiziksel ağdaki tüm üyeler aynı ağ numarasına fakat farklı üye numaralarına sahip olmalıdırlar.

-Üye Numarası atanması:

- 0 Üye numarası bu üye ya da ağ kullanmıyor manasını taşımaktadır.

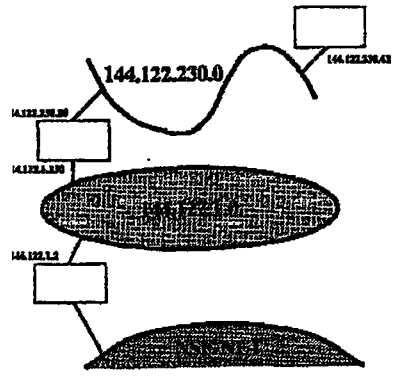
- 1 Üye numarası gatewayleri işaret etmek (numaralamak) için kullanılır.

- Bütün bitleri 1 olan , 255 veya 255.255 olan üyeler broadcast adresi olarak kullanılırlar. Bir mesaj gönderdiklerinde bu mesaj otomatik olarak kendileri ve tüm üyelere gider. Örneğin 193.1.1.0 C sınıfı bir IP numaraya sahip isek ağ numaramız 193.1.1 dir ve son byte'ın 0-255 arası değerler alabileceğini düşündüğümüzde (0,1,255 hariç- yukarıdaki sebeblerden dolayı) 2 ile 254 arasında 253 üye (bilgisayarı) kendi ağımıza dahil edebiliriz. Bu sayı eğer çok büyük bir ağa sahip değilsek bizim için tatminkardır.

Daha büyük bir ağa yapısına sahip bir kullanıcı grubu için ise B sınıfı bir numara daha uygundur.

Örneğin 144.124.0.0 ; burada üçüncü byte 254 üyelik alt- ağlar için kullanılabilir. 144.124.0.0 İ.T.Ü. nün ağ numarası ise ve İ.T.Ü. bünyesinde yer aldığı varsayılan TUBİTAK Yazılım Araştırma ve Geliştirme Grubu 144.124.235.0 ile bir alt ağ grubu numarasına sahip olabilir. Aynı zamanda bu alt grup içerisindeki bir Unix makine 144.124.235.34 ve bir PC 144.124.235.37 IP numaralarına sahip olabilir. Dolayısıyla 254 üyelik alt grup ve her alt grupta 254 üye yer alabilir. 0 ağın kendisini belirttiğinden ve 255 broadcast için ayrıldığından bu iki numara üyeler tarafından kullanılamamaktadır. Yukarıdaki örnekte İ.T.Ü. kendi alt ağ numaralarından sorumludur, kendi Internet numaralarını istediği gibi alt ağlara ayırmakta serbesttir. Ama TUBİTAK'ın kendi alt numaralarını nasıl kullanacağı ayrıntısıyla ilgilenmez.

Bu şekilde hiyerarşik yapılara bölünmüş TCP/IP ağları biribirleri ile IP router'lar yolu ile haberleşirler. IP routerlar bağlı oldukları her ağa üyedirler ve bu ağlar arasında bilgi alışverişini düzenlerler. Bütün routerlar hiyerarşik bir düzen içinde birbirleri ile bağlantı kurarak hangi ağa gidecek bilginin hangi yöne gönderilmesi gerektiğini bilirler. Şekil 2.5.4.'de ODTÜ deki TCP/IP ağının bir kısmı görülmektedir.



Şekil 1: ODTÜ'deki TCP/IP Ağı

Şekil 2.5.4. ODTÜ TCP/IP Ağı Bağlantısı

144.122.230.42 numaralı bilgisayardan 128.2.210.6 numaralı bir bilgisayara gönderilen bilgi önce 144.122.230.80 sonra 144.122.1.2 numaralı router'lardan geçerek yerine gider. [7]

2.5.5. Temel IP Yönlendirmesi (ROUTING):

Paketlerin yönlendirilmesi yalnızca IP tarafından yapılır.IP ağ numarasına bakarak yönlendirme yapar; bu esnada üye (host) numarasını dikkate almaz. Hedef routing algoritmasını basitleştirmek ve yönlendirme tablolarının hacmini küçültmektir. Dikkat edilmesi gereken nokta aynı data link ağına bağlı bütün üyelerin aynı ağ numarasına sahip olmalarıdır. Her yönlendirme (Routing) tablosu aşağıdaki formda olmalıdır. Ağ namarası ,gateway numarası, flagler 193.140.192.0 193.140.196.65 Bir yönlendirici paketleri geçirirken , framelerini çözer ve yeniden framelendirir. Bu esnada kaynak ve hedef IP adresleri değişmez. Yönlendirme Protokolü , ağ üzerindeki her yönledirme düğümündeki yönlendirme tablolarını update etmek için kullanılır. Yönlendirme protokolü , yönlendiricileri (router) ağ üzerindeki ulaşılmak istenen hedefe en etkin biçimde bir rota çizmek üzere aktif kılar. Unutulmaması gereken nokta, Yönlendirme (Routing) protokolünün IP'nin bir parçası olmadığıdır. IP aktif yönlendirmeyi, yönlendirme tablolarının nasıl inşa edildiğini bilmeksizin gerçekleştirir.

2.5.6. IP Adresleri : Subnet ve Masklar

IP adresleri bir ağ numarası,alt ağ numarası ve üye numarası olmak üzere bölümlere ayrılabilir. Bunun için sebep networke her data link veya yönlendirici (router) eklendiğinde IP numarasını register etmekten kaçınmaktır. Network çoğu zaman hiç beklenmedik şekilde büyümek zorunda kalabilir. 142.131.0.0 şeklindeki bir B sınıfı adresde, üçüncü byte alt ağ numarasını ve dördüncü byte üye numarasını belirtecek şekilde davranabilir. IP adresinin hangi kısımlarının alt ağ numarası içerdiğini belirtmek için 32 bit ağ maskeleri (mask) kullanılır. Örneğin yukarıdaki B sınıfı adres için mask : 255.255.255.0 2 biti altağ numarası ve 6 biti üye numarası olmak üzere bir C sınıfı adres için mask : 255.255.255.192 şeklinde belirtilebilir. Alt ağlara parçalanmış bir networkdeki yönlendirme, alt ağ içermeyen bir nerwordekiyle ayınıdır. Her alt ağ tek bir networkmüş gibi davranır. Yine bir B sınıfındaki 5 numaralı altağ için Broadcast adresi için bir örnek: 142.131.5.255 şeklinde verilebilir.

2.5.7.Üye (Host) İsimleri, Domain ve Alt Domain İsimleri

Internet numaralarını yorumlamak bilgisayarlar için güç olmasa bile insanlar için oldukça zordur. Bu nedenle TCP/IP internetinde ikinci bir adresleme mekanizması kullanılmaktadır. Bu mekanizmaya domain adları denir. Hafızada kalabilecek, hatırlanabilir bir üye ismi bir IP adresinden daha iyidir. Örneğin palamut, 193.140.196.65 'e göre daha hatırlanabilir bir isimdir. Bu kolaylığı sağlamak için UNIX 'de /etc/hosts adında içinde IP adresleri ve buna karşılık gelen üye isimleri bulunan bir dosya yer alır. Bu dosyanın her satırında bir IP adresi ,üye ismi ve kısaltılmış üye ismi yer alır.

Çok büyük networklerde bu isimlendirme ad serverları kullanılarak yapılır.(DNS veya NIS) Domainler üye ve/veya alt domainlerin kolleksiyonunu ifade ederler. Internet'te organizasyonlar hiyerarşik olarak adlandırılmıştır. Internet'teki her ülkenin bir domain adı vardır. Buna tek istina ABD'dir. Her organizasyon kendi altında istediği derinlikte hiyerarşi oluşturmakta serbesttir. Hiyerarşideki her seviye birbirinden domain adındaki noktalarla ayrılır. Üst düzey domainler Amerika'daki gibi organizasyonel bir şekilde (com,edu,gov,mil,net,org) veya Amerika dışında ise coğrafi olarak (uk.tr.jp şeklinde ülke kodları) ile ifade edilirler.

Türkiye'nin domain adı tr'dir. Bunun altında akademik kuruluşları içeren edu, devlet kurumlarını içeren gov gibi alt-domainler tanımlıdır.

Alt domainler belli bir domainin mantıksal bölümlerini ifade eder. Her register edilmiş domain kendini alt domainlere bölme yetkisine sahiptir.Örneğin ODTÜ tr domainini bölme yetkisine sahiptir.Bunun altında akademik kuruluşları içeren edu, devlet kurumlarını içeren gov gibi alt domainler tanımlıdır. srdc.metu.edu.tr adresi Türkiye'deki akademik bir kuruluş olan METU' daki bir araştırma kurumu olan SRDC'nin domain adıdır. Domain adlarındaki en son parça host adını belirler. Örneğin yeniköy.srdc.metu.edu.tr bir host adıdır. Her host adı bir Internet numarasına karşılık düşer. Ancak her host'un ya da her ağın bir domain adı olma zorunluluğu yoktur.

Örnek domain isimleri: cc.metu.edu.tr,cc.boun,edu.tr,cs.stanford.edu,itu.com.

Küçük networklerde domain ve altdomain isimleri UNIX altında /etc/networks dosyası altında saklanır. Burada yine daha hatırlanabilir olması açısından domain ve altdomainlere karşılık gelen isimler de saklanır. Daha büyük sistemlerde ise bu işlem isim serverları kullanılarak gerçeklenir.(DNS veya NIS)

Uygulama Katmanı Servisleri:

TCP/IP ağlarında kullanıcılara basit dosya transferinden hypertext bilgi erişim sistemlerine kadar pek çok olanak sağlanmıştır. Bu servislerin bazıları uzun süredir işletim sistemlerinin doğal bir parçası haline gelmiştir. Bazıları ise henüz yeni geliştirilen servisler olup sonradan sağlanmaları gerekmektedir. Genel olarak TCP/IP servislerinin ortak özelliği public domain olarak bulunabilmeleridir.

BÖLÜM 3. PROGRESS VTYS

3.1. Veri Tabanı Yönetim Sistemi

VTYS veritabanına bütün erişimleri idare eden bir yazılımdır. Kavramsal olarak çalışması şu şekilde ifade edilebilir.

1.) Kullanıcı bir erişim talebini, bir veri alt dili (data sublanguage) SQL, Progress 4GL, Informix 4GL vb. kullanarak ifade eder.

2.) VTYS bu talebin yolunu keser - durdurur ve onu analiz eder.

3.) VTYS sırasıyla; o kullanıcı için dış şema (external schema) , dış kavramsal tasviri (external/conceptual mapping) , iç kavramsal tasviri (conceptual/internal mapping) ve veri yapısal tanımını (storage structure definition) kontrol eder.

4.) VTYS veritabanı verileri üzerindeki gerekli hareketleri icra eder.[3]

VTYS'nin işlevleri biraz daha detaylı incelenecek olursa şu noktalar önem kazanmaktadır.

Veri Tanımlama (Data Definition)

VTYS veri tanımlarına program listesi (source) biçiminde erişebilmeli ve onları uygun object biçime çevirebilmelidir. Başka bir deyişle, VTYS çeşitli veri tanımlama dilleri için dil işlemcisine sahip olmalı ve aynı zamanda Veri Tanımlama Dili (DDL) tanımlarını anlamalıdır. Örneğin "Firma" dış kayıtlarının bir "İndirim" alanı içerdiğini anlamalı ve bu bilgiyi kullanıcı isteklerini yorumlamada ve onlara cevap vermede kullanabilmelidir (Örneğin indirim oranı %25 den düşük olan firmaların sorgulanması).

Veri İdaresi (Data Manipulation)

VTYS kullanıcı tarafından gelen, veri görüntüleme, veri ekleme, güncelleme isteklerini idare edebilmelidir. Genellikle DML istekleri planlanmış ya da planlanmamış olabilir.

- Planlanmış istek; icra edildiği anda ihtiyacı evvelden görülen istektir. VTYS böyle bir durumda fiziksel veritabanını isteğe iyi bir performansla cevap verebilmeyi garanti eder. - Planlanmamış bir istek; ters olarak aniden çıkan evvelden görülmeyen bir istektir. Fiziksel veritabanı tasarımı böyle bir istek için çok uygun olmayabilir.

Planlanmış istekler genellikle "operasyonel" veya "üretim" uygulamaları şeklinde , planlanmamış istekler "Karar destek uygulamaları" şeklinde ifade edilebilirler.

Veri Güvenliği ve Entegrasyonu

Veritabanı kullanıcı isteklerini izleyebilmeli, güvenlik ve entegrasyonu bozabilecek girişimleri geri çevirebilmelidir.

Veri Muhafazası ve Eşanlılık

VTYS'ler veya bazı eşdeğer olarak anılan yazılımlar hareket yönetici görevini üstlenmektedirler ve aynı zamanda kurtarma denetimi ve eş anlılık kontrolü görevini de üstlenmektedirler..

Veri Sözlüğü

VTYS bir veritabanı sözlüğü fonksiyonuna sahip olmalıdır. Veri sözlüğü veri hakkında veri şeklinde nitelendirilebilir. (Bazen metadata diye adlandırılır.) Özellikle bütün şemalar ve tasvirler (dış,kavramsal vs.) fiziksel olarak source ve object biçimlerde saklanabilmelidir. Geniş bir sözlük cross-reference bilgi, çeşitli gösterimleri, örneğin hangi kullanıcılar hangi raporlara ihtiyaç duyuyor, hangi programlar veritabanının hangi kısımlarını kullanıyor, hangi terminaller sisteme bağlı vb. içermelidir. Sözlüğün kendisi de bir veritabanı gibi inşa edilmeli ve bu sayede sözlük diğer veritabanları gibi sorgulanabilmelidir.

Performans

Tabii ki VTYS bu fonksiyonları en etkin biçimde sağlama görevini yerine getirmelidir.

Progress:

Progress son zamanlarda gelişme gösteren, kendi özel Dördüncü Generasyon program geliştirme diline sahip bir İlişkisel Veritabanı Yönetim Sistemidir.

Diğer Veritabanı Yönetim Sitemleri incelendiğinde (Oracle,Informix,Sybase vs.) program geliştirmede daha çok SQL'e bağımlılık gözlenmektedir. Progress ise kendine özgü Bir 4GL'e sahiptir. Genellikle bir projenin büyük bir kısmı sadece 4GL kullanılarak bitirilebilmektedir. Kendi 4GL'i dışında ayrıca ANSI SQL standardlarına da ugunluk göstermektedir. SQL komutları Progress derleyicisi altında çalışabilmektedir.

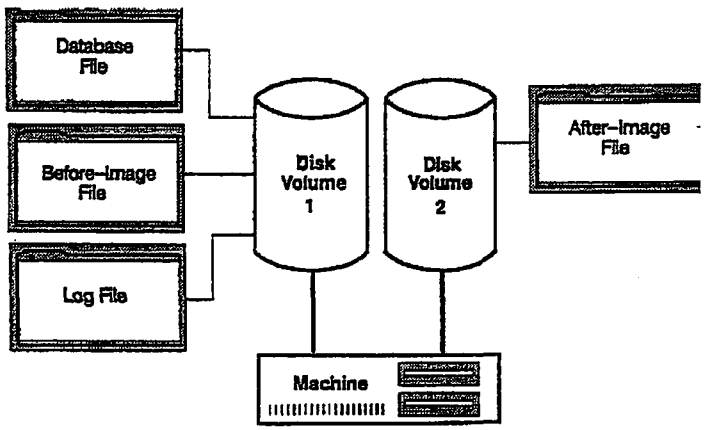
Bu bölümde Progress'i karakterize eden özellikleri veritabanı mimarisi, getirdiği sınırlamalrı, ihtiyaç duyduğu kaynakları, network üzerinde çalışma şeklini, performansını etkileyen faktörleri, çok kullanıcılı ortamda çalışma mantığı ve eşanlılık denetimi, hareket yönetimi,güvenlik ve muhafaza mantığı incelenecek v e ayrıca C.J. DATE'in dağıtıklığa ilişkin 12 kuralı baz alınarak bir DVTYS'e ne şekilde uygunluk gösterdiği belirtilecektir.

Yapılan incelemelerde tez çalışması esnasında kullanılan 6.2 versiyonu baz alınmıştır. Son zamanlarda çıkan 7.0 ve daha üstü versiyonlar grafik kullanıcı ara yüzü desteği ve nesneye dayalı bir veri tabanı yönetim sistemi olma özeliklerini de taşımaktadır. Tez konusu proje Unix ve TCP/IP platformlarında geliştirildiği için açıklamalar daha çok bu platform baz alınarak yapılmıştır.

3.2. Sınırlamalar Ve Kaynakların Kullanımı

3.2.1.Veritabanı Dosyaları:

Yeni bir progress veritabanı yaratıldığında otomatik olarak birbirleriyla ilişkili olan şu üç dosya oluşur:



Şekil 3.2.1. Veritabanı Dosyalarının Dağılımı

- Veritabanı Dosyası: Veritabanı tanım ve verilerini içeren dosyadır. Tek hacimli veritabanları için sonlarına hep .db uzantısı gelir. Çok hacimli veritabanlarında bir veya daha fazla olan .dn uzantılı dosyalardan oluşur. Burada "n" veritabanının parça sayısına göre değişen bir tamsayıdır. Çok hacimli veritabanları aynı zamanda bir veya daha fazla .bn uzantılı dosyalara sahip olabilirler.[8]

- Log Dosyası: Veritabanı başlatma ve sona erdirme işlemleri, ciddi hatalar ve veritabanındaki diğer önemli değişikliklere ilişkin tarih itibarıyla yer alan bilgilerin bulunduğu bir ASCII dosyadır. Bu dosya incelenerek veritabanının yaratıldığı andan itibaren yaşadığı gelişmeler izlenebilir.

- Before Image Dosyası: Dinamik hareketlerin geri alınması ve sistem çökmesinden dolayı zarar görmemek için, sitemin bu tür durumlardan evvelki halinin yer aldığı dosyadır. Bu dosya Progress uygulamaları esnasında ve arasında mevcut bulunmaktadır. Eğer bu dosya veritabanının bulunduğu yerden farklı bir disk üzerinde saklanmak istenirse (-g) başlama opsiyonu kullamılmalıdır. Tek hacimli veritabanlarında .bi, çok hacimli veritabanlarında .bn uzantısı alırlar.

Ayrıca Progress, Roll Forward Recovery opsiyonu ile kullanılıyorsa Progress tarafından otomatik olarak "after- image" dosyası yaratılır. Bu, veritabanının fiziksel sebeplerden dolayı (diskin fiziksel zarar görmesi gibi) gördüğü zararlardan kurtarmak üzere, veritabanının yeniden inşa edilmesinde kullanılır. Daha iyi bir korunma sağlamak üzere başka bir disk üzerinde saklanması tercih edilir. Veritabanı her başlatıldığında after-image dosyası belirtilmek üzere (-a) opsiyonu kullanılır. Progress bu dosyanın sonuna .ai uzantısını ekler.

.db, .lg , .bi ve .ai dosyaları yerleştirilirken şunlar göz önünde bulundurulmalıdır.

- Donanım kayıplarına karşı, after-image dosyası, .db ve .bi dosyalarının bulunduğu diskten farklı bir yerde saklamak daha fazla güvenlik sağlayacaktır.

- Performans ve disk alanı sorunlarından dolayı .bi dosyası veritabanı dosyasının bulunduğu yerden daha farklı yerde saklanabilir. Bu durumda dikkat edilmesi gereken .bi ve .ai'in aynı disk üzerinde olmamasıdır.

.lk Dosyası: Progress veritabanı kullanımda olduğu zaman, veritabanının bulunduğu dizin altında otomatik olarak [veritabanı].lk kilit dosyası oluşturur. Veritabanınından çıkıldığı zaman bu dosya yine Progress tarafından silinir. Fakat herhangi bir sistem çökmesi sonunda bu dosya durur ve sistem yeniden açılıp veritabanı kullanılmak istendiğinde bu dosya mevcut olduğundan dolayı girişe izin verilmez. Bu durumda veritabanının sistem ve ağ üzerinde kullanılmadığı kesin ise,

bu dosya silinir. Veritabanının kulanımı sırasında bu dosyayı silmek, veritabanına zarar verebilir. Veritabanının yeniden yüklenmesi gerekebilir.

Geçici Dosyalar: Progress veritabanı üzerinde bir uygulama icra edildiğinde uygulama içerisindeki hareketler ve değişkenlerdeki bilgileri tutmak için bir takım geçici dosyalar oluşturulur. Bunlar (.lbi) uzantılıdırlar. Uygulama bittikten sonra bunlar otomatik olarak temizlenir. Unix üzerinde (-t) (Sava Temp Files) opsiyonu kullanılmadığı sürece bu dosyalar çalışma dizininde gözükmezler; çünkü unlinked olarak yaratılmışlardır.

.lg Dosyası: Veritabanı log dosyası boyutu, kullanım arttıkça büyümektedir. Sadece son biligileri bırakıp diğerlerini silmek için "prolog [veritabanı]" komutu kullanılır.

3.2.2. Progress Sınırlamaları:

Progress'in getirdiği sınırlamalar, kullanılan donanım ve işletim sistemi versiyonu özelliklerine göre değişmektedir. Bunlar aşağıdaki gruplar halinde ifade edilebilir:

Veritabanı Limitleri:

Veritabanı	200 gigabyte (GB).
Dosyalar	Index sayısıyla sınırlıdır. Çünkü her dosya bir indekse sahip olmalıdır.
Indeksler	Her veritabanı için maksimum 1023 tanedir. Dosya başına düşen kesin bir sınırlama yoktur. Her indeksde maksimum 16 alan yer alır. Bir index içerisindeki bütün alanların karakter uzunlukları 127'den az olmalıdır.
Kayıtlar	Her kayıt için maksimum 32000 byte . (Pratikte buffer ve stack boyutu sınırlamaları bu sınırı yaklaşık 15000 byte'a düşürmektedir.)
Alanlar	Her dosya başına, kayıt boyutuyla sınırlıdır.
Kayıt Kilitlemeleri	Aynı anda SHARE veya EXCLUSIVE olarak kilitlenen maksimum kayıt sayısı makineye bağımlıdır. Fakat en azından 2000'dir.
Workfile	Minimum kayıt uzunluğu 64 byte'dır.

Veri Değerleri:

Karakter	Kayıt boyutuyla sınırlıdır.

Tarih	M.Ö: 1/32768'den M.S. 31/12/32767 arasındadır.
Desimal	Toplam 50 digitttir. 10 desimal hanenin üzerine çıkabilir.
Tamsayı	-2,147,483,648'den 2,147,483,647'ye kadar.
Mantıksal	true/false, yes/no

Veritabanı Başına Kullanıcı Sayısı:

Tek Kullanıcılı	1
Çok Kullanıcılı	Maksimum sayı makineye bağlıdır. UNIX ve VMS gibi paylaşımlı bellekli sitemlerde semafor, proses limitinin olmadığı, makine performansının kısıtlanmadığı sistemlerde 2048 kullanıcının üzerindedir. BSD 4.2. paylaşımlı belleksiz, UNIX Sistem 3 ve XENIX makineleri için bu limit her proses için kullanılan file descriptor sayısının 10 eksiğidir. Ortalama 10 ve 54 kullanıcı arasında değişmektedir. BTOS/CTOS makineleri için 64 kullanıcıdır.

Aynı Anda Veritabanı Başına Düşen Hareket Sayısı:

Tek Kullanıcılı	1
Çok Kullanıcılı	Her kullanıcı için 1 (Maksimum kullanıcı sayısı = 2048).

Editör:

Satır Sayısı	(Edit buffer boyutu / 82). 8086 ve 80286 işlemcili sistemler için edit buffer 2K-63K arasındadır. Diğer sistemler için 2K - 4GB arasındadır. Dikkat edilmesi gereken programın text uzunluğu ne olursa olsun derlenmiş hali (.r) 63 K'dan küçük olmalıdır.

INPUT/OUTPUT:

Terminal Satır Genişliği	UNIX,VMS,BTOS/CTOS : 80-132 DOS, OS/2 : 80 kolon.
Yazıcı Satır Genişliği	255'in üzerindedir.
Export Dosya	Her alan için 3000 karakter, her kayıt için 32 K karakter.

Import Dosya	Her alan için 3000 karakter, her kayıt için 32K karekter.
Stream'ler	Her prosedür (program) için 5 stream.

Sıralama:

Seviyeler	10 alan veya ifade mertebesinde.
Boyut	Her sıralanmış kayıttaki, bütün sıralı alanların birleşmesiyle oluşan boyut itibarıyla 127 karakter.

İsimler:

Prosedür İsimleri	UNIX : 1 - 12 karakter. DOS-OS/2 : 1 - 8 karakter ve .p uzantısı . VMS : 1 - 39 karakter. BTOS/CTOS: 1-50 karakter.
Dosya,alan,indeks	32 karakter uzunluğu üzerinde; a-z ve A-Z ile başlar; harf,sayı ve #,$,%,&,-,_ karakterlerinin herhangi kombinasyonları ile devam eder.
Veritabanı isimleri	UNIX : 1 - 11 karakter. DOS-OS/2 : 1 - 8 karakter. VMS : 1 - 39 karakter. BTOS/CTOS :1 - 12 karakter.

Derleyici:

	DOS, OS/2, XENIX 286 ve BTOS/CTOS üzerinde bütün aktif kayıtlar, değişkenler ve workfile'ları aktif durumdayken 1K-63K buffer alanı sınırları içerisinde olmalıdır. Diğer sitemlerde buffer alanı 4GB'a çıkabilir.
İfade	Her ifade için 4000 karakter. Her ifade için maksimum 1000 kelime . Her kelime veya işaret kelime sayılmaktadır.
Frame	Maksimum 255 karakter uzunluğundadır.
İç içe bloklar	Maksimum 255 bloktur.

3.2.3.Veritabanı Disk ihtiyacının Hesaplanması

Progress bütün veritabanı alanlarını indekslerini değişken uzunluklarınca saklamakta; karakter alanındaki boşluklar ve nümerik değerlerdeki 0'lar diskte yer kaplamamaktadırlar. Değişkenleri bu şekilde uzunluklarınca saklamak aşağıdaki avantajları getirmektedir.

1) Diskte veri saklamak için kullanılan alan azaltılmış olur.

2) Bu sayede belli bir alana verilebilecek maksimum boyut verileceğinden, uygulama sonradan meydana gelecek genişlemeler nedeniyle bölünmeyecektir.

Veritabanı Boyutunun Hesabı:

Boyut	Formül
Veritabanı Boyutu	Şema Boyutu + Veri Dosyası Boyutu + İndeks dosyası boyutu.
Şema Boyutu	15000 - 250000 arasındadır. Veritabanı üzerinde tanımlı olan dosya ve indekslerin sayısına bağımlıdır.
Veri Dosyası Boyutu	Ayrık dosya boyutlarının toplamıdır.
Ayrık Dosya Boyutu	Kayıt sayısı * alan uzunluğu * 1.5
İndeks Boyutu	Ayrık indekslerin toplamıdır.
Ayrık indeks Boyutu	İndekslenen dosyadaki kayıt sayısı * (7 + indeks içerisindeki alan sayısı + indeks alanı uzunluğu) * 2.

Alan Boyutunun Hesaplanması:

Veri tipi	Alan Uzunluğu(Byte)
Karakter	1 + karakter sayısı (boşluk hariç). Eğer karakter sayısı 240'dan büyükse, 1 yerine 3 eklenir.
Tarih	3.
Desimal	
0	1
0 olmayan	2 + (tamsayı digitleri sayısı + 1) / 2.
Tamsayı	
0	1
-+127	2
-+32767	3
-+8 milyon	4
-+2 milyar	5
Mantıksal	
false	1
true	1

Örnek: Sadece uçak dosyası olan bir veritabanının boyutu yaklaşık olarak hesaplanmak istenirse:

Uçak tescili : 6 karakter .
Uçak Tipi : 4 - 15 karakter .
Tonaj : Tamsayı 4 - 6 digit Tonaj Kodu — Devamlı 2 karakter.

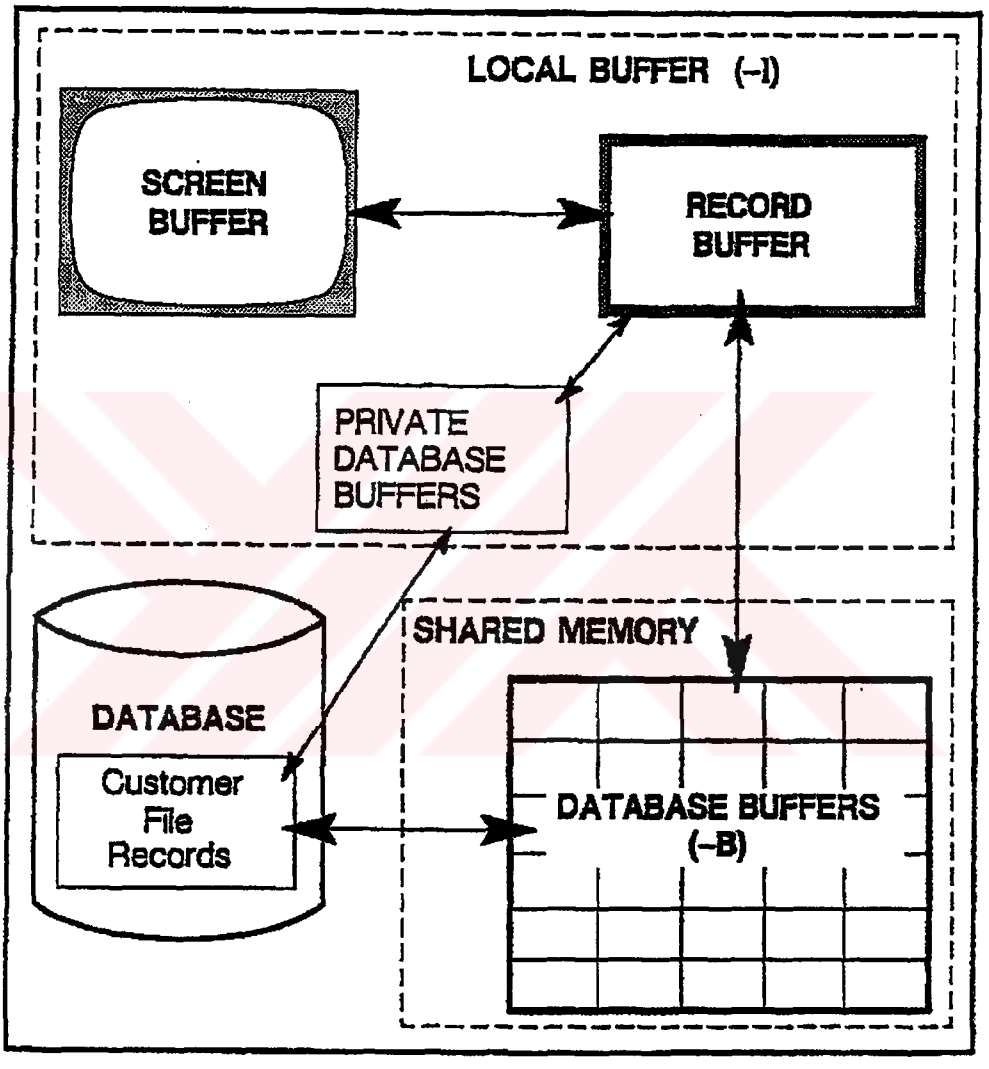
Şema Boyutu = 15000 (Sadece uçak tablosu mevcut olduğundan boş veritabanı boyutu alınmaktadır.)
Alan Uzunluğu = 6 + 10 + 5 + 2
tescil tip tonaj Kod
= 23.
Uçak Dosya Boyutu = Kayıt sayısı * alan uzunluğu * 1.5
= 3000 * 23 * 1.5 .
= 103500.
İndeks Boyutu = Kayıt sayısı * (7 + indeks içerisindeki alan sayısı + indeks alan uzunluğu) * 2.
= 3000 * (7 + 1 + 6) * 2
= 84000
Veritabanı Boyutu = Şema boyutu + Veri dosyası boyutu + indeks boyutu.
= 15000 + 103500 + 84000
= 202500 byte.

Bu değer genelde gerçektekinden biraz daha büyüktür. Şu anda hesaplanan yalnızca veritabanının büyüklüğüdür. Ayrıca geçici dosyalar, .bi dosyası ve .lg dosyalarının da uzunluğu da dikkate alınmalıdır.

Örneğin sıralama geçici dosyası için 100K - 250K arası yer bırakılmalıdır. Geçici prosedür kodu için yer bırakılmalıdır. Bu derlenen .r kodların sayısıyla bağlantılıdır. Genellikle 500K - 1500K civarındadır. .bi ve .lbi için de tahmini bir alan hesaplanmalıdır.

3.2.4. Veritabanı Buffer'ı Kullanımı:

Bütün veritabanına erişim istekleri, veritabanı buffer'ına gelir. Eğer istene veritabanı bloğu buffer'da değilse, Progres o bloğu diskten okur. Veritabanı buffer'ları (-B) parametresi ile belirtilerek paylaşımlı bellek içerisinde yer alır. Eğer veritabanı sadece okunmak için kullanılıyorsa "Özel veritabanı buffer"'ları kullanılır. Bu sayede paylaşımlı bellekteki buffer'lar boş yere işgal edilmez ve diğer



Şekil 3.2.2. Database Buffer'ın Kullanımı

güncelleme yapmak isteyen kullanıcılar engellenmemiş olur. Doğrudan veritabanına ulaşılır ve paylaşımlı bellekte yapılması gerekenler burada yapılır. Veritabanı buffer'ı haricinde kullanıcı bazında hareketler ve değişkenlerle ilgili yerel buffer'larda mevcutttur. İlgili opsiyonlar şunlardır:

-B opsiyonu, veritabanı buffer sayısını belirlemekte kullanılır.
-l , Özel buffer sayılarının kümülatif toplamı sayısını verir.
-O , her ayrı proses için özel veritabanı buffer'ı tahsis eder.
-l , Yerel kayıt buffer boyutunu belirler.
-e , prosedür editör buffer boyutunu belirler.

-B buffer sayısını belirlemek için aşağıdaki yöntem uygulanır: 5 + Her kullanıcı için: tek bir prosedürde ulaşılacak dosya sayısı + Her kullanıcı için: 3 * (Tek bir prosedürde kullanılan indeks sayısı) . -B mümkün olduğu kadar gerçek belleğin imkan verdiği ölçüde büyük seçilmelidir. Eğer veritabanı buffer'ı yeterince büyükse bütün kayıtlara hemen erişim mümkün olacak, disk I/O'su azalacaktır. Her buffer bloklama faktörüne göre 512, 1024 veya 2048 byte yer tutmaktadır. Aşağıdaki ifade ile veritabanı buffer'ı belirlenir.
proserve [veritabanı] -B buffer sayısı.
-B için herhangi bir değer verilmezse tek kulanıcı için 20, çok kullanıcılı için 8 * kullanıcı sayısı kadar otomatik olarak buffer sayısı verilmektedir. Maksimum değeri 32000'dir.

Ayrıca periyodik olarak büyük raporlar alan kullanıcılar -O opsiyonu ile kendilerine özel veritabanı buffer'ları tahsis etmelidirler. Aksi takdirde büyük bir rapor bütün veritabanı buffer'larını işgal edecek ve performansı dakikalarca düşürecektir. Herhangi bir değer verilmediği zaman -O değeri 0'dır. Minimum değeri 4' dür.

Yerel Buffer Boyutu:

Progress ekran değişkenleri, prosedürlerde kullanılan değişkenler ve ayrıca buffer'da tuttuğu workfile denen geçici çalışma dosyaları için yerel buffer'ı kullanır. Bu nedenle eğer bu tür değişken, kayıt ve workfile'lar buffer'da fazla yer tutuyorsa -l parametresini artırmak gerekir. Aksi halde aktif uygulamayı öldürücü hatalar doğabilir. Kullanışı aşağıdaki gibidir:
mpro [veritabanı] -l n ; n tamsayı.
n kilobyte cinsindendir. 386 ve üzerindeki INTEL ve diğer bütün işlemciler için yerel buffer 4GB'a kadar çıkabilir. 386'nın altındaki INTEL işlemciler için bu boyut 63K ile sınırlıdır.

Edit Buffer Boyutu :

INTEL 8086 ve 80286 için maksimum 63K; diğerleri için maksimum 4GB'dır. Kullanılışı aşağıdaki gibidir.
mpro [veritabanı] -e [buffer-boyutu].
Bu opsiyon Progress editörü buffer'ındaki bir prosedürün maksimum çıkabileceği limiti verir. Aynı zamanda derlenmiş bir prosedürün (.r dosyasının) üst sınırını da belirlemiş olur. Edit buffer her ne kadar 4GB'a kadar çıkabilse de, Progress'de yer alan ; derlenmiş bir dosyanın büyüklüğü 63K'yı geçemez kuralı edit buffer'ı belli bir prosedür için bu kadar üst bir sınırda kullanmayı dolaylı yoldan engeller. Bu yüzden Progress'le uygulama geliştiriken programların küçük parçalar halinde, modüler olması gerekmektedir. Çok büyük program parçacıkları kulanmaktan çekinilmelidir. Zaten belki de bu nedenle program kavramından çok prosedür kavramı kullanılmaktadır.

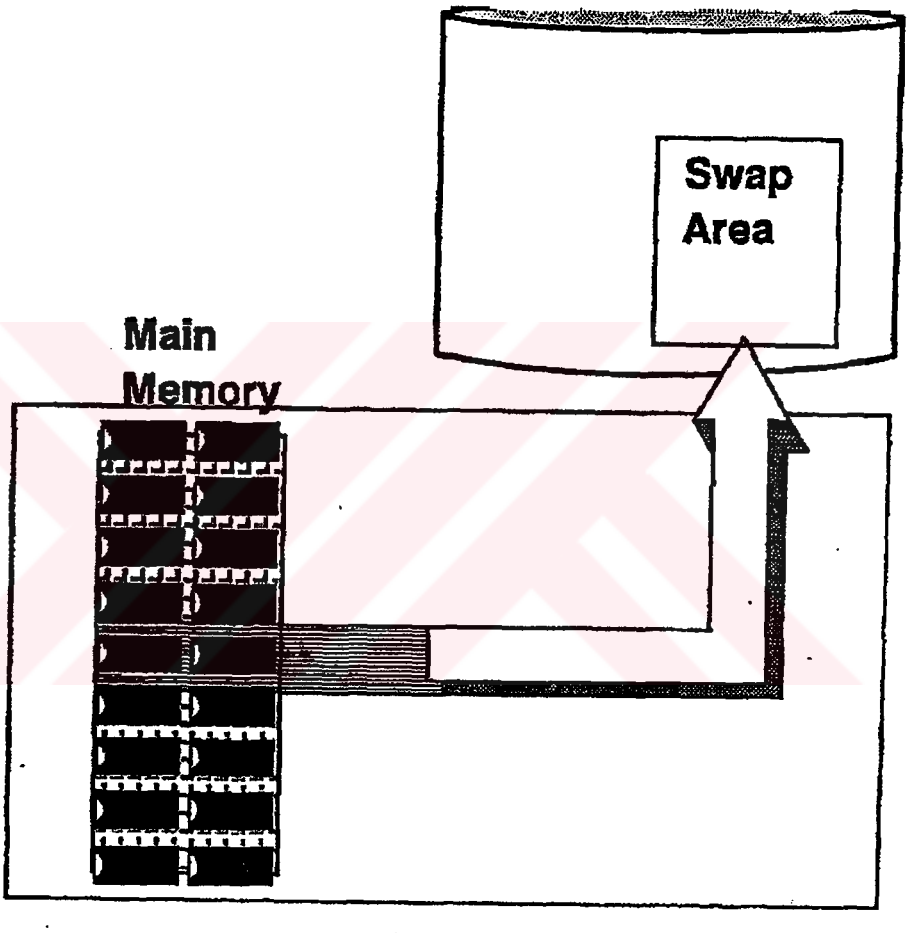
3.2.5. Bellek Kullanımı (UNIX)

Çok kullanıcılı bir sistemde bellek ihtiyacını belirlerken aşağıdaki unsurlar göz önünde bulundurulmalıdır.

- Unix tarafından ihtiyaç duyulan bellek,
- Her veritabanı broker'ı tarafından ihtiyaç duyulan bellek,
- Progress server'ı tarafından ihtiyaç duyulan bellek,
- Her Progress kullanıcısı tarafından ihtiyaç duyulan bellek,
- Tek veya çok kullanıcılı Progress kullananlar için Progress'in bir kopyasının bellekte tutacağı yer,
- Çok kullanıcılı Progress kullananlar için veritabanı server ve broker'ının paylaşımlı bellekte tutacağı yer gözönünde bulundurulur.

Eğer bütün prosesler tarafından kullanılan bellek miktarı sistemin fiziksel bellek sınırlarını aşıyorsa, belleğin taşan kısımları diske kaydırılır. Diskteki bu alana swap alanı denir. Diskteki swap alanı sistem kurukurken sistem yöneticisi tarafından belirlenir. Önerilen swap alanı büyüklüğü şu iki değerin büyük olanıdır: 5 MB ya da (2 * bellek alanı).

Swap alanının yeterli derecede seçilmesi çok önemlidir. Aksi takdirde sistem ölümcül kilitlenmeye (deadlock) uğrayabilir. Bu durum genellikle çok büyük proseslerin aynı anda çalışması esnasında gerçekleşir. Eğer çok sayıda kullanıcı çalışacaksa veya Progress proseslerinin ihtiyaç duyacağı bellek miktarında artışlar



Şekil 3.2.3. Swap Alanının Kullanımı

olacaksa , mümkünse sistemdeki swap alanı büyütülmelidir. Aksi halde sistemin her an için kilitlenmesi söz konusu olabilir.

Swap alanını büyütmek için diskin yedeği alınıp, yeniden formatlanmalıdır.

Kod Paylaşımı:

Progress tarafından desteklenen bütün modüller paylaşılabilirdir. Bunun bir sonucu olarak da, sistemde hiçbir aktif Progress kullanıcısı olmasa bile -progress modülü Unix swap alnında sistem kapanıncaya kadar kalacaktır. Bu faktör bellek kaynaklarının kullanımını daha etkin hale getirecek ve sistemin diske erişim sayısını azaltacaktır.

Bellek Kullanım Opsiyonları:

-B : Veritabanı buffer sayısı.
-c : İndeks kürsörü sayısı.
-e : editör buffer'ı sayısı.
-l : Özel buffer'ların toplam sayısı.
-l : Yerel buffer boyutu.
-L : Kilitleme tablosu girişi.
-n : Kullanıcı sayısı
-Mn : Maksimum server.
-Mxs: Paylaşımlı bellek boyutu.
-O : Kullanıcı başına düşen özel buffer sayısı.
-s : stack boyutu.

Indeks Kürsörleri:

-c opsiyonu prosedürler için gerekli olan indeks kürsör sayısını belirlemektedir. Her aktif indeks kullanan erişim (For each ve find next döngüsü), kullanıldığı prosedür için indeks kürsörüne ihtiyaç duymaktadır. Her kürsör bellekte 64 byte yer tutmaktadır. Kullanılışı aşağıdaki gibidir. proserve [veritabanı] -c n

Kilitleme Tablosu Girişi:

-L opsiyonu kilitleme tablosu girişini ayarlamaya yarar. Belli bir kayıt üzerinde bir kilitleme olduğu zaman (Share- lock veya exclusive-lock) , bu kilitleme tablosunda bir giriş olarak tutulmaktadır. Her kilit girişi bellek paylaşımlı olmayan sistemlerde 14 byte, bellek paylaşımlı sistemlerde 18 byte yer tutmaktadır. Herhangi bir kullanıcı belli bir kilidi ele geçirmeye çalıştığında veya kilit taştığında (overflow) veritabanı

server'ı devreden çıkar. Bu nedenle gerektiğinde;
proserve [veritabanı] -L n .
ile L parametresi artırılmalıdır. Fakat kilitleme tablosunun da belli bir yer kaplayacağı düşünülürse, kilit tablosunun taşmasından kurtulmak için prosedürleri ufak hareketler içerecek şekilde inşa etmek ve kayıtlara sadece okumak için erilşiliyorsa no-lock ile kilit koymadan erişmekte fayda vardır.

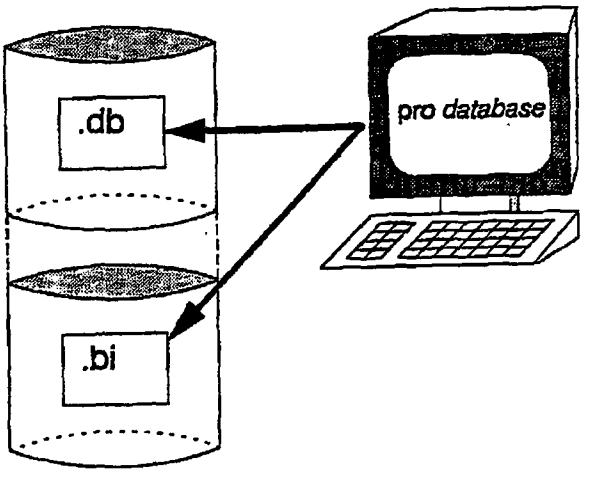
Paylaşımlı Bellek Boyutu (-Mxs):

Bu parametre paylaşımlı bellek taşma alanını büyütmek için kullanılır. Yalnızca Unix paylaşımlı bellek sistemlerinde geçerlidir. Eğer taşma alanı küçük geliyorsa Progress hata mesajı ile programdan çıkar. Kullanımı aşağıdaki gibidir.
pro -Mxs Kbyte .

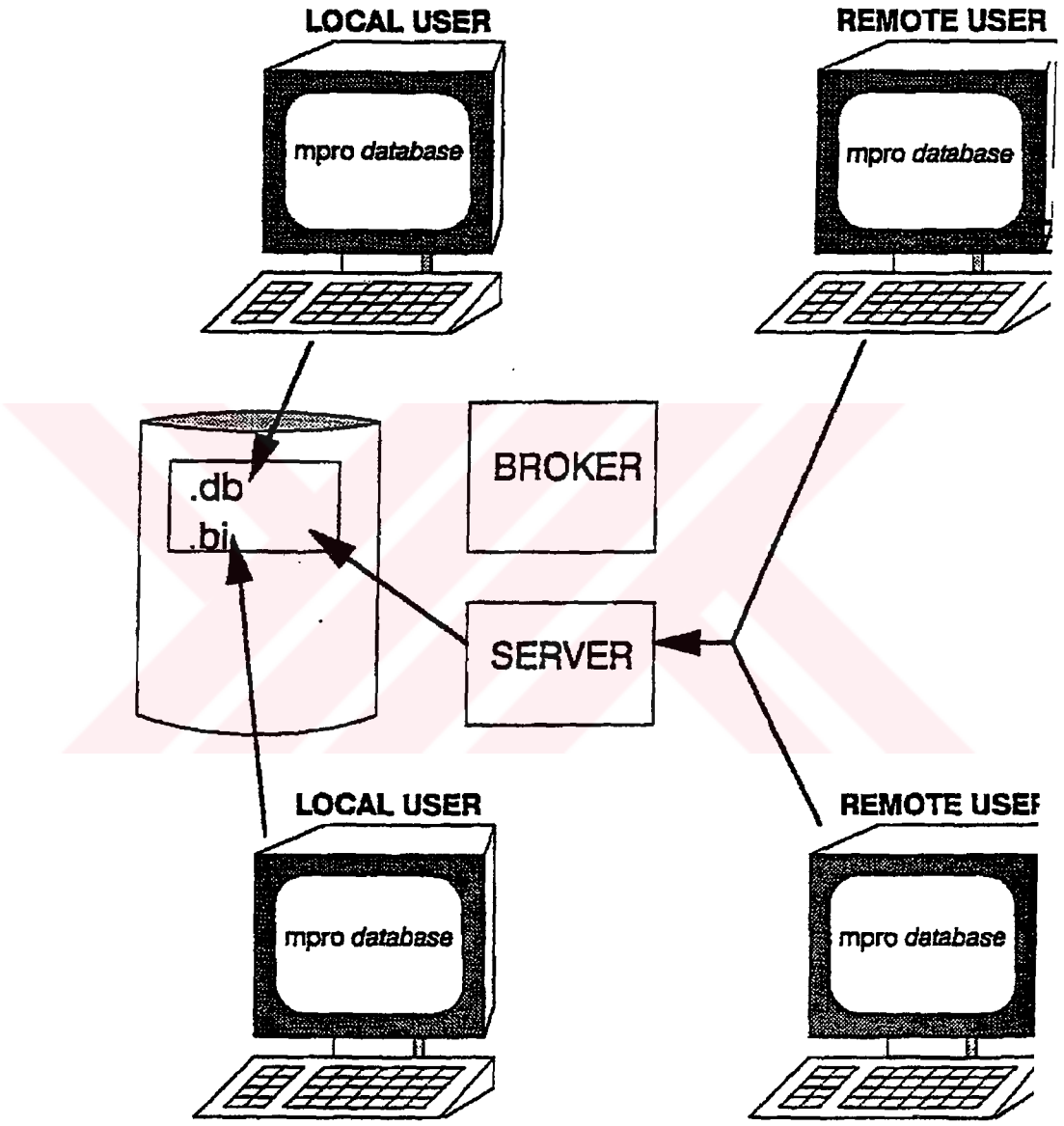
3.3. Progress Mimarisi

3.3.1. Tek Kullanıcılı Bir veritabanı mimarisi:

Tek kullanıcılı bir veritabanı komut satırından "pro [veritabanı]" şeklinde başlatılır. "pro fatura" gibi. fatura.db fatura veritabanını, fatura.bi before image dosyasını ifade eder. Son derece basit bir yapıdır. Aynı anda yalnız tek bir kullanıcı veritabanını kullanabilir. Veritabanını tek kullanıcılı olarak açan komut "pro"'dur. DOS işletim sisteminde multitasking özelliği olmadığı için bu şekilde kullanmak



Şekil 3.3.1. Tek Kullanıcılı Mimari



şekil 3.3.2. Çok Kullanıcılı Mimari

zorunluluktur. Böyle bir kullanımda dağıtıklık söz konusu olamaz. Çünkü aynı anda bu veritabanına başka bir kullanıcının bağlanması mümkün değildir[7].

3.3.2.Çok Kullanıcılı Veritabanı Mimarisi:

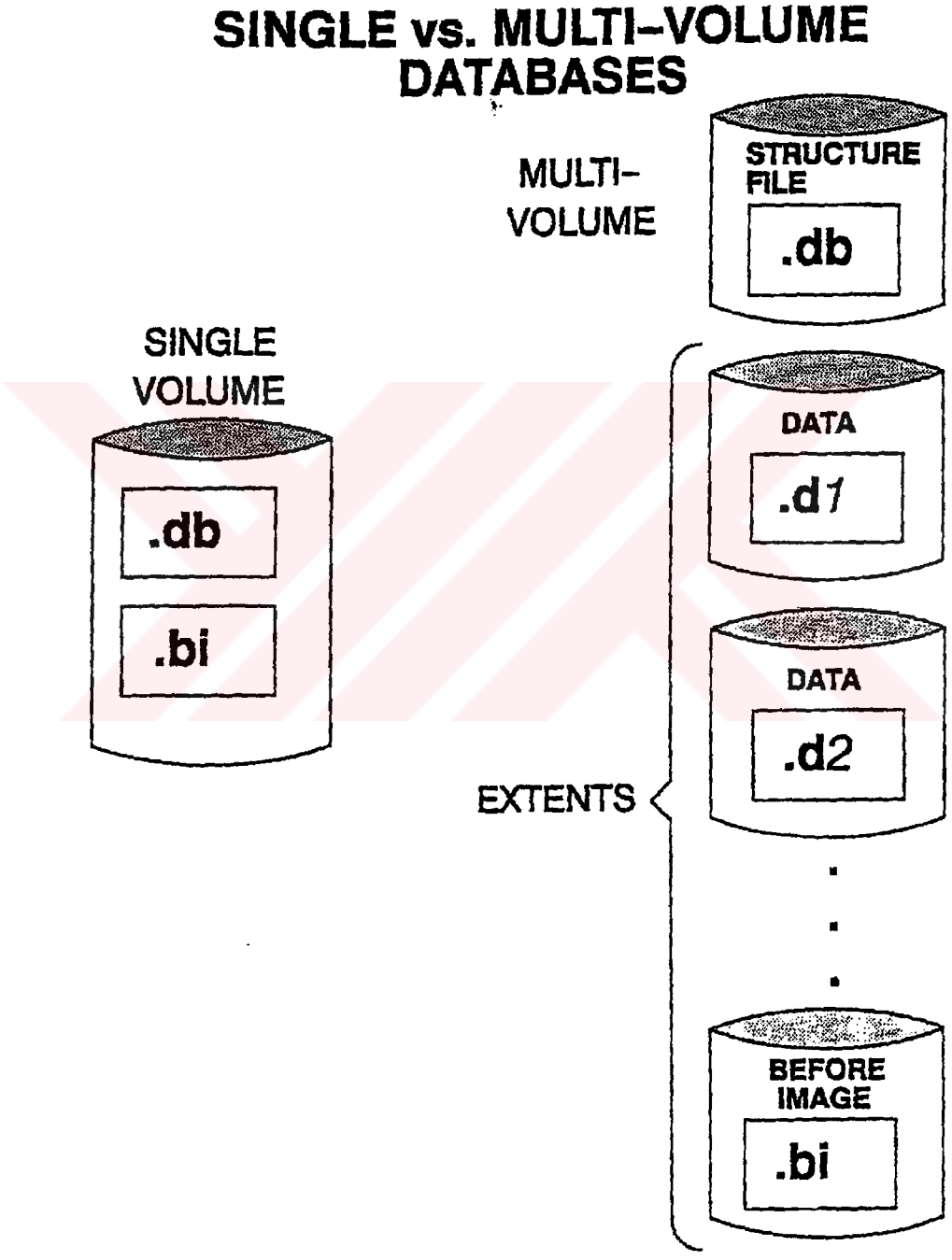
Paylaşımlı belleğe sahip bir Unix makinesi üzerinde kullanılabilir. Böyle bir sistemde yerel kullanıcılar (yani veritabanının bulunduğu makina üzerindeki kullanıcılar) self servis kullanıcılarıdır. Ağ üzerindeki başka bir makinedeki uzak kullanıcılar ise veritabanına anacak bir server prosesi ile erişebilir. Broker proses uzak kullanıcılar için gerektiğnde server prosesi başlatır. Yerel kullanıcıların (Local client) ise veritabanına erişim için server prosesi kullanmaya gereksinimleri yoktur. paylaşımlı bellek ve yerel buffer alanlarını kullanarak veritabanına erişimi sağlayabilirler. Yerel buffer alanları veritabanında okuma yapıldığı zaman kullanılır. Broker denilen proses ise paylaşımlı bellek segmentlerini idare eder.

Çok kullanıcılı olarak veritabanını başlatan komut "proserve [veritabanı] diğer opsiyonlar" şeklindedir. Eğer veritabanını yalnızca yerel kullanıcılar kullanacak ise herhangi bir servis kullanmaya gerek yoktur. Ancak veritabanına uzak kullanıcılar da erişim yapacaksa Üye ve servis ismi ve birden fazla ağ protokolü kullanılıyorsa protokol ismini de belirtilmelidir. Aksi takdirde uzak kullanıcılar ağ üzerinde ilgili servis olmadığı için bu veritabanına erişemezler."proserve" server prosesdir. Yerel bir Client "mpro [veritabanı] diğer opsiyonlar" şeklinde veritabanına erişimi sağlarlar. Ancak uzak kullanıcıların erişim yaparken servis, erişilecek veritabanının bulunduğu makinenin üye (host) ismini , gerekliyse protokol ismini verme zorunluluğu vardır. "mpro" client prosesdir.

3.3.3. Tek ve Çok Parçalı Veritabanları:

Progress'de bir veritabanı normalde tek bir bütün halinde saklanır. Her veritabanının bir tek before-image dosyası vardır.

Fakat veritabanı farklı parçalar halinde belli bir makine veya ağ üzerinde saklanmak (NFS vs. kullanılıyorsa) isteniyorsa buna da imkan sağlanmaktadır. 100 farklı dosya sistemi veya disk biriminde veritabanı tutulabilir. Tek parça bir veritabanın ulaşabileceği maksimum büyüklük dosya sistemi boyutu ile sınırlıdır. Oysa bu şekilde 100 dosya sistemi varsa 100 dosya sisteminin toplam boyutu kadar bir hacme ulaşılabilir. Veritabanının kaç parça halinde oluşacağına kullanıcı karar verir. Fakat her parçada hangi tablo ve kayıtların yer alacağına Progress karar verir. Kullanıcı bunu tayin edemez. Tek veya birden fazla before image dosyası olabilir.



Şekil 3.3.3 Tek ve Çok Parçalı Veritabanları

3.3.4.Terminoloji (Paylaşımlı Bellek Ortamında-Unix Progress):

Broker: Çok kullanıcılı veritabanı için ana veya kontrol prosesidir. Broker Proses paylaşımlı bellek üzerinde yer tahsisini idare eder.

Client: Progess veritabanı kullanıcı istemidir(session).

Yerel Client: Broker'ın çalıştığı makine üzerinde çalışan çok kullanıcılı Progress istemidir. Yerel client'lar veritabanına direk paylaşımlı belleği kullanarak erişirler. Server'a veritabanına erişim için istekte bulunmazlar. Progress "mpro" komutu ile başlatılırken bir Client proses başlatılmış olunur.

Multi Threaded (Çoklu İplikli): Paylaşımlı bellek sistemleri üzerinde (Unix) veritabanı erişimi çoklu-iplikli olarak adlandırılır. Çünkü birden fazla yerel veya ağ üzerindeki uzak Client'lar paylaşımlı belleği kullanarak veritabanına eşanlı erişimi sağlar. Progress'in eski versiyonlarında yer alan tek iplikli modelde veritabanıyla ilgili bütün istekler bir kanalda tek bir veritabanı prosesi için dizilirler.

Uzak Client: Veritabanının olmadığı uzak bir ağ düğümündeki client prosestir. "mpro" ile beraber "-H üye -S server ismi" kullanılması gerekmektedir. Uzak kullanıcılar veritabanına direkt oarak erişemezler; broker'ın başlattığı anda uzak cient-server vasıtasıyla erişimi sağlarlar.

Uzak Client-Server: Bir veya daha fazla uzak client'a sunum yapan server prosestir. Broker uzak Client-server'ları, uzak client olarak başlatır. Uzak client-server'lar, uzak client'lar adına paylaşımlı kaynaklara (veritabanı buffer'ı, kilitler gibi) erişirler. Server'lar üye bilgisayar üzerinde işlem yapar ve DECnet veya soket kullanarak uzak client'larla haberleşirler.

Semafor: Unix işletim sistemi altında paylaşımlı kaynaklara çoklu erişimleri koordine etmek için kullanılan bir çeşit kilittir. Broker, her yerel client ve her uzak client-server bir semafora ihtiyaç duyar.

Server: Progress'in bellek paylaşımını destekleyen versiyonları için uzak client-server tanımı geçerlidir. Eğer bellek paylaşımını desteklemeyen versiyonu ise burada esas veritabanı server'ıdır.

Paylaşımlı Bellek: Ana Bellek alanındaki segmentlerin çoklu kullanıcılar tarafından eşanlı olarak erişilmesidir. Progress veritabanı buffer'ı, kilitleme tablosu, index kürsör havuzunu ve diğer kaynakları paylaşımlı bellek içerisinde yer alan bütün veritabanı kullanıcıları tarafından ortak kullanılmasını sağlamaktadır. Bu kaynaklara erişim kilitlemeler ve semaforlar tarafından kontrol edilir. Broker, yerel

client'lar ve uzak client-server'ların hepsi paylaşımlı belleğe erişebilirler, yani veritabanına direkt erişirler.

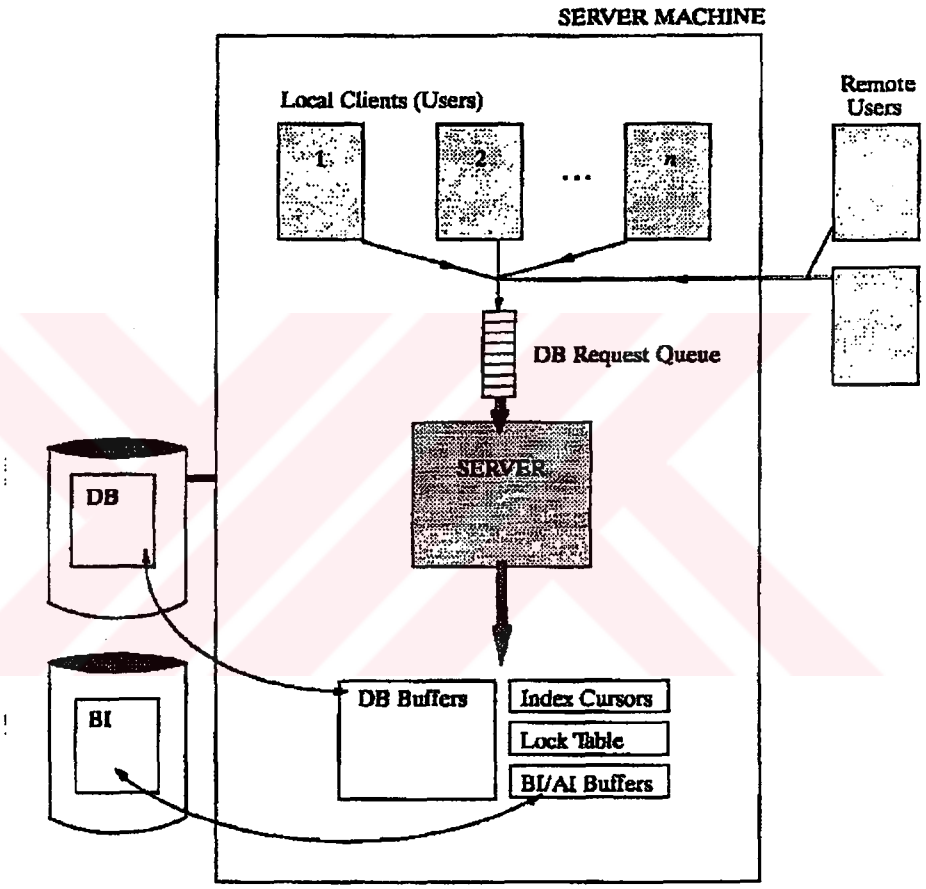
Soket: TCP/IP için geçerli kavramdır. Ara proses haberleşmesi son noktası olarak görev görürler. Soketler haberleşme içerisindeki farklı uygulama modellerini gerçekleştirmede kullanılabilirler: datagram gönderimi, sanal devreler (virtual circuit) ve sıralı paket değişimi gibi. Progress, TCP/IP ağı üzerindeki makineler arası proseslerdeki veri akışı gibi bir boru (pipe) uygulamasını gerçekleştirmek üzere Unix domain soketlerini kullanır.

3.3.5. Çoklu İplikli / Çoklu Server Mimarisi

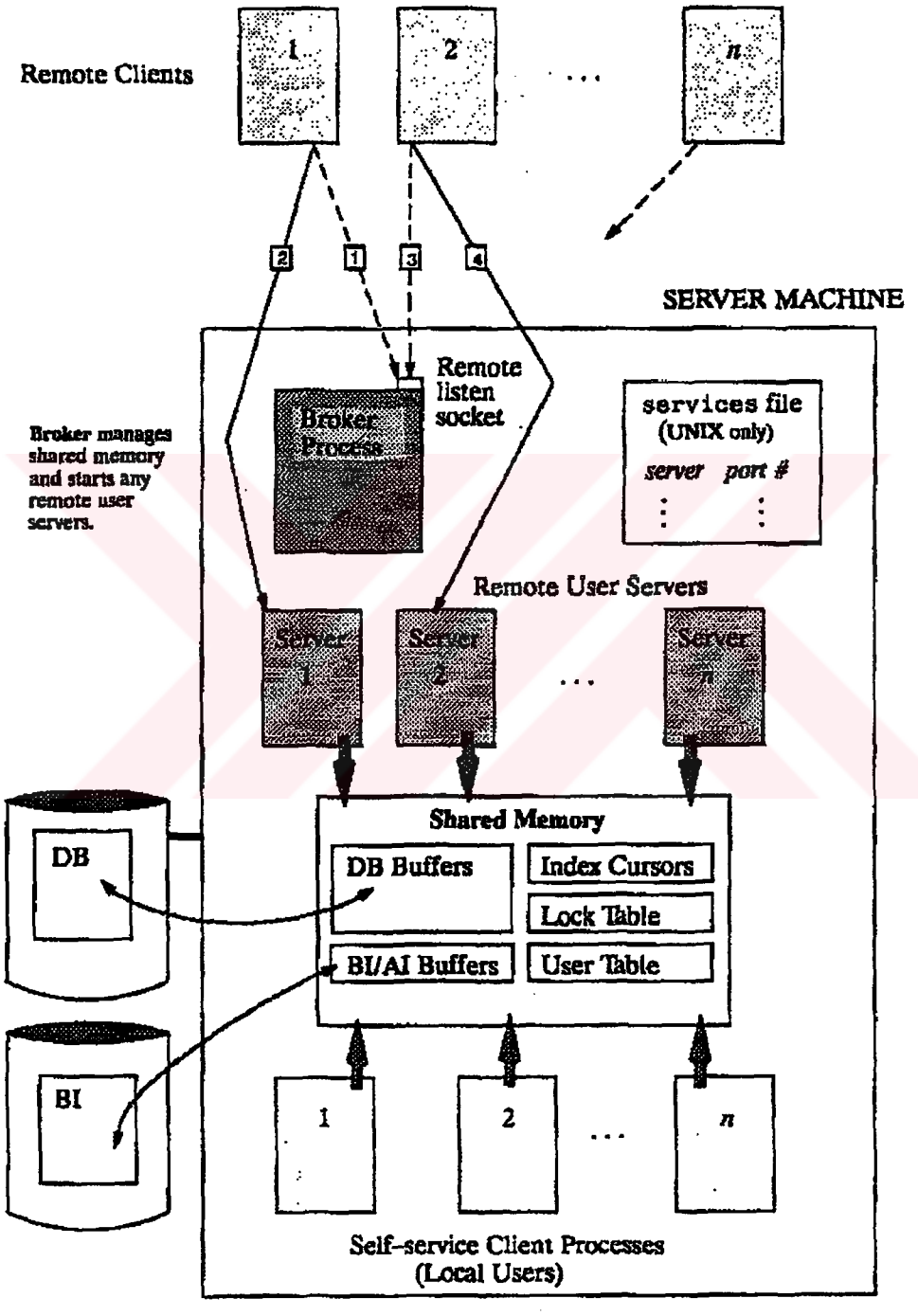
Paylaşımlı bellek, yerel client'lara, veritabanı server'ına ve dolayısıyla veritabanının kendisine direkt erişime izin verir. Ayrıca çoklu server'lar ve self servis yerel client'lar veritabanına aynı zamanda erişebilirler. Bütün veritabanı kullanıcıları tarafından ihtiyaç duyulan kaynaklar: veritabanı buffer'ı (-B), kilitleme tablosu (-L), index kürsörü (-c) paylaşımlı bellek içerisinde yer alır ve broker tarafından tahsis edilip, paylaştırılır. Aşağıdaki şekil paylaşımlı belleği kullanan yerel client'ların çoklu iplikli mimarisinde nasıl erişim yaptıklarını göstermektedir.

Broker: Her veritabanı bir broker prosese sahiptir. Unix altında bu "proserve" komutunu çalıştırmakla otomatik olarak başlar. Broker proses paylaşımlı belleği, semaforları, paylaşımlı buffer havuzunu, kilitleme tablosunu ve bütün ortak kaynakların paylaşımını tahsis eder. Broker paylaşımlı belleği temizleyerek başlatır ve normal şartlarda paylaşımlı belleği silen tek prosestir. Unix üzerinde, eğer broker proses "shutdown" komutu kullanılmadan sona erdirilirse, paylaşımlı bellek segmentlerini temizlemez. "proutil -C dbipcs" komutu ile paylaşımlı belleğin dondurulmuş olup olmadığı anlaşılır. "ipcrm -m" Unix komutu ile de paylaşımlı bellek serbest bırakılır. Ağ üzerinde broker, uzak Client bağlantılarını dinler. Mevcut client-server'lara uzak client'ları bağlar ya da ihtiyaç duyduğunda yeni server'ları başlatır. Broker aynı zamanda veritabanı kapatıldığında uzak client prosesleri de durdurur.

Client (İstemci): Client'lar bir kullanıcı tarafından çalıştırılan Progress "mpro" komutu ile isteme başlarlar. Aşağıdaki şekilde de görüldüğü gibi ağ üzerinde yer alan client'lara eklenirler. Tekrar hatırlanması gereken nokta yerel client'ların paylaşımlı belleğe direkt eriştikleri; fakat uzak client'ların direkt erişemedikleridir.



Şekil 3.3.4. Tek İplikli Mimari



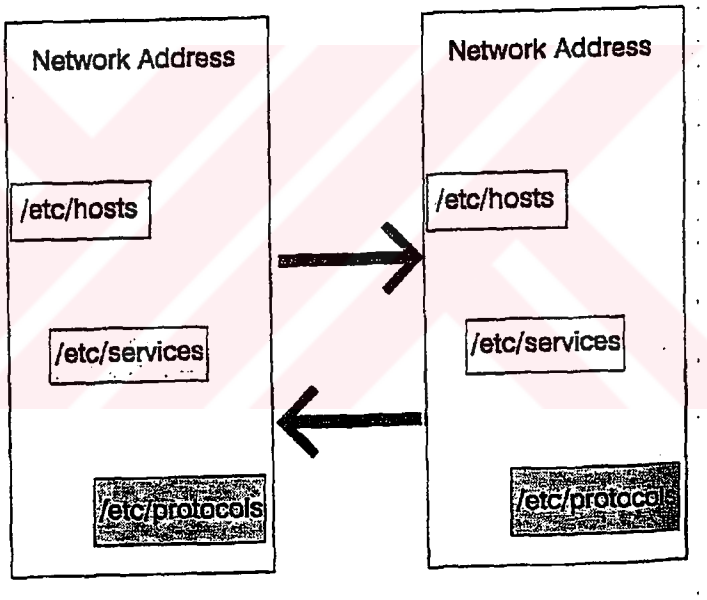
Şekil 3.3.5. Çok İplikli Mimari

3.4. Progress ve Network

Progress Unix makineler ile bir bilgisayar ağı üzerinde kullanıldığında ihtiyaç duyulan ağ TCP/IP'dir.

TCP/IP , ilişkisiz prosesler arasında bilgiye geçirmek için soketleri kullanır. Bu prosesler farklı olan makinelerde olabilir. Soketler Unix'deki pipe'lara benzetilebilir. Aralarındaki fark pipe'ların iki ilişkili proses arasında kullanılması (ebeviyen ve çocuk) ve dolayısıyla aynı makine üzerinde olmasıdır. Soketlerde proseslerin ilişkili olması ve aynı makine üzerinde yer alması zorunlu değildir.

Aşağıda TCP/IP ağı ve Unix makineleri arasındaki ilişki gösterilmektedir. Projede kullanılan her makinede aşağıdaki konfigürasyon söz konusudur.



Şekil 3.4.1.TCP/IP Network'ü üzerindeki Unix Makineleri

TCP/IP üzerindeki her Unix makinesi:

i) Bir ağ adresine sahip olmalıdır. Ağ adresi TCP/IP bölümünde de belirtildiği gibi ağ üzerindeki network ve makineyi belirleyen nümerik değerdir. Ağ üzerindeki her makina birbirleri arasındaki haberleşmeyi kolaylaştırmak için böyle bir adrese sahip olmalıdır.

ii) /etc/hosts dosyası yer almalıdır. Bu dosya ağ üzerindeki tüm makina isimleri ve ağ adreslerini içerir. Bu dosya ağ üzerindeki her makina üzerinde yer almalıdır.

iii) /etc/services dosyası, ağ üzerindeki beli bir prosese isim, port numarası ve haberleşme protokolü atandığı yerdir.

iv) /etc/protocols dosyası sistem üzerindeki bütün protokollerin tanımlı olduğu yerdir.

TCP/IP üzerindeki Progress yapısı üç muhtemel senaryo şeklinde olabilir.

- Her Unix makinenin kendi sabit diski var ise Network File System (NFS) opsiyonel olarak kullanılabilir.

- Unix makinelerden oluşmuş bir bilgisayar ağı mevcut ve en az bir bilgisayar sabit diske sahip ise NFS kullanılabilir.

- Unix,Xenix ve/veya MS-DOS makinelerden oluşmuş ve NFS kullanmayan bir TCP/IP ağı. Her makine kendi sabit diskine sahiptir.

Projede kullanılan sistem ilk senaryoya uygunluk göstermektedir. NFS istenirse kullanılabilir, fakat gerekli değildir.

Böyle bir yapıda her makinede Progress'in yer almasına gereksinim yoktur. Fakat yer alması performansı artırıcı bir etkendir. Eğer yoksa NFS bir ihtiyaç olarak doğmaktadır. Projede kullanılan her makinede Progress yüklüdür. Genel Müdürlük'teki (Merkez) makinede programlar burada geliştirildiği için Komple Progress paketi diğer makinelerde sadece programlar çalıştırıldığı için Run-Time modülü mevcuttur.

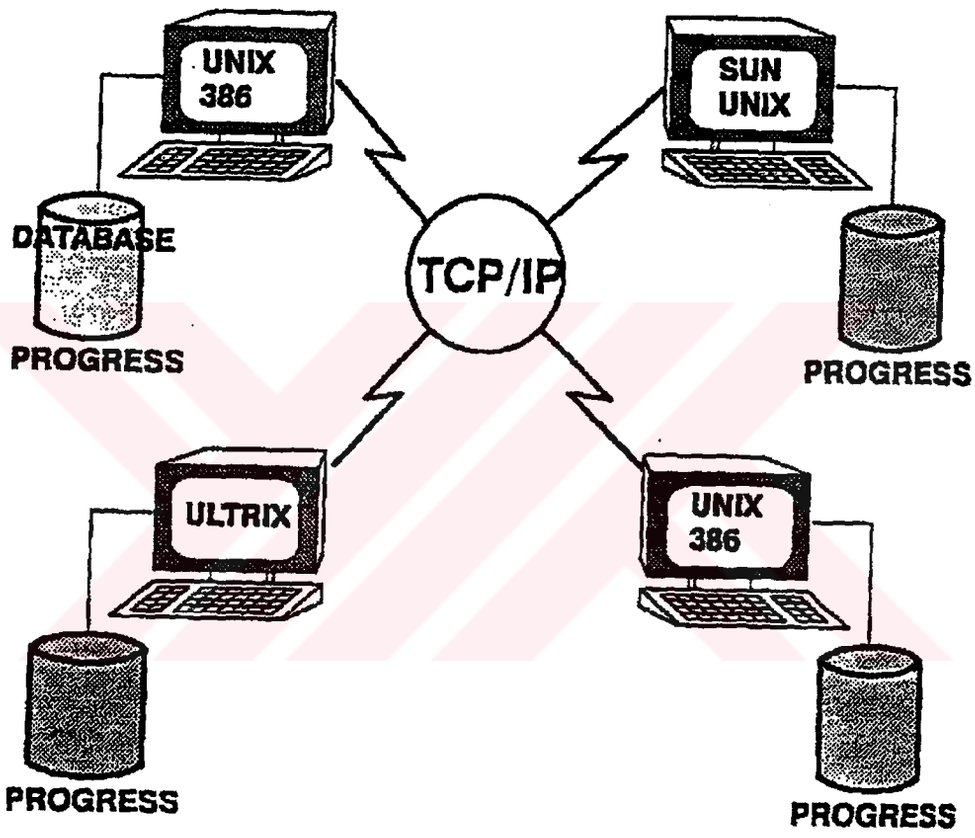
Genel olarak benzeri bir yapıda her makina tipi için en az bir makinede Progress yüklü olmalıdır.

3.4.1. /etc/hosts:

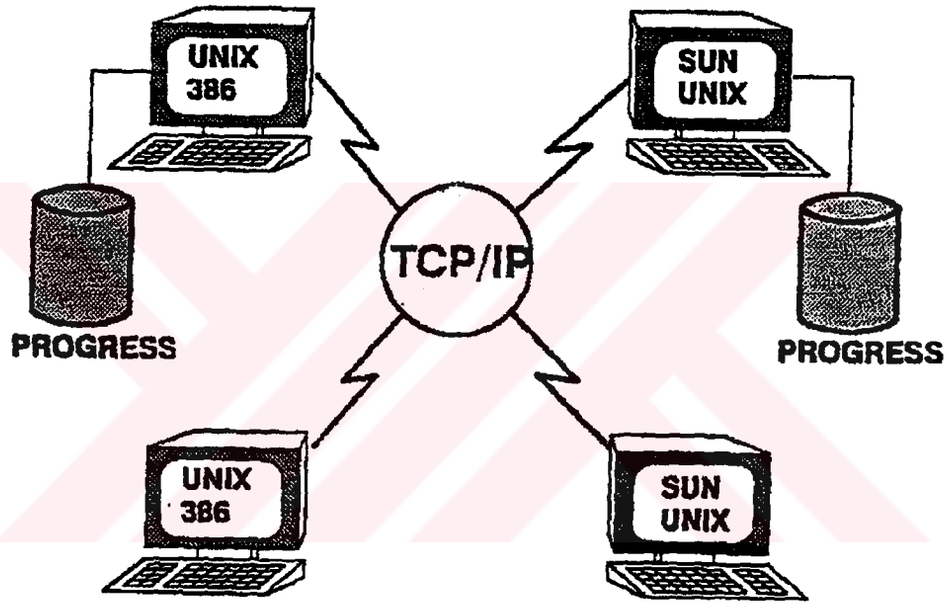
Bu dosya ağ adreslerini, her makineye verilen isimleri ve bu isimler için eş makine ismini ifade eden kısaltma isimlerini kapsar. Bu dosya ağ üzerindeki her makinede yer almalıdır. Sistemdeki her makine için makine adları ve adresleri mevcut olmalıdır.

Örneğin Antalya makinesi için "93.0.0 .4 " ağ adresini, "antalya" makine ismi, "ant" ise kısaltma ikincil makine ismini ifade etmektedir.

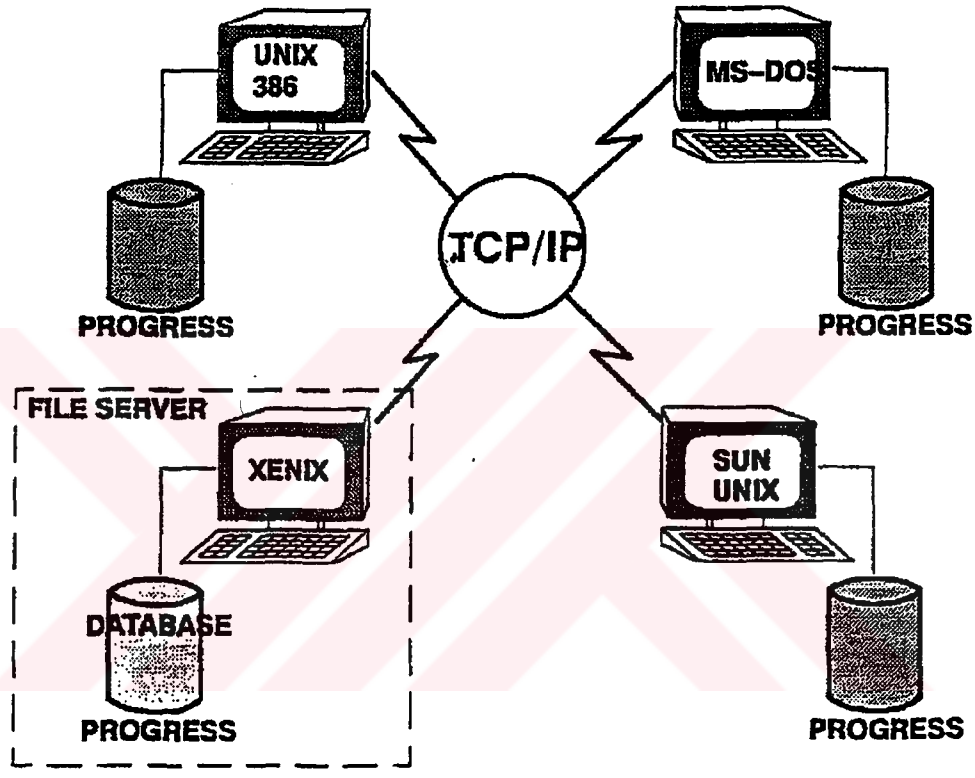
Progress server'nın çalıştığı her makinanın adı /etc/hosts içinde yer almalıdır.



Şekil 3.4.2. NFS'in isteğe bağlı olduğu TCP/IP Senaryosu



Şekil 3.4.3. NFS'in Gerekli Olduğu TCP/IP senaryosu



Şekil 3.4.4. NFS'in olmadığı TCP/IP senaryosu

Sistem Üzerindeki /etc/hosts dosyası:

```
#ident    "@(#)cmd-inet:etc/hosts
   1.1.2.1" # # Internet host table # 127.0.0.1
          localhost       loopback
93.0.0.2   merkez         mer
93.0.0.1   istanbul       ist
93.0.0.3   izmir          izm
93.0.0.4   antalya        ant
93.0.0.5   ankara         ank
93.0.0.6   dalaman        dal
```

3.4.2. /etc/services ve /etc/protocols:

Tez Konusu Projede Kullanılan Servisler:

```
istser   2101/tcp        # Progress İstanbul Server
merser   2102/tcp        # Progress Merkez Server
izmser   2103/tcp        # Progress izmir Server
antser   2104/tcp        # Pogress Antalya Server
ankser   2105/tcp        # Progress Ankara Server
dalser   2106/tcp        # Progress Dalaman Server
```

Bu dosya farklı sistem servis prosesleri hakkında bilgileri saklamaktadır. Her veritabanı için çalışan server, prosesleri isimlendirir. Aynı zamanda bunlar için bir port numarası ve her server için tcp haberleşme protokolü belirler.

Yukarıda tez konusu proje için kullanılan sistem üzerindeki /etc/services dosyası içinde yer alan servis isimleri gözükmektedir. Bu dosya her Progress server'ı için bir satıra gereksinim duyar. Bu satırlarda:

- Server ismi: Server veya Broker prosesi için seçilen isimdir. Her server farklı bir isme sahip olmalıdır.

- Port Numarası: 2000 - 2999 sayıları arasında herhangi bir port numarası kullanılabilir. (Uygulama sırasında server gerekirse uzak client'lara 1000 - 1999 arası port numaraları tahsis edebilir.)

- tcp : Progress devamlı tcp protokolünü kullanmaktadır.

Sistem üzerindeki her makinada bu dosyanın bir kopyası olmalıdır.

/etc/protocols:

Bu dosya internet içerisimde kullanılan, bilinen protokoller hakkında bilgi içerir. Her protokol için bir satır vardır. Her satır isim, numara, ikincil isim ve protokol için açıklama içerir.

TCP/IP Ağı Üzerinde Bir Progress Server'ını başlatmak:

proserve [veritabanı ismi] -S [servis ismi] -H [host(üye) ismi] -N [Protokol İsmi]

Örnek: proserve fatura -H ank -S ankser -N TCP Bu Ankara makinesi üzerindeki fatura veritabanını ankser servisi ile başlatmaktadır.

Böyle bir servisi başlatmak için Ankara makinesi üzerinde olmak gereklidir. Burada:

-H : Üye ismini belirtmektedir. Ankara'nın ikincil ismi kullanılmıştır.

-S : ankser adındaki servis ismini belirtmektedir.

-N : Hangi protokolün kullanıldığını belirtmektedir.

Unix üzerinde hep TCP kullanılmaktadır. Proserve komutu, çok kullanıcılı progresi çalıştırmadan evvel server prosesi başlatmak için kullanılır. -S opsiyonu, /etc/services dosyasında yer alan bir server ismini belirler. Eğer böyle bir server ismi verilmezse TCP/IP üzerindeki uzak bir Client ilgili Server'ı kullanamaz, server prosesin çalıştığı makinedeki veritabanına bağlantıyı gerçekleştiremez.

Yerel olarak çok kullanıcılı başlatılan Ankara server'ı için "mpro ank" komutu fatur veritabanını çok kullanıcılı olarak yeterli bir komuttur.

Ancak uzak bir makinadan örneğin İstanbul makinesinden bağlantı kurmak için: "mpro [veritabanı ismi] -H [host(üye)] -S [servis ismi] -N [protokol ismi] gerekliyse diğer opsiyonlar" yapısı kullanılır. Örnek: mpro fatura -H ank -S ankser -N TCP . İstanbul'daki kullanıcı yukarıdaki şekilde Ankara fatura veritabanına uzak bağlantı yapılabilir.

Server'ı Kapatmak:

"proshut [veritabanı ismi] gerekliyse diğer opsiyonlar" komutuyla veritabanıyla iligli server proses durdurulur. Örneğin: "proshut fatura" ile ilgili makine üzerindeki fatura veritabanını idare eden broker proses sona erdirilir.

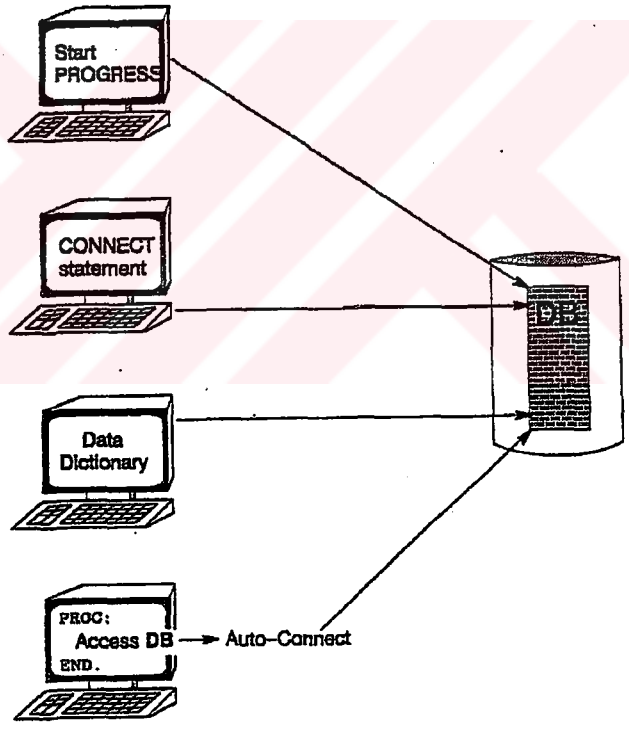
3.5 .Çoklu Veritabanları ve Bağlantıları

Özellikle dağıtık veritabanı organizasyonlarında aynı anda birden veritabanı üzerinde işlem yapmak gereklidir. Bu da veritabanı bağlantıları ile gerçekleşmektedir.

Progress tek bir uygulama programı içerisinden birden fazla veritabanına bağlanma ve onlar üzerinde işlem yapma imkanı tanımaktadır. Bu özellik dağıtıklığı gerçekleştiren temel unsurdur.

Bu özellik sayesinde:

Farklı veritabanlarının birleştirilmesi ile oluşan raporlar elde edilebilir, - Aynı anda birden fazla veritabanını güncelleyen prosesler yazılabilir, - Aynı anda



Şekil 3.5.1. Veritabanı Bağlantı Metodları

Progress, Oracle, RMS veritabanlarına erişilebilir.

Belli bir veritabanına bağlanma dört şekilde gerçekleşebilir.

i) Progress'i başlatırken, ii) "Connect" ifadesi ile, iii) Veri sözlüğü tarafından, iv) Prosedür içerisinden otomatik bağlantı ile . [4]

En önemli bağlanma parametresi "-db veritabanı ismi" parametresidir. Bu parametre ile başlama programı veya komut satırındaki Progress'i başlatan komut-la beraber birden fazla veritabanına bağlanabilir.

Standard olarak bir Progress uygulaması içerisinden en fazla 5 veritabanına bağlantı gerçekleştirilebilir. Fakat "-h" maksimum veritabanı bağlantı parametresi kullanılarak bu sayı 240'a kadar çıkarılabilir. Bağlanma parametrelari daha evvel de bahsedildiği gibi .pf parametre dosyaları içerisinden tanımlanabilir.

Yukarıda bahsedilen bağlanma şekilleri tek tek incelenirse:

3.5.1. Bağlantı Çeşitleri

i.) Progress'i Başlatırken Bağlanmak:

mpro veriabanı1 -db
veritabanı2 -db
veritabanı3 ..... şeklinde veya bir parametre dosyası kullanılarak

mpro -pf <parametre dosyası>

parametre dosyası .pf:

-db veritabanı1 -db veritabanı2 -db veritabanı3

ile sağlanabilir. Burada mpro yerine pro, bpro veya mbpro kullanılabilir.

ii) Bir prosedür içerisinden "connect" ifadesi ile bağlanma:

Connect Fiziksel veritabanı ismi [opsiyonlar] ..[NO-ERROR].
parametre dosyası [opsiyonlar]

Connect ifadesi bir veritabanına bir Progress prosedürü veya Progress editörü içerisinden bağlantı kurulması imkanı sağlar. Fiziksel isim, veritabanının disk üzerindeki esas ismidir. Connect ifadesi ile tek bir veritabanına bağlantı sağlamakta fayda vardır. Çünkü birden fazla veritabanına tek bir connect ile bağlantı sağlanmak istendiğinde, veritabanlarından biri ile bağlantı gerçekleştirilemezse ifade başarısızlığa uğrayacak ve dolayısıyla boş yere diğer veritabanı bağlantıları gerçekleştirilemeyecektir.

No-error opsiyonu ile hata durumunun pas geçilmesi sağlanır. Fakat bir hata oluştuğunda ilgili hata mesajları ekranda gözükür. Bir veritabanına bağlantının

gerçekleşip gerçekleşmediğinin tespitinde "Connected" fonksiyonu kullanılır. Buna özellikle no-error opsiyonu kullanıldığı zaman gereksinim duyulur.

Örnek:

```
connect fatura. connect istfat.
connect -pf parmfil1.p.

parmfil1.p:
-db ankfat.
```

Connect kullanılırken dikkat edilmesi gereken bir başka husus da veritabanını içeren kısımların ayrı bir prosedür içerisinde tutulmasıdır.

Aşağıdaki program çalışmamaktadır:

```
connect fatura.

For each fatura.ucak:
    display ucak.
end.
```

Şu şekilde olması gerekmektedir.

```
üstprogram.p:
connect fatura.
run altprog.p.

altprog.p:
For each fatura.ucak:
     display ucak.
end.
```

Yukarıda bahsedilen no-error opsiyonu kullanıldığında Progress belli bir veritabanına bağlantı yapıldığını bilmemektedir. İşte bu durumlarda "connected" fonksiyonunu kullanmakta fayda vardır.

Örnek:

```
if not connected(izmfat)  then do:
   message "İzmir'e bağlanılıyor.".
  connect fatura -H izm -S izmser -ld adbfat no-error.
 end.
```

```
if connected(adbfat)  then run altprog.p.
else message "İzmir'le bağlantı gerçekleştirilemiyor.".
```

Görüldüğü gibi connected fonksiyonu veritabanı bağlantılarının gerçekleşip gerçekleşmediğini anlamakta çok önem taşımaktadır.

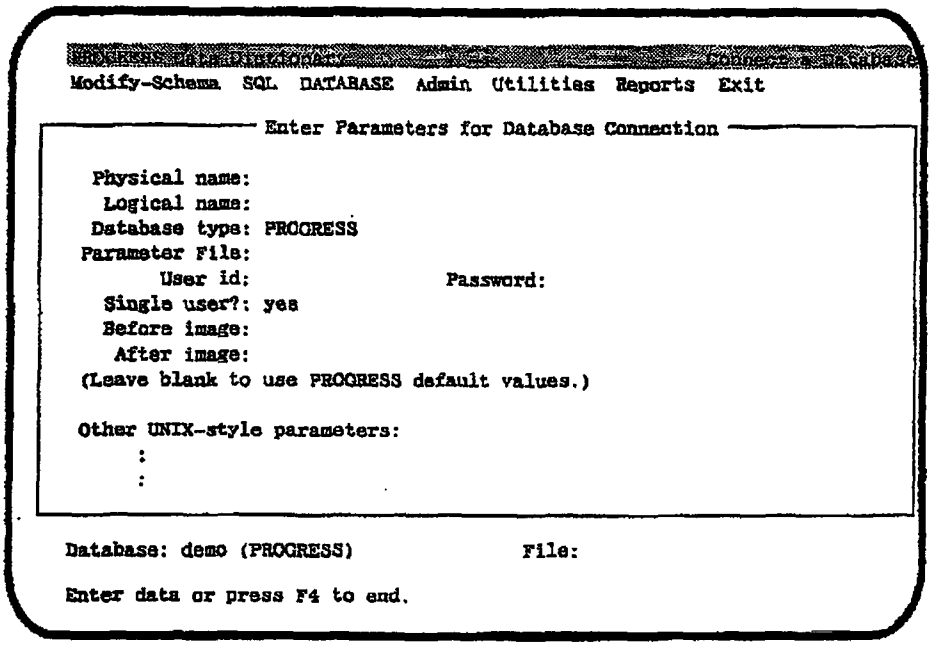
Progress Fonksiyonu	Açıklaması
Connected	Belli bir veritabanının bağlı olup olmadığını test eder.
DBtype	Bağlanılan veritabanı tipini belirler. ("Progress, Oracle,RMS vs)
Dbrestrictions	Bağlanılmış olan veritabanı için Progress tarafından desteklenmeyen komutların listesini veren bir karakter katarı gönderir. Örneğin Oracle veritabanına bağlantı kurulmuşsa "Last, Prev, Recid, Setuserid, use-index" katarı geri dönmektedir.
Gateways	Mevcut Progress versiyonunun desteklediği veritabanlarını bir karakter katarı ile içerisinde göstermektedir. Örnek: "Progress, RMS, Oracle".
Num-dbs	Bağlantı kurulan veritabanı sayısını vermektedir.
Ldbname	Aktüel olarak veritabanının mantıksal ismini vermektedir.
pdbname	Bağlanılan veritabanının fiziksel ismini vermektedir.
sdbname	Belli bir veritabanı için şema bilgilerini tutan veritabanının ismini verir.

iii) Veritabanı sözlüğünden yapılan bağlantı:

Aşağıda görülen Progress sözlüğü içindeki "database" bölümünde yer alan connect opsiyonlu pencere içerisinden yapılır.

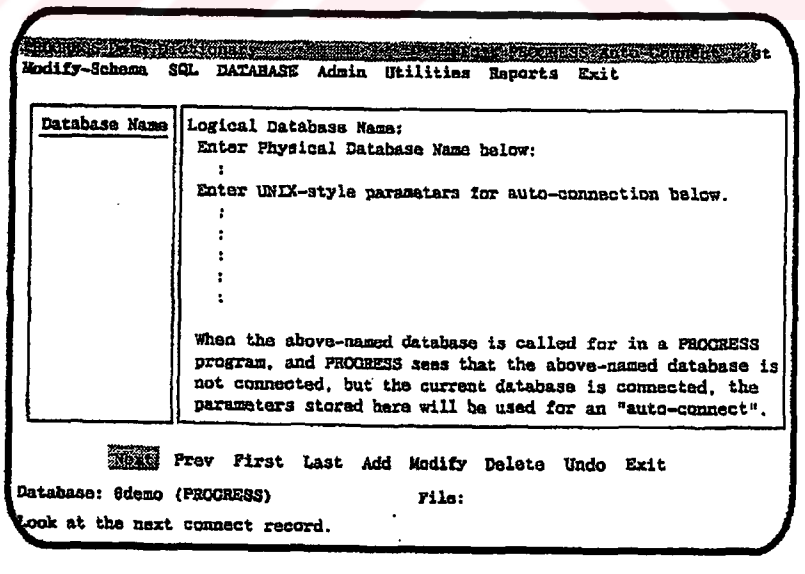
iv) Otomatik Bağlantı:

Bu özellik, belli bir veritabanı içindeki otomatik bağkantı bilgilerini kullanarak, belli bir program içerisinde ikinci bir veritabanı bilgisine rastlandığında o veritabanıyla bağlantı sağlanmadıysa otomatik olarak bağlantıyı gerçekleştirir. Yalnız bu veritabanın veritabanı sözlüğü içerisindeki auto-connect listesi içerisinde yer alması gereklidir.



Şekil 3.5.2. Veritabanı Sözlüğünden Bağlantı

Belli bir veritabanına daha önce auto-connect ile bağlanılır ve sonra tekrar connect ifadesi ile bağlantı gerçekleştirilmeye çalışılır ise connect ve auto-connectdeki bağlantı bilgileri birleşerek bağlantı gerçekleştirilir. Bu durumda connect ifadesindeki bilgiler önceliğe sahiptir.



Şekil 3.5.3. Otomatik Yapılan Bağlantı

3.5.2. Genel Veritabanı Bağlantı Kavramları:

Mantıksal Veritabanı İsimleri:

Mantıksal veritabanı ismi bağlanılan fiziksel veritabanı ismini temsil eder. Bu isim hangi veritabanı ile bağlantı kurulduğunun ayrıştırılmasında faydalı olmaktadır. özellikle dağıtık bir sistemde hangi veritabanının nereye ait olduğunun anlaşılmasında faydalı olmaktadır.

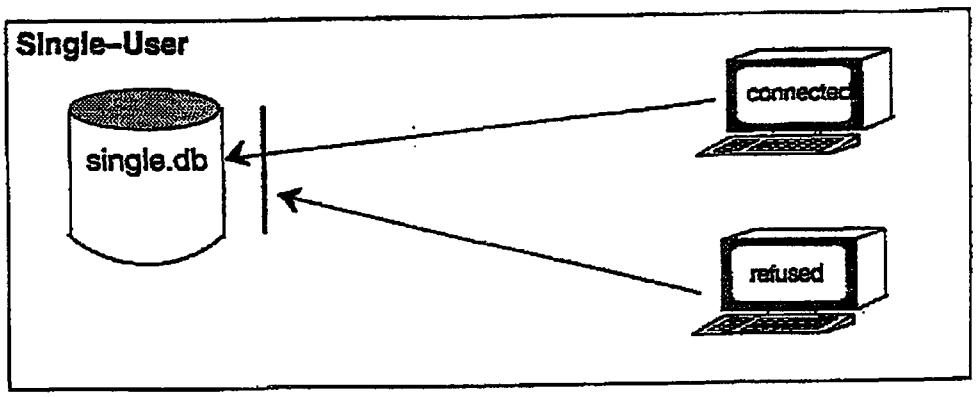
Örnek: connect fatura -H ant -S antser -N TCP -ld aytfat.

Eğer dağıtık bir sistemdeki bütün veritabanlarının fiziksel ismi aynı ise yukarıda örnekte olduğu gibi "aytfat" mantıkasl ismi verilerek diğer veritabanlarından ayrıştırma yapılabilir. Burada "-ld" bağlantı parametresi, standard fiziksel isimden başka bir mantıksal veritabanı ismi vermeyi sağlamaktadır.

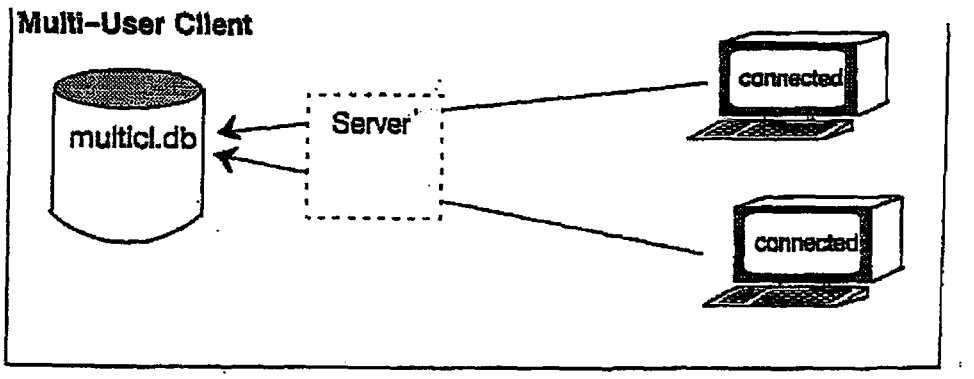
Mantıksal veritabanı isimlerinin başka bir avantajı da derlenmiş bir programın yeniden derlenmesine gereksinim bırakmaksızın çalışmasını sağlamaktır. Örneğin fatura veritabanına bağlantı sağlandıysa ve işlem yapan programlar "fatura2" adlı başka yerdeki bir veritabanı için derlendiyse fatura veritabanını için ayın programların çalışması söz konusu ise "-ld fatura2" kullanılarak programlar yenideb fatura veritabnı için derlenmeden çalışabilir.

Daha önce bağlantı sağlnmış bir veritabanı ismiyle aynı bir bağlantı talebi başarısızlığa uğrar.

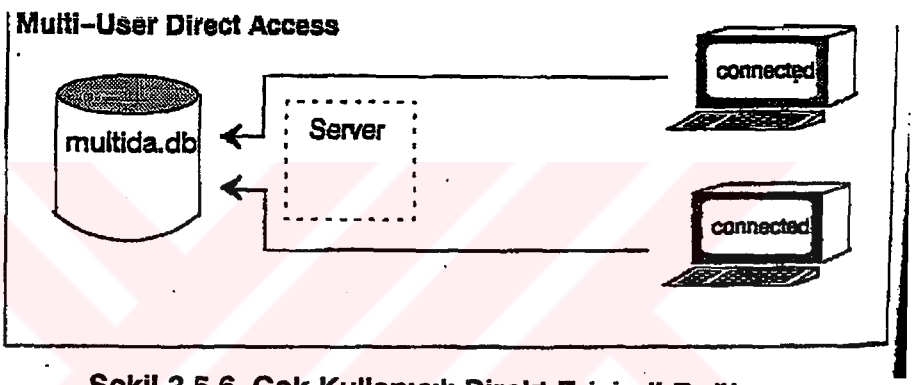
Bağlantı Modları:



Şekil 3.5.4. Tek Kullanıcılı Bağlantı Biçimi



Şekil 3.5.5. Çok Kullanıcılı Bağlantı Biçimi



Şekil 3.5.6. Çok Kullanıcılı Direkt Erişimli Bağlanma

Her veritabanı bağlantı esnasında kurulan bir bağlantı moduna sahiptir. Veritabanı bağlantı modu o veritabanini aynı anda kaç kullanıcının kullanabileceğini belirler.

Eğer veritabanına, çok kullanıcılı başlatma komutları mpro veya mbpro ile bağlanılırsa kullanılan makine için komut satırında belirtilen bütün veritabanlarına otomatik olarak çok kullanıcılı modda bağlantı sağlanmış olur.

Pro komutu veya "connect -1" parametresi ile gerçekleşen bütün bağlantılar tek kullanıcılı moddadır.

3.5.3. Veritabanı Tipi

Progress harici bir veritabanına bağlantılar uygun veritabanı gateway'leri kullanılarak gerçekleştirilir. Progress altından Oracle veya RMS gibi veritabanlarına bağlanmak için ek olarak gateway denilen geçitler vardır.

a) Şema bilgisi tutan (schema-holder) bir veritabanı oluşturulur.
b) Progress harici veritabnı için şema tutucu içerisinde gateway şeması

oluşturulur.
c) Şema tutucu veritabanına bağlanılır.
d) Gateway şemasına bağlanılır.

örnek:

Bir Oracle veritabanına bağlanmak işlemi aşağıdaki gibi gerçekleştirilir.

pro -pf oraprm.pf.

oraprm.pf: -db holddb -ld holder -db x -ld ora1 -U orauid -P orauid.

Gateway Terminolojisi:

Progress 4GL Uygulaması:

For each firma:	Progress
display firma_ad	Harici
End.	Veri

Veritabanı gateway'i Progress 4GL ile geliştirilmiş bir uygulamanın progress harici bir veritabanı üzerinde işlem yapmasına olanak vermektedir.

Schema holder veritabanı, bir veya daha fazla Progress harici veritabanı için şema tanımlarını içermektedir. Gateway schema ise, schema holder içerisindeki Progress harici veritabanına ait yapısal açıklamaları içermektedir. Schema holder veritabanı, Progress harici veritabanı gateway şeması haricinde herhangi bir progress şemasını da içerebilir.

3.5.4. Veritabanı Lokasyonu

Veritabanı lokasyonu, uygulama.veritabanlarını iki genel tipe ayırır. Bunlar yerel veritabanları ve uzak veritabanlarıdır.

Veritabanı konfigürasyonu ise üç şekildedir.

1) Birleşik (Federated)

2) Dağıtık-(Distributed)

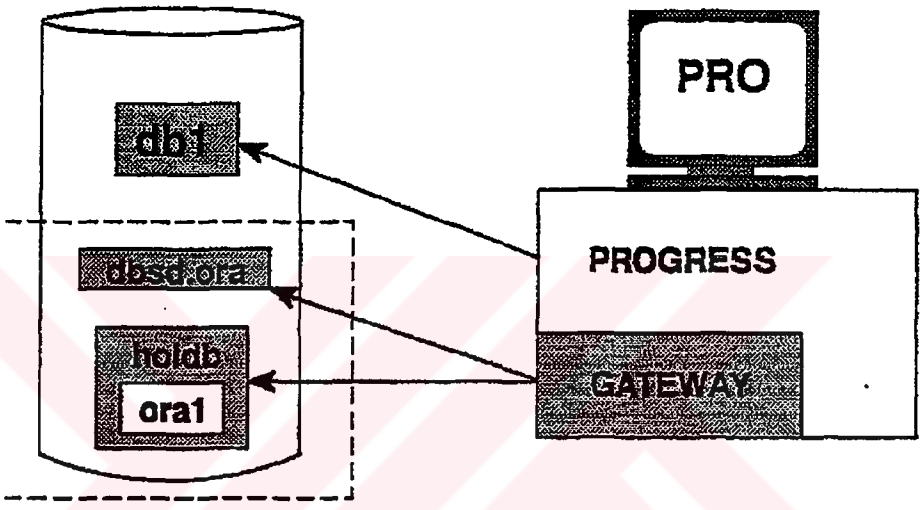
3) Dağıtık/Aynı anda Networkler.

Yerel veritabanları, uygulamanın işlediği aynı makine üzerinde yer alırlar. Uzak veritabanları, network üzerinde uygulamanın yer aldığı makinadan farklı bir makine

üzerinde yer alırlar. Birleşik veritabanı konfigürasyonunda bütün uygulama veritabanları, uygulama prosesinin çalıştığı makine üzerindedir. Dağıtık veritabanı konfigürasyonunda, uzak veritabanları uygulamanın gerçekleştiği makineye tek bir ağ protokolü ile bağlıdır.

Birleşik (Federated) Senaryo:

Yukarıdaki senaryoda Progress uygulaması iki yerel veritabanına erişmektedir.



Şekil 3.5.7. Birleşik Veritabanı Senaryosu

Veritabanları: db1 ve dbsd.ora

pro -pf parmfl3.pf.

parmfl3.pf: -db db1 -ld birincidb -db holdd -1 -l ikincidb -db x -ld ora1 -1 -U orauid -P orapwd

Dağıtık (Distrubuted) Senaryo:

Uzak makinelerdeki veritabanlarına tek bir ağ protokolü kullanarak erişilmektedir. Bunun için tez konusu projedeki senaryo örnek verilebilir.

Burada istfat, adbfat, esbfat, aytfat, dlmfat veritabanları uzak muhasebe, genfat veritabanı yerel veritabanlarıdır.

Bunlara erişim aşağıdaki şekildedir.

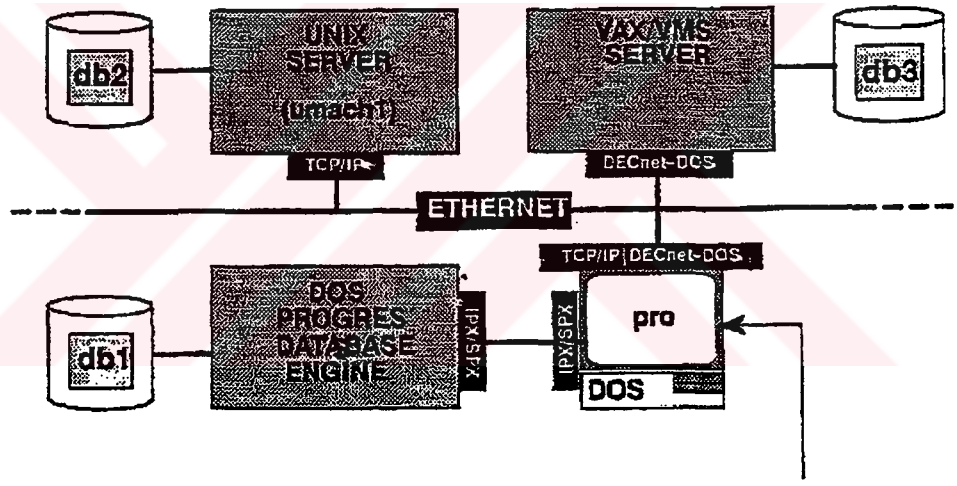
mpro -pf baglan.pf.

baglan.pf:

-db fatura -ld genfat -db muhasebe -db fatura -ld istfat -H ist -S istser -N TCP
-db fatura -ld izmfat -H izm -S izmser -N TCP
-db fatura -ld esbfat -H ank -S ankser -N TCP
-db fatura -ld aytfat -H ant -S antser -N TCP
-db fatura -ld dlmfat -H dal -S dalser -N TCP

Burada -H ile bağlanılmak istenen veritabanın bulunduğu makine, -S ile ilgili veritabanına hangi servis kullanılarak bağlanılacağı, -N ile hangi network protokolü kullanılarak bağlanılacağı belirtilmektedir. -ld ise ilgili veritabanına hangi mantıksal isim kullanarak bağlanılacağı belitilmektedir.

Dağıtık/Aynı Anda Çoklu Network Senaryosu:



Şekil 3.5.8. Aynı Anda Çoklu Ağ Üzerinden Dağıtık Senaryo

Progress, uzak veritabanlarına aynı anda birden fazla ağ protokolü kullanarak erişme imkanı vermektedir. Uygulama prosesinin yer aldığı makine diğer makinelere ulaşmak için protokollerle ilgili yazılım ve donanıma sahip olmalıdır.

Yukarıdaki şekilde db1 (IPX/SPX) , db2 (TCP/IP), db3 (DECnet-DOS) protokollerini kullanmaktadır.

Aşağıdaki görüldüğü gibi bağlantılar gerçekleştirilmektedir.

pro -pf parmfl5.pf.

parmfl5.pf:

-db db1 -ld birincidb -N SPX -S db1sv -db db2 -ld ikincidb -N TCP -H umach1 -S db2sv -db db3 -ld ucuncudb -N NETBIOS -S db3sv

Burada -N parametresi ile hangi network protokolünün kullanılacağı belirtilmektedir.

3.5.5.Veritabanı Bağlantılarını Koparmak:

Standard olarak her Progress uygulamasının sonuna bağlanılan veritabnaları kopmaktadır. Bunun dışında uygulama esnasında bir veritabanından kopmak istenirse "disconnect" komutu kullanılır. Disconnect komutu aktif uygulama ile ilgili işlemler bitmedikçe veya geri çevrilmedikçe icra edilmemektedir. Kullanım şekli: disconnect antıksal veritabanı ismi

Örnek:

```
connect -db muhasebe -1.
run altprog.p.
```

altprog.p:

```
run altprog2.p.
  For each muhasebe.doviz:
     display doviz_alis doviz_satis efektif_alis    efektif_satis tarih.
  end.
```

altprog2.p:

```
disconnect muhasebe.
connect muhasebe.
```

Eğer bir veritabanı, zaten evvelden bağlanılmış, fakat disconnect ile sona erdirilmiş ise, kendi üzerinde yeniden connect komutu çalıştırılırsa veritabanı yine bağlıymış gibi davranacak , dsiconnect ifadesi etkisiz olacaktır.

3.5.6. Bağlantılar ve Problemleri:

Bağlantı problemi, belli bir uygulamanın veritabnına bağlanması için geçen süreyle ilgilidir. Veritabanına ne kadar çok kullanıcı bağlanırsa, veritabanı broker'ının bunlara vereceği cevap ve işlemlerin tamamlanması süresi de o kadar artacaktır. Bu nedenle veritabanına optimum bağlantı sayısını sistem kaynakları da göz önünde bulundurularak -n kullanıcı sayısı parametresi ile belirlemekte fayda vardır. Belli bir uygulama içerisinden yapılan her veritabanı bağlantısı veya kopması extra bir maliyet getirmektedir. Bu sorun daha çok yerel networkler için önemlidir.

Wide Area Networkler'de ise özellikle ülkemizde kullanılan network protokolünün hızı en önemli faktör haline gelmektedir. Ülkemizdeki PTT alt yapısı dağıtık organizasyonlara cevap verebilmek açısından cok vavas kalmaktadır.

3.6. Hareket Yönetimi

Hareket denildiği zaman herhangi bir işin, ya tamamıyla yapıldığı ya da hiçbir kısmının yapılmadığı birim ifade edilmektedir. Fizksel veritabanı kurtarma denetiminin temel taşını oluşturmaktadır.

Progress içerisinde blok ve hareket kavramı iç içe yer almaktadır. Bu bloklar aşağıdaki gibidir.

```
Repeat:
  <Komut >        ====> Blok ====> Hareket
  <Komut>____
End.
 For each:
     <Komut>      ====> Blok ====> Hareket
     <Komut>
End.
Do Transaction:
    <Komut>       ====> Blok =====> Hareket
    <Komut>
End.
Do on error:
    <Komut>       ====> Blok ====> Hareket
    <Komut>
End.
```

Repeat ve For each iterasyon bloklarıdır. Her bir blok veya her bir iterasyon belli bir harekete karşılık gelmektedir. Veri güvenliğni sağlamak için hareketleri doğru şekilde düzenlemek ve kontrolünü yapmak çok önem taşımaktadır. Çünkü veritabanı işleyişinde herhangi bir beklenmeyen kesinti olduğunda, veritabanını yeniden başlatırken çökme anındaki yarım hareketler ayıklanır.

Progress hereket bloğunu standard olarak; en dıştaki, kayıdı exclusive-lock koyarak okuyan veya güncelleyen bloğu seçer. Bir prosedür bloğu her zaman için en dış en dış hareket bloğu kabul edilir. Her kullanıcı belli bir anda aktif olarak sadece tek bir hareket içerisinde bulunabilir.

Dağıtık veritabanları üzerinde belli bir hareket için: Dağıtık sistem üzerindeki her veritabanı için , eğer o veritabanı kullanılıyor veya değişikliğe uğruyorsa bir hareket başlamış demektir. Dağıtık veritabanlarında da normal de olduğu gibi belli bir hareket içerisinde iki aşamalı taahüt mantığı geçerlidir. Bu dağıtık veritabanı yönetim sistemleri için veri muhafazası ve güvenliği için oldukça önem taşımakta ve herhangi bir network problemindeb dolayı oluşacak tutarsız veri oluşması durumunu engellemektedir.

Progress hareket kontrolünü yaparken özellikle before-image dosyalarından faydalanmaktadır. Her veritabanı için bir bir before-image dosyası bulunmaktadır. Sistemin çökmesinden dolayı veritabanında oluşacak zararları önlemek açısından .bi dosyasına hareket esnasındaki bilgiler direkt buffer'dan değil hareket biter bitmez özel olarak disk üzerinde kendisine işlenir.

.bi'a bilgiler yazılırken disk şu şekilde kullanılır:

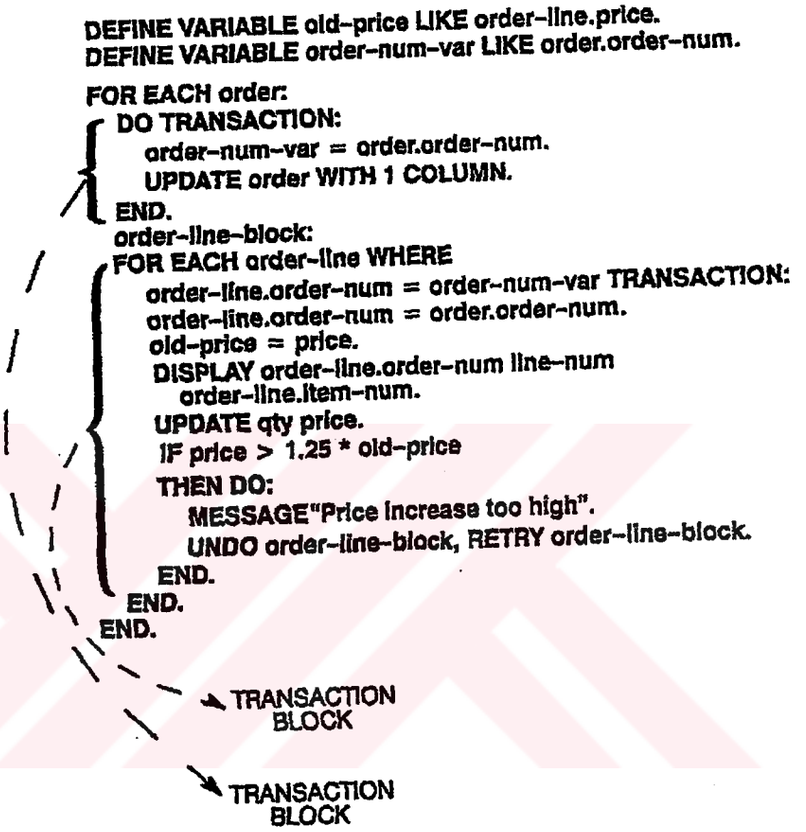
Tek Kullanıcıda: Her hareket bir evvel ki hareketin kullandığı alanı alarak yeniden kullanır. Büyük hareketler için .bi dosyasının hacmi de büyür.
Çok Kullanıcıda: .bi alanı ancak o cluster içerisinde hiçbir aktif veritabanı hareketi olmadığı takdirde yeniden kullanılır. Çok ağır işleyen, büyük on-line hareketlerde .bi dosyası oldukça büyümektedir. .bi dosyasına belli bir hareket esnasında bilgi işlendiği gibi, belli bir veritabanı kaydı veya indeksi modifiye edildiği zaman da bilgi işlenmektedir.

Kayıtlardaki yenilenmeler, her kayıt alanının sonunda gerçekleşir. İndeksteki yenilenmeler ise veri değerlerine son hali verildiği zaman yapılmaktadır. .bi dosyası modifiye edilecek kayıtların içeriklerini, belli bir hareket içerisine girmeden evvel tutmaktadır. Böylece hareket esnasında herhangi bir sorun çıktığında , geriye tutarlı verilere dönüş mümkün olabilmektedir. hareket esnasında değişikliğe uğrayan kayıtlar ancak hareketin sonunda kabul edilmektedir.

Before Image dosyasının sonuna bilgiler;
i) Her hareketin başlangıcında,
ii) Assign, insert, set, update gibi indeks değişikliğine yol açan komutlardan sonra,
iii) Her hareketin sonuda yazılır.

Mümkün olduğu kadar çok fazla kayıdı kilitlememek , .bi'ı büyütmemek ve sistem performansını arttırmak için hareketlerin boyutlarını küçültmekte fayda vardır. Bu aşağıdaki örnekle açıklanabilir.

En dıştaki For each bloğu direkt olarak veritabanı üzerinde herhanfi bir değişiklik yapmadığı için bir hareket bloğu değildir. Bütün değişiklikler içerideki iki hareket bloğu tarafından yapılmaktadır. Do transaction bloğunun, her iterasyonu ayrı bir harekettir. Order güncellemesi esnasında herhangi bir hata söz konusu

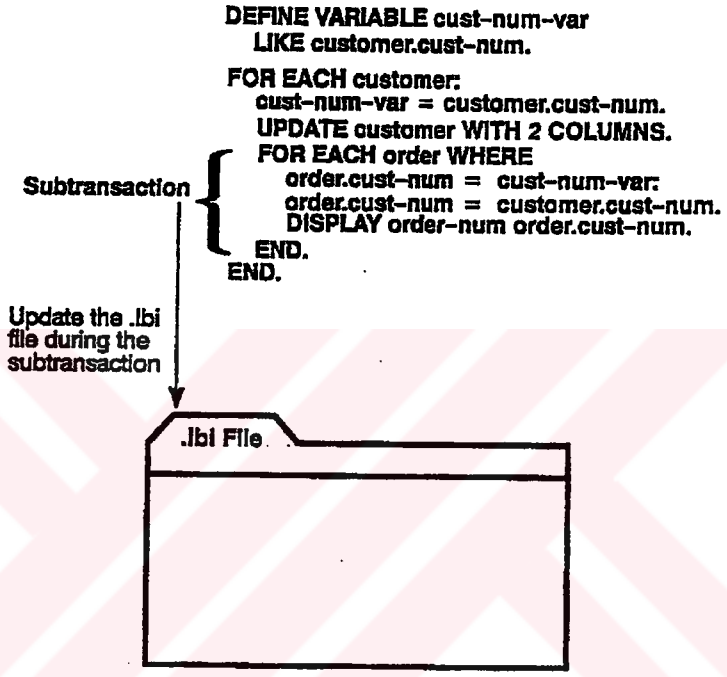


Şekil 3.6.1. Hareketleri daha küçük tutmak

olduğunda order kayıdında yapılan bütün değşiklikler geri alınacaktır. For each içbloğu da bir hareket bloğudur. Her iterasyon ayrı bir hareketi ifade etmektedir. O blok esnasında herhangi bir hata meydana geldiğinde order-line kayıdı üzerinde yapılan bütün değişiklikler geri alınacaktır. Fakat order kayıdındaki değişiklikler kabul edilmiş olacaktır. Tamamlanmış bir hareketin geri alınması söz konusu değildir. Burada bir hareketin boyutunun küçültülmesi ya da büyütülmesinin önmei görülmektedir. order-line'da bir hata olduğu zaman , order'da da yapılan değişikliklerin hepsi alınmak isteniyorsa hareket boyutu order ve order-line dosyalarının ikisini de içine alacak şekilde büyütülmelidir.

Before - image dosyasından başka alt hareket yönetiminde local before image dosyaları (.lbi) yardımcı olmaktadır. :bi 'dan farkı,I/O buffer'ına yazılıyor olmasıdır. Aşağıdaki örnekle .lbi dosyasının nasıl kullanıldığı açıklanabilir.

Alt hareketlerdeki bütün aktiviteler hareket esnaında .lbi dosyasına yazılmaktadır. Herhangi bir hata söz konusu olduğunda .lbi kullanılarak hareket



Şekil 3.6.2.Alt Hareketler ve Yerel Before Image Mekanizması

geri alınır. Bir alt prosedür çağrıldığı zaman aktif olan hareket , alt prosedür süresince de aktif kalır. Bir alt prosedür esnasında, sistem çökmesi dışında herhangi bir hatadan dolayı hareketin geri alınması gerekiyorsa, alt prosedür geri çevrilir.; fakat alt prosedürü çağıran programdaki değişiklikler geri çevrilmez; çünkü alt prosedür bir alt hareket olarak kabul edilmektedir.

Bir hareket esnasında sistem çökmesi olursa , veritabanı yeniden başlatıldığında hareket esnasındaki nbütün değişiklikler geri alınır, hareketten evvelki duruma dönülür. Her client için bir .lbi dosyası mevcutttur. Belli bir hareket geri alındığı zaman veritabanı üzerindeki değişikliklerden başka değişkenler üzerinde meydana gelen değişiklikler de geri alınır. Eğer bu değişikliklerin geri alınması istenmiyorsa değişkenler tanımlanırken no-undo opsiyonu kullanılmalıdır. Değişkenlerle ilgili bilgiler yine .lbi dosyası içerisinde tutulur. Dolayısıyla eğer çok fazla değişken

kullanılmış ve no-undo opsiyonu kullanılmamışsa .lbi dosyasının boyutu büyüyecektir. Veritabanını başlatırken (-l) opsiyonunu büyütmak gerekecektir.

Hareketlerin getireceği maliyetleri azaltmak için :

i) Başlayan ve biten hareketlerin sayısı azaltılabilir. Alt hareketler kullanılabilir.
ii)(-i) opsiyonu kullanılabilir. Bu opsiyon kullanıldığında çökmeye karşı hiçbir muhafaza olmayacaktır. Bu nedenle bu opsiyonu kullanmadan evvel veritabnının yedeğini almakta fayda vardır.
iii) değişkenlerin getireceği maliyet azaltılamak için no-undo kullanılabilir.

Dağıtık Çok Kullanıcılı Veritabanı Üzerinde Hareket Yönetimi:

Belli bir hareket bloğu içerisinde hareket başlangıcından evvel bağlanılmış bütün veritabanları bu hareketten etkilenirler. Veritabanındaki değişikliklerin gerçekleşmesi için iki açamalı tahhüt kuralı uygulanır. Genellikle bir hareket bloğu içerisinde belli bir veritabanına bağlanmamakta fayda vardır. Çünkü bağlantı süresinin uzaması, hareket içerisinde kilitlene kayıtların gereksiz yere kullanılamamasına yol açacaktır. Özellikle coğrafi bir alan üzerine yayılmış bilgisayar ağlarında (Wide Area-Geographic Network) habaerleşme altyapısında sorunlar varsa bu hususa dikkat etmek gerekir.

Örnek:

```
if connected("istfat") and connected ("ayfat") then

run hareket.p. /** Hareket Bloğu**/

else message "Hareket icra edilemiyor.".
```

Dağıtık bir sistemde iki aşamalı taahüdün ilk safhasında, hareketin etkilediği veritabanlarındaki değişikliklerin tamamlanıp tamamlanmadığı kontrol edilir. Eğer bu veritabanlarından birine ulşılamıyorsa, .bi kullanılarak bütün veritabanlarında yapılan değişiklikler geri alınır. Eğer bütün veritabanları ulşılabilir ve değişikliğin başarı ile yapıldığı görülüyor ise ikinci aşamaya geçilerek hareket esnasındaki bütün değişiklikler kabul edilir.

Yalnız Progress harici (Oracle veritabanı üzerinde) iki aşmalı taahüt işlememektedir. İlk aşamada Progress veritabanları kontrol edilir. İkinci aşamada Progress veritabanındaki değişiklikjler gerçekleşmeden Oracle veritabanındaki değişiklikler gerçeklenir. Dolayısıyla Progress veritabanlarındaki değişiklikler gerçeklenmeden Oracle veritabnı tarafından bir sorun doğabilir.

3.7. Çok Kullanıcılı Ortam ve EşAnlılık Kontrolü

Bir VTYS çok kullanıcılı ortamında, aynı anda aynı kayıtlar üzerinde işlem esnasındaki kontroller veri bütünlüğü ve güvenliği açısından çok önem taşımaktadır. Çoğu VTYS'lerde bu konuda, üçüncü kuşak dillerine göre oldukça büyük kolaylıklar ve güvenil getirilmiştir.[9]

Progress de bu konuda birtakım kontrol mekanizmaları içermektedir. Eş anlılık kontrolünü sağlamak gerçekten çok kolay; temel bir takım kilitleme mekanizmalarına dayalı olarak gerçekleşmektedir. Fakat Başarılı olmak , diğer kullanıcıları kilitlememek için önemli ola bu kontrollerin (kilitlerin) sistemi ve diğer kullanıcıları nasıl etkilediğini kavrayarak, kayıtlar üzerinde hareket etme politikasını ona göre tayin etmektir. Progress eşanlılık kontrolünü otomatik olarak getirdiği kayıt kilitlemeleri sayesinde gerçekleştirmektedir. Normal olarak bir kayıt okunduğunda SHARE-LOCK, güncellendiğnde EXCLUSIVE-LOCK adlı kilitler Progress tarafından konulmaktadır.

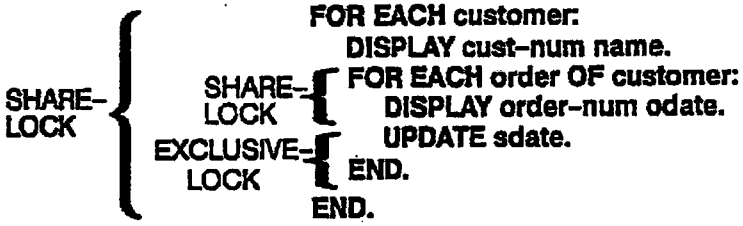
Progress belli bir kayıda share-lock koyduğunda , diğer kullanıcılar o kayıdı okuyabilir, fakat o kayıt üzerinde güncelleme yapamaz. Bir kayıt güncellendiğnde otomatik olarak Exclusive-lock konulduğundan dolayı, share-lock veya Exclusive-lock ile aynı kayıda erişmek isteyen diğer kullanıcılar okuma ve güncelleme yapamazlar. Eğer bir kullanıcı herhangi bir kayıda Başka bir kullanıcı share-lock koyamaz. Yine bir kullanıcı herhengi bir kayıda share lock koyduysa başka bir kullanıcı aynı kayıda exclusive-lock koyarak erişemez.

Tablo 3.7.1. Kilitler ve Etkileri

Bir kayıt		Diğerleri o kayıda
No-Lock	ise	Exclusive-Lock Share-Lock No-Lock
Share-Lock	ise	Share-Lock No-Lock (Yalnız okumak için)
Exclusive-Lock	ise	No-lock

Yukarıdaki tablo kayıt kilitlemelerinin diğer kullanıcıları nasıl etkilediğini göstermektedir.

Örnek:

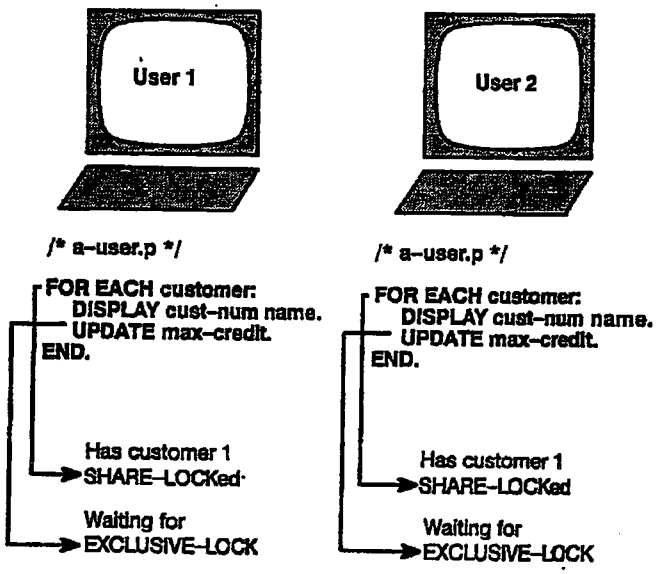


Progress kayıt alanının bitiminde share-lock'ı kaldırmaktadır. Hareketin sonunda exclusive-lock'ı kaldırmaktadır. en dıştaki for each içerisindeki iterasyonlarda diğer kullanıcılar ilgilili kayıtları okuyabilir; fakat güncelleyemezler. Çünkü share-lock söz konusudur.

Share-lock konulan bir kayıtda daha sonra bir güncelleme yapılacaksa kayıt kilidi otmatik olarak exclusive-lock'a çevrilir. Bu esnada diğer kullanıcılar ancak no-lock kullanarak okuna amacıyla erişim yapabilirler.

Kilit konulan bir kayıttaki kilit , "release" komutu ile iade edilir. Fakat henüz hareket içerisinde olan bir kayıdın kilidini değiştiremez.

Ölümcül Durum:

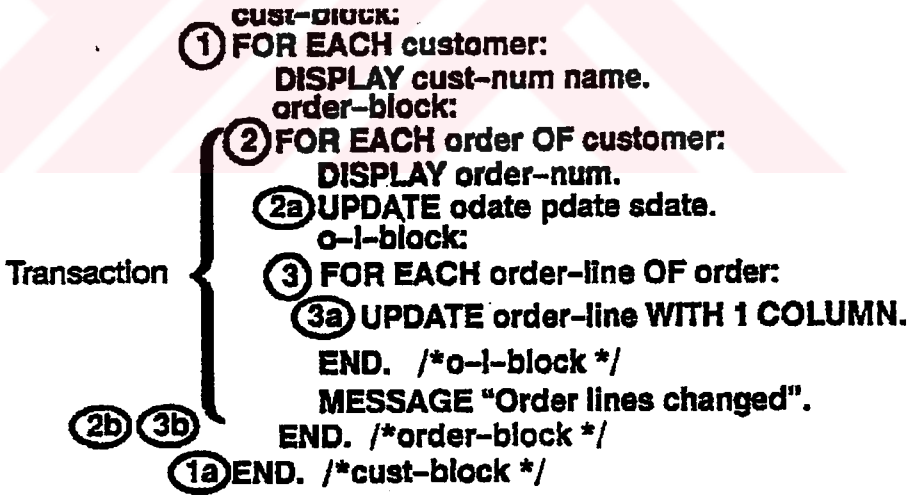


Şekil 3.7.1. Ölümcül Kilitlenme

Merkezi veritabanlarında eşanlı programlarda en çok dikkat edilmesi gereken husus, ölümcül kilitlenmelere uğramamaktır. Progress'de aynı anda iki kullanıcı, aynı kayıda share-lock ile erişmişse , güncelleme yapmak istediğnde böyle bir duruma düşerler. Çünkü her iki kulllanıcı da güncelleme yapabilmek için exclusive-lock koymak zorundadır. Eğer karşı kulanıcı hala share-lock ile kayıt üzerinde ise diğer kullanıcı, karşı kullanıcının share-lock'ı kaldırmasını beklemek zorundadır. Aynı şey karşı kullanıcı için de geçerlidir. İki kullanıcı da share-lock'ı devam ettirirse ölümcül kilitlenme olur. Bu tür durumlardan kaçınmak için yalnızca okunacak kayıtlara diğer kullanıcılara zarar vermemek açısından no-lock ile erişmek, güncelleme yapılacak kayıtlara, diğer bir kullanıcının share-lock'ından etkilenmemek için exclusive-lock ile erişilmelidir.

Hareketin boyutu ne kadar büyükse o kadar kayıdı kilitleme tablosu içine alacak ve bu kayıt sayısı çok artarsa -L (Lock Table) parametresini arttırma gereksinimi doğacaktır.

Aşaağıdaki örnekte büyük hareketlerin, kayıt kilitlenmelerini nasıl etkilediği görülmektedir.



1.) Customer share-lock

1a.) Customer share-lock'ını kaldır.

2.) order share-lock koy.

2a.) order exclusive-lock koy.

2b.) order exclusive-lock ve share-lock'ını kaldır

3.) order-line'a share-lock koy.

3a.) order-line'a exclusive-lock koy.
3b.) For each order hareketi bititğinden dolayı order-line'a ait exclusive-lock ve share-lock'ı kaldır.

Yukarıdaki örnekte exclusive-lock uzun süre tutulmakta, bu süre içerisinde diğer kullanıcılar, güncelleme yapamamaktadırlar. İçeriği azaltmak ve aynı anda çok kullanıcının aynı kayıdı kullanabilmesini sağlamak açısından uzun süre exclusive-lock kullanmaktan kaçunmak gereklidir. Ufak hareketler kilitleme zman uzunluğunu da azaltmaktadır. Fakat hareketler kısaltılrken fonksiyonlarda kayıbın olmamasına özen gösterilmelidir.

Yukarıdaki program ufak boyuttaki hareketlerle aşağıdaki şekilde daha optimize edilmiş olarak çalışabilir.

```
cust-block:
(1) FOR EACH customer:
      DISPLAY cust-num name.
    order-block:
(2) FOR EACH order OF customer:
      DISPLAY order-num.
Transaction {
      DO TRANSACTION:
(2a)    UPDATE odate pdate sdate.
(2b)  END.
Transaction {
      o-l-block:
(3)   FOR EACH order-line OF order:
(3a)    UPDATE order-line WITH 1 COLUMN.
(3b)  END. /*o-l-block */
      MESSAGE "Order lines changed.".
(2c) END. /*order-block */
(1a) END. /*cust-block */
```

1.) Customer share-lock'lı.
1a.) Customer share-lock'ı kalkıyor.
2.) order share-lock'lı
2a.) order'a exclusive-lock konuluyor.
2b.) order axclusive-lock'ı kaldırılıyor.
2c.) order share-lock'ı kaldırılıyor.
3.) order-line'a share-lock konuluyor.
3a.) order-line'a eclusive-lock lonuluyor.
3b.) order-line exclusive ve share-lock'ı kaldırılıyor.

Yukarıdaki örnekte büyük hareketler küçük hareketlere parçalanmıştır. Bu sayede diğer kullanıcıları etkileyen exclusive-lock'ların sayısı azaltılmıştır.

Diğer kullanıcıları hiç etkilemeyen kilitleme yöntemi no-lock'tır. Özellijkle arka planda çalışan raporlamalarda kullanılmasında fayda vardır. Bir kayıda sadece okumak için erişilecekse, diğer kullanıcıları etkilememek ve sisteme maliyet getirmemek açısından no-lock kullanmakta fayda vardır.

No-lock ile okunan kayıtlar güncellenmek istendiğinde, exclusive-lock ile yeniden erişilmeleri gerekir. Bir kayıda no-lock konulduğunda diğer kullanıcılar o kayıdı okuyabilir, güncelleyebilir ve silebilir. No-lock ile okunan kayıtlar kilitleme tablosunda yer tutmaz, dolayısıyla (-L) parametresini arttırmaya gerek duyulmaz ve eşanlılık performansı artar.

3.8. Güvenlik Denetimi

Bir veritabanı yönetim sistemninde güvenlik denetimi, veritabanı sistemini istenmeyen kullanıcılara karşı koruyan , uygulamalar ve veritabanı sözlüğü üzerinde denetimi tamamıyla veritabanı yöneticisinin kontrolündeki kişilere belli ölçülerde veren bir mekanizmadır. Belirli bir uygulamanın sağlıklı gidebilmesi için gerekli bir unsurdur.

Veritabanı güvenliği iki yolla sağlanmaktadır. Uygulamaların kullanımında kullanıcılara göre kısıtlamalar getirerek veya veritabanını, tabloları ve alanları hangi kullanıcıların hangi haklarla kullanıcağı belirtilerek. İyi bir güvenlik için her ikisinin de sağkanmasu gereklidir.

Progress'de güvenlik, veritabanı metaşemasında yer alan kullanıcı tanımları sayesinde sağlanır. Uygulamalar bazında güvenlik sağlamak da veritabanında, yönetici tarafından oluşturulan "aktiviteler" ve "kullanıcılar" veya benzeri alanlardan oluşan tablolar sayesinde gerçekleştirilebilir.

Kullanılan işletim sistemi de güvenliği arttırıcı etkenelrden biri olabilir. İşletim sisteminin dosyalar üzerine getirdiği kullanım yetkileri güvenliği sağlamada yardımcı olabilir. Örneğin Unix işletim sisteminin getirdiği güvenlik mekanizması sayesinde veritabanı içerisinden hiçbir korunmaya gereksinim duyulmaksızın bazı kullanıcıların veritabanını kullanması engellenebilir.

Unix işletim sistemi üzerindeki Progress veritabanı veri güvenliği , sisteme login etmiş kullanıcı isimleri kullanılarak yapılır. Unix işletim sistemine girebilmek için bir kullanıcı ismi girmek gereklidir. Kullanıcı ismi ise /etc/passwd denilen dosyada tanımlı olmak zorundadır. Ayrıca .profile dosyası dosyası içerisinden kullanıcı ortamı tanımlanabilmektedir.

Progress başlatıldığı zaman, otomatik olarak aktif kullanıcı ismine bakar ve onu veritabanı için kullanıcı ismi olarak kabul eder. Progress içerisindeki userid fonksiyonu da yine bu kullanıcı ismini alır.

Unix /etc/passwd dosyasının yapısı aşağıdaki gibidir.

userid:password:idnum:groupid:userinfo:homedir:loginsh

userid: Kullanıcının Unix ismi

password: Kripte edilmiş şifre

idnum : Uniqe-tek bir kimlik numarası

groupid : Sistem üzerindeki kullanıcı grupları için olan unique(tek) bir kimlik numarasıdır. /etc/grou dosyasını referans alır.

userinfo: Kullanıcı hakkında bilgi; genellikle kullanıcının tam ismi yer alır.

homedir: Kullanıcının home dizinini belirler.

loginsh: Kullanıcının başlangıç shell'i.

3.8.1. Progress Kullanıcıları ve _User Dosyası:

Progress veri güvenliğini büyük ölçüde, metaşema içerisinde yer alan _User dosyasını kullanarak sağlar. Bu dosyanın içerisinde, kullanıcı ismi, şifre ve tam kullanıcı ismi alanları yer alır. Bu alanların gerçek isimleri aşağıdaki gibidir.:

userid : 12 karakterlik stringdir. Progress uygulaması ile bağlantılıdır ve boş ("") olabilir.

password: 16 karakterlik stringdir. Sadece kullanıcı tarafından bilinir. Herhangi bir kkullanıcının girdiği şifre Progress "encode" fonksiyonu ile çözülebilir.

user-name: Kullanıcı hakkında açıklayıcı bilgilere sahip olan 40 karakterlik bir alandır. Genellikle kullanıcının tam ismi yer alır.

Bu dosya ile veritabanı sözlüğü içerisindeki "Data Security" alt menüsü vasıtasıyla işlem yapılabilir. Her ne kadar Unix kullanıcı isimleriyle bir bağlantı varsa da Progress kullanıcı isimleri , Unix kullanıcı isimlerinden farklıdır. Veritabanına girerken verilen isim kullanıcı ismidir. Otomatik olarak girilirse standard olarak Unix kullanıcı ismi kabul edilir. Eğer yönetici tarafından özel kullanıcı isimleri

tanımlanırsa, veritabanına her girişte Unix'den farklı bir kullanıcı ismi grilmesi istenir. Bu esanada girilen ism neyse kullanıcı ismi olarak da o kabul edilir.

Bellli bir program içerisinden hangi kullanıcının girdiği öğrenilmek isteniyorsa USERID fonksiyonu kullanılır. Aktüel kullanıcının ismini veren bir fonksiyondur. Bir veya daha fazla kullanıcı tanımlanması halinde Progress'i başlatan login.p programında herhangi bir değişiklik yapılmadığı takdirde, veritabanına girerken otomatik olarak kullanıcı ismi ve şifresi sorulur. Girilen bilgilerin veritabanında _user dosyasındaki bilgilerle uyuşup uyuşmadığı kontrol edilir. Eğer uyşmuyorsa veritabanına girilmesine izin verilmez . Eğer uyuşuyor ise Userid fonksiyonu girilen ismi alır. Prosedürler kullanılırken, veritabanı dosyalarına ulaşılırken bu kullanıcı ismi sayesinde güvenlik kontrolü yapılabiliğr.

3.8.2. Kullanım Anında Kullanıcıları Kontrol Etmek

Yukarıda da belirtidiği gibi güvenlik kontrolü iki şekilde yapılmaktadır. Bunlarda bir tanesi aktiviteler, yani programlar bazında yapılan kontrollerdir. Bu kontrol programlar içerisinden sağlanabilmektedir. Bu tür kontroller için özellikle "Can-Do" fomksiyonu kullanılmaktadır. True veya false değerini almaktadır.[8]

Daha evvel de bahsedildiği gibi eğer veritabanı için _user dosyası içerisinde bir veya daha fazla kullanıcı tanımlandıysa, Progress'i başlatan prostart.p içerisnde yer alan login.p prosedürü veritabanının userid'sini almakta ve bu userid'nin _user dosyası içerisinde tanımlı olup olmadığına bakmaktadır. Eğer girilen kullanıcı ismi burada tanımlı değilse veritabanını kullanmaya izin vermemektedir. Can-do fonksiyonu login.p tarafından alınan kullanıcı isminin, programı kullanma yetkisine sahip olanlar içinde olup olmadığını kontrol etmektedir. Aşağıda bununla ilgili bir örnek verilmiştir.

```
Örnek:
 if not can-do("anlasma,mali")
     then  do:
           message "Bu programı kullanmaya yetkili değilsiniz.".
              Return.
      end.
Repeat:
   insert firma with 2 columns.
end.
```

Yukarıdaki örnekte Can-do fonksiyonu tırnak içerisindeki kullanıcı isimleri ile, userid fonksiyonunun aldığı değeri karşılaştırmakta; eğer userid değeri bu tırmak içerisindeki değerlerden birisine tekabül ediyorsa true değerini almaktadır. Burada Can-do true değerini alırsa kullanıcı yeni bir firma bilgisi girmektedir. Eğer false değeri alırsa mesaj gözükmekte ve proframdan çıkılmaktadır.

Yıukarıda Can-do fonksiyonu ile yapılan kontrol "userid" fonksiyonu ile de aşağıda olduğu gibi yapılabilir.

Örnek:

```
if userid <> "anlasma"
      then do:
        message "Bu programı kullanmaya yetkili değilsiniz!".
          return.
      end.
Repeat:
      insert firma.
end.
```

3.8.3. Aktiviteler Bazında Güvenlik Kontrolü

Belli bir uygulama belli program ve altprogram gruplarından oluştuğu için, programlar bazında da yine kullanıcı isimlerinden faydalanarak güvenlik kontrolü yapılabilir. Bunu gerçekleştirmek için veritabanına aktiviteler ve bu aktivitelerin hangi kullanıcılar tarafından kullanılabileceğini tutan bir tablo oluşturmak gereklidir. Aktivite alanında programın ismi ("istasyon.p") gibi, çalıştıranlar alanında ("anlasma,mali,IST") şeklind bu programı kullanmaya yetkili kullanıcı isimleri virgüllerle ayrılmış vaziyette tutulmalıdır. İndeks alanı da aktivite alanına göre unique olmalıdır.

Böyle bir dosyanın ismi "yetki" olduğu ve aktivite , yetkili alanlarından oluştuğu varsayılırsa:

```
DO FOR Yetki:
      find yetki "firmasil.p" no-lock.
        if not can-do(yetkili)
            then do:
                message "Bu programı kullanmaya yetkili değilsiniz!".
                    return.
             end.
End.
```

Prompt-for firma.uh-kod.
Find firma using uh-kod.
Delete firma.

Aktivite	Yetkili
firmasil.p	anlasma,mali
ghcngir.p	IST,ESB,ADB,AYT,DLM
istasyon.p	anlasma,IST,ESB,ADB,AYT,DLM
.....	.....
.....	.....

Yukarıdaki programda firmasil.p'yi kullanmaya yetkili olanlarla userid fonksiyonunun aldığı değerin çakışıp çakışmadığı kontrol edilmekte; eğer firmasil.p için tanımlı yetkili alanında yer alan kullanıcı isimleri ile çakışmadığı görülürse programın kullanılmasına izin verilmemektedir.

3.8.4. Dosya ve Alan Bazında Derleme Esnasında Sağlanan Güvenlik Kontrolü

Güvenlik uygulamalar bazında sğlandığı zaman herhangi bir kullanıcı kendi programını yazarak bu güvenlik kontrolünü delebilir. Bu tür tehlikeler söz konusu ise daha iyi bir güvenlik için tablo(file) ve alanlar (field) bazında güvenlik kontrolü yapılması gereklidir.

Bu kontrol veritabanı sözlüğü içerisinde ulaşılabilen "Permission" dosyası tarafından sağlanabilmektedir.

File İzinleri:

Bir file için dört kademeli güvenlik söz konusudur. Bunlar:
CAN-READ

Bu alanda o file'ı okuma yetkisine sahip kullanıcılar bilgisi yer almaktadır. (*) işareti bu file'ı bütün kullanıcıların okuyabilme yetkisine sahip olduğunu ifade etmektedir. Fakat (*) işareti yerine "anlasma, mali" varsa bu dosyayı sadece anlasma ve mali adlı kullanıcıların okuma yetkisine sahip oldukları analaşılmalıdır.

Eğer "!mali" gibi (!) işaret ve arkasından bir kullanıcı ismi kullanılmışsa bu o file'ı, o kullanıcı dışında herkesin okuyabileceği anlamına gelmektedir.

CAN-WRITE

Burada belli bir file'a hangi kullanıcıların yazma yetkisi olduğu yer almaktadır. Bu file üzerinde günceleme yapmak isteyen her kullanıcı, buradaki kullanıcı isimleri ile karşılaştırılır.

CAN-CREATE

Burada kayıt yaratma yetkisine sahip kullanıcı ismlerinin listesi yer almaktadır.Yeni bir kayıt yaratıldığında userid fonksiyonunun aldığı değer bu liste ile karşılaştırılmaktadir.

CAN-DELETE

Kayıt silme yetkisine sahip kullanıcı listesini vermektedir.

Bu dört alan için muhtemel ifadeler ve anlamları aşağıdaki gibidir.

İFADE	ANLAMI
*	Bütün kullanıcılar erişmeye yetkili.
<kullanıcı>	Bu kullanıcı erişmeye yetkili .
!<kullanıcı>	Bu kullanıcı erişmeye yetkili
<string>	Yalnız bu string ile başlayan kullanıcılar erişime yetkili.

Alan Bazında Güvenlik:

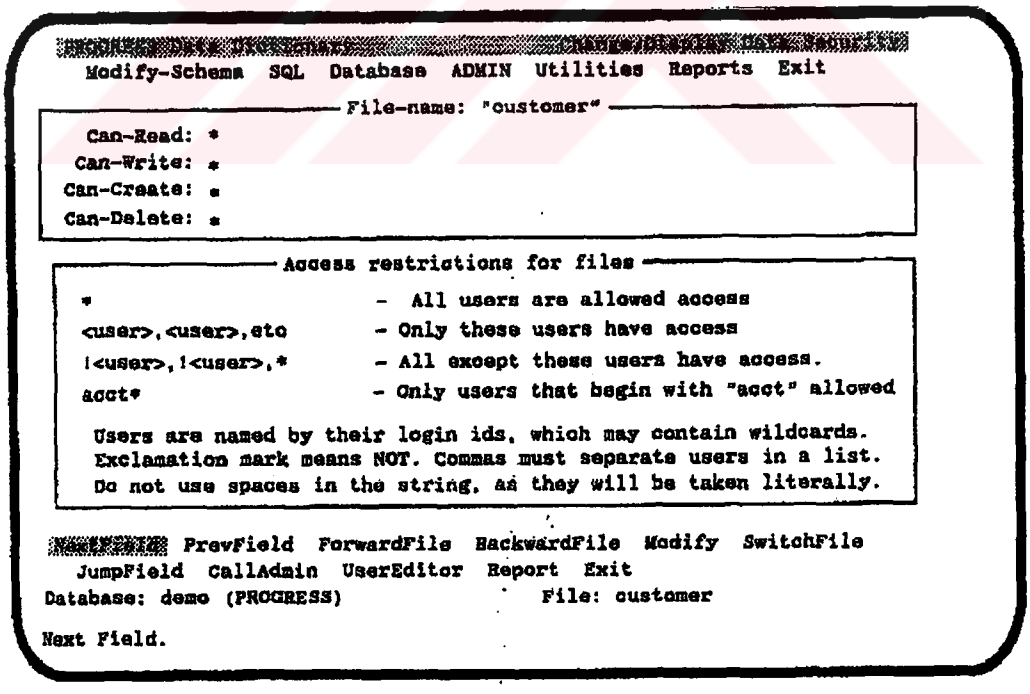
File (dosya) bazında güvenlik kontrolü yaoılabildiği gibi; alan bazında da yapılabilmektedir. Her ikisini de yapmak için veritabanı "Security Administrator" - Güvenlik Yöneticisi olmak gereklidir.

Bir file içerisinde bir veya daha fazla alanın herkes tarafından erişilmesi sakıncalı olabilir. Bu tür durumlarda alan bazında güvelik kontrolü gereklidir. İki kademeli olmaktadır.
can-read ve can-write . Burada kullanılacak ifadelerin anlamları yukarıdaki tablodaki gibidir.

Dosya ve alan bazında güvenlik kontrolünü yönetmek, yeni kullanıcılar tanımlamak ve şifrelerini vermek için veritabanı "Security Administrator" olmak gereklidir. Bu veritabanı sözlüğü içerisinden gerçekleşmektedir.

Security Administrator'ı veritabanı yöneticisi belirler. Kullanıcı bilgileri üzerimndeki değişiklikleri, yalnızca Security Administrator gerçekleştirir. Veritabanı güvenlik yöneticisi tarafından tanımlanan isimleri ayrıca herhangi bir yere çıktı göndermek için de kullanılmakta faydalı olamaktadır. Çok kullanıcılı bir ortamda kullanıcı ismine baklıarak, belli bir rapor istenilen yazıcı veya dosyaya yönlendirilebilir. Örneğin anlasma isimli kullanıcı karşısındaki yazıcı ismine tekabül eden anlasmap adlı yazıcıya çıktılarını gönderir.Bu da yine yetki dosyası gibi tablo tarafından kontrol edilir.

Belli bir file'ın tanımlarının (file yapısı, indeks yapısı) değiştirilmesi istenmiyorsa varitabanı sözlüğü içerisinde yer alan freeze/unfreeze seçeneği kullanılarak ; *freeze* ile dondurlarak tanımda bir değişiklik yapılması engellenir, unfreeze ile dondurukan bir file eski normal haline getirilir.



Şekil 3.8.1. Progress Sözlüğü Güvenlik Denetimi

3.9. Veritabanı Muhafazası (Kurtarma Denetimi)

Bir veritabanının muhafazasını sağlamak için, herhangi bir sistem çökmesi, veritabanı işleyişinin aksatılması vb. sorunlar karşısında o veritabanı yönetim sisteminin sağladığı, veritabanının aksadığı sorundan evvelki haline getirecek kurtarma mekanizmasının iyiliği ve güvenilirliği çok büyük önem taşımaktadır.

Progress'de yukarıda beirtilen şekildeki bir kurtarma mekanizması orjinal adıyla "Progress Roll Forward Recovery" bu görevi üstlenmektedir. Bu kurtarma denetimi ile veritabanı kayıplarından oluşacak zarar minimum düzeye inmektedir.Progress bu kurtarma denetimini iki şekilde yapmaktadır.

i) Before İmaging:

Bu özellik otomatik olarak işlemektedir. Veritabanını sistem çökmelerine karşı korumaktadır. Fakat veritabanının üzerinde bulunan disk vs. donanım problemlerine karşı bir korunma getirmemektedir. Bu tür durumlarda veritabanı backup'ı restore edilip , kayıp hareketler yeniden gerçekleştirilerek sorundan evvelki hale dönülür. Before imaging mekanizmasını çalıştırmak için özel bir işlem yapılmasına gerek yoktur. Yeni bir veritabanı yaratıldığında otomatik olarak .bi dosyası oluşturulur ve bu mekanizma aktif duruma geçer. Ancak büyük veri yüklemeleri gerektiği zaman veya başka herhangi bir sebepten dolayı bu mekanizma istendiği zaman devre dışı bırakılabilir[8].

ii) After Imaging/Roll Forward Recovery:

Bu özellik veritabanı medya kayıpları; yani disk, diğer donanım vs. kayıplarından korumaktadır. Bu mekanizma before imaging gibi otomatik olarak aktif hale gelmemektedir. Ayrı bir işlemle aktif hale getirilebilmektedir.

3.9.1. Before Image Mekanizması

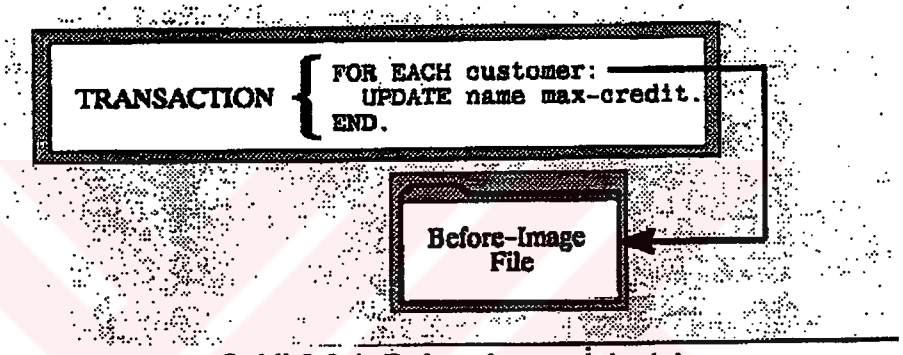
Bu özellik yukarıda da belirtildiği gibi sistem çökmelerinden korunmak amacıyla Progress tarafından otomatik olarak çalıştırılan bir mekanizmadır. Aşağıdaki bu programda bütün müşterilerle (customer) ilgili name ve max-credit alanları güncellenmektedir.

```
For each customer:
   update name max-credit.
End.
```

1 ve 2 numaralı müşterilerin güncellendiğini ve 3. müşteri güncellenirken makinenin elektrik kesintisi yüzünden kapandığı varsayılsın. Progress yeniden başlatıldığında sistem çökmesi esnasındaki bütün aktif hareketlerin geri alındığını ve veritabanı muhafazasının tamamlandığını belirten bir mesaj gözükecektir.

Kayıtlara bakıldığında 1 ve 2 numaralı müşterilere ait güncellenmiş bilgilerin aynen muhafaza edildiği fakat 3 numaralı müşteriye ait bilgilerin, güncelemeden evvelki haliyle mevcut olduğu görülecektir. Bilginin bu şekilde yüklenmesi before-imaging mekanizması sayesinde gerçekleşmektedir. Aşağıdaki şekil bu mekanizmanın nasıl işlediğini göstermektedir.

Progress bir güncelleme işleminden evvelki aktüel veritabanı durumunun



Şekil 3.9.1. Before Image İşleyişi

kopyasını alıp , before image dosyasına yazmaktadır. Bu aktivite update ifadesinin sonunda başlamaktadır. Prpgress veritabnını yeniden başlatırken bu bilgiyi before image dosyası içerisinden alarak, belli bir hareket esnasında meydana gelen sistem çökmesi durumunda, hareketten evvelki hale dönmektedir.

Daha evvel de belirtildiği gibi before imaging :

- Otomatiktir. Aktfif hale getirmeye gerek yoktur.

- Sistem Çökmesine karşı bir korunma getirmektedir.

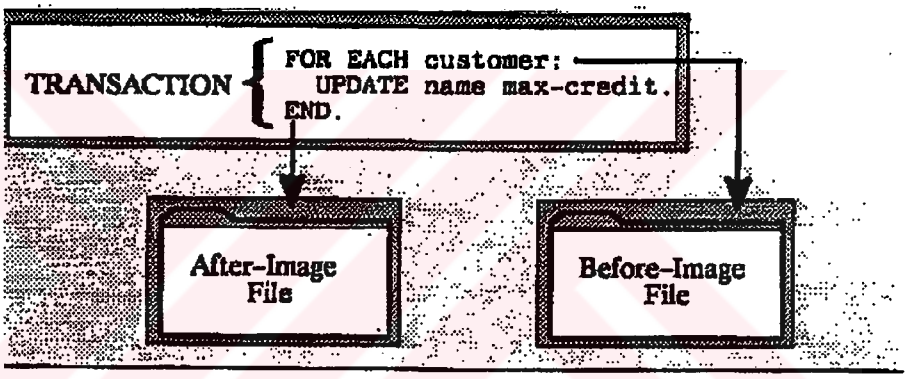
- Medya Kayıplarına karşı herhangi bir korunma getirmemektedir.

Medya kayıplarına karşı kprunmayı ise Roll Forward Recovery mekanizması sağlamaktadır. Eğer bu mekanizma kullanılmazsa .db ve .bi dosyaları yok olacağından dolayı veritabanı ancak son yedeklenmiş haliyle yeniden restore edilerek kurtarılabilecektir.

3.9.2. Roll Forward Recovery

Bu mekanizma devreye sokulduğunda Progres belli bir güncelleme işlemi sırasında ek birtakım işler yapmaktadır. Yukarıda verilen program örneği ele alınırsa:

1 ve 2 numaralı müşterilere ilişkin bilgilerin güncellendiği ve 3. müşteriye ait bilgiler güncellenirken veritabanının bulunduğu makine üzerinde fiziksel bir hasar olsun. Bu durumda veritabanını son tutarlı haliyle çalıştırmak için before-image dosyasının hiçbir afydası olmayacaktır. Çünkü diskte fiziksel bir hasar meydana geldiğinden dolayı veritabanı ve before image dosyası yok olacak ve dolayısıyla beforei-mage kurtarma mekanizması bir işe yaramayacaktır. Fakat Roll Forward mekanizması kullanılıyorsa veritabanı kurtarılabilecektir. Aşağıdaki şekil bu meknizmanın nasıl işlediğini göstermektedir.



Şekil 3.9.2. Progress'in After Image Dosyasını Kullanımı

Güncelleme işlemi başlamadan evvelki durum kopyalanıp update ifadesinin sonunda before-image dosyasına yazılmaktadır. Ayrıca her hareketin sonunda da güncellenmiş kayıt after image dosyasına yazılmaktadır.

1 ve 2 numaralı müşteriler güncellendikten sonra, 3. müşteri güncellenmesi esnasında diskte bir problem olursa bir evvelki backup ve backup'dan sonraki hareketlerin tutulduğu after-image dosyasının yüklenmesi halinde hiçbir hareket kaybına uğramadan veritabanı kurtarılması gerçeklenir.

Roll Forward Rocovery ve After Imaging:

-Otomatik değildir. Roll forward Recovery'i kullanmak için after-imaging'i enable etmek gereklidir.

- Belirli bir planlanmış periyodik yedekleme işlemi yapılması gereğini duyar .

- Veritabanını medya kayıplarına karşı korur.

Roll Forward Recovery'i Gerçekleştirme Yöntemi :

Bu mekanizmanın çalışabilmesi için iki tane gerekli unsur vardır.

i.) Progress veritabanı dosyaları için uygun yerlerin seçilmesi .

ii.) Düzenli bit yedekleme stratejisinin tesbit edilmesi.

i.) Veritabanı Dosyaları için uygun lokasyonun seçilmesi :

Yeni bir Progress veritabanı yaratıldığında otomatik olarak birbiriyle ilişkili olan veritabanı dosyası, log dosyası ve before-image dosyası oluşturulur.

Roll Forward Recovery ile veritabanı kullanıldığında ise Progress otomatik olarak after-image dosyası oluşturur. Bu dosyanın içerdiği bilgiler veritabanını yeniden inşa etmek için kullanılır. Özellikle donanımla ilgili kayıplar söz konusu olduğunda kullanılmaktadır. Before-şmage dosyasının bulunduğu diskten farklı bir diskte yer alması halinde donanım kayıplarına karşı maksimum korunma sağlanmış olur. Çünkü after-image dosyası genellikle before-image ve veritabanı dosyasının bulunduğu diskte zarar olduğunda kullanılmaktadır. Dolayısıyla before-image ile aynı diskte bulunduğu durumda after-image dosyası, dsik zarar gördüğünden dolayı kullanılamayacaktır.

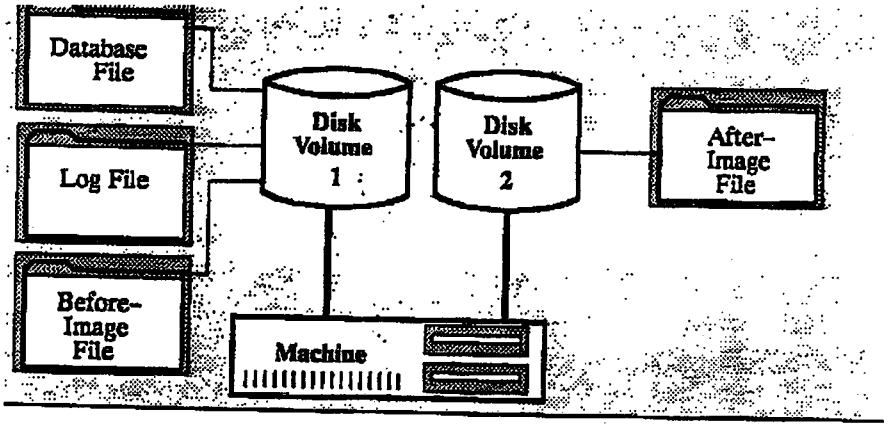
After image dosyası, son alınılan backup ile beraber düşünüldüğünde aslında veritabanının kendisini verdiği görülmektedir. Bu nedenle bir veritabanını yeniden oluşturmakta kullanılabilir. Progress başlatılırken (-a) opsiyonu kullanılarak bu dosyanın lokasyonu belirtilir. .ai uzantısı ekleyen Progress bu dosyayı istenen lokasyonda oluşturur.

Aşağıdaki şekil ilgili dosyaların tipik lokasyonunu göstermektedir.

.db , .lg, .bi ve .ai dosyalarının lokasyonunu belirlemek için aşağıdakiler göz çnünde bulundurulmalıdır:

- Donanım kayıplarına karşı, after image dosyasını .bi, .db, ve ,lg dosyalarının bulunduğu diskten farklı bir diskte saklamakta büyükj fayda vardır.

- Performans sebeplerinden dolayı before-image dosyası, .db dosyasının bulunduğu diskten farklı bir disk bırıminde saklanabilir. Yalnız dikkat edilmesi gereken after-image dosyası ile aynı yerde saklanmamasıdır.



Şekil 3.9.3. Veritabanı Dosyalarının Optimum Yerleşim Şekli

3.10. Performansı Etkileyen Faktörler:

Bellli bir sistem üzerindeki herhangi bir uygulama ve sistem performansı arasındaki ayırımı yapmak pek kolay değil, hatta keyfidir. Performansdan bahsedildiği zaman kendini etkileyen birçok faktör belirmektedir. Bu nedenle Progress veritabanı yönetim sistemiyle geliştirilecek bir uygulama için performansı etkileyen bütün faktörler göz önünde bulundurulmalıdır. Bu faktörler aşağıdaki gibi sınıflandırılabilir.[9]

i.) Performans (Progress)

Hız

Tradeoff

ii.) Uygulama Performansı

Cevap Süresi

Saniyedeki Hareket Sayısı

Kullanıcı Sayısı

iii.)Sistem Performansı

Saniyedeki Hareket Sayısı

Throughput

Cevap Süresi

Kullanıcı Sayısı

Belli bir uygulamanın tasarımı kod tasarımı ve kullanılan programlama tekniklerine bağlıdır. Performansın ölçümü ise sistem performansına bağlı göreceli olarak değişmektedir.

Progress performansı ise kendi içerisindeki tasarım ve kodlama mantığıyla beraber, kendi tabiatındaki sistem kısıtlamalarıyla orantılıdır. Geliştirilmekte olan bir uygulamanın performansı, sistem performansı, Progress veritabanı yöneticisi performansı ve tabiiki yazılımcının tasarım ve kodlama tekniğine bağlıdır. Bu bağımlılıklar performans analizini zorlaştırmaktadır.

Progress ile yazılmış bir uygulama için, veritabanı yöneticisi sistemin bir parçası olarak düşünülebilir.

3.10.1. Kodu Uygun Hale Getirmek

Sistem, performans üzerinde büyük etki gösterir. Fakat daha iyi bir performans için sistemde değişiklik o kadar kolay olmadığı için, kodlama mantığı gözden geçirilerek daha iyi bir performans sağlanabilir. Farklı kültürdeki yazılımcılarım geliştirdiği aynı görevi gören yazılımlar, yazılımcıların kültür ve bilgi birikimlerine bağlı olarak farklı performanslar gösterebilir. Belli bir yazılımcının raporlama programı ,çalıştırıldıktan iki saat sonra sonuç verirken, aynı rapor başka bir yazılımcının yazdığı programla iki dakikada elde edilebilir. Bu fark yazılımcılar arasındaki kodlama tekniklerinden doğan farklılıklardan kaynaklanabilmektedir.

Daha iyi bir peformans için aşağıdaki unsurlara dikkat edilmelidir.

-İndeksler: Doğru indeks seçilmelidir. Bu seçim yapılırken şu dört unsur göz önünde bulundurularak yapılan işin ihtiyacına göre bir indeks seçilmelidir..

a) Kayıtlara daha hızlı erişmek ve görüntülenmesini sağlamak.
b) Kayıtların otomatik sıralanmasını sağlamak .
c) Verilerin unique'lüğünü sağlamak.
d) Dosyalar arası işlemleri daha hızlı hale getirmek.

Yukarıdaki dört unsur ve yapılmak istenen işlem göz önünde bulundurularak en uygun indeksi seçmek gereklidir. Örneğin tarih aralığında bir raporlama yapmak istendiğnde, tarih alanı içeren indeks yerine yalnız kod alanından oluşan bir indeks kullanılması performansı çok önemli ölçüde azaltacak, raporlama süresi uzayacaktır.

-Hareketlerin boyutu küçük tutulmalıdır. Daha evvelde belirtildiği gibi büyük hareketlerin maliyeti de büyük olmaktadır.

-I/O sayısı mümkün olduğu kadar minimize edilmelidir.

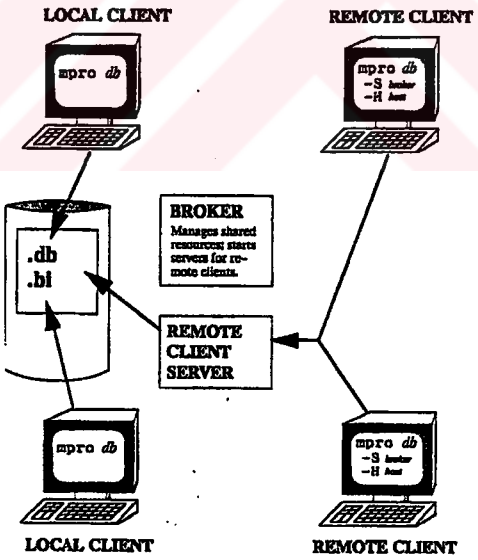
-Değişkenler tanımlanırken NO-UNDO opsiyonu kullanılmalıdır. Böylece .lbi dosyasına daha az I/O olacaktır.

-Program esnasında işletim sistemine çıkışların sayısı mümkün olduğu kadar azaltılmalıdır.

-Rekürsif programlardan mümkün olduğu kadar kaçınılmalıdır. Çünkü bu tür programlar bellekte önemli darboğazlara sebep olabilir.

3.10.2. Sistem Kaynaklarının Verimli Kullanılması

Paylaşımlı belleğe sahip bir sistemde çalışan Progress için paylaşımlı bellek, semaforlar, file descriptor'ları, buffer'lar, kilitler ve prosesler optimum düzeyde kullanılmalıdır.



Şekil 3.10.1. Paylaşımlı Bellekli Bir Sistemden Erşim

Paylaşımlı Bellek (Shared Memory):

Broker, yerel client'lar ve uzak client-server'ların hepsi paylaşımlı belleğe erişmektedirler. Host makine üzerinde çalışan bütün prosesler, uzak makinelerde

çalışmakta olan tüm uzak Client Prosesleri , server kullanmadan paylaşımlı belleğe erişemezler.

Sadece broker prosesi, paylaşımlı bellek üzerinde başlatma, silme ve tahsis etme işlemlerini yapabilir. Her aktif Progress veritabanı için bir veya daha fazla bellek segmenti tahsis edilir. Broker mümkün olduğu kadar büyük miktarda segment parçasını tutar. Bu miktar sistemden sisteme değişmektedir. İhtiyaç duyulan paylaşımlı bellek (shared memory) miktarı aşağıdaki paramatreler ile belirlenir.

(-n) Kullanıcı sayısı
(-Mn) Maksimum Server
(-B) Veritabanı Buffer'ları (Her buffer için bir disk bloğu)
(-c) İndeks kürsörü
(-L) Kilitleme tablosu girişi (Locking Table Entry)

Unix işletim sisteminde her shared memory segmenti unique bir niceleyici (identifier) alır. Bu niceleyicilerin sayısı Unix Kernel'ı içerisinde yer alan SHMNI parametresinin değeri ile sınırlıdır. Her veritabanı prosesi, kendi veritabanı için ayrılmış bütün segmentlere bağlıdır. Bu bağlantı sayısı genellikle 1 veya 2'dir ve SHMSEG kernel parametresi ile belirlenmektedir. Maksimum segment boyutu SHMMAX parametresi ile ve aktif segmentlerin sayısı, SHMALL parametresi ile sınırlanmaktadır.

Unix İşletim sistemi üzerinde "ipcs -m" komutu ile kullanımda olan shared memory segmentleri görülebilir. Broker sistem üzerindeki bir başka kullanıcı tarafından öldürülür, yani veritabanı shutdown komutu kullanılmadan sona erdirilirse, bağlantılı olduğu memory segmentlerini terketmez ve diğer prosesler bu segmentleri kullanamaz. Bu durumda "proutil -C dbipcs" kullanılarak shared memory'nin durumu tesbit edilebilir.

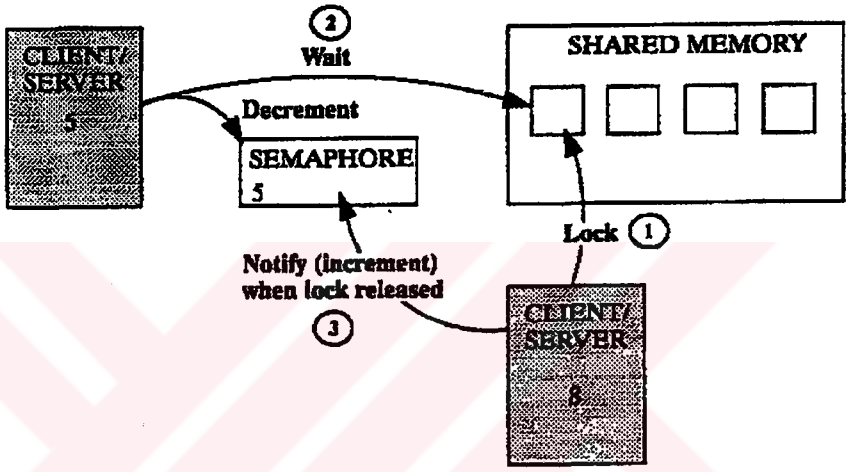
```
PROGRESS SHARED MEMORY STATUS
  ID   ShMemVer   Seg#   InUse    Database
  68          -      -       -    (not PROGRESS)
 100          3      0     Yes    /db/work5/demo
 101          3      1       -    /db/work5/demo
 120          3      0      No    /db/work5/test
 150          2      -       -    (unsupported shared memory version)
```

Şekil 3.10.2. Paylaşımlı Belleğin Kullanımı

Unix komutu olan "ipcrm -m" ile bu segmentler boşa çıkarılır.

Semaforlar:

Semafor paylaşımlı kaynaklara çoklu erişimleri denetlemek için konulan bir çeşit kilittir. Her proses bir semafora sahiptir. Bu şekilde paylaşılan bir kaynak kullanılmak istendiğinde proses bekletilmektediir. Aşağıdaki şekil semaforların eşanlı prosesleri nasıl idare ettiğini göstermektedir.



Şekil 3.10.3. Semaforlar

Şekilde görüldüğü gibi bir semaforlar dizisi mevcuttur. Her veritabanı prosesine bir semafor düşmektedir. Proses 5, Proses 8 tarafından kullanımda ve kilitli olan kayıt,indeks veya başka bir paylaşılan kaynağın kilidinin kaldırılmasını beklerken, semaforu azaltmaktadır. Kilit tutucu (8) kullandığı kaynağı geri iade ettiğnde, bekleyen proses (5) uyanmakta ve semaforu arttırmaktadır.

Semaforlar proserve ile veritabanı başlatıldığında tahsis edilmektedir. Çünkü yukarıda da belirtildiği gibi her proses bir semafora ihtiyaç duymaktadır. Veritabanında erişim yapacak proses sayısına göre o veritabanı için gerekli semafor sayısı belirlenmektedir. Belli bir veritabanı için semafor sayısı (SEM#) aşağıdaki gibi hesaplanır.

#SEM = Max. Kullanıcı sayısı (-n) + max. mümkün server (-Mn) + 4.

Kullanılacak maksimum semafor sayısı yine Unix işletim sistemi içerisindeki SEMMNS parametresi ile belirlenir. Eğer veritabanını kullananların sayısı artarsa bu parametrenin değeri de arttırılmak zorunda kalabilir. Semafor kullanımı

azaltılmak isteniyorsa maksimum kullanıcı sayısı parametresi (-n) ve maksimum server parametresi (-Mn) değeri azaltılmalıdır.

Progress ayrıca kendi içerisinde iki semafor kullanmaktadır.

File Descriptor'lar:

Bir file açıldığında ona verilen kimliktir (ID). File Descriptor'ları için belli bir sistem limiti mevcuttur. Her veritabanı prosesi (client,uzak client-server ve broker) 15 veya daha fazla file descriptor'unu kullanabilir. Bu nedenle sistemin file descriptor limitini, kullanılan proseslerin 15 katı kadar bir sayıya ayarlamak gerekir. İşletim sistemi için de 10 file descriptor'u bırakacak şekilde bir ayarlama yapmak gereklidir.

Buffer'lar:

Farklı kullanıcılarda veritabanına erişim aktiviteleri gerçkleştiğinde, veritabanı blokları, veritabanı buffer'ları içerisinden okunur. Bu veritabanı buffer'ları (-B) başlangıç parametresi ile belirlenir ve shared memory içerisinde yer alır. Buffer sayısı arttıkça daha az disk I/O'su olmakta ve dolayısıyla buffer sayısının artması sistem performansını herhangi bir yük getirmemektedir. Dolayısıyla shared memory'nin izin verdiği ölçüde mümkün olduğunca veritabanı buffer'ları (yüzlerce veya daha fazla) tahsis etmek performans açısından avantaj sağlayacaktır (Buradaki shared memory gerçek memory'dir. Swap area tarafından tahsis edilen sanal memory değildir.).

Veritabanı buffer'ları hem indeks hem de data bloklarını tutmaktadır. -B için gerekli minimum değer şöyle tespit edilebilir.

-B : Her kullanıcı için :Tek bir prosedürden erişilen dosya# +

Her kullanıcı için : (3 x tek prosedürde kullanılan indeks#)

-B mümkün olduğu kadar gerçek belleğin izin vediği ölçüde büyük seçilmelidir. Eğer buffer havuzu yeteri kadar büyükse bütün kayıtlara direkt erişim yapılabilir ve dolayısıyla performans artar. Disk I/O sayısı düşer.

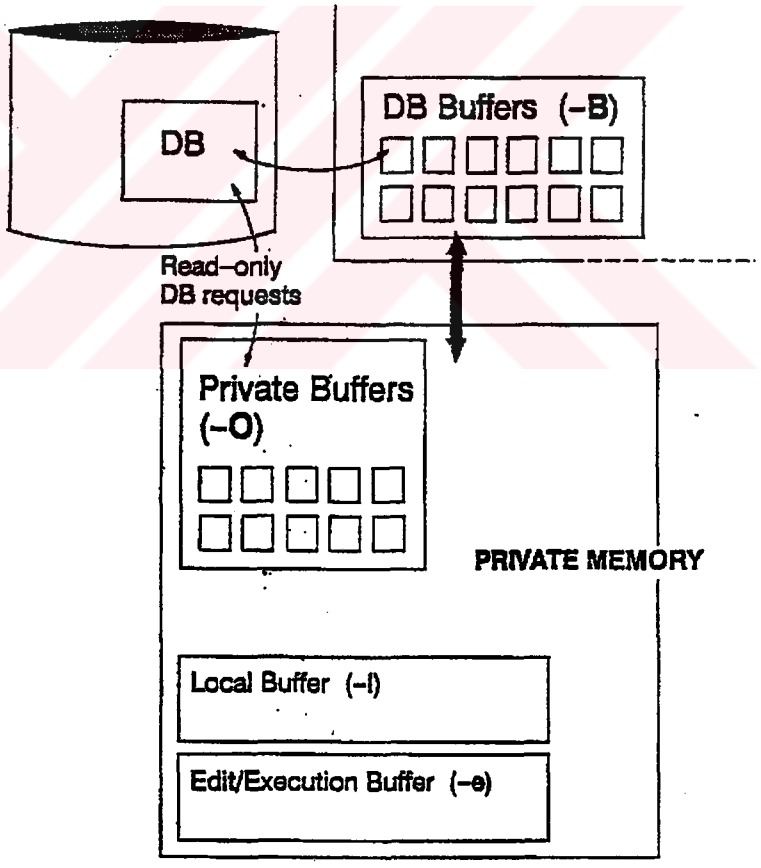
Özel Veritabanı Buffer'ları:

(-O) başlangıç parametresi ile belirlenmektedir. Shared memory içerisinde değil, her kullanıcının ya da server'ın yerel buffer'ında yer almaktadır. Özel buffer'lar daha çok sadece okunmak için kullanılan veritabanları için kullanılır. Bu şekilde paylaşımlı bellekten yer kapmak için diğer kullanıcılar daha az beklemiş olur.

Çoğunlukla okuma yapan yalnız birkaç güncelleme yapan veritabanı kullanıcıları için, paylaşımlı belleğe düşen yükü hafifletmek açısından özel veritabanı buffer'ı kullanmakta fayda vardır. Böyle bir durum için 8-12 özel buffer tahsis etmek yeterli olacaktır. Toplam özel buffer'ların sayısı ise (-l) başlangıç parametresi ile belirlenmektedir.

Eğer özel veritabanı buffer'ları kullanılmış ve bir proses güncelleme işlemi yapıyorsa, güncellenecek kayıtların özel buffer'daki kopyaları havuzdan silinir ve güncelleme bititkten sonra yeniden görünür. Bu nedenle eğer okuma/yazma oranı 5/1'den küçükse özel buffer kullanmak performansı olumsuz olarak etkileyecektir. (-l) okuma/yazma oranı 5/1 'den büyük prosesler için kullanılmalıdır.

Özel veritabanı buffer'ı için bir değer verilecekse minimum 4 olmalıdır. Aşağıdaki şekil bir client proses ile vertitabanı buffer'ları arasındaki ilikiyi göstermektedir.



Şekil 3.10.4. Özel veritabanı buffer'ları

Kilitler:

Daha evvelde bahsedildiği gibi kilitleme giriş tablosunda yer tutmakta; bu da shared memory'de yer işgal etmektedir. Dolayısıyla bir veritabanı için sadece okuma yapılacaksa -RO (read only) opsiyonu kullanılmalıdır. Böyelece ne kilitleme giriş tablosonda yer kaplanmış ne de .bi file 'ına I/O yapılmış olur. Dolayısıyla Shared memory kullanılmamış olur.

Prosesler:

Unix üzerinde NPROC parametresi, sistemdeki aktif proseslerin toplam sayısını belirlemektedir. Bu sayı genellikle 50 ve 200 arasındadır. MAXUP parametresi ise tek bir kullanıcı ID'si tarafından başlatılan eşanlı proses sayısını belirler; genellikle 15 ile 25 arasındadır. Eğer aynı isimde birden fazla kullanıcı girerse MAXUP değeri artar. Eğer MAXUP değeri yetersiz kalırsa sistem yeniden konfigüre edilmelidir.

Disk Kullanımı:

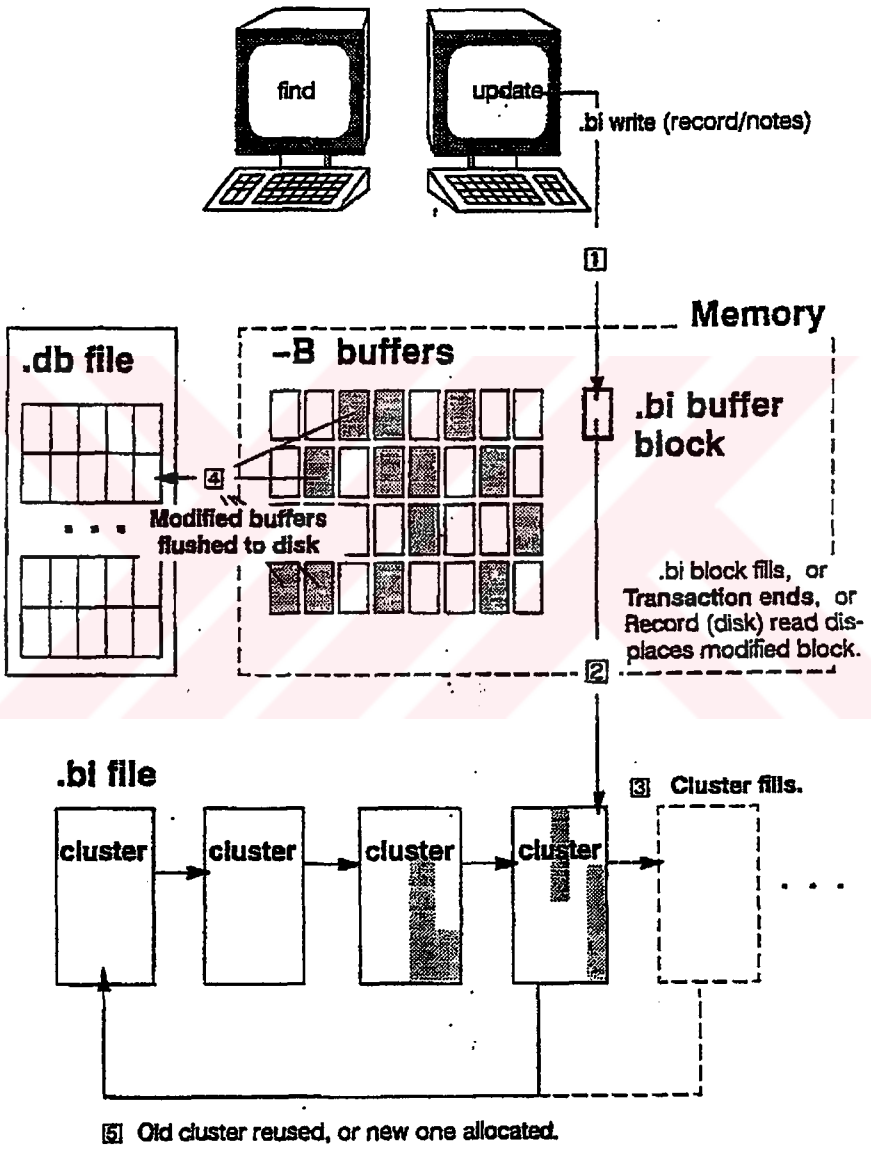
Disk ve bellek kaynakları birbirleriyle yakından ilişkilidir. Çoğu zaman birbirlerinin eksiklerini kapatmaktadırlar. Örneğin (-B)'yi arttırmak kullanılan bellek miktarını arttırır; Fakat disk I/O'sunu azaltır. Yine benzer olarak işletim buffer'ı (-e)'yi arttırmak bellek kullanımını arttırır. Fakat diske yüklenen prosedür yükünü azaltır.[8]

Disk kullanımında dikkat edilmesi gereken nokta uygun bir swap alanının ayrılmasıdır. Bu da 5MB x 2 x RAM'den büyük olmalıdır. Bunun haricinde Progress, Progress uygulamaları ve geçici dosyalar için yeteri kadar yer olmalıdır.

Before Image ve Disk:

Before Image ve local before image mekanizması disk I/O'sunu önemli ölçüde arttırmaktadır. .bi dosyası oldukça büyüyebilmekte ve bütün kullanıcı ve hareketlerle ilgili kayıt ve notları tutmaktadır.

.bi dosyası diskte cluster'lar üzerinde yer alır. Her bilgi işlemede en az dört cluster tahsis edilir. Her bir cluster dolduğunda, modifiye edilmiş veritabanı buffer blokları diske aktarılır ve yeni bir cluster tahsis edilir ya da eski bir tanesi kullanılır. Bir .bi buffer bloğu; dolduğu zaman; hareket bittiğinde, modifiye edilmiş bir buffer, yeni blok için yer açmak amacıyla yazdırılması gerektiğinde yazdırılır.



Şekil 3.10.5. Before Image Mekanizması

Bir .bi cluster'ı, aktif bir hareketin bilgilerini kullanıyorsa yeniden kullanılamaz. Aynı zamanda yeni bir cluster eklendiğinde de kullanılamaz. Aşağıdaki şekil bu mekanizmayı göstermektedir.

Normal olarak bir cluster boyutu 16 KB'dir. (bi) opsiyonu ile cluster boyutunu arttırmak sistem performansını da arttıracaktır; Fakat bu sefer disk kullanımı da azalacaktır. Kullanımı aşağıdaki gibidir.

proutil <veritabanı> -C truncate bi [kilobyte]

Artan cluster boyutunun getireceği iki maliyet vardır:

1.) Büyük cluster'lar disk alanını işgal eder.
2.) Sistem çöktükten sonra veritabanı yeniden başlatılmak istendiğinde, veritabnını eski haline getirme süresi oldukça uzayacaktır.

Bellek Darboğazı:

CPU kullanımı eğer ağırsa (%95'in üzerindeyse) sistem zorlanmakta demektir. Muhtemelen buna sebep bellek miktarının yetersiz kalması ve disk üzerindeki swap alanının fazlasıyla kullanılmasıdır.

Böyle bir bellek darboğazından kurtulmak için iki seçenek vardır. Bellek kullanımını azaltmak ya da daha fazla bellek eklemektir.

Böyle bir durumda server/broker yeniden başlatılırken -B,-c ve -L değerleri azaltılmalıdır. Kullanıcılar için ise yine -l, -e, -D ve -s değerleri azaltılmalıdır. Özel veritabanı buffer'ları bu sefer arttırılmak yerine azaltılmalıdır. Swap alanı 8 MB veya daha üzerine çıkarılmalıdır. Gerekirse paylaşımlı bellek alanını arttırmak için sistem yeniden konfigure edilmelidir. -n opsiyonu ile verilen kullanıcı sayısı kısıtlandırılmalıdır. Sistem üzerindeki gereksiz prosesler sona erdirilmelidir.

Disk Darboğazı

Diskle ilgili bir darboğaz yaşandığında aşağıdakiler yapılmalıdır.

Tek Server: .db ve .bi dosyası aynı disk üzerinde olmalıdır. Özel dosyalar (temporary) bağımsız başka bir disk birimi üzerinde tutulmalıdır.

Çoklu-Server: .db ve .bi dosyaları farklı disklerde olmalıdır. Multi Volume extent veritabanları oluşturularak farklı disklere dağıtılmalıdır. Özel dosyalar da yine bütün disklere dağıtılmalıdır.

Bunun dışında yapılması gerekenler: -Mf parametresi, server başlatılıken pozitif bir sayı değeri almalıdır. Başlangıç değeri 3 saniyedir. Bu her hareketin sonunda .bi dosyasına hemen bilgi yazılmasını önler ve belirgin şekilde performans artar. Server yeniden başlatıldığında -B, maksimum değere (500) çıkarılmalıdır. Yeteri kadar bellek varsa, daha fazla prosedürü bellekte tutmak için -e değeri arttırılmalıdır. Veritabanı server'ının önceliği arttırılmalıdır. Kodlama kontrol edilmeli alt hareketler minimize edilmeli, no-undo ve no-lock kullanılmalı, kayıt boyutları mümkünse küçültülmelidir.

Multi Volume - Çok Parçalı Veritabanları

Multi Volume veritabanları kullanılarak disk darboğazları aşılabilir ve performans arttırılabilir. Çoklu veritabanı konfigürasyonu kullanılarak, bir veritabanı 100 üzerinde dosya sistemde ya da ayrı disk birimi üzerinde parçalara ayrılarak bağlantılı olarak çalışabilir.

Hangi verinin hangi parçada yer alacağına Progress karar verir. Kullanıcı bu konuda değişiklik yapamaz. Veritabanı büyüdükçe yeni disk birimleri eklenip, yeni parçalar tanımlanır. Bir veya daha fazla parça halinde before-image dosyası olabilir. Genellikle tek parça olması yeterlidir.

CPU Darboğazı

Birden fazla bağımsız diskler eklendiğinde CPU'nun işi ağırlaşır ve bu CPU'yu bir darboğaza götürebilir. Bu tür durumlarda yeni ve daha hızlı CPU'lar eklemek gerekebilir.

3.11.Progress ve DVTYS Özelliklerine Uygunluk

Progress Veritabanı Yönetim Sistemi kendi başına tamamen dağıtık veritabanı özelliklerini taşımamaktadır. Ancak analist ve programcı tarafından kullanılan bazı yöntemlerle bu özeliklerin hemen hemen hepsi sağlanabilir.

Bu bölümde direkt Progress tarafından veya yazılımcının katkılarıyla bu özelliklerin hangi şekilde sağlandığı veya sağlanamadığı incelenecektir. İncelemeler İlk bölümde yer alan ünlü VTYS uzmanı C. J. Date'in ifade ettiği kurallar üzerine yapılacaktır. Gerektiği yerlerde tez kunusu projeden örneklemelerde bulunulacaktır.

3.11.1. Yerel Özerklik:

Bu özellik hemen hemen tamamıyla sağlanabilmektedir. Dağıtık bir sistem üzerindeki bütün ayrı VTYS'ler biribirinden bağımsız olarak çalışabilmektedir. Her yerel yapı içerisindeki işlemler o yerel yapı tarafından kontrol edilmektedir. Her farklı sitemin verisi kendi yerel makinesi üzerinde saklanmakta ve bu veriler uzak sistemler tarafından erişilebilir olsalar bile yönetim yine yerel sistemlere aittir: Uzak sistemler network üzerinden karşı yerel sistemdeki veritabanı sitemine ulaşabilme özgürlüğüne sahiptir. Fakat veritabanı sözlüğü içerisinde bir değişiklik yapabilmesi için, önce eğer karşı sistem yöneticisi tarafından belli kulanıcılar tanımlı ve her kullanıcının yetkisi sınırlıysa bu engeleri geçmesi gereklidir . Aynı zamanda karşı yerel sisteme erişim için oradaki veritabanı yöneticisinin diğer sistemdeki kullanıcılar için tanımlı olan servisi başlatması gereklidir. Servis aşağıdaki gibi başlatılmaktadır.

proserve fatura -H ist -S istser -ld istfat....

Burada veritabanının bulunduğu makine ismini, -S başlatılan TCP/IP servis ismini, -ld ise veritabanının mantıksal ismini ifade etmektedir. Bu işlemin İstanbul istasyonundaki fatura veritabanı için yapılacaksa İstanbul makinesi üzerinde çalıştırılmak zorondadır. Diğer yerel sistemlerden birisi Client-Server olarak bu veritabanını kullanmak istiyorsa kendi makinesi üzerinde, İstanbul makinesinin ismi (-H) kendi makinesindeki /etc/hosts dosyası içerisinde ve yime İstanbul servisinin ismi (-S) /etc/services dosyası içerisinde tanımlı olmalıdır. Örneğin Ankara sistemi üzerineki bir kullanıcı İstanbul veritabanı yönetim sistemine bağlanmak istiyorsa; connect fatura -H ist -S istser -ld istfat .... şeklinde bağlanabilmektedir. Fakat bu bağlantının gerçekleşebilmesi için İstanbul üzerinden istser adlı servisin başlatılması gereklidir.

Görüldüğü gibi yerel sistemler tamamıyla yerel veritabanı yöneticilerinin denetim ve kontrolü altında tutulabilmekte ve yönetim açısından diğer sistemlere bağımlılık söz konusu olmamaktadır. Fakat yerel özerklik içerisinde sistemlerin birbirine bağımlı olmadan çalışması beklendiği hatırlanırsa, bu özelliğin tamamıyla sağlanmadığı görüşüne varılabilir. öyle ki; örneğin İstanbul sisteminde bulunan verilen hizmetler verileri başka bir sistem tarafından faturalandırma için kullanılacaksa (Genel Müdürlük), faturalandırma işlemin başarıyla sonuçlanabilmesi için İstanbul veritabanı üzerindeki ilgili dönem ve firmaya ait verilen hizmet bilgilerine ulaşmak gereklidir. Dolayısıyla Genel Müdürlüğün fatura kesebilmesi için İstanbul sistemine bir bağımlılık doğmaktadır. Eğer o anda İstanbul sistemine herhangi bir nedenden dolayı ulaşamıyorsa diğer sistemdeki faturalama

işlemi yapılamayacaktır. Fakat dağıtık bir sistemdeki bütün parçaların biribiriyle belli bir şekilde ilişki içerisinde olduğu düşünülürse bu durumun yerel özerkliği fazla zedelemediği söylenebilir. Bu ilişkiler tasarımın sonucu olarak bazı bağımlılıklara sebep olmaktadır.

Bu tür bağımlılıklar bazı verilerin kopyalanarak sistem üzerinde başka bir sistemde bulunması vs. metodlarla giderilebilir. Örneğin yukarıda verilen örnekte, İstanbul'da verilen hizmet bilgileri belli bir metodla bir veya daha fazla sistemde daha bulunsaydı, İstanbul makinesinde bir sorun olduğunda diğer alternatif bir sistem veya sitamlerden gerekli bilgilere ulaşılabilirdi.

3.11.2. Merkezi Bir Sisteme Bağımlılık Olmaması

Bu özellik yerel özerkliğin bir sonucu olarak ortaya çıkmaktadır. Tüm sistemler eşit haklara sahiptir. Hiçbir sistem diğerine göre bir ayrıcalık göstermez ve işlemler hiçbir merkez sisteme bağımlı olmaksızın yürütülmektedir. Böyle bir özelliği Progress tamamen desteklemektedir. Yani teknik açıdan hiçbir merkezi sisteme bağımlılık yoktur.

Fakat burada merkezi bir sisteme bağımlılık ancak tasarım sebebiyle söz konusu olabilir. Belli bir sistem çöktüğünde eğer diğer sitemdeki işler aksıyorsa bu özellik sağlanmıyor demektir. Tasarım belli bir merkez sistemi dayanak almayacak şekilde yapılır ve veriler buna göre dağıtılırsa bu özellik sağlanmaktadır.

Bu özelliğin sağlanması için veritabanı yöneticilerine büyük görev düşmektedir. Verilerin dağılımı ve yapısı öyle şekilde yapılmalıdır ki, belli bir işlem başka bir sistemdeki sorundan dolayı başarısızlığa uğramadan gerçekleşsin.

Sonuç olarak doğru tasarım ile bu özellik sağlanmaktadır.

3.11.3. Sürekli İşletim :

Bu özellik sağlanmaktadır. Bütün yerel sistemler, bir başka bir sistemdeki kesintiden dolayı veya sisteme yeni bir makine ve veritabanı eklenmesinden dolayı işlerine ara vermeden çalışmalarına devam edebilmektedir.

Ancak bazı durumlarda planlı kapama gereksinimi doğmaktadır. Bir veritabanı üzerinde yeni alanların tanımlanması, yeni dosyaların eklenmesi, indeks modifikasyonları ve eklemeleri işlemleri o veritabanı üzerindeki çalışmaları durduracak ve değişikliklerle ilgili programlar yeniden derlenmek zorunda kalacaktır. Bu o veritabanını kullanan kullanıcıların beklemesine sebep olacaktır. Aslında bu

normal bir sonuçtur. Bu özelliğin sonucu olarak beklenen bu tür olaylardan sonra değişikliğin olmadığı sistemlerin etkilenmemesidir. Fakat bu durumda istisnai olarak gerçekleşebilmektedir. Örneğin bazı yerel sistemlerde versiyon değişikliği oluyorsa onu kullanmaya mecbur diğer sistemler bu değişikliğin tamamlanmasını beklemek zorunda ve bu veritabanlarını kullanan programlarını yeniden derlemek zorundadır.

3.11.4. Yer Bağımsızlığı:

Bu özellik tamamen sağlanmaktadır. Progress'in sahip olduğu Client-Server mantığı ve veritabanlarının bağlanmasıyla bu özellik oldukça basit bir şekilde sağlanmaktadır.

Bu özelliği sağlamak için dağıtık sisteme ait her veritabanı için iligili broker/server başlatılmalı; bu esnada her sistem için ayrı bir servis kullanılmalıdır. Örneğin:

Dalaman İstasyonu: proserve fatura -H dal -S dalser -ld dlmfat -n 5 .....

İstanbul İstasyonu: proserve fatura -H ist -S istser -ld istfat -n 7 .....

Ankara İstasyonu: proserve fatura -H ank -S ankser -ld esbfat -n 5 ....

. . . gibi.

Yukarıdaki prosesler server proseslerdir. Bu prosesler sayesinde yerel veya uzak Client'lar istedikleri veritabanına bağlantı sağlayabilmektedir. Örneğin Genel Müdürlükteki bir kullanıcı program içerisinden:

connect fatura -ld dlmfat -H dal -S dalser -U DLM -P DLM...

ile "dalser" servisini kullanarak Dalaman veritabanına bağlanabilmektedir. Bağlantı sağlandıktan sonra erişilmek istenen dosya ister yerel (Genel Müdürlük) ister uzak (Dalaman istasyonu) makinede yer alsın sanki yerel makinedeymiş gibi işlem yapılabilmektedir. Örneğin Genel Müdürlükte girilen yeni bir hizmet tanımı ve fiyatını içeren bilgi Dalaman istasyonuna aşağıdaki program parçacığı halinde aktarılabilmektedir.

```
find dlmfat.shizmet of genfat.shizmet no-error.
if not available dlmfat.shizmet
   then create dlmfat.shizmet.
{shizmet.i dlmfat.shizmet genfat.shizmet}
```

Burada genel müdürlükten girilen hizmet bilgisinin Dalaman istasyonundaki hizmet dosyasında (shizmet) olup olmadığı kontrol ediliyor. Eğer yoksa yeni bir hizmet kayıdı yaratılıyor. shizmet.i include prosedür de Dalaman istasyonuda konumlanılan shizmet kayıdına genel müdürlükteki verileri assigm ediyor. Görüldüğü gibi Genel Müdürlük sistemi üzerinden, Dalaman veritabanı kayıtları üzerinde sanki kendi veritabanı üzerindeymiş gibi işlem yapılabilmektedir.

3.11.5. Dilimlenme Bağımsızlığı:

Bu özellik otomatik olarak sağlanamamaktadır. Veriler ancak yaratıldığı sistem üzerinde saklaanabilmektedir. Dolayısıyla verilerin fizksel erişim sebeplerinden olarak belli bir ilişki bazında otamatik olarak en çok kullanıldıkları yerde saklanacak şekilde dilimlenmesi söz konusu değildir.

Bir veritabanını farklı disk birimleri üzerinde farklı parçalara ayırma (Multi Volume Databases) özelliği mevcuttur. Fakat Burada verilerin ilişkiler bazında belli yerlerde sakalanbilmesi mümkün değildir. Progress böyle bir mantık gözetmeksizin, disk maliyetinin performans üzerine getirebileceği olumsuz etkileri engellemek açısından belli bir mantık gözetmeksizin verileri farlı veritabanı parçalarında saklamaktadır.

Fakat uygulmayı geliştiren kişi tarafından veritabanı içerisine hangi verinin nerede saklandığı göstergeleri konulursa bu özellik kısmen sağlanabilir.

3.11.6. Kopyalama Bağımsızlığı :

Belli bir ilişki veya ilişkinin belli bir kısmını kullanım amacıyla otomatik olarak başka sistemlere aktaran bir araç yoktur. Fakat bu şekildeki veriler programcı tarafından basit program parçacıklarıyla diğer sistemlere kopyalanabilir.

Örneğin yeni bir firmayla yapılan anlaşma bilgileri aşağıdaki şekilde daha etkin bir kullanım sağlamak açısından aşağıdaki gibi kopyalanabilir.

```
prompt-for genfat.firma.uh-kod.
    ---------
    ---------
For each genfat.fiyat no-lock
    where genfat.fiyat.uh-kod = input genfat.firma.uh-kod :
    find istfat.fiyat of genfat.fiyat no-error.
    if not available istfat.fiyat
    then create istfat.fiyat.
```

```
/*****************************************/
   {fiyat.i  istfat.fiyat genfat.fiyat}
/* Fiyat İstanbul veritabanına assign ediliyor.*/
end. .
```

Yukarıdaki program parçacığından ekrandan girilen firmaya ait fiyat skalası İstanbul İstasyonuna kopyalanmaktadır. Bu kopyalama çeşidi daha farklı yöntemlerle de olabilmektedir.

3.11.7. Dağıtık Sorgu İşleme:

Bu özelliğin sağlanabilmesi kullanılan network performansı ile yakından ilgilidir. Eğer network hızı düşükse yerel sistem üzerindeki sorgulama ile, uzak bir sistem üzerinden yapılan sorgulama arasında çok vakit olarak çok büyük farklılıklar doğabilmektedir.

Bu sorun özellikle Coğrafi bir alan üzerinde yayılmış sistemlerde söz konusu olmaktadır. Fakat network yeteri kadar hızlı olduğu takdirde Progress Broker'ı sorugulama hangi sitemden istenirse istensin gerekli optimizasyonu yapmaktadır. Prosesleri ve dolayısıyla sorgulamaları yönlendiren Broker'dır.

Bir sorgulamanın performansı kullanıcı tarafından da kullanılan programlama tekniklerine bağımlı olarak değişebilmektedir.

3.11.8. Dağıtık Hareket Yönetimi:

Bu özellik iki aşamada değerlendirilebilir: Kurtarma Denetimi ve eşanlılık denetimi.

Bu iki özellik de sağlanmaktadır. Dağıtık sistem üzerindeki hareketler iki aşamalı taahhüt mabtığıyla atomik olarak icra edilmektedir. Uzak bir sistem üzerinde yapılan harekette, hareketle beraber hareketin başarımı ile ilgili sistemler arasında mesajlar gidip gelmekte, eğer başarıyla sonuçlandığı taahhüt edilirse hareketin tamamı kabul edilir. Eğer bu taahüt gerçekleşmez ve hareket yarıda kesilirse o hareket içerisindeki bütün değişiklikler geri alınır ve kabul edilmez . Örneğin bir güncelleme işlemi bütün sistemler üzerinde tek bir hareket içerisinde yapılıyorsa, yalnız bir sistemden kaynaklanan sorun yüzünden hareketin başarıyla sonuçlandığı taahhüt edilmediğinden dolayı bütün yapılan güncellemler iki aşamalı taahhüt mantığı gereğince geri alınacaktır. Dolayısıyla böyle bir vakit kaybından kurtulmak için hareketeleri ufak parçalara bölmekte fayda vardır.

Kurtarma denetimi gibi aynı anda kullanılan kaynaklar için eşanlılık denetimi yapılmaktadır. Bu denetim normal dağıtımlı olamayan sistemlerde olduğu gibi kilitleme mekanizması ile gerçekleşmektedir. Yerel özerkliğin de bir sonucu olarak her sistem kendi kilitleme mekanizmasından sorumludur. Bunu her yerel veritabanının Broker'ı idare etmektedir. Fakat bu idare yalnız kendi yerel veritabanı üzerinde gerçekleştiğinden dolayı farklı sistemlerdeki kilitlemelerden haberdar olamamakta ve farklı sistemlerin kilitleme dolayısıyla bir başka sistemdeki kilidin kaldırılmasını beklemesinden dolayı bölüm 1'de anlatılan ölümcül kilitlenmeler gerçekleşmektedir.

Bu tür kilitlenmelerden kurtulamak ancak TCP/IP içerisinde mevcut olan belli bir süre boyunca hiçbir mesaj alınamamasından dolayı meydana gelen time-out sayesinde veritabanları arasındaki bağlantının kopmasıyla mümkün olabilmektedir. Ancak bu durumda da bağlantısı kopan veritabanına yeniden bağlanmak gerek-lidir. Bu durumda hareket yarıda kaldığı için bütün değşiklikler geri iade edilecektir.

3.11.9. Donanım Bağımsızlığı:

Bu özellik büyük ölçüde sağlanmaktadır. Progress ile geliştirilen bir uygulama aynı işletim sistemine sahip donanım cinsi ve markası ayırt edilmeksizin bütün makineler üzerinde çalışabilmektedir. Yani bir uygulama farklı donanımlara taşınabilirdir.

Yalnız donanımların teknik performans, kapasite ve sınırlamalarından dolayı veritabanı başlangıç parametrelerinin uygun seçilmesi gerekebilir. Fakat sonuç olarak belli bir uygulama aynı işletim sistemi üzerindeki başka bir donanım üzerinde; herhangi bir teknik sınırlama olmadığı takdirde çalışabilirdir.

Aynı şekilde farklı bir donanımdan, başka bir donanım üzerindeki Progress veritabanı üzerinde işlem yapılabilmektedir. Örneğin X marka bir donanım üzerindeki kullnıcı hem kendi veritabanı üzerinde hem de Y marka başka bir makindeki Progress veritabanı üzerinde işlem (sorgulama, güncelleme vs.) yapabilmektedir.

Bu özelliğin sağlanması için gerek koşul işletim sisteminin aynı olmasıdır.

3.11.10. İşletim Sistemi Bağımsızlığı:

Bu özellik kısmen sağlanmaktadır. Her işletim sistemi için ayrı bir Progress Compiler veya Run-time gereklidir. Örneğin DOS işletim sistemi altındaki bir veritabanı, doğal olarak Unix İşletim siteminde kulanılamamaktadır. Bu aslında her

işletim sisteminin farklı bir dosyalama sistemine sahip olmasından kaynaklanmaktadır. Her bir işletim sitemi için ayrı Progress modülleri gerekmektedir.

Ancak bir veritabanı farklı bir işletim sistemine dolaylı da olsa, ASCII formattaki dosyaların bütün işletim sistemlerince tanınabilir olması özelliğinden dolayı taşınabilirdir. Belli bir veritabanının dosya, indeks, alan, kullanıcı vs. tanımları ASCII formatta bir dosyaya atılıp; Bu tanım dosyası (.df) farklı işletim sistemlerinde boş olarak yaratılmış veritabanlarına yüklenerek aynı yapıdaki veritabanı oluşturulabilir. Verileri de kopyalamak için: her dosyaya ait veriler ASCII formattaki dosyalara atılıp bu dosyalar içerisindeki bilgiler farklı işletim sistemindeki veritabanlarına yüklenebilir. Progress sözlüğü içerisinde bu işlemler için özel utility'ler mevcuttur.

Yine İşletim sistemi bağımsızlığı için sorun çıkaran özelliklerden birisi de ; belli bir işletim sistemindeki derlenmiş bir programın başka bir işletim sistemi üzerinde çalışamamasıdır. Farklı bir işletim sistemindeki aynı yapıdaki veritabanını kullanmak için programın o veritabanı için derlenmesi gerekmektedir.

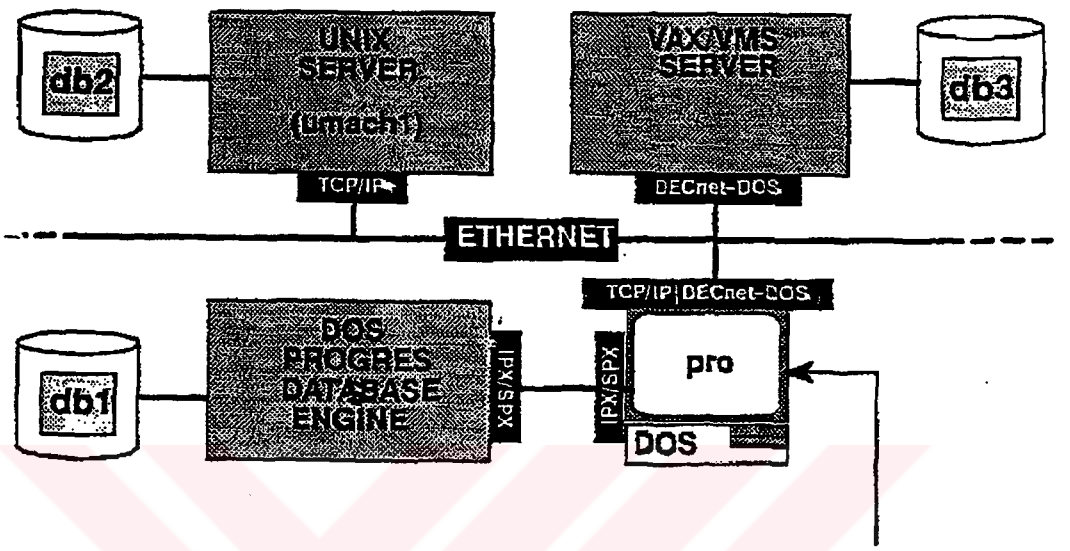
Fakat sonuç olarak farklı farklı işletim sistemleri üzerindeki veritabanları Dağıtık bir siteme eşit ortak olarak katılabilmektedir. Unix işletim Sistemindeki bir veritabanı başka bir işletim sitemindeki (örneğin VMS) veritabanına bağlantı yapıp gerekli sorgulama, güncelleme vs. işlemlerini yapabilmektedir.

3.11.11. Ağ Bağımsızlığı:

Belli bir Progress veritabanı farklı ağ protokolleri altından geçerek diğer Progress veritabanlarına bağlanabilmektedir. Gerekirse Ethernet'le birbirine bağlı makinelerden oluşmuş yerel bir network üzerindeki veritabanları üzerinde ve gerektiğinde bir WAN (Wide Area Network) üzerinde telefon hatlarını kullanarak TCP/IP vasıtasıyla veya başka bir protokol vasıtasıyla Coğrafi alan üzerindeki bir veritabanı üzerinde işlem yapabilmektedir.

3.11.12. VTYS Bağımsızlığı:

Bu özelllik Progress içerisinde ilgili modülleri yazılmış VTYS'ler için geçerli olmaktadır. Yani Progress harici bir veritabanı için gerekli arayüzler mevcut Progress versiyonunda varsa, bu veritabanları ile haberleşme mümkün olmakta, dağıtık siteme eş bir ortak olarak katılım sağlayabilmektedirler. Bu Progress paketi içerisindeki gateway'ler ile sağlanmaktadır.



Şekil 3.11.1 Örnek Bir Progress Dağıtık Veritabanı Senaryosu

BÖLÜM 4. UYGULAMA HAKKINDA BİLGİ

Bu bölümde tez konusu olan uygulama hakkında açıklayıcı bilgiler yer almaktadır. Havaş GHCN - Faturalama sistemi hakkında bilgi verilmekte nasıl bir organizasyon tasarlandığı anlatılmakta ve dağıtıklığa ilişkin temel program örnekleri verilmektedir.

4.1.Proje ve Konusu

Havayolu şirketlerine verilen hizmetler, cash veya kredili olarak faturalanarak tahsil edilmektedir. Dolayısıyla, şirketlerle yapılan anlaşmalar da cash veya kredili anlaşma şeklindedir. Anlaşma süresince bir şirketten anlaşmaya uygun olarak ya hep cash ya da hep kredili olarak tahsilat yapılır. Anlaşması olmayan şirketlerden ise ücret, ilgili uçuş için verilen hizmetler bedeli kadar cash olarak tahsil edilir. Cash anlaşmalı şirketler için tahsilat, hizmet verilen uçak yerdeyken ilgili uçuş esnasında verilen hizmetlerin bedeli kadar yapılır. İlgili uçuş için Ground Handling Charge Note denen hizmetleri gösteren form ve fatura karşılığı tahsilat yapılır.

Kredili anlaşma yapılmış şirketlerde ise beli bir periyodik dönemde o havayolu şirketine verilen hizmetler bedeli olarak ücret tahsil edilir. Bu tür şirketlere, o periyodik dönem içerisinde her uçuş için almış olduğu hizmetleri ve bunların indirimsiz fiyatlarını gösteren hizmet cetveli, şirket anlaşmasına göre yapılan indirimlerin yansıtıldığı fatura , her uçuş için verilen hizmetleri gösterir Ground Handling Charge Note denilen hizmet formları , fatura ve hizmetlerle ilgili mektup gönderilir. Gönderilen mektupda kanunen geçerli olan bir son ödeme tarihi verilir ve bu süreye kadar ücretler tahsil edilir. Havayolu şirketinin isteğine göre belli bir hizmete, belli bir uçuş tipine vs. göre ayrı faturalar kesilebilir.

Kredili ve cash faturaların kesildiği yerler bellidir. Hizmetin verildiği istasyonlar yalnızca Cash fatura kesme yetkisine sahiptir. Ayrıca her istasyon yalnız kendi verdiği hizmetler karşılığında fatura kesmekte yalnız kendi verdiği hizmetler için Ground Handling Charge Note tanzim etmektedir. İstasyonlar yalnız Cash anlaşmalı şirketler için fatura kesmektedir. Fakat Cash veya Kredi anlaşmalı şirket olması farketmeksizin bütün şirketler için Ground Handling Charge Note düzenlemektedir.

Kredili fatura kesme yetkisi ise yalnızca Genel Müdürlüğe aittir. Her istasyon için ayrı olmak üzere; Kredili şirketlere ait gelen GHCN'lara göre, belli periyod içerisinde ilgili istasyonda verilen hizmetlere istinaden faturalama yapılır. Verilen hizmetlere ilişkin GHCN'lar hizmetlerin verildiği istasyonlardan; faturalar, hizmet cetvelleri ve mektuplar Genel Müdürlük'ten çıkarılır.

Havayolu Şirketleriyle anlaşmalar Genel Müdürlük tarafından yapılır. İstasyonlar Cash faturaları, yapılan anlaşmalar doğrultusunda keserler. Ayrıca her istasyon kendisiyle ilgili verilen hizmetler, GHCN'lerle ilgili istatistiki bilgileri ve çeşitli raporları çıkarır. Bazı istatistiki bilgileri Genel Müdürlüğe gönderir. Genel Müdürlük ise istasyonlardan gelen bazı istatistiki bilgiler dışında, genel olarak birtakım envanter ve istatistiki raporlar hazırlar.

Genel olarak işler, yapıldığı yerden bağımsız olarak düşünüldüğünde üç ana bölümde kategorize edilebilir.

Şirketlerle anlaşmaların yapılması:

Genel Müdürlük hizmetler bazında her havayolu şirketiyle ayrı bir anlaşma imzalar. Anlaşmalar yukarıda da belirtildiği gibi ya cash ya da kredilidir.

Fiyatlandırma açısından verilen hizmetler üç ana grupta toplanabilir. a) Temel Hizmetler: Bu tür hizmetler değişken kriterlere göre fiyatlandırılmaktadır. Fiyatlandırmaya etkileyen temel kriterler uçağın tonajı, firmanın yerli veya yabancı olduğu,zamlı tarifeye girip girmediği ve uçuşun dış hat olup olmadığıdır. Bunun yanında yine her şirket için ayrı olmak üzere her uçuş tipine, uçuş yerine vs. göre farklı fiyatlandırmalar yapılabilmektedir.

Havaş ile anlaşması olmayan şirketler için IATA tarafından kabul edilmiş yine uçak tonajı, iç hat veya dış hat olması, yerli veya yabancı firma olmasına vs.göre değişen fiyat tarifesi uygulanır.

İstasyonlarda hizmet verilirken bu tür hizmetler için o şirket için uygulanan tarife neyse o fiyat alınmak zorundadır. On üç tonaj dilimi mevcuttur. Normal standard tarifeye göre her tonaj diliminde farklı ücretler vardır. Bir şirketle öyle bir anlaşma yapılabilir ki; ilk dört tonaj dilimi için örneğin trafik ve ramp adlı hizmetlerden hep aynı fiyat alınsın boş uçuşlarından trafik ve ramp ücreti alınmasın veya kargo uçuşları için aynı tonajdaki normal yolcu uçağına göre farklı fiyatlandırma yapılsın vs.. Bunun gibi farklı şekillerde farklı anlaşmaların olması söz konusudur. Bu tür hizmetler de adet veya saat cinsinden tutar yoktur. Hizmet verilip verilmediği önemlidir. Hizmet verildiyse o şirkete ait fiyat tarifesindeki ücret neyse o alınır.

Genellikle her şirket belli hizmetleri alır.Ama bazen alınan hizmetler değişebilir. Bu hizmetlerin hepsi apron içi hizmetlerdir. Genel indirime dahildirler.

b) İsteğe Bağlı Hizmetler:

Bu tür hizmetler temel hizmetler gibi devamlı alınan değil duruma göre istekte bulunulup alınan hizmetlerdir. Bu tür hizmetler için de firma bazında yapılan anlaşmalar söz konusudur. Yalnız fiyat tarifesini oluşturan hizmetleri belirleyen faktörler temel hizmetlere göre daha belirgindir. Bu faktörler, firma ve adet veya süredir. Bu faktörlere göre her firma için bir fiyat tarifesi mevcuttur. Hizmet verilirken hizmet miktarı (adet,süre vs.) belirtilmesi esastır. Genelde alınacak ücret hizmetin fiyat tarifesi * miktar şeklindedir.

Fiyatlandırma dışında diğer önemli bir nokta bu hizmetlerin bazılarının genel indirime dahil olması bazılarının da olmamasıdır.

Bu tür hizmetlerin hepsi apron üzerinde verilmektedir.

c) Dışarıdan Sağlanan Hizmetler:

Apron dışında verilen veya başka bir şirket tarafından karşılanan belli bir ücret tarifesi olmayan hizmetlerdir. Bu hizmetler de kendi içerisinde ikiye ayrılmaktadır. 1) Havaş tarafından dışarıdan sağlanan hizmetler. 2) DHMİ hizmetleri.

Havaş tarafından sağlanan hizmetler şirketin o uçuşuyla beraber verilen diğer hizmetlerle beraber fatura edilir. Hizmet apron dışında verilmiş veya farklı bir kuruluş tarafından sağlanmıştır. Ama yapılan hizmet Havaş adınadır. DHMİ hizmetleri ise DHMİ tarafından verilmektedir ve DHMİ adınadır. Havaş yalnızca belli bir komisyon alarak müşterisine DHMİ adına fatura kesmektedir. Bu tür hizmetler genel indirime dahil değildir.

Hizmet bazında oluşturulan ücret tarifeleri ham ücretlerdir. GHCN'lerde yer alan fiyatlar da bu ham fiyatlardır.Tek bir GHCN'ye veya birden fazla GHCN'ye göre faturalama yapılırken her şirketle yapılan özel anlaşmalarla belirlenen hemen hemen her şirkette değişen indirimler uygulanmaktadır. Üç tür indirim vardır. Yapılış sırasına göre şöyledirler.

i) Trafik + Ramp (Üçlü) indirimi: Şirketten alınması gereken trafik ve ramp hizmeti ücretlerinden yapılan yüzde indirim. Bu indirim sadece üç ayaklı uçuşlarda yapılır.

ii) Trafik İndirimi : Trafik hizmetinden yapılan indirim yüzdesi.

iii.)Genel İndirim: Genel İndirime dahil hizmetlerden yapılan indirim yüzdesi.

Bu indirimler sırayla yapılır. Görüldüğü gibi eğer üç indirim de uygulanırsa trafik hizmetinden üç defa arka arkaya indirim yapılacaktır. Örneğin indirim oranları sırasıyla %10 %50 %10 olsun: trafik için şirketten ham olarak 1000 $ alınması gerekiyorsa indirim yapıldığında (1000 - .10*1000) - ((1000 - .10*1000) * .50) - (((1000 - .10*1000) * .50) * .10) $ fatura üzerinde trafik hizmeti ücreti olarak yer alacaktır.

Ground Handling Charge Note'ların (GHCN) oluşturulması:

GHCN formları, yalnız istasyonlar tarafından düzenlenmektedir. Bu formlarda hizmet verilen uçuşa ilişkin bilgiler, firmaya ait bilgiler, uçağa ait bilgiler ve hangi hizmetlerin verildiği gibi bilgiler vardır. Temel ve isteğe bağlı hizmetler için GHCN'yi tanzim eden kişinin yorum hakkı yoktur. Hangi hizmetten ne kadar aldığı veya alınıp alınmadığı işaretlenmekte, fiyatlandırma bu bilgiye göre o firmanın ücret tarifesinden gerçekleşmektedir. Hizmetin verildiği uçuş için üçlü indirim (trafik + ramp) yapılıp yapılmayacağı bu form üzerinde belirlenir.

3) Faturalama :

Hizmet ile ilgili faturalama yapılırken GHCN'lar baz alınır. Cash Şirketler için her bir GHCN'ye bir fatura karşılık gelir. Kredili şirketler için ise belli bir dönemdeki GHCN'ların tamamı tek bir fatura üzerinde toplanır. Faturalarda şirketlerle yapılan özel ücretlendirmelerden başka ayrıca gerektiği zaman üç tür indirim uygulanır. Ayrıca GHCN dışında kesilen iş fişleri denilen belgelere dayanan ve hizmet dışı sebeplerden ötürü kesilen faturalar vardır.

Konuyla ilgili iş dağılımı kısaca yukarıda anlatıldığı gibidir. Bu organizasyon yapısı içerisinde, havayolu şirketleriyle yapılan anlaşmaların yanlış yorumlara uğramadan GHCN ve faturaların düzenlenmesini hatasız ve kısa sürede tamamlamak, verilen hizmetler karşılığı kesilen faturaların muhasebeleşmesini on-line ve hatasız yapmak, verilen hizmetlerle ilgili istatistiki bilgileri daha sağlıklı almak için ilgili proje gerçekleştirilmiştir.

4.2. Neden Dağıtık Bir Organizasyon

Yukarıda da anlatıldığı gibi yapılan işler istasyonlar ve Genel Müdürlük bazında farklılıklar göstermektedir. Genel Müdürlük'te Anlaşmalar Dairesi Başkanlığı Şirketlerle anlaşmalar imzalamakta, İstasyonlar uçaklara verilen hizmetlerle ilgili GHCN'leri tanzim etmekte ve hizmet verilen uçak Cash anlaşmalı bir şirkete ait ise

ilgili GHCN'ye ilişkin faturalama yapmakta, Genel Müdürlük'te Mali İşler Dairesi Başkanlığı ise Kredili Anlaşmalı şirketlere ait GHCN'lere istinaden aylık veya haftalık bazlardaki döneme ait faturalama işlemlerini yapmakta ve bunların muhasebeleşmesini gerçekleştirmektedir.

Görüldüğü gibi bazı işlemler yalnız belli yerlerde yapılmaktadır. Örneğin şirkerlerle yapılan anlaşmaları yalnız Genel Müdürlük Anlaşmalar Dairesi Başkanlığı yapmakta; verilen hizmetlerle ilgili bilgileri yalnız istasyonlar girmekte ve her istasyon yalnız kendi verdiği hizmet bilgilerini girmeye yetkili olmakta ; Kredili şirketler için faturalama istasyonlar bazında yalnızca Genel Müdürlük Mali İşler Dairesi Başkanlığı'nca yapılmakta, Cash anlaşmalı şirketler için Faturalar yalnız hizmetin verildiği istasyonlar tarafından kesilmekte; bütün faturaların muhasebeleşmesi Genel Müdürlük Mali İşler Dairesi Başkanlığı'nca yapılmaktadır. Ayrıca her istasyon kendi verdiği hizmetlere ilişkin; Genel Müdürlük ise Kredili firmalara ve istasyonlara ilişkin hizmet cetvelleri ve çeşitli istatistiki raporları oluşturmaktadır.

Yapılan işlemler farklı birimlerde gerçekleşmesine rağmen hepsi birbiriyle ilişki içerisindedir. Örneğin istasyonlar verilen hizmetlerle ilgili GHCN'leri tanzim ederken yalnız hangi hizmetin verildiğini ve kaç birim verildiğini belirtme yükümlülüğü içerisindedirler ilgili fiyatlandırmalar tamamıyla Genel Müdürlük Anlaşmalar Dairesi tarafından o firma için ilgili uçuşa istinaden yapılan anlaşma baz alınarak yapılmaktadır; Faturalar ise GHCN'larda yer alan ham fiyatlar üzerinden yine Genel Müdürlük Anlaşmalar Dairesi tarafından ilgili firma için yapılan indirimler baz alınarak kesilmektedir. Mali İşler Dairesi Başkanlığı ise Kredili şirketlere ilişkin faturalamayı yaparken istasyonlar tarafından tanzim edilen GHCN'leri ve ilgili indirimleri yapmak için Anlaşmalar Dairesi tarafından yapılan anlaşmaları baz almaktadır.

Görüldüğü gibi her birim yeri geldiği zaman özerk olarak çalışmakta yeri geldiği zaman diğer birimlerle bağlantılı çalışmaktadır. Böyle bir organizasyona en uygun yapı dağıtık organizasyon yapısıdır: Her birim kendine ait işleri yaparken özerk çalışmalı; fakat gerektiği zaman diğer birimlerle bağlantılı olarak çalışabilmelidir. Getirilen başlıca teknik faydalar ise şöyledir:

1) Her sistem özerk çalışdığından dolayı, o sistem üzerindeki her hareket merkezi bir sisteme gitmek zorunda kalmamakta ve dolayısıyla olası network maliyeti oldukça azalmaktadır.

2) Yine her sistem özerk çalışdığından dolayı, yerel bir işlem yalnız kendi sistemine bağımlı olmakta ve dolayısıyla network yavaşlığı veya problemlerinden dolayı bu işlemin geç gerçekleşmesi veya başarısızlığa uğraması ihtimali azalmaktadır. Dolayısıyla özellikle istasyonlar üzerindeki performans merkezi bir yapıya göre çok daha yüksek olmaktadır.

3) Mümkün olduğu kadar hiçbir sistem (istasyon, Genel Müdürlük) kendi işini yapmak için bir diğerine bağımlı olmamaktadır. Örneğin Genel Müdürlük'teki sistemde işleyişin durması Antalya istasyonunda hizmet verilmesini etkilememektedir.

4) İşlem yükü dağıtıldığından dolayı merkezi bir sisteme aşırı bir yük binmemekte, dolayısıyla bu durumdan kaynaklanan bir darboğaz yaşanmamaktadır. Her sistemin kapasite ve performansı kendi yerel işlemleri doğrultusunda tespit edilmektedir.

5) Madde 4'ün bir sonucu olarak; çok güçlü bir merkezi makine olmasına gerek kalmamasından dolayı donanım maliyetleri de azalmaktadır.

6) Verilerin dağılımında bir optimizasyon söz konusu olmaktadır. Veriler daha çok kullanıldıkları yerlerde yer almaktadırlar. Fakat gerektiğinde bazı veriler birden fazla yerde yer alabilmektedir.

4.3. Görev Dağılımı

GENEL MÜDÜRLÜK - Anlaşmalar Dairesi Başkanlığı:

Bu birim, havayolu şirketleri ile yapılacak olan anlaşmanın içeriğini tespit etmekte ve GHCN' ların düzenlenmesi , Cash ve Kredili faturaların kesilmesi, hizmet verilen havayolu şirketleriyle yapılan yazışmalar için gerekli bilgilendirmelerin eksiksiz ve vaktinde yapılması görevini üstlenmektedir.

Bir şirket için GHCN'lerin düzenlenmesi ve faturaların yalnışsız olarak kesilmesi ve ilgili yazışmaların yapılabilmesi için ilk önce o şirketle ilgili sabit bilgilerin girilmesi gereklidir. Bu bilgiler Anlaşmalar Dairesi Başkanlığı tarafından girilir. Girilen bilgiler: firmalarla yapılan indirimler, Komisyon yüzdeleri, firma adres, telefon vs. bilgileri, temel hizmetlerle igili her tonaj dilimi ve herbir tonaj dilimi içerisindeki farklı pozisyonlar için (Teknik İniş, Tarifeli Uçuş, Boş Uçuş, Kargo Uçuşu, iç hat ve dış hat uçuşu olması vs.) fiyatlandırmaları, request hizmetleri için alınacak ücretler ve bu hizmetlerin indirme dahil olup olmaması ve diğer anlaşma bilgileridir.

Yeni bir hizmet türünün ve ilgili bilgilerin girilmesi, bu hizmetlerin tüm firmalar bazında fiyatlandırılması, Türkiye ve Dünya üzerindeki havaalanları ile ilgili bilgilerin girilmesi ,bu bilgilerle ilgili raporların oluşturulması yine bu birim tarafından yapılmaktadır.

Bu birim tarafından girilen bilgiler diğer bütün işlemler için baz olmaktadır. Ücretlendirmeyi yapan ve faturaları kesen birimlerin bu birim tarafından yapılan fiyatlandırmalar dışına çıkması , fiyatları değiştirmesi veya anlaşma dışı indirim yapması söz konusu değildir. Şirket ve anlaşmalara ilişkin bilgilerin idare edildiği tek birimdir. Bu birim, ilgili işlemler yapılırken arka planda kontrol edilen temel bilgilerin bu birim tarafından girilmesinden dolayı, işlemlerin vaktinde ve hatasız gerçekleşmesi için hayati önem taşımaktadır.

Mali İşler Dairesi Başkanlığı:

Bu birim, kredili anlaşması olan şirketler için Anlaşmalar Dairesi tarafından girilen sabit bilgiler ve istasyonlar tarafından girilen GHCN bilgileri doğrultusunda faturalama işlemini yapmakta, her faturanın içerdiği hiznet bilgilerini içeren hizmet cetvellleri oluşturmakta, bunları belli bir son ödeme tarihi ile şirketlere mektupla yollamakta ve ayrıca faturaların muhasebeleşmesi için firma ve hizmeterle ilgili muhasebe kodları, maliyet merkezi kodları firma teminat tutarı vs. bilgileri girmektedir. Kredili faturalar, fatura kesildikten hemen sonra on-line olarak muhasebe kayıtlarına işlenmektedir. Yine bu bölümde GHCN harici faturalandırmaya cevap veren ayrı bir "Extra Faturalama" modülü kullanılmaktadır. Havayolu şirketleri için kesilenleri yine otomatik olarak muhasebeye işlenmektedir.

Faturalar belli periyodlar içerisinde ve firma bazında ; firmanın isteğine göre İç hat, dış hat, Boş Uçuş, bütün uçuşlar için vs. kesilebilmektedir. Bunun için istasyonlarda yer alan faturanın kesileceği döneme ait GHCN'ler baz alınmaktadır: Bir firma beli bir istasyona 60 uçuş yapmışsa o firmaya ait aldığı hizmetleri ve birtakım uçuş bilgilerini gösteren 60 adet GHCN düzenlenmiştir. Bu firma İç hat Uçuşlarını ve dış hat uçuşlarını tarifeli ve boş uçuşlar bazında faturalandırılmasını isteyebilir. Bu durumda o döneme ait GHCN'lerden otomatik olarak bu istekler doğrultusunda program tarafından ayrıştırma yapılarak faturalar kesilmekte ve ilgili hizmet cetvelleri, mektuplar oluşturulmaktadır. Eğer 10 uçuş dış hat tarifeli uçuş ise, dış hat tarifeli uçuş faturası yalnız ilgili 10 GHCN baz alınarak , 5 adet boş uçuş söz konusu ise ilgili 5 GHCN baz alınacaktır.

Ayrıca istasyonlar bazında sorgulama yapılarak verilen hizmet ve GHCN'lere ilişkin istatistiki bilgiler de bu birimden alınmaktadır.

Görüldüğü gibi bu birim kredili faturaların kesilmesi ve muhasebeyle ilişkili işlerin takibi sorumluluğunu almıştır.

İstasyonlar:

Network'e bağlı istasynlar İstanbul, Ankara, İzmir, Antalya ve Dalaman'dır. Her istasyonun görev ve sorumululuğu kendi verdiği hizmetlerle ilgilidir. Her istasyon kendi yerel yapısı içinde aynı işlemleri yapmaktadır.

Sabit uçak bilgilerinin girilmesi , verilen hizmet bilgilerinin girilmesi ve GHCN'lerin oluşturulması , Cash anlaşmalı şirketlere faturaların kesilmesi ve yerel yapılarındaki hizmet ve GHCN'lere ilişkin raporlamaların yapılması istasyonlar tarafından gerçekleştirilmektedir.

Hizmet bilgilerinin girilmesi Harekat ve Apron Teknik Hizmet Bölümü tarafından yapılmaktadır. Bu bilgiler hizmet esnasında uçak yerdeyken yapılmaktadır. Apron üzerinde hizmet verilen her uçak için ayrı GHCN'ler düzenlenmektedir. Yerli havayolu şirketleri için geliş ve gidiş uçuşları için ayrı GHCN'ler, yabancı havayolu şirketleri için geliş ve gidiş için tek bir GHCN düzenlenmektedir.

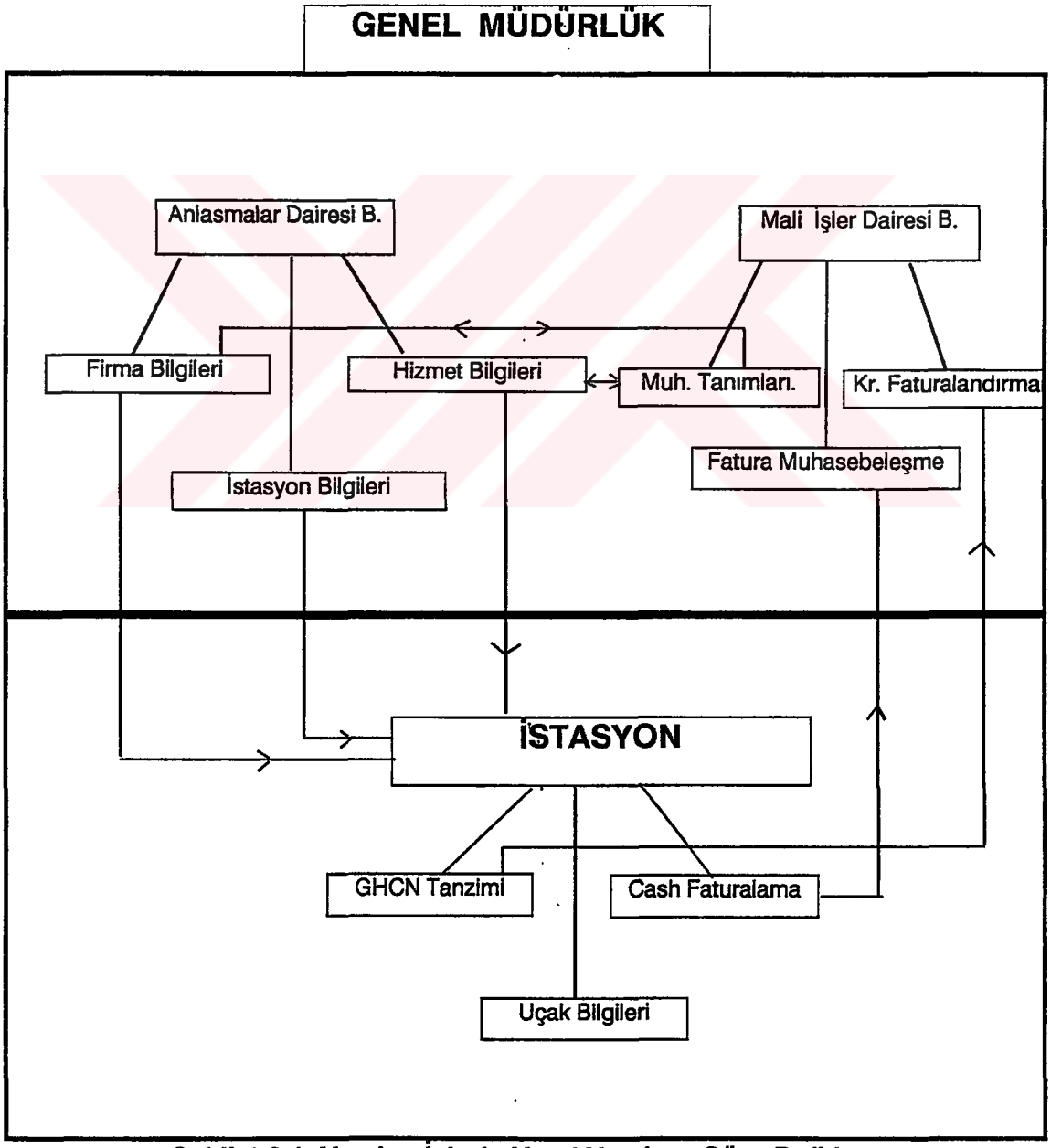
GHCN'lar Harekat Bölümü tarafından düzenlenmektedir. Temel hizmetlerin girilmesini harekat yapmakta, isteğe bağlı hizmetler bilgi girişi ise apron üzerinde farklı yerlerde bulunan Harekat ve Apron Teknik Hizmet Müdürlüğü tarafından yapılmakta ; bu bilgiler direkt ya da dolaylı olarak GHCN'lere kaydedilmektedir.

GHCN için verilen hizmetler girilirken fiyatlandırma otomatik olarak Anlaşmalar Dairesi Başkanlığı tarafından girilen bilgiler doğrultusunda olmaktadır. Kullanıcı sadece hangi hizmetin verildiğini veya kaç birim verildiğini girmektedir. Uçuşla ilgili bilgiler doğrultusunda o firmayla yapılan anlaşmaya göre fiyatlandırma yapılmaktadır. Buradan Anlaşmalar Dairesi Başkanlığı tarafından bilgilerin doğru ve zamanında girilmesinin ne kadar önem taşıdığı görülmektedir. Yanlış veya geç girilen bilgi yanlış fiyatlandırmaya sebep olmaktadır.

Hizmet verilen firma Cash anlaşmalı ise her GHCN için bir fatura kesilir. Fatura kesilirken yapılan indirimler, hangi hizmetlerden indirim yapılacağı yine Anlaşmalar Dairesi Başkanlığı tarafından girilen bilgiler doğrultusunda yapılır. GHCN ve faturanın bir nüshası uçak kaptanına verilir. Faturalama işlemi tahakkuk birimi tarafından yapılmaktadır. Yeri geldiği zaman GHCN harici voucher denilen bel-

gelere göre veya eksik tahsilat için faturalar da kesilmektedir. Bunun için ayrıca "Extra Faturalama" denilen modül kullanılmaktadır.

Kredili şirketler için ise yalnız GHCN düzenlenir ve bir nüshası yine uçak kaptanına verilir. Bu şirketlerle ilgili faturalama işlemi, Genel Müdürlük tarafından, istasyonda tanzim edilen GHCN'ler sorgulanarak yapılır. Periyodik olarak (daha çok aylık bazda) o döneme ait GHCN'ler Anlaşmalar Dairesi Başkanlığı tarafından girilen bilgiler doğrultusunda indirimler yapılarak Genel Müdürlük Mali İşler Dairesi Başkanlığı tarafından faturalanır.



Şekil 4.3.1. Yapılan İşlerin Yerel Yapılara Göre Dağılımı

4.4. Veritabanı ve Uygulamalar Hakkında Bilgiler:

Veritabanı içerisinde kullanılan dosya, alan, kayıt, indeks yapıları sayfa .... da yer alan Progress veritabanı sözlüğünden alınan veri sözlüğü raporunda detaylı bir şekilde görülmektedir. Burada gerekli açıklamalar detaylı olarak incelenmektedir.

Proje içerisinde kullanılan veritabanları yerel yapılar üzerinde homojen dağılım göstermektedir. Yalnız , faturaların muhasebeleşmesi işlemi Genel Müdürlük'te yapıldığı için Genel Müdürlük veritabanında ek birtakım alan ve dosyalar yer almaktadır. Fakat istasyonlarda yer alan veritabanlarının hepsi özdeş ve genel müdürlük veritabanının bir alt veritabanı şeklindedir.

Şekil 4.1. de yapılan işlerin kabaca dağılımı gözükmektedir. Bu yapı içerisinde Genel Müdürlük daha çok kontrol ve organizasyon işlevini görmekte, istasyonlar ise Genel Müdürlük tarafından girilen bilgiler doğrultusunda 24 saat durmaksızın uçaklara verilen hizmet bilgilerini (GHCN) girmekte ve Cash anlaşmalı şirketler için yine Genel Müdürlük tarafından girilen bilgiler doğrultusunda faturalama yapmaktadır. Genel Müdürlük ise istasyonlar tarafından girilen hizmet bilgilerini kullanarak, kredili anlaşması olan şirketler için haftalık, aylık vs. dönemlerde verilen hizmetlere ilişkin firma bazındaki hizmet cetvellerini çekmekte ve gerekli indirimleri yaparak faturaları kesmekte ve bu faturalar anında muhasebeleşmektedir. Bu işlemlerden başka istasyonlar bazında gerekli raporlar alınmaktadır.

Bu bilgi sistemi üzerinde yapılan temel işler ve kullanılan temel dosyalar aşağıdaki gibidir.

Firma Bilgilerinin girilmesi :

Bir havayolu şirketine ilişkin hizmet verilebilmesi için o şirkete ilişkin anlaşmaları ve indirimleri içeren bilgilerin girilmesi gerekmeketedir. Dolayısıyla yeni anlaşma yapıldıktan sonraki ilk bilgi girişi firma bilgileri dosyalarından başlamaktadır. Firma Bilgilerini firma, fiyat, request, adres, ulke dosyalarında yer alan verileri oluşturmaktadır.

Buradaki temel dosya firma dosyasıdır. Bu dosya içerisinde firma kodu (uh-kod) alanı ki bu alanda uluslarası hava taşımacılığında kullanılan üçlü uçuş kodları girilmektedir; firma adı , fatura adresi, şirketin yerli olup olmadığı, anlaşmanın Cash veya kredili olduğu, birinci , ikinci ve üçüncü indirim yüzdeleri, üçlü indirim anlaşmasının olup olmadığı, kargo ücreti vs. alanları yer almaktadır. Firmayla ilgili sabit bilgiler buradan alınmaktadır. Burada özellikle indirimlere ilişkin bilgiler

faturalandırma yapılırken kullanılmaktadır. Her firma için anlaşmaya göre ham ücretler üzerinden farklı indirimler uygulanmaktadır. Birinci indirim, firmadan alınan trafik hizmeti fiyatı üzerinden yapılan indirim; ikinci indirim, firma için indirime tabi olan bütün hizmetlerden alınan fiyatlar üzerinden yapılan indirim, üçüncü indirim ise dış hat uçuşu yapmış ve Türkiye içerisinde iki istasyona uğramış uçuşlar için anlaşma varsa , trafik + ramp hizmetlerinden yapılan indirimdir. Bu indirim bilgilerinin istasyonda farklı vardiyalardan çalışan, kullanıcıların hepsinin bilmesinin zor ve sakıncalı olacağından dolayı gerek Cash faturalandırma yapabilmek, gerekse Kredili faturalandırma yapabilmek açısından vaktinde girilmesi ve istasyonlara dağıtılması büyük önem taşımaktadır. Bu dosya görüldüğü gibi faturalandırma yapmak için hayati önem taşımaktadır.

Fiyat dosyası veritabanı içerisinde belki de en hayati öneme sahip olanıdır. Bu dosya içerisinde Havaş'ın verdiği temel hizmetlere ilişkin farklı durumlar için alınacak ücret bilgileri bulunmaktadır. Bu hizmetler çoğunlukla alındığı için GHCN'ler üzerinde fiyatlandırmanın doğru olabilmesi için bu firmaların eksiksiz ve her duruma cevap verebilecek şekilde girilmesi gereklidir. Belirli bir temel hizmetin fiyatlandırılmasındaki temel kriterler uçuşun iç hat veya dış hat olduğu, teknik iniş yapıldığı, boş uçuş yapıldığ, uçağın tonajı yerli veya yabacı bir firmaya ait olması , vb. unsurlardır. Bunun dışında farklı kriterlere göre de anlaşmalar yapılmaktadır.

Örnek1: Trafik hizmeti tarifeli uçuşlar için 80 tonluk bir uçak için 400 $ iken , Teknik İnişler için 200$, Eğer boş uçuş ise: yerli firma için fiks bütün farklı tonajlı uçaklar için iç hat uçuşu yapmış ise 50 $, dış hat uçuşu yapmış ise normal alınması gereken trafik ücretinin yüzde onu + 100$'dır; eğer firma yabancı ise yine 400$ ücret alınmaktadır.

Örnek2: Bir firma için Tahran'dan gelen yolcusuz gelen veya kargo taşıyan ve yine kargo taşıyarak bir dış hat uçuşu yapan bir evvelki aynı tonajdaki uçak için 300$ trafik ücerti, eğer dış hat uçuşunda yocu taşımış ise 400$ trafik ücreti alınmaktadır. Diğer bütün kargo uçuşları ve tarifeli uçuşlar için 400$ alınmaktadır.

Örnek3: Yine başka bir firmayla yapılan anlaşmaya göre belli bir temel hizmet için ; beş tonun altındaki uçaklar için boş uçuş yapılmısa diğer firmalardan çoğunlukla 50$ alınmasına rağmen ücret alınmamaktadır.

Bütün bu fiyatlar her tonaj dilimi için değişmektedir. Genelede Uluslarası havayolu kuruşları tarafından saptanan fiyatlar alınmaktadır. Fakat yukarıdaki örneklere benzer olarak normalde belli bir tonaj ve belli bir uçuş için 200$ olan bir ücret belli bir havayolu ile yapılan anlaşma gereği 170$ olabilmektedir. Bazen ise

Örnek 2' de olduğu gibi belli bir yerden gelip kargo taşıyarak giden uçaklar için IATA tarafından belli bir ücret tarifesi olmayan çok özel anlaşmalar imzalanabilmektedir. Bu nedenle IATA tarafından belirlenen ücret tarifesi her anlaşma için geçerli olmamaktadır. Fakat bu tarife bilgileri veritabanı içerisinde "tarife" adlı bir dosyada tutulmaktadır. Burada her hizmet, her tonaj dilimi (13 tonaj dilimi), iç hat, dış hat için ücretler yer almaktadır. Yeni bir firma bilgisi girildiğinde tarife bilgilerinden faydalanılarak belli bir fiyat skalası oluşturulmaktadır. Bu tamamıyla kullanıcıya daha az veri girişi sağlamak için yapılmaktadır. Kullanıcı özel anlaşmalı durumları fiyat dosyasında güncelleyerek Havaş'ın fiyat tarifesini oluşturmaktadır.

Fiyat dosyasında firma kodu, pozisyon kodu, tonaj kodu, hizmet kodu, tutar alanları mevcuttur.
Firma kodu: Firma dosyasında yer alan normal IATA üçlü uçuş kodudur.
Pozisyon kodu: Her uçuş tipi için fiyatlandırma yapabilmek için , bu uçuşları farklı pozisyon kodlarıyla temsil etmek gereği doğmuştur. Her firma için en azından altı pozisyon koduna ait fiyatl kayıtları mevcuttur. Bu pozisyonlar iç hat normal uçuş, iç hat boş uçuş, iç hat teknik uçuş, dış hat normal uçuş, dış hat boş uçuş, dış hat teknik uçuşlarıdır.
Tonaj Kodu: Uçak tonajları 13 dilime bölünmüştür. Örneğin 0-5 ton arası uçaklar birinci dilimde, 5-10 ton arası uçaklar ikinci dilimde vs. yer almaktadır. Her tonaj dilimi için 01, 02, ... gibi tonaj kodları verilmiştir.
Hizmet kodu : Ücet alınacak hizmetin sahip olduğu koddur.
Tutar: Belli bir firma, tonaj, pozisyon, hizmete ait alınacak ücret yer almakatadır.

Bir firma için en az 6 temel hizmet, 6 pozisyon kodu, 13 tonaj kodu için fiyat skalası mevcuttur. Yani 13 x 6 x 6 adet o firma için fiyat skalasını belirleyen kayıt mevcutttur. Bunın dışıında her özel durum için yapılan anlaşma, yeni bir pozisyon kodu verilerek fiyatlandırılmakatadır. yani her yeni durum için 6 x 13 adet kayıt oluşmaktadır.

Fiyat skalasının oluşturulmasındaki amaç istasyonlarda farklı vardiyalarda çalışan personelin hizmetleri fiyatlandırırken bütün anlaşma detaylarını öğrenmek gibi bir zorluk çekmemesi ve yanlış yorumlamalarda bulunarak hatalı fiyatlandırma yapmamasıdır. İstasyondaki kullanıcıdan beklenen sadece hizmet verilen uçağın uçuşuna ilişkin bilgilerin girilmesi ve hangi hizmetlerin alındığının işaretlenmesidir. Bu bilgiler fiyat dosyasındaki bir kayıda tekabül etmekte ve otomatik olarak fiyatlandırma hatasız ve çabuk olarak sağlanmaktadır.

Request Dosyası:

Bu dosya kullanım amacı olarak fiyat dosyasına benzemektedir. Bu dosya içerisinde isteğe bağlı alınan hizmetlere ilişkin fiyatlandırma biligileri yer almaktadır. Bu hizmetlerdeki fiyatlandırma uçak tonajı veya uçuş tipine göre değişmemektedir. Fakat belli bir hizmet için her firmaya farklı bir fiyat tarifesi uygulanabilmektedir.

Bu dosya içerisinde request kodu, birim /süre, tutar, indirime dahil mi ve firma kodu alanları yer almaktadır.

Request Kodu: Her geçen gün şirket yeni bir request hizmeti verebilir. Dolayısıyla böyle bir işlem için hizmet tanımlarının girilmesi gereklidir. Bu tanımlar reqfiyat adlı dosyadan girilmektedir. Bu dosyada verilen standard ücret bütün şirketlere (request dosyasına) aktarılmaktadır. Bu alan verilen request hizmetinin ,reqfiyat dosyasıyla bağlantısını sağlayan hizmet kodu alanıdır.

Birim/süre: Request hizmetleri belli bir süre ya da belli adet olarak verilmektedir. örneğin 3 saat "GPU" hizmeti veya 3 adet "Transport Arrival By Bus " hizmeti verilmiş olabilir . Kaç birim kadar veya süre boyunca hizmet verilmişse fiyat * birim olarak alınması gereken ücret hesaplanmaktadır. Örneğin 3 saat GPU 3 * 40 = 120 $ olarak fiyatlandırılmaktadır.

Tutar : Her firma, her request hizmeti için birim başına fiyatlar bu alanda girilmektedir. Her request , her firma için ayrı ayrı kayıtlarda fiyatlandırılmıştır.

İndirime dahil mi: Bazı hizmetler, bazı firmalar için Genel İndirim denilen indirime dahil bazıları ise değildir. Faturalama yapılırken bu bilgiye bakılarak bu hizmet üzerinden indirim yapılıp yapılmamasına karar verilmektedir.

Fiirma Kodu: Bu alanda yine firmanın uçuş kodu yer almaktadır.

Request dosyası GHCN tanzim edilirken kullanıcıyı fiyatları tek tek girme zorluğundan ve yanlış fiyatlandırma sakıncasından korumakta ve faturalandırma yapılırken indirimleri hesaplamada yardımcı olmaktadır. Kullanıcı hangi hizmetten kaç birim verildiğini girmekte ve request bilgilerine göre otomatik olarak fiyatlandırma yapılmaktadır. O şirket için fatura kesilirken, şirketle yapılan bir Genel indirim anlaşması varsa indirimlere dahil request hizmetleri üzerinden indirim yapılmaktadır.

Bütün firma bilgileri her yerel yapı içerisinde yer almaktadır. Fakat bu bilgilerin girilmesi ve değiştirilmesi yetkisi tamamıyla Genel Müdürlü'ğe aittir. Bu bilgilerin

istasyonlarda yer almasının başlıca sebepleri performans ve merkez bir sisteme bağlılık olmaması içindir. Mevcut bilgisayar ağının hızı ağ üzerinde acil on-line transaction'lar için çok yavaş olduğundan dolayı, yerel bir işlemi merkez bir sisteme bağlı kılarak çalıştırmaktansa bu sabit bilgilerin her yerel yapı içerisinde yer alması avantaj sağlamaktadır. Burada belli bir şekilde dağıtık veritabanı yönetim sistemleri içerisinde kulanılan veri kopyalama metodu kullanılmaktadır. Aynı zamanda dağıtıklığın karşı olduğu merkezi bir sisteme bağımlılık yoktur kuralına karşı gelinmemektedir. Eğer bu bilgiler yalnız Genel Müdürlük'te yer alsa, günün 24 saati hizmet verilen istasyonlarda Genel Müdürlük sisteminde meydana gelebilecek bir arıza veya hatlardaki bir sorun yüzünden bütün GHCN ve faturalama ilşlemleri duracak veya herhangi bir sorun olmasa bile işlemler arada bilgisayar ağı olması nedeniyle çok ağır gidecektir.

Burada bilgilerin Genel Müdürlük'ten istasyonlara vaktinde hatasız olarak aktarılması büyük önem taşımaktadır. Bu aktarma yöntemleri bölüm .... da anlatılmaktadır.

Hizmet Tanımlarına ilişkin Bilgilerin Girilmesi:

Verilen hizmetlerle ilgili temel tanımlama bilgileri reqfiyat, shizmet, tarife, diskod, dhmkod adlı dosyalarda tutulmaktadır. Reqfiyat dosyası içerisinde isteğe bağlı request hizmet kodları, tanımları, hizmetin hangi birimler bazında fiyatlandırılacağı, standard fiyatı, muhasebe hesap kodları ve maliyet merkezi kodları yer almaktadır. Bu hizmetlerden belli bir tarifeye göre ücret alınmaktadır. Yeni tanımlanan bir hizmet bütün mevc7ut firmalar için reqfiyat dosyasına standard fiyatlarıyla aktarılmaktadır. Bu hizmetler her firmaya göre değişken tarifeli olabildiği için firma bazında tanımlı oldukları reqfiyat dosyası içerisinden yapılan anlaşmaya göre ücretlendirilmektedirler. GHCN tanzim edilirken fiyatlar reqfiyat dosyası içerisinden alınmaktadır.

Shizmet dosyası içerisinde temel hizmetlere ilişkin; hizmet kodu, muhasebe hesap kodları, maliyet merkezi kodları, hizmet tanımı yer alamkatdır. Fiyatlandırmaya ilişkin sabit bilgiler ise tarife adlı dosyada yer almaktadır. Burada bütün temel hizmetler için mevcut tonaj dilimleri ve iç hat veya dış hat uçuşu olması bazında IATA tarafından belirlenen standard fiyatlar yer almaktadır. Yeni bir firma yaratıldığında, o firma için fiyat skalası olarak başlangıç parametrik değerleri bu dosyadan alınmaktadır sonra bu parametereler fiyat dosyası üzerinde firmayla yapılan anlaşmaya göre güncellenmektedir. GHCN'lar tanzim edilirken temel hizmetlerin fiyatlandırılmasında fiyat dosyası baz alınmakta; fiyat dosyası için firma bazında parametrelere dayalı (pozisyon kodu, tonaj kodu, hizmet kodu, firma

kodu) fiyatlandırma için ise tarife dosyası baz alınmaktadır. Tarife ve fiyat dosyası içerisinde yer alan tonaj kodu alanı ise ücretin hangi tonaj dilimi için geçerli olduğunu göstermektedir.

Diskod dosyası içerisinde dışarıdan sağlanan hizmetlere ilişkin kod, tanım, muhasebe hesap kodları , maliyet merkezi kodları yer almaktadır. Fiyatlandırma için herhangi bir bilgi girilmemektedir. GHCN veya faturalar düzenlenirken fiyatlar o anki durum itibarıyla girilmektedir. Apron dışında verilen veya firma tarafından istenen fakat Havaş dışında bir kuruluş tarafından temin edilen hizmetler bu kategoriye girmektedir. Bu nedenle de belli bir fiyat tarifesine sahip değildirler.

Dhkod ,içerisinde de Dhmi hizmetlerine ilişkin bilgiler aynen dışarıdan sağlanan hizmetler de olduğu şekilde bulunmaktadır. Bu hizmetler DHMİ (Devlet Hava Meydanları İşletmesi) tarafından verilmekte fakat belli bir komisyon karşılığında ilgili firmaya Havaş tarafından fatura edilmektedir.

Hizmetlerle ilgili bilgiler Genel Müdürlük tarafından girilmektedir. Bütün yerel veritabanlarında yer almaktadır. İstasyonlara Karşı yerel sistem üzerinden çalıştırılan bir programla ya da direkt veritabanı bağlantısıyla hizmet bilgileri aktarılmaktadır.

GHCN Bilgileri:

Daha evvel de anlatıldığı gibi faturalandırma için temel bilgileri Ground Handling Charge Note - GHCN denilen bilgiler oluşturmaktadır. Bu bilgiler hem bilgisayarda saklı tutulmakta hem de resmi GHCN formlarına bastırılmaktadır.

Bir GHCN içerisinde GHCN kayıt numarası, pozisyon kodu, hizmet verilen uçağın uçuş bilgileri, firma ismi, uçak tescil bilgisi, geliş gidiş yeri ve tarihleri bilgileri , verilen hizmetler ve bu hizmetlerden alınan indirimsiz ücretler yer almaktadır. GHCN bilgileri iki dosyadan oluşmaktadır. Bunlar Gso ve oreq adlı dosyalardır. Uçuş bilgileri, temel hizmet bilgileri , isteğe bağlı request bilgileri Gso dosyası içerisinde yer almaktadır; bu dosya esas GHCN bilgilerini tutmaktadır. Oreq dosyası ise dışarıdan sağlanan hizmetler ve uçuş bilgilerini tutmakta Gso ile ilişkilendirilme uçuş bilgileri tarafından sağlanmaktadır.

GHCN'ler yalnız istasyonlarda tanzim edilerek print-out'ları alınmaktadır. Fiyatlandırma yapılırken istasyondaki kullanıcı sadece hangi hizmeti verdiği ve kaç birim verdiği hakkında yorum yapabilmekte, fiyatlandırma otomatik olarak firma sabit anlaşma bilgilerini içeren dosyalar kullanılarak gerçekleştirilmektedir. GHCN düzenleyen kullanıcının şirketlerle yapılan anlaşmaları bilmesine gerek

kalmamaktadır. Temel hizmetlerde alınan hizmetler Evet/Hayır şeklinde seçilmekte Evet denilen hizmetler fiyat dosyasındaki sabit bilgilerden faydalanılarak fiyatlandırılmaktadır. Burada önemli olan kullanıcının uçuş bilgilerini doğru ve eksiksiz olarak girmesidir. Fiyatlandırma yapılırken uçuş bilgileri tarafından uygun parametreler (pozisyon kodu, firma kodu, tonaj kodu, hizmet kodu) belirlenmekte ve bunlarla ilgili fiyat dosyasına erişilerek fiyatlandırma yapılmaktadır. Bu fiyatlar üzerinden ayrıca gece saat 22.00 - 06.00 arası, Cumartesi günleri saat 13.00'dan sonra , Pazar ve diğer resmi tatil günlerinde %50 zamlı tarife uygulanmaktadır. Program içerisinden ve tatil günlerinin girilmiş olduğu tatil dosyasından faydalanılarak belirlenen fiyatlar üzerinden zamlı tarife otomatik olarak uygulanmaktadır; tatil dosyasında resmi tatil günleri ve bu tatillerin hangi saatler içerisinde geçerli olduğu bilgisi yer almaktadır.

Uçağın zamlı (surcharge) tarifeye girip girmediği GHCN tarihi ve uçağın yere iniş veya kapı kapanış saatleri baz alınarak yapılmaktadır. GHCN tarihi olarak, varsa uçağın gidiş tarihi; yoksa uçağın geliş tarihi baz alınmaktadır. Yerli havayolu şirketleri için hem geliş hem de gidiş için ayrı GHCN tanzim edilmekte; yabancı havayolu şirketleri için geliş ve gidiş uçuşları için tek bir GHCN düzenlenmektedir. Aşağıdaki örnek fiyatlandırmanın nasıl yapıldığının anlaşılmasında yardımcı olacaktır.

Örnek:

Air France şirketine ait bir uçuş yapıldığı varsayılsın. Bazı uçuş bilgileri şöyle olsun:
Geliş Yeri : Paris
Gidiş Yeri : Tahran
Geliş Tarihi : 31/05/94
Giidiş Tarihi . 01/06/94
Yere iniş saati: 23.30
Kapı kapanış saati: 12.30
Uçuş tipi : S (schedule-tarifeli)
Uçak Tescili : fr112
Belli bir temel hizmet için evet seçildiyse fiyatlandırma yaparken; Geliş yeri ve gidiş yerine bakılarak uçuşun iç hat veya dış hat olup olmadığı kontrol edilecektir. Eğer geliş veya gidiş yerlerinden bir tanesi yabancı ülkedeki bir istasyon veya uçuşu yapan firma bir yabancı havayolu şirketi ise uçuş dış hat uçuşu olarak aksi takdirde iç hat uçuşu olarak değerlendirilecektir. Burada Air France bir yabancı havayolu şirketi olduğu için otomatik olarak dış hat uçuşu olarak kabul edilecektir. Yabancı

bir şirket olmasaydı; Paris ve Tahran'ait istasyon bilgilerine istasyon dosyasından bakılacak her ikisi için de yerli istasyon bilgisine rastlanıyorsa iç hat olarak kabul edilecektir. İstasyonlarda biri yabancı istasyon olarak gözüküyorsa bu sefer dış hat olarak kabul edilecektir. Bu örnekte Tahran veya Paris yabancı istasyonlar olduğu için dışhat olarak kabul edilecektir.

Uçuşun dış hat veya içhat oluşu pozisyon kodu parametresini belirleyen bir unsurdur. İç hat ise pozisyon kodunun ilk karakteri 0, değilse 1 olarak tespit edilecektir. Pozisyon kodunu belirleyen ikinci unsur uçuş tipidir. Uçuş tipine göre pozisyon kodunun ikinci karakteri tespit edilmektedir. Tarifeli ve fiyatlandırmayı etkilemeyen normal uçuşlar için ikinci karakter 1, boş uçuşlar için 2, teknik uçuşlar için 3 vs.dir. Pozisyon kodları ve tanımları pozisyon adlı dosyada tutulmaktadır. Çok özel yapılan bir anlaşma için ayrı bir pozisyon kodu verilip ayrı bir fiyatlandırma yapılabilmektedir. Örneğin Yine Air France firmasıyla Tahran'dan gelen ve Türkiye'den yolcu almadan giden uçuşlar için özel bir anlaşma imzalandıysa "15" gibi bir pozisyon kodu verilip fiyat dosyasında onunla ilgili kayıtlar yaratılarak fiyatlandırma yapılabilir. Bu örnekteki uçuş tarifeli olduğu ve dış hat tarifeli uçuş kategorisine "11" geldiği için pozisyon kodu olarak kabul edilecektir.

Fiyatlandırma için diğer bir parametre tonaj kodudur. Bunun için girilen uçak tescilen ucak adlı dosyada karşılık gelen kayıda bakılacak ve uçağın tonaj kodu buradan alınacaktır. Bu uçağın 60 ton olduğu ve "07" tonaj koduna sahip oduğu uçak dosyasından tespit edilecektir.

Şu anda fiyat dosyasına erişmek için bütün parametreler belirlenmiş durumdadır. Eğer seçilen hizmet Catering ise hizmet kodu olarak "06" kabul edilecektir. Fiyat dosyasında firma kodu : AF , hizmet kodu : "06",
pozisyon kodu : "11" , tonaj kodu : "07" olan kayıda erişilerek ilgili tutar GHCN üzerine işlenecektir.

Gso dosyası üzerinde ayrıca isteğe bağlı hizmetler fiyatlandırılmaktadır. Buradaki fiyatlandırma request adlı dosya baz alınarak yapılamktadır. Hizmet kodu ve kaç birim olarak verildiği kullanıcı tarafından girilmekte firma kodu ve request kodu parametreleri ile reqfiyat dosyasına ulaşılmakta birim * tarife fiyatı kadar ücret Gso dosyasına işlenmektedir.

Dışarıdan sağlanan hizmetler işe oreq adlı dosyadan girilmekte burada fiyatlandırma kullanıcı tarafından yapılmaktadır. Uçuş kodu , geliş flight no, gidiş flight no ve uçuş tarihi bilgileri aracılığıyla ilgili GSO ile ilişki kurularak beraberce faturalanırlar.

Bu örnekte de görüldüğü fiyatlandırma için GHCN'yi düzenleyen kullanıcı için önemli olan uçuş vs. bilgilerini doğru girmesidir. Diğer önemli bir faktörde yapılan anlaşma bilgilerinin en kısa sürede istasyonlara iletilmesidir.

Gso dosyası içerisinde temel hizmet ve request hizmetlerine ilişkin bilgiler Progress veritabanının sağladığı extent özelliğinden faydalanarak dizilerde tutulmaktadır. Seçilen dizinin kodu , tutarı vs. bu dizilerin elemanlarına diğer sabit bilgi dosyalarıyla ilişkilendirilerek atanmaktadır.

Faturalandırma İşlemi:

Havayolu şirketleriyle yapılan anlaşmalara göre Cash veya Kredili olmak üzere verilen hizmetler fatura edilmektedir. Bu faturalar için GHCN'ler baz alınmaktadır. Ayrıca GHCN'lerde gösterilmeyen , iş fişleri denen belgelerde belirtilen hizmetler Extra Faturalama seçeneği ile Cash anlaşmalı şirketler için İstasyonlarda , Kredili şirketler ,için Genel Müdürlük'te fatura edilmektedir.

Fatura bilgileri fatura adlı dosyada saklanmaktadır. Ayrıca hangi GHCN'lerin hangi faturalara ait olduğunu tutan gsofat adlı bir dosya mevcuttur. Fatura dosyaında fatura numarası, firma kodu, fatura tarihi, mektup tarihi , son ödeme tarihi; diziler içinde hizmet bazında tutulan hizmet kodu , muhasebe kodu, indirim miktarı; fatura toplamı, doviz cinsi sayfa .... da veri sözlüğünde yer alan alanlar mevcutttur. Fatura numaraları otomatik olarak verilmektedir.

Cash Faturalama:

Cash fatura , Cash anlaşmalı sirketler için istasyonlar tarfından kesilmektedir. Her bir GHCN'ye tek bir Cash fatura karşılık gelmektedir. Bir Cash fatura kesilebilmesi için önce verilen hizmetle ilgili indirimsiz fiyatların yer aldığı GHCN'lerin düzenlenmesi gerekmektedir.

GHCN düzenlendikten , print-outları alındıktan sonra , Cash faturalama işlemi seçilip; GHCN'nin firma kodu, geliş-gidiş uçuş numaraları ve GHCN tarihi girilerek ilgili GHCN'ye erişilip, verilen hizmetler firma bilgilerinde girilmiş olan indirim, yüzdeleri, komisyon oranı doğrultusunda dolar bazında fatura edilir.

Burada faturaların sağlıklı kesilebilmesi için şirketlerle yapılan anlaşmaların vaktinde istasyonlara iletilmesi gereklidir. Firma dosyasında yer alan indirim bilgileri ve komisyon yüzdesi faturalama için büyük önem taşımaktadır. İlk önce trafik ve ramp hizmetleri tutarlarından yapılan üçlü indirim, daha sonra kalan tutar üzerinden yalnız trafik hizmetinden yapılan birinci indirim ve nihayet en sonunda

kalan tutar üzerinden bütün temel hizmetler ve indirime dahil isteğe bağlı request hizmetleri için ikinci genel indirim yapılmaktadır. Bu indirimlerin yapılabilmesi için şirketlerle ilgili indirim anlaşmalarının imzalanması gerekmektedir. Üçlü indirim dış hat uçup Türkiye üzerinde iki istasyona uğramış uçaklar için mevcut bir anlaşma varsa yapılmaktadır.

Hizmetler yalnız anlaşmalı olan şirketlere verilmemektedir. Anlaşmasız olan herhangi bir şirket gelip hizmet isterse normal standard oluşturulmuş bir tarife ile ücretlendirme yapılmaktadır. Bu tür şirketler için özel analmında "SPL" firma kodu ve ayrıca ikincil özel kod alanında firmanın adını tanımlayan şirket kodu girilmektedir.

Kredili Faturalama:

Bu faturalar yalnızca Kredili anlaşması olan şirketler için kesilmekte ve bu yetki sadece Genel Müdürlüktedir. Kredili faturalar belli bir dönem içerisinde, belli bir istasyonda , belli bir firma için kesilmektedir.

Kredili faturalar istasyonlarda bilgi girişi yapılan GHCN'ler baz alınarak yapılır. İlgili döneme ait GHCN'ler tampon bir dosyaya aktarılır. Gerekli kontroller yapılır. Belli bir sorun kalmadığı takdirde esas çalışılan GHCN dosyasına aktarılır. GHCN'lerin Genel Müdürlük veritabanına aktarılması TCP/IP vasıtasıyla gerçekleştirilen veritabanı bağlantısı ile olmaktadır.

Kredili faturalar, üçlü indirim anlaşması olan şirketler için her dönemde iki adet kesilmektedir. Biri üçlü uçuşlara ait hizmet cetveli ve fatura, diğeri ise üçlü olmayan uçuşlara ait hizmet cetveli ve fatura şeklinde kesilmektedir. Bu iki tür faturanın dışında belli uçuş tiplerine göre ayrı ayrı hizmet cetvelleri çekilebilmekte ve de fatura edilebilmektedir. Örneğin belli bir şirket her ay iç hat ve dış hat üçlü indirime dahil uçuşlarını ve dahil olmayan uçuşlarını ayrı ayrı gösteren dört adet fatura ve hizmet cetveli isteyebilir. Veya üçlü başka bir şirket hem iç hat için hem dış hat için ayrı ayrı olmak üzere boş uçuşlarını ayrı faturalandırmak, boş uçuş olmayan üçlü indirim uçuşlarını ve üçlü indirim olmayan uçuşlarını ayrı faturalandırmak isteyebilir. Kredili faturalama modülü bu şekildeki farklı uçuşlara göre olan isteklere cevap verecek şekilde geliştirilmiştir. Yukarıda bahsedilen fatura tiplerinin hepsi ve daha fazlası kredili faturalama menüsünde bir seçenek olarak yer almaktadır. Ayrıca bir kaç uçuş çok özel bir özelliğe sahipse, uçuşların ait olduğu GHCN'lerde verilen özel kodlar bazında faturalama yaparak her türlü faturalama isteğine cevap verilmekltedir. Örneğin bir şirket başka bir şirket adına yaptığı uçaşları ayrı faturalatmak

isteyebilir. Bu durumda bu uçuşlara ait GHCN'lerdeki özel kod alanı kullanılarak verilen özel koda göre ayrı faturalandırma gerçekleştirilebilir.

Faturalar kesilmeden önce her GHCN'de verilen hizmetleri detaylı bir şekilde gösteren GHCN ücretlerinin yer aldığı hizmet cetvelleri çekilir. Hizmet Cetvelleri incelendikten sonra hangi döneme ait hizmetlerin fatura edilmek istendiği girilerek o dönem içerisindeki ilgili GHCN'ler gerekli indirimler yapılıp ve komisyonlar eklenerek faturalanır. İndirimlerin yapılması ve komisyonların eklenmesi mantığı Cash faturalamada anlatıldığı gibidir. Faturalar kesildikten sonra bir son ödeme tarihi ile mektup firmalara gönderilir.

Burada Cash faturadan farklı olarak birden fazla GHCN'ye tek bir fatura karşılık gelmekte ve firmalara hizmet cetvelleri ve mektupla bir son ödeme tarihi verilerek gönderilmektedir.

Ayrıca genel müdürlükte kesilen faturalar on-line olarak aynı makine üzerindeki muhasebe veritabanında yer alan cari kart bilgilerine işlenmekte ve buradan da muhasebeleşme gerçekleşmektedir. Cash faturalarda henüz on-line muhasebeleşme söz konusu değildir.

Cash ve Kredili faturalama dışında ayrıca GHCN dışında belgelere dayalı kesilen daktilo gibi kullanılabilen, fakat yalnız veritabanında tanımlı her türlü hizmet için belli bir tarifeye fiziksel olarak bağlı olmayan extra fatura kesimi hem Kredili şirketler için Genel Müdürlük, Cash anlaşmalı şirketler için istasyonlar tarafından yapılmaktadır. Yine Genel müdürlükte ayrıca hizmet dışı uygulamalar teknik şartname vs. faturaları da yine extra faturalama seçeneği ile kesilebilmektedir.

Bu temel işlevler dışında istasyon bilgi girişi, uçak bilgi girişi, tatil bilgi girişi, özel kod bilgi girişi vs. gibi ekran outputlarında da görülen bir takım sabit bilgi girişleri ve bütün modüllerle ilgili raporlamalar mevcuttur.

4.5. Ağ Üzerinden Bilgi Akışı ve Dağıtıklık

Ağ üzerinden biligi akışı daha evvelki bölümlerde de açıklandığı üzere Progress veritabanının veritabanı bağlantıları ve Client-Server mimarisi özelliğinden faydalanılarak gerçekleşmektedir.

Bunun için her yerel yapıdaki veritabanlarının dağıtık bir sisteme eş ortak olarak kullanılabilmesi için her sistemdeki veritabanına ait broker/server başlatılmaktadır. Başka sistemlerden bu veritabanlarına erişmek için her sistemde yer alan /etc/hosts makine isimlerini içeren dosya ve özellikle Broker/server'ı kullanmak için

gerekli olan TCP/IP servis isimlerinin bulunduğu /etc/services dosyasındaki veritabanı servis ismi kullanılmaktadır. Her sistemde veritabanı broker/server'ı aşağıdaki gibi başlatılmaktadır.

İstanbul İstasyonu:
proserve fatura -H ist -S istser -N TCP -n 5 .

Ankara İstasyonu:
proserve fatura -H ank -S ankser -N TCP -n 5

İzmir İstasyonu :
proserve fatura -H izm -S izmser -N TCP -n 5

Antalya İstasyonu:
proserve fatura -H ant -S antser -N TCP -n 5

Dalaman İstasyonu:
proserve fatura -H dal -S dalser -N TCP -n 5

Genel Müdürlük:
proserve fatura -H mer -S merser -N TCP -n 10

Yukarıdaki gibi her sistemdeki veritabanı için başlatılan broker/server'lar yerel Client ve diğer uzak Client-Server'lar tarafından gelen istekleri organize etmektedir. Yukarıdaki -H parametresi Bölüm ... da da anlatıldığı gibi Progress server'ının bulunduğu makineyi -S parametresi ise bu veritabanına o makindeki hangi servis ile ulaşılabileceğini belirtmektedir. -N TCP/IP protokolünün kullanılacağı, -n ise kaç kullanıcıya hizmet verebileceğini belirtmektedir. Bölüm ... da bu konuyla ilgili detaylı bilgiler verilmektedir.

Yukarıdaki şekilde başlatılan veritabanı çok kullanıcılı servislerini kullanabilmek için yerel client'lar için aşağıdaki gibi bir scriptle (İstanbul İstasyonu veritabanı için) bağlanmaktadır:

Uzak client-server'lar ise programlar içerisinden aşağıdaki gibi bir ifade ile bağlantı sağlamaktadırlar.

connect fatura -H ist -S istser -N TCP -ld istfat.

Burada İstanbul makinesi dışından başka bir kullanıcı İstanbul fatura veritabanına bağlanmaktadır. Kendi fatura veritabanı ile bağlanacağı veritabanının fiziksel isimleri aynı olduğu için bağlanılacak fatura veritabanına -ld parametresi ile

mantıksal bir isim vermekte ve program içerisinden istanbul veritabanındaki dosyalar istfat.<dosya ismi>.<alan ismi> şeklinde kullanılmaktadır. Örneğin Genel Müdürlükten yukarıdaki gibi İstanbul İstasyonu veritabanına bağlanan bir kullanıcı British Airways firmasının indirimlerinde olan değişikliği İstanbul İstasyonuna aktarmak için güncellemeyi aşağıdaki program parçacığı ile yapabilir:

```
find fatura.firma using fatura.firma.uh-kod no-lock no-error.
find istfat.firma of fatura.firma  exclusive-lock no-error.
if available istfat.firma then
  assign istfat.firma.f-ind1 = fatura.firma.f-ind1
         istfat.firma.f-ind2 = fatura.firma.f-ind2
         istfat.firma.f-ind3 = fatura.firma.f-ind3.
```

Yerel yapılar arasındaki bilgi akışı temelde üç ana grupta gerçekleşmektedir.

i.) Sabit bilgilerin Genel müdürlükten diğer istasyonlara veri kopyalaması metodu kullanarak aktarılması.

ii.) İstasyonlarda tanzim edilien GHCN'lerin Genel Müdürlük tarafına aktarılıp gerekli değişikliklerin yapılarak faturalandırılması.

iii.) Yerel yapılar bazında gerekli sorgulamaların yapılması.

i.) Sabit bilgilerin Genel Müdürlükten diğer istasyonlara veri kopyalama metodu kullanılarak aktarılaması:

Daha evvel de anlatıldığı gibi istasyonda hizmet akışı sırasında GHCN'lerin tanzim edilmesi ve faturaların kesilmesi için Genel Müdürlük tarafından bilgi girişi yapılan firmaya ve hizmetlere ait bilgiler kullanılmaktadır. Tüm sistemin sağlıklı çalışabilmesi için bu bilgilerin istasyonlar tarafından sorunsuz olarak erişilebilmesi büyük önem taşımaktadır.

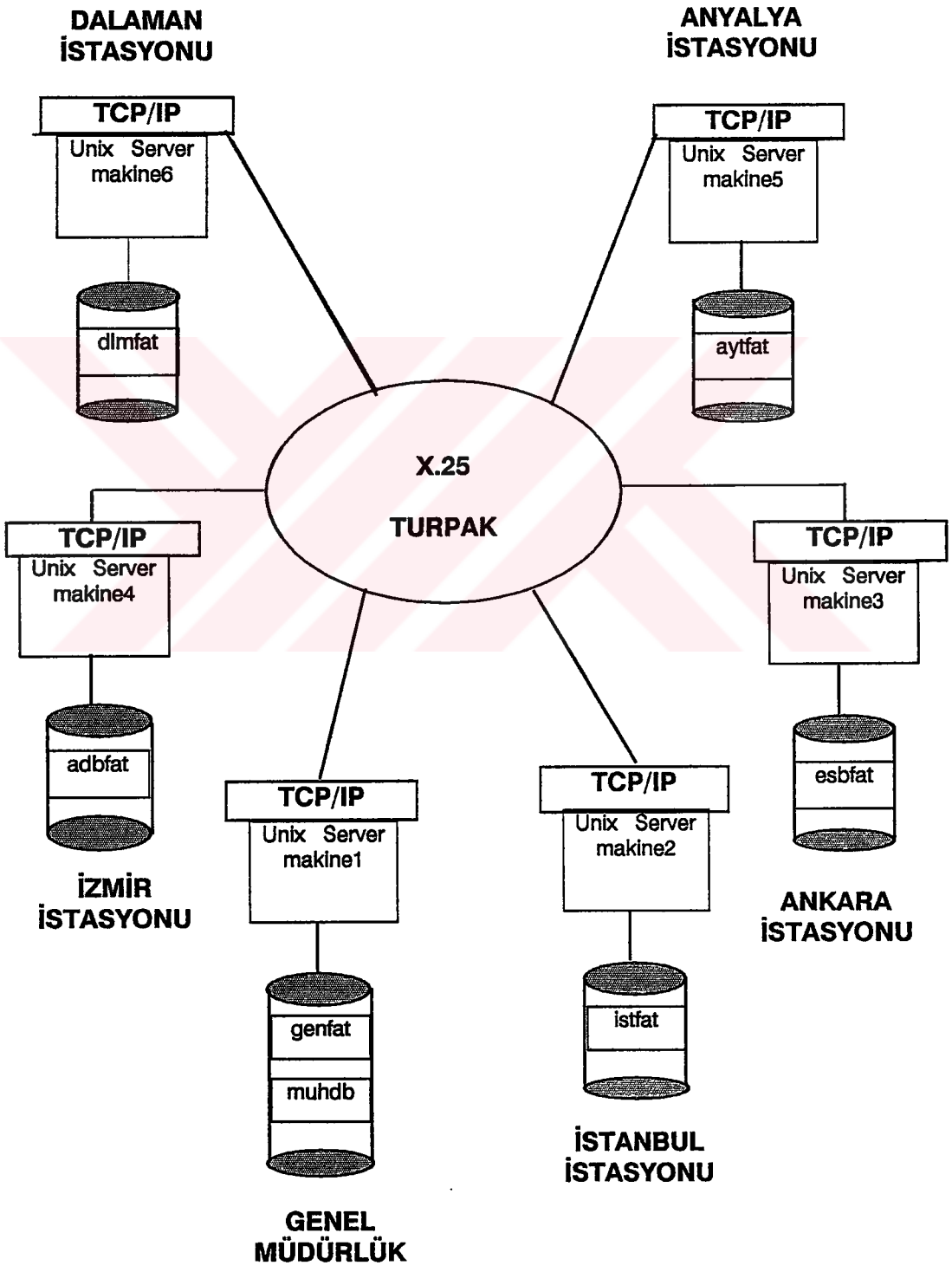
İstasyonlardaki iş akışı uçuş trafiğine bağlı olarak çok hızlı olması gerektiğinden dolayı bu bilgilere de çok hızlı bir şekilde erişmek gerekmektedir. Bilgisayar ağı (PTT Turpak) bu ihtiyacı karşılamak için çok yetersiz olduğundan dolayı , bu bilgilerin her yerel yapı içerisinde performansı arttırmak açısından yer alması gereği doğmuştur. Ayrıca yerel özerklik ve merkezi bir sisteme bağlılık yoktur dağıtıklık kurallarına göre verinin tek bir merkezi sistemde(Genel Müdürlükte) yer alması sakınca yaratmaktadır. Bu nedenle yine bir dağıtıklık kuralı olan veri kopyalama bağımsızlığı kullanılarak Genel Müdürlük'te güncellenen veya yeni girilen veriler diğer istasyon veritabanlarına bağlantılar yardımıyla toplu olarak aktarılmaktadır.

Belli bir firmanın anlaşmasında değişiklik olduğu veya yeni bir firma anlaşması girildiği zaman bu bilgiler ilk önce Genel Müdürlük veritabanında girilmektedir. Daha sonra ilgili veriler aşağıdaki program (firmadag.p) ile toplu olarak bütün istasyonlara aktarılmaktadır. Bu aktarma veritabanı bağlantısı gerçekleştirilerek TCP/IP üzerinden sağlanmaktadır. Fakat bazen bu işlem hatlardaki sorunlar yüzünden gerçekleşemediği zaman ikinci bir alternatif olarak; verilerin aktarılacağı sisteme X.25 üzerinden pad ile bağlanılıp karşı sistem üzerinden firma bilgi girişi programı çalıştırılarak bu değişiklik gerçekleştirilmektedir. Fakat bu yöntem son ihtimal olarak düşünülmektedir. Çünkü bu şekilde veri entegrasyonunda tehike olabilme riski artmaktadır. Veritabanı bağlantısı ile yapılan güncelleme başarı ile sonuçlanırsa bilgiler eksiksiz olarak aktarılımış olmaktadır.

Burada performansı arttırmak için başka bir yöntem olarak şu yöntem izlenebilir: Genel Müdürlük'te güncellenen veriler işletim sistemi üzerinde bir ASCII dosyaya export edilir. Daha sonra bu dosya ftp veya uucp kullanılarak güncelleme yapılacak sistemlere kopyalanılır. Sonra bu sistem üzerinden belli periyodlarda arka planda çalışan veya belli bir program içerisine serpiştirilen program parçacığı ile bu veriler veritabanına import edilebilir. Bu yöntem performans açısından daha iyi bir sonuç verebilir. Çünkü veritabanı bağlantısı ile yapılan direkt güncellemede her kayıt için iki aşamalı taahüt gereğince başarıyla sonuçlanıp sonuçlanmadığına ilişkin birçok mesajlar gidip geldiğnden dolayı süre uzamaktadır. Ancak bu yöntemde de iki aşamalı taahüt mekanizmasının çalışmaması veri tutarlılığında sorunlar yaratabilmektedir. Çünkü diğer sistemde güncellemenin başarıyla gerçekleşip gerçekleşmediği tahhüt edilememektedir.

Kullanılan programlar içerisinde performans çok önemli bir dezavantaj sağlanmadığı sürece güncellemlerın daha güvenilir olması açısından veritabanı bağlantısıyla ilk anlatılan yöntem kullanılmaktadır. Yalnız firmadag.p programında da görüldüğü gibi herbir istasyon için yapılan güncellemeler ayrı bir hareket olarak (do transaction blokları) işlenmektedir. Eğer bütün istasyonlar tek bir hereketle güncellenmiş olsalardı, herhangi bir sistemde meydana gelen sorun yüzünden hareket geri alınacak ve diğer istasyonlarda bu değişiklikler yansıtılamayacaktır. Tek bir sistemdeki sorun yüzünden diğerleri de etkilenecektir. Hareketler ayrı olduğunda güncellemenin başarı ile yapıldığı veritabanları aynen muhafaza edilecektir.

Veri Tabanı Organizasyonel Yapısı



Şekil 4.5.1. Veritabanı Org. Yapı

```
/*****************firmadag.p**********************/
def var fkod like firma.uh-kod.
set fkod label "Firma Kodu"
help "İstasyonlara aktarılacak firmanın kodunu giriniz."    with side-labels row 4
centered.
unix silent date  > bas.
find firma where firma.uh-kod = fkod no-lock no-error.
 if available firma then do:
   message "Gerekli İşlemler yapılıyor.
  Lütfen bekleyiniz".

 do transaction:
  connect /home/fatura/fatura -ld istfat -H ist -S istser -N TCP -U IST -P IST.
  run istpro.p(fkod).   run istreq.p(fkod).
  disconnect istfat.
end.

do transaction:
       connect /home/fatura/fatura -ld adbfat
      -H izm -S izmser -N TCP  -U ADB -P   ADB.
       run adbpro.p(fkod).
       run adbreq.p(fkod).
      disconnect adbfat.
end.

do transaction:
 connect /home/fatura/fatura -ld aytfat -H ant -S antser -N TCP -U AYT -P AYT.
 run aytpro.p(fkod).   run aytreq.p(fkod).   disconnect aytfat.
end.

do transaction:
 connect /home/fatura/fatura -ld esbfat
-H ank -S ankser -N  TCP -U ESB -P ESB.
 run esbpro.p(fkod).   run esbreq.p(fkod).
disconnect esbfat.
end.
```

```
do transaction:
  connect /home/fatura/fatura -ld dlmfat -H dal -S dalser
 -N TCP -U DLM -P     DLM.
  run dlmpro.p(fkod).   run dlmreq.p(fkod).   disconnect dlmfat.
end.
end.
 message "aktarma başarı ile tamamlandı".
unix silent date  son.


/*******istpro.p***************/
def input parameter fkod like fatura.firma.uh-kod.
find fatura.firma where fatura.firma.uh-kod = fkod no-lock no-error.
if connected("istfat") then do:
  find istfat.firma exclusive-lock where istfat.firma.uh-kod = fkod no-error.
  if not available istfat.firma then   create istfat.firma.
  {firma.i istfat fatura}
end.

 for each fatura.fiyat of fatura.firma no-lock:
    find istfat.fiyat of fatura.fiyat exclusive-lock no-error.
    if not available istfat.fiyat then create istfat.fiyat.
    {fiyat.i istfat.fiyat fatura.fiyat}
   release istfat.fiyat.
 end.
/*****************************************************/


/********istreq.p**************/
def input parameter fkod like fatura.firma.uh-kod.
find fatura.firma where fatura.firma.uh-kod = fkod no-lock no-error.
if connected("istfat") then do:
   find istfat.firma exclusive-lock where istfat.firma.uh-kod = fkod  no-error.
   if not available istfat.firma then   create istfat.firma.
  {firma.i istfat fatura}
end.
for each fatura.request of fatura.firma no-lock:
```

```
  find istfat.request of fatura.request exclusive-lock no-error.
  if not available istfat.request then create istfat.request.
  {request.i istfat.request fatura.request}
  release istfat.request.
end.
```

ii.) istasyonlardaki kredili şirketlere ait GHCN'lerin Genel Müdürlük veritabanına aktarılması:

Daha evvel de belirtildiği gibi GHCN'ler hizmet verildikçe istasyonlar tarafından tanzim edilmektedir. Tanzim edilen GHCN'lerin yazıcıdan çıktılarının bir nüshası Genel Müdürlüğ'e gelmektedir. Genel Müdürlük ise bu gelen GHCN evrakalarına ait bilgileri belli bir tarih aralığı ile istasyonlardan Genel Müdürlük veritabanına çekmektedir. Bu veriler Genel Müdürlük sisteminde yer alan tampon bir alanda kontrol edilmekte ve sorun yoksa esas çalışma veritabanına aktarılmakta, tampondan silinmektedir.Bu örnek programlarda gösterilmektedir.

Aslında bu yöntemde bir istasyonlardaki verinin Genel Müdürlük veritabanına kopyalanması extra bir işlem olmaktadır. Fakat bu yapılmadan verilerin direkt istasyonlardan sorgulanarak işlem görmesi, Genel Müdürlükte bu GHCN'ler üzerinde yapılacak hizmet cetveli çekimi, faturalama işlemlerini oldukça yavaşlatmaktadır. Hatlardaki hız çok daha yüksek bir seviyeye geldiğinde bu veriler üzerinde işlemler direkt olarak istasyonlar üzerinden veritabanı bağlantısı ile yapılan sorgulamalarla geçekleşebilir.

```
DEFINE VARIABLE menu AS CHARACTER EXTENT 4.
    DO WHILE TRUE:
       HIDE ALL.
       DISPLAY    SKIP(1)
        "  1. GSO AKTARILMASI                    " @ MENU[1] SKIP
        "  2. DI. S. HİZMET AKTARMASI            " @ MENU[2] SKIP
        "  3. DHMİ BİLGİLERİNİN AKTARILMASI " @ MENU[3] SKIP
        "  4. ÇIKIŞ                              " @ MENU[4] SKIP
     WITH FRAME choices NO-LABELS ROW 5.
CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices      TITLE "
İSTASYONDAN BİLGİ AKTARMA MENÜSÜ " CENTERED .
  HIDE FRAME choices.
  IF     FRAME-INDEX = 1 THEN RUN baglan1.p.
```

```
  ELSE IF FRAME-INDEX = 2 THEN RUN baglan2.p.
  ELSE IF FRAME-INDEX = 3 THEN RUN baglan3.p.
  ELSE IF FRAME-INDEX = 4 then do :
        if  connected("istfat") then
       disconnect istfat.
       return.
  end.
END.
/*****************************************************/

 /*****************baglan1.p*************************/
if  not connected("istfat") then do:
   if userid("fatura1") = "IST"
  then connect /home/fatura/fatura -ld istfat -H ist -S istser -N TCP -U IST -P IST.
  else if userid("fatura1") = "ESB" then
  connect /home/fatura/fatura -ld istfat -H ank -S ankser -N TCP -U ESB -P ESB.
  else if userid("fatura1") = "ADB" then
 connect /home/fatura/fatura -ld istfat -H izm -S izmser -N TCP -U ADB -P ADB.
  else if userid("fatura1") = "DLM" then
connect fatura -ld istfat -H dal -S dalser -N TCP -U DLM -P DLM.
 else if userid("fatura1") = "AYT" then
 connect /home/fatura/fatura -ld istfat -H ant -S antser -N TCP -U AYT -P AYT.
 if connected("istfat") then do:
    message userid("fatura1")  " ile bağlantı Sağlandı.".
    run aktar3.p.
    disconnect  istfat.
 end.
end.
/*************************************************************/


/******aktar3.p******/
def var tarih1 as date.
def var tarih2 as date. def workfile w_gso1 like fatura1.gso.
 form "Tarih1 :" tarih1 skip
       "Tarih2 :" tarih2  with frame tarfrm row 7 no-label centered
       title "Gso Aktarma Menusu" .
 prompt-for tarih1 auto-return tarih2 auto-return with frame tarfrm.
```

```
message "export işlemi başladı.".
output to gso.dat.

for each istfat.gso where istfat.gso.uh-tarih = input tarih1 and
   istfat.gso.uh-tarih  input tarih2 no-lock:
  export gso.
end.
hide message no-pause.
output close.
message "export tamamlandı.".

input from gso.dat.
repeat:
  create w_gso1.
  import w_gso1.
end.
input close.

for each w_gso1:
 find  fatura1.gso  where fatura1.gso.gsono = w_gso1.gsono
 exclusive-lock no-error.
 if available fatura1.gso then delete fatura1.gso.
end.

input from gso.dat.
 repeat:
     create fatura1.gso.
    import fatura1.gso.
   end.
input close.
/*************************************************************/
```

İki aşamalı taahhüt kuralı gereğince bağlantılı veritabanları arasındaki bir güncellemenin gerçekleşebilmesi için karşı sisteme önce hareketin başarıyla sonuçlandığının teyidinin gitmesi gerekmektedir. Bu nedenle bu tür hareketlerde veritabanları arasında bol mesaj alışverişi olmaktadır. Bu mesaj alışverişleri extra bir maliyet getirmektedir. Bu darboğazı hafifletmek için mümkün olduğu kadar veritabanları arasındaki ara dosyaları kullanmamakta fayda vardır. Burada da

istasyonlardaki GHCN'lerin verisi Genel Müdürlük veritabanıyla ilişkide bulundurulmadan direkt Genel Müdürlük Sistemindeki bir ASCII dosyaya atılmakta, sonra bağlantı kesilmekte ve bu dosyadan Genel Müdürlük GHCN dosyasına veriler import edilmektedir.

iii.) Raporlamalar ve İstatistiki verilerin alınması.

Her istasyon kendi yerel verilerini sorgulamaktadır. Ancak Genel Müdürlük idari konumda olduğu için Her yerel yapı üzerinde raporlar almaktadır. Bu raporlar veritabanı bağlantıları ile gerçekleşen sorgulamalarla olmaktadır.

Örneğin Genel Müdürlük istasyon bazında firmaların anlaşma bilgilerinin takibini yapabilmekte, tanzim edilen GHCN'ler ile ilgili envanter takibi yapabilmekte, İstasyon veritabanının işleyişini etkileyen sabit bilgilerin takibini vs. yapabilmektedir. Veriler kullanıldıkları yerlerde durmasına rağmen, Genel Müdürlük tarafından sanki kendi yerel sistemindeymiş gibi sorgulanabilmektedir. Burada sorgulamanın kendi yerel sisteminde oluyormuşcasına gibi hissettirmemesine sebep tek unsur var olmaktadır ; o da yine performans problemidir. Fakat bu sorgulamaların zaman açısından faturalama ve hizmet cetvellerinin dökümü gibi bir acileyeti olmadığı için tahammül edilebilir düzeydedir.

GHCN envanter dökümü bu sorgulamalardan birine örnek olarak verilebilir:

```
DEFINE VARIABLE menu AS CHARACTER EXTENT 7.
DO WHILE TRUE:
 HIDE ALL.
  DISPLAY    SKIP(1)
   "  1. GENEL MÜDÜRLÜK          " @ MENU[1] SKIP
   "  2. İSTANBUL İSTASYONU      " @ MENU[2] SKIP
   "  3. ANKARA  İSTASYONU       " @ MENU[3] SKIP
   "  4. İZMİR   İSTASYONU       " @ MENU[4] SKIP
   "  5. ANTALYA İSTASYONU       " @ MENU[5] SKIP
   "  6. DALAMAN İSTASYONU       " @ MENU[6] SKIP
   "  7. ÇIKIŞ                   " @ MENU[7] SKIP
    WITH FRAME choices NO-LABELS ROW 5.
if connected("fatura") or  connected("istfat") then run kop.p.
   CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices
   TITLE "  İSTASYON ANA MENU  " CENTERED .
   HIDE FRAME choices.
   IF     FRAME-INDEX = 1 THEN do:
```

```
if connected("fatura") or c onnected("istfat") then run kop.p.
else      run merbaglan.p.
  if connected("fatura") or connected ("istfat") then
 message "Merkez ile bağlantı sağlandı.".
 run rapormenu.p.
 run kopist.p.
end.
IF   FRAME-INDEX = 2 THEN do:
  run istbaglan.p.
  if connected("istfat") then
     message userid("istfat") " ile bağlantı sağlandı.".
     run rapormenu.p.
     run kopist.p.
end.
ELSE IF FRAME-INDEX = 3 THEN do:
  run esbbaglan.p.
  if connected("istfat") then message userid("istfat") " ile bağlantı sağlandı.".
     run rapormenu.p.
     run kopist.p.
end.
ELSE IF FRAME-INDEX = 4 THEN do:
 if connected("fatura") then run kop.p.
    run adbbaglan.p.
 if connected("istfat") then  message use rid("istfat") " ile bağlantı sağlandı.".
   run rapormenu.p.
   run kopist.p.
end.

ELSE IF FRAME-INDEX = 5 THEN do:
  if connected("fatura") then run kop.p.
    run aytbaglan.p.
 if connected("istfat") then
    message userid("istfat") " ile bağlantı sağlandı.".
    run rapormenu.p.
    run kopist.p.
end.
```

```
 ELSE IF FRAME-INDEX = 6 THEN do:
   if connected("fatura") then
    run kop.p.
    run dlmbaglan.p.
   if connected("istfat") then
   message userid("istfat") " ile bağlan tı sağlandı.".
   run rapormenu.p.
   run kopist.p.
  end.
 else if frame-index = 7 then do:
     quit.
 end.

END.
/*****istbagla.p*************************************************/
connect /home/fatura/fatura -ld istfat -H ist -S istser -N TCP -U IST -P IST.
/*****************************************************************/



/*********rapormenu.p******/
 DEFINE VARIABLE menu AS CHARACTER EXTENT 10.
DO WHILE TRUE:
 HIDE ALL.
  DISPLAY    SKIP(1)
   "  1. Tarih Aralıklı GHCN Dökümü              " @ MENU[1] SKIP
   "  2. Tarih Aralıklı D.S. Hizmetler Dökümü    " @ MENU[2] SKIP
   "  3. Request Hizmetleri Dökümü               " @ menu[3] skip
   "  4. Dışarıdan Sağlanan Hizmetler Dökümü     " @ menu[4] skip
   "  5. DHMİ Hizmetleri  Dökümü                 " @ menu[5] skip
   "  6. Firma Bilgileri Dökümü                  " @ menu[6] skip
   "  7. Uçak Bilgileri Dökümü                   " @ menu[7] skip
   "  8. İstasyon Bilgileri Dökümü               " @ menu[8] skip
   "  9. Tarih Aralıklı Fatura Listesi           " @ menu[9] skip
   " 10. Çıkış                                   " @ menu[10] skip
    WITH FRAME choices NO-LABELS ROW 5.
    CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices
    TITLE "  RAPORLAMALAR   " CENTERED .
  HIDE FRAME choices.
```

```
 IF     FRAME-INDEX = 1 THEN do:
   compile rapor2.p.
   RUN rapor2.p.
 end.
 ELSE IF FRAME-INDEX = 2 THEN do:
  compile rapor1.p.
   RUN rapor1.p.
 end.
 ELSE IF FRAME-INDEX = 3 THEN do:
    compile rapor3.p.
     RUN rapor3.p.
end.
 ELSE IF FRAME-INDEX = 4 THEN do:
    compile rapor4.p.
    RUN rapor4.p.
end.
 else if frame-index = 5 then do:
     compile dhmrapor.p.
     run dhmrapor.p.
end.
else if frame-index = 6 then do:
     compile firrapmenu.p.
     run firrapmenu.p.
  end.
 else if frame-index = 7 then do:
     compile ucakrapor.p.
     run ucakrapor.p.
 end.
 else if frame-index = 8 then do:
     compile istrapor.p.
     run istrapor.p.
end.
else if frame-index = 9 then do:
     compile tarfat.p.
     run tarfat.p.
end.
else if frame-index = 10 then return.
END.
```

```
/************************rapor2.***********************/
 def var tarih1 like gso.uh-tarih.
 def var tarih2 like gso.uh-tarih.
 def var istk like gso.uh-istk.
 def var tip like  ucak.uh-tip.
 def var sec as char format "x(10)" extent 3.
 sec[1] = "Ekran" .
 sec[2] = "anlasma1p".
sec[3] = "anlasma2p"
   . form "Tarih1   :" tarih1 skip
          "Tarih2   :" tarih2 skip
          "İstasyon :" istk  with
           title "GHCN Tarihleri" frame gstar_frm centered row 6      no-label.
 form header "İSTASYON KODU :
" istk          "İLK TARİH : " at 40
tarih1          "SON TARİH : " at 70
tarih2 with frame gs_frm.
    assign istk = substring(userid("fatura"),1,3).
update tarih1 auto-return
     tarih2 auto-return
     istk   auto-return
     with frame gstar_frm.
display "Göndermek istediğiniz yeri seçiniz." skip
     sec[1] sec[2] sec[3] at 50
     with frame sor_frm centered row 12 overlay       no-label .
    choose field sec with frame sor_frm.
if frame-index = 1 then    Terminal = "wy60tw".
if frame-index = 2 or frame-index = 3 then do:
  output through lp -d value(frame-value) -s paged no-echo.
  put control "~017".
end.
for each gso where gso.uh-istk  = istk and
                   gso.uh-tarih = tarih1 and
                   gso.uh-tarih  tarih2
no-lock by gsono:
  find ucak of gso no-lock no-error.
   if available ucak then tip = ucak.uh-tip.
```

```
  else   tip = "".
  find firma of gso no-lock no-error.
       disp gso.gsono label "GHCN No"
            gso.uh-kod label "Kod"
            firma.firma_ad label "Firma Adı"
            uh-no1 label "Fltno1"
            uh-no2 label "Fltno2"
            gso.uh-reg  label "Tescil"
            tip    label "Uçak Tipi"
           gso.uh-gel  label "Geliş Y."
           gso.uh-git  label "Gidiş Y."
           gso.uh-tarih label "Tarih" with frame gs_frm 34 down
           width 120 centered.
           down  with frame gs_frm.
end.
output close.
message "Devam etmek için bir tuşa basınız.".
readkey.
 terminal = "wy60".


/*********************kopist.p*********************/
 if connected("fatura") then   disconnect fatura.
 if connected("istfat") then    disconnect istfat.
/*************************************************/
```

Genel Müdürlükte kesilen faturaları otomatik olarak cari karta işleyen program ise aşağıda görüldüğü gibidir. Burada fatura numarası çağıran program tarafından verilmekte ve bu fatura muhasebe veritabanında ilgili yerlere işlenmektedir.

```
/*******************Faturaların Cari Karta İşlenmesi programı*************/
def var fisnum like hcari.carifis_no.
def var i as int.
def var top1 as decimal.
def workfile w_hcari like hcari.
def var dovcins like fatura.doviz.
def var dovtutar as decimal.
def var cnt as int initial 0.
def input param fatnum like fatura.fat_no.
```

```
run muhbaglan.p.
find fatura where fatura.fat_no = fatnum no-lock no-error.
find firsabit where firsabit.uh-kod = fatura.uh-kod no-lock no-error.
 find last scari where scari.ana_kod  = firsabit.cari_ana and
                       scari.alt_kod  = firsabit.cari_alt and
                       scari.tali_kod = firsabit.cari_tali use-index scind  no-error.
if available scari then fisnum = scari.fnum + 1.
else do:
fisnum = 1.
create scari.
end.
assign  scari.ana_kod  = firsabit.cari_ana
        scari.alt_kod    = firsabit.cari_alt
        scari.tali_kod   = firsabit.cari_tali
        scari.fnum       = fisnum.
find muhist where muhist.uh-istk = fatura.uh-istk no-lock no-error.
if not available firsabit then do:
  message "Bu firma Mevcut değil!".  undo, retry.
end.
dovtutar = 0.
dovcins = "".
 if fatura.doviz = "TL" and fatura.kur  1 then assign dovtutar = fatura.kur.
else if fatura.doviz  "TL" then
find gundov where gundov.tarih = fatura.fat_tar and
                  gundov.kis   = fatura.doviz no-lock no-error.
if available gundov then do:
 dovtutar = gundov.doviz_alis.
 dovcins  = gundov.kis.
end. cnt = 0.
 /*******Repeat Basla********/
repeat i = 1 to 25:
   if (fana_kod[i] = "600" or fana_kod[i] = "601") and
     falt_kod[i] = "20"   and
     fatura.muh-kod[i] = "00003" then next.
    if kod[i]  "" and fatura.tutar[i] = 0 then do:
  find shizmet where shizmet.skod = kod[i] no-lock no-error.
  if available shizmet then do:
    create w_hcari.
```

```
  cnt = cnt + 1.
  assign w_hcari.ana_kod    = fatura.fana_kod[i]
       w_hcari.alt_kod    = fatura.falt_kod[i]
       w_hcari.tali_kod   = fatura.muh-kod[i]
       w_hcari.cari_ana   = firsabit.cari_ana
       w_hcari.cari_alt   = firsabit.cari_alt
       w_hcari.cari_tali  = firsabit.cari_tali
       w_hcari.cari_sira  = cnt
       w_hcari.aciklama   = firsabit.firma_ad + " " +
       string(fatura.fat_no)
       w_hcari.bolum      = shizmet.bolum
       w_hcari.uretim_yeri = shizmet.uretim_yeri.
end.
 find reqfiyat where reqfiyat.reqkod = kod[i] no-lock no-error.
if available reqfiyat then do:      create w_hcari.
  cnt = cnt + 1.
 assign w_hcari.ana_kod    = fatura.fana_kod[i]
        w_hcari.alt_kod    = fatura.falt_kod[i]
        w_hcari.tali_kod   = fatura.muh-kod[i]
        w_hcari.aciklama   = firsabit.firma_ad + " " +
        string(fatura.fat_no)
        w_hcari.cari_ana   = firsabit.cari_ana
        w_hcari.cari_alt   = firsabit.cari_alt
        w_hcari.cari_tali  = firsabit.cari_tali
        w_hcari.cari_sira  = cnt
        w_hcari.bolum      = reqfiyat.bolum
      w_hcari.uretim_yeri = reqfiyat.uretim_yeri.
end.
 find diskod where diskod.orkod = kod[i] no-lock no-error.
 if available diskod then do:
  create w_hcari.
  cnt = cnt + 1.
  assign w_hcari.ana_kod    = fatura.fana_kod[i]
         w_hcari.alt_kod    = fatura.falt_kod[i]
         w_hcari.tali_kod   = fatura.muh-kod[i]
         w_hcari.aciklama   = firsabit.firma_ad + " "  +
         string(fatura.fat_no)
         w_hcari.cari_ana   = firsabit.cari_ana
```

```
        w_hcari.cari_alt    = firsabit.cari_alt
        w_hcari.cari_tali   = firsabit.cari_tali
        w_hcari.cari_sira   = cnt
        w_hcari.bolum       = diskod.bolum
        w_hcari.uretim_yeri = diskod.uretim_yeri.
 end.
 find dhkod where dhkod.dhm-kod = kod[i] no-lock no-error.
 if available dhkod then do:
   create w_hcari.       cnt = cnt + 1.
   assign w_hcari.ana_kod     = fatura.fana_kod[i]
          w_hcari.alt_kod     = fatura.falt_kod[i]
          w_hcari.tali_kod    = fatura.muh-kod[i]
          w_hcari.cari_ana    = firsabit.cari_ana
          w_hcari.cari_alt    = firsabit.cari_alt
          w_hcari.cari_tali   = firsabit.cari_tali
          w_hcari.cari_sira   = cnt
          w_hcari.aciklama    = firsabit.firma_ad + " " +
          string(fatura.fat_no)
        w_hcari.bolum       = dhkod.bolum
        w_hcari.uretim_yeri = dhkod.uretim_yeri.
 end.
 assign    w_hcari.cari_ana    = firsabit.cari_ana
           w_hcari.cari_alt    = firsabit.cari_alt
           w_hcari.cari_tali   = firsabit.cari_tali
           w_hcari.doviz_tutar = fatura.tutar[i]
           w_hcari.kur_cinsi   = 1    w_hcari.cari_sira   = cnt
  w_hcari.kur_fiyatı  = dovtutar
  w_hcari.istasyon    = muhist.muh-istk
  w_hcari.carifis_no  = fisnum
   w_hcari.b_a        = "a"
   w_hcari.tutar      =  dovtutar * fatura.tutar[i]
   w_hcari.kur_tarihi  = fatura.fat_tar
   w_hcari.doviz       = dovcins
    w_hcari.fat_no     = fatura.fat_no.
 end.
end.
/****Kod son ***/
/******************Repeat son*********************/
```

```
/********* Fatura Toplam Tutarının Cari Karta İşlenmesi*********/
if fatura.odenen  0 then do:
create w_hcari .
cnt = cnt + 1.
assign    w_hcari.cari_ana    = firsabit.cari_ana
          w_hcari.cari_alt    = firsabit.cari_alt
          w_hcari.cari_tali   = firsabit.cari_tali
          w_hcari.ana_kod     = firsabit.cari_ana
          w_hcari.alt_kod     = firsabit.cari_alt
          w_hcari.tali_kod    = firsabit.cari_tali
          w_hcari.cari_sira   = cnt
          w_hcari.doviz_tutar = fatura.odenen
  w_hcari.kur_cinsi   = 1    w_hcari.kur_fiyatı  = dovtutar    w_hcari.istasyon
= muhist.muh-istk    w_hcari.carifis_no  = fisnum    w_hcari.b_a       = "b"
w_hcari.tutar      =  dovtutar * fatura.odenen    w_hcari.aciklama    =
string(fatura.fat_no)    w_hcari.kur_tarihi  = fatura.fat_tar    w_hcari.doviz      =
dovcins    w_hcari.fat_no      = fatura.fat_no.    find scari where scari.ana_kod
= w_hcari.ana_kod and                scari.alt_kod  = w_hcari.alt_kod and
scari.tali_kod = w_hcari.tali_kod                  no-error.    if not available scari
then create scari.    {scari.i scari w_hcari}
end.  /*********SON Fatura toplam*********/
/********* Komisyon Toplamının aktarılması************/
if fatura.komisyon = 0 then do:
  create w_hcari . cnt = cnt + 1.
  assign    w_hcari.cari_ana    = firsabit.cari_ana
            w_hcari.cari_alt    = firsabit.cari_alt
            w_hcari.cari_tali   = firsabit.cari_tali
            w_hcari.alt_kod     = "20"
            w_hcari.tali_kod    = "00003"
            w_hcari.doviz_tutar = fatura.komisyon
            w_hcari.kur_cinsi   = 1
            w_hcari.kur_fiyatı  = dovtutar
            w_hcari.istasyon    = muhist.muh-istk
           w_hcari.carifis_no  = fisnum
            w_hcari.cari_sira   = cnt
           w_hcari.aciklama    = firsabit.firma_ad + " " +
   string(fatura.fat_no)
   w_hcari.b_a        = "a"
```

```
   w_hcari.tutar      =  dovtutar * fatura.komisyon
   w_hcari.kur_tarihi  = fatura.fat_tar
   w_hcari.doviz       = dovcins
   w_hcari.fat_no      = fatura.fat_no.
if firsabit.f-yerli then w_hcari.ana_kod = "600".
else w_hcari.ana_kod = "601". end.
 /********SON Komisyon Toplam*********/
/*********İndirim Toplamı************/    if (fatura.ind1_t + fatura.ind2_t +
fatura.ind3_t)  0 then do:       create w_hcari.       cnt = cnt + 1.       assign
w_hcari.ana_kod    =  firsabit.isana_kod            w_hcari.alt_kod     = fir-
sabit.isalt_kod            w_hcari.tali_kod    = firsabit.istali_kod
w_hcari.aciklama    = string(fatura.fat_no)         /*  w_hcari.bolum       =
dhkod.bolum             w_hcari.uretim_yeri = dhkod.uretim_yeri*/
w_hcari.cari_ana    = firsabit.cari_ana             w_hcari.cari_alt    = fir-
sabit.cari_alt             w_hcari.cari_tali   = firsabit.cari_tali
w_hcari.doviz_tutar = fatura.ind1_t + fatura.ind2_t + fatura.ind3_t
w_hcari.kur_cinsi   = 1              w_hcari.kur_fiyatı  = dovtutar
w_hcari.cari_sira   = cnt
          w_hcari.istasyon    = muhist.muh-istk
          w_hcari.carifis_no  = fisnum
          w_hcari.b_a         = "b"
          w_hcari.tutar       = dovtutar * w_hcari.doviz_tutar
          w_hcari.kur_tarihi  = fatura.fat_tar
          w_hcari.doviz       = dovcins
          w_hcari.fat_no      = fatura.fat_no.
    end.
/******* İndirim Toplamı Son **********/
for each w_hcari break by w_hcari.ana_kod
                  by w_hcari.alt_kod
                  by w_hcari.tali_kod:
       top1 = top1 + w_hcari.doviz_tutar.
       if last-of(w_hcari.tali_kod) then do:
         create hcari.          {hcari.i hcari w_hcari}.
         top1 = 0.
      end.
end.
do for carifisacik:
   find istasyon of fatura no-lock no-error.
```

```
   create carifisacik.
assign carifisacik.cari_ana    = firsabit.cari_ana
       carifisacik.cari_alt    = firsabit.cari_alt
       carifisacik.cari_tali   = firsabit.cari_tali
       carifisacik.carifis_no   = fisnum
       carifisacik.FIS_ACIK[1] = istasyon.istad +  " İstasyonunda "
 +      string(fatura.fat_no) + " no'lu fatura ile "
        carifisacik.FIS_ACIK[2] =      firsabit.firma_ad + "'e verilen hizmetlerin
mahsubu.".
end.
/******************Cari Kart Son*****************************************/

/******Muhbaglan.p*******/
connect  /home/muhasebe/muhasebe no-error.
if not connected("Muhasebe") then do:
  message "Muhasebeye aktarılamıyor.".
  undo, retry.
end.
 /*************************/
```

SONUÇLAR VE ÖNERİLER:

Dağıtık Veri Tabanı Yönetim Sistemelrinin getirdiği faydalar gerçekten çok büyüktür. Fakat bununun yanında çok verimli sonuçların alınması için özellikle bilgisayar ağı hızının çok iyi olması gerekmektedir. Dolayısıyla böyle bir uygulama coğrafi alan üzerine yayılmış bir ağ üzerinde gerçekleşiyorsa bu ağın mümkün olduğu kadar iyi bir performans göstermesi gerekmektedir. Yerel bir ağda hız o kadar önemli bir sorun olmamaktadır.

Tez konusu çalışma geniş alana yayılmış bir bilgisayar ağı üzerinde gerçekleşmektedir. Bilgisayarlar arası iletişimi X.25 (PTT- Turpak) sağlamaktadır. Makineler birbirleri ile maksimum 9600 bps hızıyla görüşmektedir. VTYS'lerin genellikle yazıldıkları yer olan Amerika'da ise bu hız 42 mega bit per second'dır. Görüldüğü gibi Türkiye'de bu konudaki performans anormal derecede düşüktür.

Dağıtıklık özelliğnin yerine getirilmesindeki en önemli kurallardan birisi sistemlerin özerk çalışmasıdır. Bununla beraber gerektiğinde on line transactionların gerçekleşmesi de oldukça zaruridir. On line transaction'ların performansı da direkt olarak ağ hızıyla ilgilidir. Gerekli transaction'lar vaktinde yapılamadığı zaman farklı veritabanlarında farklı verilerin yer alması nedeniyle veri tutarsızlığı gözükmektedir. Bu nedenle farklı veritabanları arasındaki hareketler, iletişim hızı ve hareketin aciliyeti göz önünde bulundurularak düzenlenmelidir. Eğer bilgisayar ağının sağladığı iletişim hızı çok yetersiz kalıyorsa sabit bilgilerin bütün veritabanlarında yer almasında fayda vardır. Sabit bilgilerin diğer yerel veritabanlarına aktarımı ise toplu veya arka planda çalışan bir proses vasıtasıyla olmalıdır. İletişim hızının düşük olduğu ağlarda, gerektiğinde veritabanı bağlantıları yerine; kaynak veritabanındaki verileri diğer veritabanlarının anlayacağı şekilde formatlı bir text dosyaya atıp , bu dosyayı diğer sistemlere transfer edip ; dosyaların transfer edildiği sistemler üzerinden gerekli hereketleri yaparak işlem yapmak daha çabuk olabilir.

Dağıtıklık konusunda VTYS'ler hala bir gelişme içerisindedirler. Dağıtık bir veritabanı yönetim sisteminin bütün özelliklerini kullanılan VTYS ticari VTYS yazılımı tek başına verememektedir. DVTYS özelliklerini tam anlamıyla sağlamak görevi veritabanı yöneticisi ve yazılımcılara düşmektedir. Örneğin dilimleme olayı hemen-hemen çoğu VTYS'lerde direkt olarak sağlanmamaktadır; veya VTYS bağımsızlığı

yine tam analamıyla sağlanamamaktadır. Bütün veritabanları arasında karşılıklı gateway mevcut değildir.Örneğin A VTYS'si B VTYS'si için gateway özelliği içerirken; B VTYS'si A VTYS'si için gateway içermemektedir. Tabii ki bu problemler aynı zamanda VTYS'nin ve işletim sistemlerinin bağımsızlığı ile ilgilidir. Açık Sistem kavramı ne kadar gelişirse , bundan dağıtık VTYS'ler o kadar fayda görecektir.

Dağıtıklıktan iyi verim alabilmek için donanım ve yazılım kısıtlamaları çok iyi hesap edilmelidir. Teorik anlamda dağıtıklığın gerçeklenmesi için geliştirilen uygulamalar faydadan çok, zarar yaratabilir. Hangi verilerin hangi yerel yapılarda yer alacağı, bunlar arasında nasıl bir mantıksal bağlantı olacağı, veritabanları arasındaki hareketlerin hangi yönde ve zamanda gerçekleşeceği, hataya ne kadar töleranslı olacağı uygulamanın başarasını etkileyen önemli faktörlerden biridir.

Ben de tez çalışmamda, mevcut donanım ve yazılımın imkanlarını göz önünde bulundurarak bir organizasyonel yapı kurup bu uygulamayı gerçekleştirdim.

KAYNAKLAR:

[1]ÖZSU, M.Tamer-University Of Alberta Edmonton, VALDURIEZ, Patrick - INRIA Paris France, Principles Of Distrubuted Database Systems, Printence Hall, (1991).

[2] CERI, Stefano - PELAGATTI, Giuseppe, Distrubuted Databases Principles and Systems, Politecnico Di Milano, Mc Graw Hill Book Company, (1984).

[3] DATE, C.J., Database Systems, Volume 1 Addison-Wesley Publishing Company, (1984).

[4] AJOUZ, Mohammed, Database Systems, Açık Sistem 94 Bildirisi, (1994).

[5] ÇAĞLAYAN, M. Ufuk, TCP/IP Ağ Yönetimi - Açık Sistem 94 Tutorial Notları, (1994).

[6] ERŞEN, Metin, Açık Sistemler TCP/IP Protokolu, TCP/IP ile İletişim, Bilişim 92 Bildirisi, (1992).

[7] DENGİ, Cevdet - DOĞAÇ Asuman, Internet Kullanarak Bilgiye Erişim, Bilişim Dergisi, (1994).

[8] PROGRESS, Software Corparation, System Administration General, (1991).

[9] PROGRESS, Software Corporation, Progress Programming Handbook, (1991).

EKLER

```
06/10/94 20:43:57        PROGRESS Data Dictionary Report                Page
Database: fatura                                                        (PROGRESS
Filename: apron1
```

Order	Field-Name	dType	Ext	Flgs	Format	Initial
30	uh-kod	char		i	x(3)	
50	uh-no1	char		i	x(5)	
140	uh-no2	char		i	x(5)	
380	uh-tarih	date		i	99/99/99	?
390	birim	inte			>>>>>>	0
400	reqkod	char		i	x(3)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
uh-kod	FIRMA_KODU	
uh-no1	FLT_NO	
uh-no2	FLT_NO	
reqkod	REQUEST KODU	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
ap1_ind*	yes	uh-tarih	1	yes	no
		uh-kod	2	yes	no
		uh-no1	3	yes	no
		uh-no2	4	yes	no
		reqkod	5	yes	no

Field Validation Criteria, Validation Messages

uh-kod:
+ Criterion: uh-kod <> ""
+ Message:

Help Messages and Descriptions

uh-kod:
+ Desc: firma kodu

uh-no1:
+ Desc: gelis flt no

uh-no2:
+ Desc: gidis flight nosu

uh-tarih:
+ Desc: uçuş tarihi

birim:
+ Desc: Request in birim süre, adet veya ağırlık miktarı.

reqkod:

+ Help: Request kodunu giriniz.
+ Desc: Request kodu

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page
Database: fatura (PROGRESS
Filename: dhkod

Order	Field-Name	dType	Ext	Flgs	Format	Initial
50	kom-kod	char			x(5)	
60	ana_kod	char			x(3)	
70	alt_kod	char			x(2)	
80	dhm-kod	char		i	x(3)	
90	aciklama	char		i	x(20)	
100	muh-kod	char			x(5)	
130	bolum	char			x(2)	
140	kalt_kod	char			x(2)	
150	kana_kod	char			x(3)	
180	uretim_yeri	char			x(1)	
190	istasyon	char			x(2)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
aciklama	Açıklama	
kana_kod	Komisyon Kodu	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
dhad	yes	aciklama	1	yes	no
dhkod*	yes	dhm-kod	1	yes	no

Field Validation Criteria, Validation Messages

Help Messages and Descriptions

kom-kod:
+ Desc: Muhasebe Komisyon Tali Kodu

ana_kod:
+ Desc: Muhasebe Ana Hesap kodu

alt_kod:
+ Desc: Muhasebe alt hesap kodu

dhm-kod:
+ Desc: DHMİ hizmet kodu

aciklama:
+ Desc: Hizmet Adı

muh-kod:
+ Desc: Muhasebe Tali Kodu

bolum:

+ Help: Bolümü giriniz
+ Desc: Maliyet merkezinin ait oldugu bölüm

kalt_kod:
+ Desc: komisyon alt kod

kana_kod:
+ Desc: Komisyon anakod

uretim_yeri:
+ Help: Üretim Yerini giriniz
+ Desc: Maliyet Merkezi Üretim Yeri Kodu

istasyon:
+ Help: İstasyonu giriniz
+ Desc: Maliyet merkezi İstasyon Kodu

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 1
Database: fatura (PROGRESS)
Filename: dhmi

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	uh-kod	char		i	x(3)	
20	uh-istk	char		i	x(3)	
30	uh-reg	char		i	x(6)	
40	dhm-tarih	date		i	99/99/99	?
70	saat	char		i	xx.xx	
80	dhm-kod	char		i	x(3)	
90	tutar	deci2			->>>,>>9.99	0

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
uh-kod	FİRMA_KODU	
uh-istk	IST_KOD	
uh-reg	UCAK TESCİLİ	
tutar	Tutar	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
dhm-ind*	yes	dhm-tarih	1	yes	no
		uh-istk	2	yes	no
		uh-reg	3	yes	no
		uh-kod	4	yes	no
		saat	5	yes	no
		dhm-kod	6	yes	no

Field Validation Criteria, Validation Messages

uh-kod:
+ Criterion: uh-kod <> ""
+ Message:

uh-istk:
+ Criterion: uh-istk <> ""
+ Message:

uh-reg:
+ Criterion: uh-reg <> ""
+ Message: Uçak Tescilini girmek zorundasınız...

Help Messages and Descriptions

uh-kod:
+ Desc: firma kodu

uh-istk:
+ Desc: istasyon kodu

uh-reg:
+ Desc: uçak tescili

dhm-tarih:
+ Desc: Dhmi hizmet veriliş tarihi

saat:
+ Desc: Dhmi hizmet veriliş saati

dhm-kod:
+ Desc: Dhmi Hizmet Kodu

tutar:
+ Desc: Dhmi hizmet tutarı

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 1
Database: fatura (PROGRESS)
Filename: diskod

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	orkod	char		i	x(3)	
20	daciklama	char		i	x(20)	
30	muh-kod	char			x(5)	
40	vmuh-kod	char			x(5)	
50	kom-kod	char			x(5)	
60	ana_kod	char			x(3)	
70	alt_kod	char			x(2)	
80	uretim_yeri	char			x(1)	
90	istasyon	char			x(2)	
100	bolum	char			x(2)	
110	valt_kod	char			x(2)	
120	vana_kod	char			x(3)	
130	kana_kod	char			x(3)	
140	kalt_kod	char			x(2)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
orkod	KOD	
daciklama	Açıklama	
kana_kod	Komisyon Kodu	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
dad	yes	daciklama	1	yes	yes
dkod*	yes	orkod	1	yes	no

Field Validation Criteria, Validation Messages

Help Messages and Descriptions

orkod:
+ Desc: Muhasebe dışarıdan sağlanan hizmet kodu

daciklama:
+ Desc: Dışarıdan Sağlanan Hizmet Adı

muh-kod:
+ Desc: Muhasebe Tali Hesap kodu

vmuh-kod:
+ Desc: KDV tali hesap kodu

ana_kod:
+ Desc: Muhasebe ana hesap' kodu

```
alt_kod:
    + Desc: Muhasebe alt hesap kodu

uretim_yeri:
    + Help: Üretim Yerini giriniz....
    + Desc: Maliyet Merkezi Üretim Yeri Kodu

istasyon:
    + Help: İstasyon Kodunu giriniz...
    + Desc: Maliyet Merkezi istasyon kodu

bolum:
    + Help: Bolum kodunu giriniz.....
    + Desc: Maliyet Merkezi Bölüm Kodu

valt_kod:
    + Desc: KDV alt hesap kodu

vana_kod:
    + Desc: KDV ana hesap kodu

kana_kod:
    + Desc: Komisyon ana hesap kodu

kalt_kod:
    + Desc: komisyon alt hesap kodu
```

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 1
Database: fatura (PROGRESS)
Filename: extkod

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	ext_kod	char		i	x(3)	
20	eaciklama	char		i	x(35)	
30	ana_kod	char			x(3)	
40	alt_kod	char			x(2)	
50	muh-kod	char			x(5)	
110	valt_kod	char			x(2)	
120	vana_kod	char			x(3)	
130	vmuh-kod	char			x(5)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
ext_kod	Kod	
eaciklama	Açıklama	
ana_kod	Ana Kod	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
ext_inda	yes	eaciklama	1	yes	yes
ext_indk*	yes	ext_kod	1	yes	no

Field Validation Criteria, Validation Messages

Help Messages and Descriptions

ext_kod:
+ Desc: Extra satış kodu

eaciklama:
+ Desc: Extra Satış Adı

ana_kod:
+ Desc: Muhasebe extra satış ana hesap kodu

alt_kod:
+ Desc: Muhasebe extra satış alt hesap kodu

muh-kod:
+ Desc: Extra satış tali hesap kodu

valt_kod:
+ Desc: Extra satış KDV alt hesap kodu

vana_kod:
+ Desc: Extra satış KDV ana hesap kodu

vmuh-kod:
+ Desc: Extra satış KDV tali hesap kodu

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 1
Database: fatura (PROGRESS
Filename: fatura

Order	Field-Name	dType	Ext	Flgs	Format	Initial
20	FAT_TAR	date		i	99/99/99	
30	uh-istk	char			x(3)	
40	uh-kod	char			x(3)	
620	fkod	char			x(4)	
660	KOMISYON	deci2			->>>,>>9.99	0
680	ODENEN	deci2			->>,>>9.99	0
700	TUTAR	deci2	25		->>>,>>>.99	0
710	IND1_T	deci2			->>,>>9.99	0
720	IND2_T	deci2			->>,>>9.99	0
730	IND3_T	deci2			->>,>>9.99	0
750	VOUC	inte			>>>	0
820	uclu	logi			EVET/HAYIR	HAYIR
850	kod	char	60		x(3)	
860	muh-kod	char	60		x(5)	
870	tar1	date			99/99/99	?
880	tar2	date			99/99/99	?
890	aciklama	char	60		x(34)	
900	toplam	deci2			->>>,>>9.99	0
910	vat	deci2			->>,>>9.99	0
920	doviz	char			x(3)	
930	kur	deci2			>>>,>>9.99	1
940	fat_no	inte		i	999999	0
950	mek_tar	date			99/99/99	?
960	itotal	deci4	25		>>>,>>>.9999	0
970	fat_muc	date			99/99/99	?
980	iptal	logi			Evet/Hayır	Hayır
990	fana_kod	char	25		x(3)	
1000	falt_kod	char	25		x(2)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
uclu	UÇLÜ UÇUŞ İND ?	
tar1	Tarih1	
tar2	Tarih2	
aciklama	Açıklama	
fat_no	Fatura No	
mek_tar	Mektup Tarihi	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
fatno*	yes	fat_no	1	yes	no
fattar	yes	FAT_TAR	1	yes	no
		fat_no	2	yes	no

Field Validation Criteria, Validation Messages

Help Messages and Descriptions

FAT_TAR:
+ Desc: Fatura Tarihi

uh-istk:
+ Desc: Hizmetin verildiği istasyon kodu.

uh-kod:
+ Desc: Fatura edilen firmanın kodu

fkod:
+ Help: Özel tip bir fatura ise fatura kodunu giriniz ...
+ Desc: fatura için kullanılacak ozel kod

KOMISYON:
+ Desc: Komisyon Toplamı

ODENEN:
+ Desc: Ödenmesi gereken tutar.

TUTAR:
+ Desc: hizmet bazındaki bürüt tutar

IND1_T:
+ Desc: Birinci İndirim Tutarı- trafik indirimi

IND2_T:
+ Desc: İkinci indirim tutarı- Genel indirim

IND3_T:
+ Desc: Üçüncü indirim tutarı- Üçlü İndirim

VOUC:
+ Desc: Voucher sayısı

uclu:
+ Desc: Üçlü Uçuşlara ait fatura olup olmadığı

kod:
+ Desc: Hizmet Kodu- temel,request, dışarıdan saglanan, dhmi, extra satış.

muh-kod:
+ Desc: hizmet tali hesap kodu

tar1:
+ Help: Başlangıç tarihini giriniz...
+ Desc: Faturanın ait olduğu hizmet döneminin başlangıcı.

tar2:

```
    + Help: Bitiş Tarihini giriniz...
    + Desc: Faturanın ait olduğu hizmet döneminin sonu.

aciklama:
    + Desc: Hizmet tanımı- dizi şeklinde

toplam:
    + Desc: Fatura brüt toplam tutarı

vat:
    + Desc: KDV toplam tutarı

doviz:
    + Desc: Döviz cinsi

kur:
    + Desc: Doviz kuru

fat_no:
    + Desc: Fatura Numarası

mek_tar:
    + Desc: mektup tarihi

itotal:
    + Desc: Belli bir hizmetten yapılan toplam indirim

fat_muc:
    + Desc: Son Ödeme tarihi

iptal:
    + Desc: Faturanın iptal olup olmadığı durumu

fana_kod:
    + Desc: Muhasebe ana hesap kodu

falt_kod:
    + Desc: Muhasebe alt hesap kodu
```

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 14
Database: fatura (PROGRESS)
Filename: firm

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	ist_kod	char		i	x(3)	
30	an_kod	char			x(3)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
f_ind*	yes	ist_kod	1	yes	no

Field Validation Criteria, Validation Messages

Help Messages and Descriptions

ist_kod:
+ Help: İstasyonda kullanılan firma kodunu giriniz.....
+ Desc: Firma uçuş Kodu

an_kod:
+ Help: Anlasmada kullanılan firma kodunu giriniz....
+ Desc: Firma bilgilerinde baz alınan uçuş kodu

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 1
Database: fatura (PROGRESS
Filename: firma

Order	Field-Name	dType	Ext	Flgs	Format	Initial
30	uh-kod	char		i	x(3)	
40	firma_ad	char			x(30)	
50	f-ind1	inte			%>9	0
60	f-ind2	inte			%>9	0
70	f-ind3	inte			%>9	0
320	kargo_fiy	deci2			>>>,>>9.99	0
330	cash	logi			CASH/KREDI	KREDI
340	f-yerli	logi			EVET/HAYIR	HAYIR
350	komisyon	inte			%>9	0
360	anlasmali	logi			EVET/HAYIR	HAYIR
370	uclu	logi			EVET/HAYIR	HAYIR
400	aciklama	char	6		x(60)	
410	adres	char	4		x(50)	
430	ozel	logi			Evet/Hayır	Hayır
440	ulke_kod	char			x(3)	
450	adryer	logi			Evet/Hayır	Evet

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
uh-kod	FİRMA_KODU	
firma_ad	Firma Adı	
f-ind1	İNDİRİM1	
f-ind2	INDIRIM2	
f-ind3	İNDİRİM3	
komisyon	Komisyon	
anlasmali	Anlaşmalı mı?	
uclu	Üçlü Uçuş İnd.?	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
fkod*	yes	uh-kod	1	yes	no

Field Validation Criteria, Validation Messages

uh-kod:
+ Criterion: uh-kod <> ""
+ Message:

Help Messages and Descriptions

uh-kod:
+ Desc: firma kodu

firma_ad:
+ Desc: firma adı

f-ind1:
+ Desc: Birinci indirim yüzdesi

f-ind2:
+ Desc: İkinci indirim yüzdesi

f-ind3:
+ Desc: Üçüncü indirim yüzdesi

kargo_fiy:
+ Desc: kargo fiyatı

cash:
+ Desc: Firmanın Cash veya Kredili anlaşmalı olduğu

f-yerli:
+ Desc: firma yerli mi?

komisyon:
+ Desc: Firmadan alınan komisyon yüzdesi.

anlasmali:
+ Desc: firma anlasmali mi ?

uclu:
+ Desc: üçlü uçuş indirimi var mı?

aciklama:
+ Desc: Firmaya ilişkin açıklamalar

adres:
+ Desc: firma fatura adresi.

ozel:
+ Desc: İstasyonda fiyatlar ekranda gösterilsin mi?

ulke_kod:
+ Desc: Firmanın ait olduğu ülke kodu

adryer:
+ Desc: Mektup adresinin yerli olup olmadığı- (son ödeme tarihi için
)

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 1
Database: fatura (PROGRESS
Filename: firsabit

Order	Field-Name	dType	Ext	Flgs	Format	Initial
30	uh-kod	char		i	x(3)	
40	firma_ad	char			x(30)	
330	cash	logi			CASH/KREDI	KREDI
340	f-yerli	logi			EVET/HAYIR	HAYIR
360	anlasmali	logi			EVET/HAYIR	HAYIR
370	uclu	logi			EVET/HAYIR	HAYIR
400	aciklama	char	6		x(60)	
410	adres	char	4		x(50)	
430	ozel	logi			Evet/Hayır	Hayır
440	ulke_kod	char			x(3)	
450	cari_ana	char			x(3)	
460	cari_alt	char			x(2)	
470	cari_tali	char			x(5)	
480	fir_ana	char			x(3)	
490	fir_alt	char			x(2)	
500	fir_tali	char			x(5)	
510	telno	char			x(15)	
520	telexno	char			x(15)	
530	faxno	char			x(15)	
540	temtutar	deci2			>>>,>>>,>>>,>>9.99	0
550	temtar	date			99/99/99	?
560	vade_tut	deci2			>>>,>>>,>>>,>>9.99	0
570	vade_tar	date			99/99/99	?
580	tltutar	deci2			>>>,>>>,>>>,>>9.99	0
590	dovtut	deci2			>>>,>>>,>>9.99	0
600	disana_kod	char			x(3)	
610	disalt_kod	char			x(2)	
620	distali_kod	char			x(5)	
630	isana_kod	char			x(3)	
640	istali_kod	char			x(5)	
650	isalt_kod	char			x(2)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
uh-kod	FİRMA_KODU	
firma_ad	Firma Adı	
anlasmali	Anlaşmalı mı?	
uclu	Üçlü Uçuş İnd.?	
cari_ana	Cari Kod	
temtar	Teminat Tarihi	
vade_tar	Vade Tarihi	
tltutar	TL Tutar	
dovtut	Döviz Tutarı	

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 1
Database: fatura (PROGRESS)
Filename: firsabit

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
fir1*	yes	uh-kod	1	yes	no

Field Validation Criteria, Validation Messages

uh-kod:
+ Criterion: uh-kod <> ""
+ Message:

Help Messages and Descriptions

uh-kod:
+ Desc: firma kodu

firma_ad:
+ Desc: firma adı

cash:
+ Desc: cash,kredili ?

f-yerli:
+ Desc: firma yerli mi?

anlasmali:
+ Desc: firma anlasmali mi ?

uclu:
+ Desc: üçlü uçuş indirimi var mı?

aciklama:
+ Desc: Muhasebe firma bilgilerine ilişkin açıklamalar

ozel:
+ Desc: İstasyonda fiyatlar ekranda gösterilsin mi?

ulke_kod:
+ Desc: Ulke kodu

cari_ana:
+ Desc: Firma cari ana hesap kodu

cari_alt:
+ Desc: Firma cari alt hesap kodu

cari_tali:
+ Desc: Firma cari tali kodu

fir_ana:
+ Desc: Firma muhasebe ana hesap kodu

```
fir_alt:
    + Desc: firma muhasebe alt hesap kodu

fir_tali:
    + Desc: Firma muhasebe tali hesap kodu

telno:
    + Desc: Firma telefon numurası

telexno:
    + Desc: Firma teleks numarası

faxno:
    + Desc: Faks Numarası

temtutar:
    + Desc: Firma TL teminat tutarı

temtar:
    + Desc: Firma teminat tutarı

vade_tut:
    + Desc: Vade Tutarı

tltutar:
    + Desc: Firma Borç/Alacak bakiyesi

dovtut:
    + Desc: Firma döviz teminat tutarı

disana_kod:
    + Help: Dışarıdan Saglananlar ana kod

disalt_kod:
    + Help: Dışarıdan Saglananlar Alt Kod

distali_kod:
    + Help: Dışarıdan Saglananlar Tali Kod.......

isana_kod:
    + Help: İskonto ana_kod
    + Desc: Firma indirimler ana hesap kodu

istali_kod:
    + Help: İskonto tali kod
    + Desc: Firma indirimler tali hesap kodu

isalt_kod:
    + Help: İskonto Alt Kodu Giriniz.
    + Desc: Firma indirimler alt hesap kodu
```

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 2
Database: fatura (PROGRESS
Filename: fiyat

Order	Field-Name	dType	Ext	Flgs	Format	Initial
30	uh-kod	char		i	x(3)	
40	poz	char		i	x(2)	
250	ton_kod	char		i	x(2)	
260	tutar	deci2			->>,>>9.99	0
270	skod	char		i	xx	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
uh-kod	FİRMA_KODU	
ton_kod	TONAJ KODU	
tutar	Tutar	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
fiy_ind*	yes	uh-kod	1	yes	no
		skod	2	yes	no
		poz	3	yes	no
		ton_kod	4	yes	no

Field Validation Criteria, Validation Messages

uh-kod:
+ Criterion: uh-kod <> ""
+ Message:

poz:
+ Criterion: fiyat.poz <> ""
+ Message:

Help Messages and Descriptions

uh-kod:
+ Help: Firma kodunu giriniz
+ Desc: firma kodu

poz:
+ Help: Pozisyon kodunu giriniz
+ Desc: pozisyon kodu

ton_kod:
+ Help: Tonaj kodunu giriniz
+ Desc: tonaj kodu

tutar:
+ Desc: hizmet tutarı

skod:
+ Desc: standard hizmet kodu

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 2
Database: fatura (PROGRESS
Filename: f_adr

Order	Field-Name	dType	Ext	Flgs	Format	Initial
20	uh-istk	char		i	x(3)	
30	uh-kod	char		i	x(3)	
420	m_adr	char	4		x(50)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col Label
uh-istk	IST_KOD	
uh-kod	FIRMA_KODU	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
adr_ind*	yes	uh-kod	1	yes	no
		uh-istk	2	yes	no

Field Validation Criteria, Validation Messages

uh-istk:
+ Criterion: uh-istk <> ""
+ Message:

uh-kod:
+ Criterion: uh-kod <> ""
+ Message:

Help Messages and Descriptions

uh-istk:
+ Desc: istasyon kodu

uh-kod:
+ Desc: firma kodu

m_adr:
+ Desc: Firma mektup adresi

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 2:
Database: fatura (PROGRESS)
Filename: gso

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	gsono	char		i	x(8)	
20	uh-istk	char		i	x(3)	
30	uh-kod	char		i	x(3)	
40	gel_tarih	date			99/99/99	?
50	uh-no1	char		i	x(5)	
70	asaat2	char			xx.xx	
100	uh-gel	char			x(3)	---
110	uh-git	char			x(3)	---
120	uh-gels	char			xx.xx	
130	git_tarih	date			99/99/99	?
140	uh-no2	char		i	x(5)	
150	dsaat1	char			xx.xx	
160	uh-gits	char			xx.xx	
210	ucus_tipi	char			x(1)	S
220	uh-reg	char			x(6)	
380	uh-tarih	date		i	99/99/99	?
390	stand	logi			STANDARD/SURCHARGE	STANDARD
460	rkod	char	50		x(3)	
500	rtoplam	deci2	50		->>,>>9.99	0
510	rbirim	inte	50		>>>>9	0
520	uclu	logi			EVET/HAYIR	HAYIR
530	uh-saat	char			xx.xx	
550	stutar	deci2	10		->>,>>9.99	0
560	sflag	logi	10		Evet/Hayır	Hayır
570	skod	char	10		xxx	
590	Toplam	deci2			>>>,>>>,>>9.99	0
600	fkod1	char			x(3)	
610	fkod2	char			x(3)	
620	fkod	char			x(4)	NORM
630	poz	char			x(2)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
gsono	GSO_NO	
uh-istk	IST_KOD	
uh-kod	FIRMA_KODU	
gel_tarih	GELIS TARIHI	
uh-no1	FLT_NO	
uh-no2	FLT_NO	
uh-reg	UCAK TESCILI	
uclu	UÇLU UÇUŞ IND ?	
stutar	Tutar	
skod	Kod	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
gsono*	yes	gsono	1	yes	no
uh-ind	yes	uh-tarih	1	yes	no
		uh-istk	2	yes	no
		uh-kod	3	yes	no
		uh-no1	4	yes	no
		uh-no2	5	yes	no

Field Validation Criteria, Validation Messages

gsono:
+ Criterion: gsono <> ""
+ Message:

uh-istk:
+ Criterion: uh-istk <> ""
+ Message:

uh-kod:
+ Criterion: uh-kod <> ""
+ Message:

Help Messages and Descriptions

gsono:
+ Desc: Ground Handling Charge Note Numarası

uh-istk:
+ Desc: istasyon kodu

uh-kod:
+ Help: Uçuşa ait firma kodunu giriniz ....
+ Desc: firma kodu

gel_tarih:
+ Desc: Gelis Tarihi

uh-no1:
+ Desc: gelis flt no

asaat2:
+ Desc: kapı acılıs saati

uh-gel:
+ Desc: gelis yeri

uh-git:
+ Desc: gidis yeri

uh-gels:
+ Desc: Geliş saati (Yere İniş)

```
git_tarih:
    + Desc: gidis tarihi

uh-no2:
    + Desc: gidis flight nosu

dsaat1:
    + Desc: kapi kapanis saati

uh-gits:
    + Desc: gidis saati

ucus_tipi:
    + Desc: ucus_tipi

uh-reg:
    + Desc: uçak tescili

uh-tarih:
    + Desc: GHCN Tarihi

stand:
    + Desc: Zamlı tarife mi?

rkod:
    + Desc: Alınan request hizmet kodu

rtoplam:
    + Desc: Alınan request hizmet toplamı

rbirim:
    + Desc: Alınan Request Hizmetinin Birimi

uclu:
    + Desc: üçlü uçuş indirimi var mı?

uh-saat:
    + Desc: GHCN saati

stutar:
    + Desc: Standard hizmet tutarı

sflag:
    + Desc: Standard Hizmet Alıp almadığı

skod:
    + Help: Yardım için F10 tuşuna basınız.
    + Desc: Standard hizmet kodu

Toplam:
    + Desc: GHCN Toplamı
```

fkod1:
+ Desc: Aktüel geliş uçuş kodu

fkod2:
+ Desc: Aktüel gidiş Uçuş kodu

fkod:
+ Help: Özel bir uçuş tipi ise kodunu giriniz.....
+ Desc: fatura için kullanılacak ozel kod

poz:
+ Desc: Pozisyon Kodu

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 2
Database: fatura (PROGRESS)
Filename: gsofat
Hangi gso, hangi faturaya ait tespit ediliyor.

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	ghcno	char		i	x(8)	
940	fat_num	inte			999999	0

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
ghcno	GHCNO	
fat_num	Fatura No	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
ghc_ind*	yes	ghcno	1	yes	no

Field Validation Criteria, Validation Messages

ghcno:
+ Criterion: ghcno <> ""
+ Message:

```
06/10/94 20:43:57        PROGRESS Data Dictionary Report                Page 2(
Database: fatura                                                        (PROGRESS)
Filename: istasyon
```

Order	Field-Name	dType	Ext	Flgs	Format	Initial
20	uh-istk	char		i	x(3)	
30	istad	char			x(18)	
40	ist_yerli	logi			EVET/HAYIR	EVET
50	tel	char			x(16)	
60	sehir	char			x(10)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
uh-istk	IST_KOD	
istad	ISTASYON ADI	
tel	Tel No	
sehir	Şehir	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
istk*	yes	uh-istk	1	yes	no

Field Validation Criteria, Validation Messages

uh-istk:
+ Criterion: uh-istk <> ""
+ Message:

Help Messages and Descriptions

uh-istk:
+ Desc: istasyon kodu

istad:
+ Desc: istasyon adı

ist_yerli:
+ Desc: istasyonun yerli mi oldugu

sehir:
+ Help: İstasyonun bulunduğu şehri giriniz...

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 2
Database: fatura (PROGRESS
Filename: muhist

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	uh-istk	char		i	x(3)	
20	muh-istk	char			x(2)	
40	vmuh-kod	char			x(5)	
50	kom-kod	char			x(5)	
110	valt_kod	char			x(2)	
120	vana_kod	char			x(3)	
130	kana_kod	char			x(3)	
140	kalt_kod	char			x(2)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
uh-istk	İstasyon Kodu	
muh-istk	Muh. Kod	
kana_kod	Komisyon Kodu	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
istind1*	yes	uh-istk	1	yes	no

Field Validation Criteria, Validation Messages

uh-istk:
+ Criterion: uh-istk <> ""
+ Message:

Help Messages and Descriptions

uh-istk:
+ Help: Uçuş İstasyon Kodunu giriniz...
+ Desc: istasyon kodu

muh-istk:
+ Help: Muhasebe istsayon kodunu giriniz...
+ Desc: Muhasebe Database indeki istasyon kodu.

vmuh-kod:
+ Desc: vat muhasebe kodu

kana_kod:
+ Desc: Komisyon anakod

kalt_kod:
+ Desc: komisyon alt kod

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 3(
Database: fatura (PROGRESS)
Filename: oreq

Order	Field-Name	dType	Ext	Flgs	Format	Initial
30	uh-kod	char		i	x(3)	
50	uh-no1	char		i	x(5)	
140	uh-no2	char		i	x(5)	
380	uh-tarih	date		i	99/99/99	?
390	orkod	char		im	x(3)	
400	ortutar	deci2			->>,>>9.99	0
410	orvat	deci2			->>,>>9.99	0
420	oradet	inte			->,>>>,>>9	0
430	orkom	logi			Evet/Hayır	Hayır
520	uclu	logi		im	Evet/Hayır	Hayır
530	uh-istk	char		i	x(3)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
uh-kod	FİRMA_KODU	
uh-no1	FLT_NO	
uh-no2	FLT_NO	
orkod	KOD	
orkom	Komisyon	
uclu	UÇLU UÇUŞ İND ?	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
orkd*	yes	uh-[illegible]	1	yes	no
		uh-tarih	2	yes	no
		uh-kod	3	yes	no
		uclu	4	yes	no
		uh-no1	5	yes	no
		uh-no2	6	yes	no
		orkod	7	yes	no

Field Validation Criteria, Validation Messages

uh-kod:
+ Criterion: uh-kod <> ""
+ Message:

Help Messages and Descriptions

uh-kod:
+ Desc: firma kodu

uh-no1:
+ Desc: gelis flt. no

uh-no2:

+ Desc: gidis flight nosu

uh-tarih:
+ Desc: gso açılma tarihi

orkod:
+ Help: Yardım için F10 tuşuna basınız.

uclu:
+ Desc: üçlü uçuş indirimi var mı?

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 3:
Database: fatura (PROGRESS)
Filename: pozisyon

Order	Field-Name	dType	Ext	Flgs	Format	Initial
40	poz	char		i	x(2)	
50	aciklama	char			x(20)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
aciklama	Pozisyon Adı	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
poz_kod*	yes	poz	1	yes	no

Field Validation Criteria, Validation Messages

Help Messages and Descriptions

poz:
+ Help: Pozisyon kodunu giriniz
+ Desc: pozisyon kodu

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 3
Database: fatura (PROGRESS
Filename: reqfiyat

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	reqkod	char		i	x(3)	
20	reqtur	char		i	x(32)	
30	reqsure	inte			>>>9	0
40	reqfiyat	deci2			->>,>>9.99	0
50	muh-kod	char			x(5)	
60	ana_kod	char			x(3)	
70	alt_kod	char			x(2)	
80	uretim_yeri	char			x(1)	
90	istasyon	char			x(2)	
100	bolum	char			x(2)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
reqkod	REQUEST KODU	
reqtur	REQUEST TURU	
reqsure	REQUEST SÜRESİ	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Ab
rad	yes	reqtur	1	yes	nc
rkod*	yes	reqkod	1	yes	nc

Field Validation Criteria, Validation Messages

reqkod:
+ Criterion: reqfiyat.reqkod <> ""
+ Message:

Help Messages and Descriptions

reqkod:
+ Help: Request kodunu giriniz.
+ Desc: Request kodu

uretim_yeri:
+ Help: Üretim Yerini Giriniz

istasyon:
+ Help: İstasyonu giriniz...

bolum:
+ Help: Bölümü giriniz....

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	reqkod	char		i	x(3)	
30	reqsure	inte			>>>9	0
40	reqfiyat	deci2			->>,>>9.99	0
50	uh-kod	char		i	x(3)	
60	indirim	logi			EVET/HAYIR	EVET

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
reqkod	REQUEST KODU	
reqsure	REQUEST SÜRESİ	
uh-kod	FIRMA_KODU	
indirim	IND.?	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
kod1*	yes	uh-kod	1	yes	no
		reqkod	2	yes	no

Field Validation Criteria, Validation Messages

reqkod:
+ Criterion: request.reqkod <> ""
+ Message:

uh-kod:
+ Criterion: uh-kod <> ""
+ Message:

Help Messages and Descriptions

reqkod:
+ Help: Request kodunu giriniz.
+ Desc: Request kodu

uh-kod:
+ Desc: firma kodu

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 3
Database: fatura (PROGRESS)
Filename: scode

Order	Field-Name	dType	Ext	Flgs	Format	Initial
620	fkod	char		i	x(4)	NORM
630	sc-aciklama	char			x(12)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
sc-ind*	yes	fkod	1	yes	no

Field Validation Criteria, Validation Messages

Help Messages and Descriptions

fkod:
+ Desc: fatura için kullanılacak ozel kod

sc-aciklama:
+ Desc: Firma özel kodlarının açıklaması

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 3(
Database: fatura (PROGRESS)
Filename: shizmet

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	skod	char		i	x(2)	
20	saciklama	char		i	x(16)	
30	muh-kod	char			x(5)	
60	ana_kod	char			x(3)	
70	alt_kod	char			x(2)	
80	uretim_yeri	char			x(1)	
100	bolum	char			x(2)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
skod*	yes	skod	1	yes	no
s_ac	yes	saciklama	1	yes	yes

Field Validation Criteria, Validation Messages

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 3'
Database: fatura (PROGRESS
Filename: tarife

Order	Field-Name	dType	Ext	Flgs	Format	Initial
250	ton_kod	char		i	x(2)	
260	ictut	deci2			->>,>>9.99	0
270	distut	deci2			->>,>>9.99	0
280	skod	char		i	x(2)	
290	tonaj1	inte			>,>>>,>>9	0
300	tonaj2	inte			>,>>>,>>9	0

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
ton_kod	TONAJ KODU	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abb
tonkod*	yes	ton_kod	1	yes	no
		skod	2	yes	no

Field Validation Criteria, Validation Messages

Help Messages and Descriptions

ton_kod:
+ Desc: tonaj kodu

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 3
Database: fatura (PROGRESS
Filename: tatil

Order	Field-Name	dType	Ext	Flgs	Format	Initial
10	tarih	date		i	99/99/99	?
20	aciklama	char			x(25)	
30	saat1	char			xx.xx	
40	saat2	char			xx.xx	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
tar*	yes	tarih	1	yes	no

Field Validation Criteria, Validation Messages

06/10/94 20:43:57 PROGRESS Data Dictionary Report Page 3
Database: fatura (PROGRESS
Filename: ucak

Order	Field-Name	dType	Ext	Flgs	Format	Initial
220	uh-reg	char		i	x(6)	
230	tonaj	inte			>,>>>,>>9	0
240	uh-tip	char			x(12)	
250	ton_kod	char			x(2)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Field-Name	Label	Col-label
uh-reg	UCAK TESCİLİ	
tonaj	UÇAK AĞIRLIĞI	
uh-tip	UÇAK TİPİ	
ton_kod	TONAJ KODU	

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abbr
uckod*	yes	uh-reg	1	yes	no

Field Validation Criteria, Validation Messages

uh-reg:
+ Criterion: ucak.uh-reg <> ""
+ Message: Uçak Tescilini girmek zorundasınız...

Help Messages and Descriptions

uh-reg:
+ Desc: uçak tescili

tonaj:
+ Desc: uçak agirligi

uh-tip:
+ Desc: uçak tipi

ton_kod:
+ Desc: tonaj kodu

Order	Field-Name	dType	Ext	Flgs	Format	Initial
440	ulke_kod	char		i	x(3)	
450	ulke_ad	char			x(30)	

+ Data Dictionary Report Legend +
c - field is case-sensitive : m - field is mandatory
i - field participates in an index : v - field is a view component

Index Name (* indicates primary)	Unique	Field Name	Seq	Asc	Abl
ulke_ind*	yes	ulke_kod	1	yes	no

Field Validation Criteria, Validation Messages

Help Messages and Descriptions

ulke_kod:

```
/************Cash Faturalama Programı*************/
/************Cash3.p*********************/
{fatura.i} /****Tanımlamalar***/
status input "Gerekli Bilgileri Giriniz veya Çıkmak için <F4> tuşuna basınız.".
FORM
  skip(1)
  "Fatura No      : " at 2 fat_no
  "Fatura Tarihi : " at 2 fat_tar
  "Özel Kod       : " at 2 fatura.fkod
  WITH FRAME FATGIR WITH NO-LABELS OVERLAY CENTERED ROW 8
       TITLE "Cash Faturalama".
form "Firma Kodu      :" at 2 gso.uh-kod
     "G.H.C.N Tarihi :" at 2 gso.uh-tarih
     "Flight No       :" at 2 gso.uh-no1 "/" gso.uh-no2
     with frame gsofrm row 2  centered  no-label
     title "Faturalanacak GHCN 'ye İlişkin Bilgiler".
 FORM  fatura.aciklama AT 1 tut AT 40 WITH FRAME FATURA
 2 COLUMNS.

/* disp  " F4 : Çıkış        F8: Fatura İptali " with row 18 centered.*/
 find last fatura use-index fatno no-lock no-error.
 if not available fatura then old_no = 0.
 else old_no = fatura.fat_no.
 hide all no-pause.
 prompt-for gso.uh-kod auto-return
 gso.uh-tarih auto-return
 gso.uh-no1 auto-return
 gso.uh-no2 with frame gsofrm.
 find gso where gso.uh-istk  = userid("fatura") and
  gso.uh-kod   = input gso.uh-kod and
  gso.uh-tarih = input gso.uh-tarih and
  gso.uh-no1   = input gso.uh-no1  and
  gso.uh-no2   = input gso.uh-no2
 no-lock no-error.
 if not available gso then do:
    message "Böyle bir G.H.C.N. mevcut değil.".
    undo , retry.
 end.
 create fatura.
 assign fatura.fat_no = old_no  + 1
 fatura.uh-kod = gso.uh-kod
 fatura.fat_tar = today
 fatura.fkod    = gso.fkod
 fatura.uh-istk = userid("fatura").

 old_no = fatura.fat_no.
     disp fatura.fat_no with frame fatgir.
     update
     fatura.fat_tar auto-return
     fatura.fkod     auto-return  with frame fatgir.

 /**********Mevcut indirimler uygulansın mı**********************/
  display "Mevcut İndirimlerin yapılmasını istiyor musunuz?" skip
   sec[1] at 5 sec[2] at 30 with centered frame x no-label.
  choose field sec with frame x.
  if frame-index = 1 then indirim_flag = true.
  else indirim_flag = false.
 /*****************************************************************/

 find firma where firma.uh-kod = fatura.uh-kod no-lock no-error.
 if not available firma then do:
    message "Böyle bir firma kodu mevcut değil.".
    undo , retry
```

```
 end.
 pp = 1. kl = 1. tvat = 0.
 fatura.komisyon = 0. BOSTOP = 0. s_top = 0. xx = 1. k = 1. t = 0.
for each shizmet where saciklama begins "traf":
 tno = integer(skod).
end.
for each shizmet where saciklama begins "ram":
 rno = integer(skod).
end.

 find gso where
   gso.uh-tarih = input gso.uh-tarih  and
   gso.uh-istk  = userid("fatura") and
   gso.uh-kod   = input gso.uh-kod    and
   gso.uh-no1   = input gso.uh-no1    and
   gso.uh-no2   = input gso.uh-no2    use-index uh-ind no-lock.
 if not available gso then do:
    message "Bu GSO mevcut değil. Lütfen yeniden deneyiniz.".
    undo, next.
 end.
 else do:
 assign fatura.tar1 = gso.uh-tarih
 fatura.tar2 = gso.uh-tarih.
/*****************************************************************************/
/*             Dışarıdan Sağlananların Hesaplatılması                        */
/*****************************************************************************/


  for each oreq where oreq.uh-kod    =  gso.uh-kod   and
       oreq.uh-tarih  =  gso.uh-tarih and
       oreq.uh-no1    =  gso.uh-no1   and
       oreq.uh-no2    =  gso.uh-no2   and
       oreq.uclu      =  gso.uclu     and
       oreq.uh-istk   =  gso.uh-istk no-lock :
    for  each diskod where diskod.orkod = oreq.orkod:
      okod[pp] = orkod.
      oT[pp] = oT[pp] + oreq.ortutar.
      vt[pp] = vt[pp] + oreq.orvat.
      if oreq.orkom then do:
   kt[pp] = kt[pp] + (oreq.ortutar + oreq.orvat) .
      end.

      tvat = tvat + oreq.orvat.
   B1:
     DO LL = 1 TO (K1 - 1):
        IF oKOD[k1] = oKOD[LL] THEN
   DO:
     oT[LL] = oT[LL] + oreq.ortutar.
     vt[ll] = vt[ll] + oreq.orvat.
     if oreq.orkom then
     kt[ll] = kt[ll] + oreq.ortutar .
     oKOD[K1] = "".
     oT[K1] = oT[K1] - oreq.ortutar.
     vt[k1] = vt[k1] - oreq.orvat.
     if oreq.orkom then
     kt[k1] = kt[k1] - oreq.ortutar .
     pp = pp - 1.
     LEAVE B1.
   END.
     END.
   pp = pp + 1. kl = pp .
       END.
  end.
   IF gso.ucus_tipi = "f" THEN BOSTOP = BOSTOP + gso.stutar[tno].
    TOT = 0. EXT = 0. tut = 0.
    DO TT = 1 TO 15 .
```

```
 find request where request.uh-kod = fatura.uh-kod and
      request.reqkod = gso.rkod[tt] no-error.
 if available request then do:
 if not request.indirim then indx_top = indx_top + gso.rtoplam[tt].
/* IF gso.REQFLAG[TT] THEN fatura.komisyon = komisyon + gso.R_TOPLAM[TT].*/

 IF gso.RKOD[TT] <> "" THEN
   DO:
    KOD[XX] = gso.RKOD[TT].
    T[XX]   = T[XX] + gso.RTOPLAM[TT].
B1:
  DO LL = 1 TO (K - 1):
    IF KOD[K] = KOD[LL] THEN
DO:
  T[LL] = T[LL] + gso.RTOPLAM[TT].
  KOD[K] = "".
  T[K] = T[K] - gso.RTOPLAM[TT].
  XX = XX - 1.
  LEAVE B1.
END.
  END.
XX = XX + 1. K = XX .
END.
end.
    END.
     do h = 1 to 6:
      s_top[h]  =  s_top[h] + gso.stutar[h] .
     end.
END.
    I = 1. kno = 1.
h = 0.
for each shizmet by skod:
 h = h + 1.
 if  s_top[h] > 0 then do:
  fatura.aciklama[i]  = saciklama.
  tut[i]        = s_top[h].
  fatura.kod[kno]    = shizmet.skod.
  fatura.muh-kod[kno]    = shizmet.muh-kod.
  fatura.tutar[kno] = s_top[h].
  /* tnot trafic indis göstergesi */
  /* rnot ramp   indis göstergesi */
  if integer(fatura.kod[kno]) = tno then tnot = kno.
  else if integer(fatura.kod[kno]) = rno then rnot = kno.
  else
itotal[kno] = firma.f-ind2 * fatura.tutar[kno] / 100.
  kno = kno + 1.
  tot = tot + tut[i].
  i = i + 1.
     end.
    end.

REPEAT LL = 1 TO (k + 2):
     FIND REQfiyat WHERE REQfiyat.REQKOD = KOD[LL] no-error.
     IF AVAILABLE reqfiyat and t[11] > 0 THEN DO:
fatura.kod[kno]    = reqfiyat.reqkod.
fatura.muh-kod[kno]    = reqfiyat.muh-kod.
fatura.tutar[kno] = t[11].
   find request where request.uh-kod = fatura.uh-kod and
        request.reqkod = reqfiyat.reqkod no-lock no-error.
   if  indirim_flag and request.indirim then
fatura.itotal[kno] =  (firma.f-ind2 / 100) * fatura.tutar[kno].

kno = kno + 1.
fatura.aciklama[I] = reqfiyat.REQTUR.
tut[I] = T[LL].
TOT = tut[I] + TOT.
```

```
I = I + 1.
      END.
 END.
 REPEAT LL = 1 TO (pp + 2):
      FIND diskod WHERE diskod.orkod = oKOD[LL] no-error.
      IF AVAILABLE diskod THEN DO:
if ot[ll] > 0 then do:
fatura.kod[kno]     = diskod.orkod.
fatura.muh-kod[kno]     = diskod.muh-kod.
fatura.tutar[kno]  = ot[ll].
indx_top = indx_top + fatura.tutar[kno].
kno = kno + 1.
fatura.aciklama[I]  = diskod.daciklama.
tut[I] = oT[LL].
TOT    = tut[I] + TOT.
I = I + 1.
      end.
if vt[ll] > 0 then do:   /* Eğer KDV si varsa */
 fatura.kod[kno]    = diskod.orkod.
 fatura.muh-kod[kno]    = diskod.vmuh-kod.
 fatura.tutar[kno] = vt[ll].
indx_top = indx_top + fatura.tutar[kno].
 kno = kno + 1.
 fatura.aciklama[i] = diskod.daciklama + " (VAT)".
 tut[i] = vt[ll].
 tot    = tut[i] + tot.
 i = i + 1.
end.

if kt[ll] > 0 then do:
   fatura.kod[kno]    = diskod.orkod.
   fatura.muh-kod[kno]    = diskod.kom-kod.
   fatura.tutar[kno]  = kt[ll] * firma.komisyon / 100.
   indx_top = indx_top +  fatura.tutar[kno].
   fatura.komisyon = kt[ll] + fatura.komisyon.
   kno = kno + 1.
  /*fatura.aciklama[i] = diskod.daciklama + " (KOM.)".*/
   tut[i] = kt[ll] * firma.komisyon / 100.
   tot    = tut[i] + tot.
   i = i + 1.
end.
      END.
  END.


      TOP = 0.
      DO LL = 1 TO I - 1:
       TOP = tut[LL] + TOP.
      END.
      fatura.komisyon = firma.komisyon * fatura.komisyon / 100.
      FIND FIRMA WHERE FIRMA.uh-kod = fatura.uh-kod no-lock NO-ERROR.
      IF AVAILABLE FIRMA and fatura.komisyon > 0 then
      DO:
       KOMCH = STRING(FIRMA.KOMISYON).
       fatura.ACIKLAMA[I] = "COM. (%" + KOMCH + ")".
       fatura.tutar[i] = fatura.komisyon.
       tut[I] = fatura.komisyon .
       I = I + 1.
      END.
      fatura.aciklama[I] = "T O T A L".
      tut[I] = TOP .
      xx = i.
      I = I + 1.

       TR = s_top[tno] - BOSTOP.
       BA = s_top[rno].
```

```
disc = 0.

  if  indirim_flag = true then do:
    FIND FIRMA WHERE FIRMA.uh-kod =  fatura.uh-kod NO-ERROR.
IF  gso.uclu THEN
 DO:
   if tr > 0 then
   itotal[tnot] = firma.f-ind3 * (tr / 100).
   if ra > 0 then
   itotal[rnot] = firma.f-ind3 * (ra / 100).
   FATURA.IND3_T =  (TR + RA) * FIRMA.f-IND3 / 100.
   TR = TR - (TR * FIRMA.f-IND3 / 100).
   RA = RA - FIRMA.f-IND3 * RA / 100.
 END.

IF FIRMA.f-IND1 > 0 THEN DO:
    if tr > 0 then
    itotal[tnot] = itotal[tnot] + (firma.f-ind1 * tr / 100).
    FATURA.IND1_T = firma.f-ind1 * tr / 100.
    tr = tr - fatura.ind1_t.
END.

      IF FIRMA.f-iND2 > 0 THEN DO:
    if tr > 0 then
    itotal[tnot] = itotal[tnot] + ((firma.f-ind2 / 100) * tr).
    if ra > 0 then
    itotal[rnot] = itotal[rnot] + (firma.f-ind2 * ra / 100).
 FATURA.IND2_T = ( TOP - (fatura.ind1_t + fatura.ind3_t + indx_top
     + bostop)) *
     FIRMA.f-IND2 / 100.
      END.

   IND1ST = STRING(FIRMA.f-IND1).
   IND2ST = STRING(FIRMA.f-IND2).
   IND3ST = STRING(FIRMA.f-IND3).
   disc = fatura.ind1_t + fatura.ind2_t + fatura.ind3_t.
 end. /* indirim_flag */
if disc > 0 then do:
   fatura.aciklama[I] = "DISCOUNT".
   tut[I] = disc.
   DISC = tut[1].
   I = I + 1.
end.

      ASSIGN FATURA.ODENEN = top - disc
      fatura.toplam = top.

      run cashyaz3.p(gso.gsono,fatura.fat_no).
      return.
```

```
/*****************Extra Faturalama Programı******************/
def new shared buffer  b_fatura for fatura.
def var cnt as int.
def var son_tarih as logical format "Evet/Hayır" initial "Evet".
def var kom_flag as logical format "Evet/Hayır" initial "Hayır" extent 10 .
def new shared workfile w_fatura like fatura.
def var vat_flag as logical initial "false" extent 10.
def var found_flag as logical.
def var firm_ad like firma.firma_ad.
def var ref as int format ">>".
def var indekr as decimal format ">>,>>>.99" extent 10.
def var ind_top as decimal.
def new shared var vat_tut as decimal format "->>>,>>>,>>9.99" extent 10.
def var vat_yuz as integer format "%99" extent 10.
define  new shared var n as integer.
def new shared var reff as int .
def var i as integer format ">>".
def new shared var kdv as int format "%99".
def new shared var h_tanim as char format "x(20)" extent  30.
def var old_no like fatura.fat_no.
def var komm as int format "%99".
def var faturano like fatura.fat_no.
status input " F2 : Standard Hizmet F3 : Request F5 : Disaridan Saglananlar".
form
"Firma Kodu    : "  w_fatura.uh-kod  no-label
firma.firma_ad  at 36 format  "x(30)" no-label  skip
"Fatura Tarihi : " w_fatura.fat_tar  no-label
"Özel Kod      : " at 36 w_fatura.fkod no-label skip(1)
 "   KOD:    HİZMET TANIMI :   TUTARI:     VAT(KDV): INDİRİM:" skip
 "1 " w_fatura.kod[1]  h_tanim[1]  w_fatura.tutar[1] format  "->>>,>>>,>>9.99"
 space(2) vat_yuz[1]   indekr[1] skip
 "2 " w_fatura.kod[2]  h_tanim[2]  w_fatura.tutar[2] format "->>>,>>>,>>9.99"
 space(2) vat_yuz[2]  indekr[2] skip
 "3 " w_fatura.kod[3]  h_tanim[3]  w_fatura.tutar[3] format "->>>,>>>,>>9.99"
 space(2) vat_yuz[3]  indekr[3] skip
 "4 " w_fatura.kod[4]  h_tanim[4]  w_fatura.tutar[4] format "->>>,>>>,>>9.99"
 space(2) vat_yuz[4]  indekr[4] skip
 "5 " w_fatura.kod[5]  h_tanim[5]  w_fatura.tutar[5] format "->>>,>>>,>>9.99"
 space(2) vat_yuz[5]  indekr[5] skip
 "6 " w_fatura.kod[6]  h_tanim[6]  w_fatura.tutar[6] format "->>>,>>>,>>9.99"
 space(2) vat_yuz[6]  indekr[6] skip
 "7 " w_fatura.kod[7]  h_tanim[7]  w_fatura.tutar[7] format "->>>,>>>,>>9.99"
 space(2) vat_yuz[7]  indekr[7] skip
 "8 " w_fatura.kod[8]  h_tanim[8]  w_fatura.tutar[8] format "->>>,>>>,>>9.99"
 space(2) vat_yuz[8]  indekr[8] skip
 "9 " w_fatura.kod[9]  h_tanim[9]  w_fatura.tutar[9] format "->>>,>>>,>>9.99"
 space(2) vat_yuz[9]  indekr[9] skip
 "10" w_fatura.kod[10] h_tanim[10] w_fatura.tutar[10] format "->>>,>>>,>>9.99"
 space(2) vat_yuz[10] indekr[10] skip
 "Komisyon : "  w_fatura.komisyon  "Total        :"  at 27 w_fatura.toplam  skip
 "Discount : "  w_fatura.ind2_t no-label
 "Net Ödenecek :" at 27 w_fatura.odenen format "->>>,>>>,>>9.99"
 with frame ext_frm centered with no-labels
 title "Extra Fatura Tanzimi".
 form i w_fatura.kod[i] h_tanim[i] kom_flag[i] vat_yuz[i]
 with 5 down frame x overlay.
 view frame ext_frm .
 pause 0.
status input "Bilgi giriniz ya da cikmak icin "
       + kblabel("end-error")
```

```
        + " tusuna basiniz".
clear frame ext_frm.    pause 0.
   basla:
   repeat :
   find first w_fatura no-error.
   if available w_fatura then
   delete w_fatura.
   create w_fatura.
   clear frame ext_frm.
   vat_tut = 0. h_tanim = "". indekr = 0. vat_yuz = 0.
   assign
   w_fatura.uh-istk = userid("fatura")
   w_fatura.fat_tar = today
   w_fatura.fkod    = "NORM"
   w_fatura.mek_tar = today.

status input "F2 : Standard Hizmet F3 : Request F5 : Dış. Sağ. F6 : Extra ".
pause 0.
   w_fatura.uh-istk = userid("fatura").
   update
       w_fatura.uh-kod
       w_fatura.fat_tar auto-return
       w_fatura.fkod
       w_fatura.kod[1]    w_fatura.tutar[1]   vat_yuz[1]  indekr[1]
       w_fatura.kod[2]    w_fatura.tutar[2]   vat_yuz[2]  indekr[2]
       w_fatura.kod[3]    w_fatura.tutar[3]   vat_yuz[3]  indekr[3]
       w_fatura.kod[4]    w_fatura.tutar[4]   vat_yuz[4]  indekr[4]
       w_fatura.kod[5]    w_fatura.tutar[5]   vat_yuz[5]  indekr[5]
       w_fatura.kod[6]    w_fatura.tutar[6]   vat_yuz[6]  indekr[6]
       w_fatura.kod[7]    w_fatura.tutar[7]   vat_yuz[7]  indekr[7]
       w_fatura.kod[8]    w_fatura.tutar[8]   vat_yuz[8]  indekr[8]
       w_fatura.kod[9]    w_fatura.tutar[9]   vat_yuz[9]  indekr[9]
       w_fatura.kod[10]   w_fatura.tutar[10]  vat_yuz[10] indekr[10]
       with frame ext_frm
       editing:
   readkey.
   if frame-field = "uh-kod" and lastkey = keycode("enter") and
      frame-value <> "SPL" then do:
      find firma where
      firma.uh-kod = input w_fatura.uh-kod no-lock  no-error.
      find firsabit where
      firsabit.uh-kod = input w_fatura.uh-kod no-lock  no-error.

      if not available firma then do:
         message "Geçersiz Firma İsmi!".
         next.
      end.
      disp firma.firma_ad with frame ext_frm.
   end.

   if frame-field = "fkod" and lastkey = keycode("enter") and
      input w_fatura.uh-kod = "SPL" then do:
      find scode where scode.fkod = input frame ext_frm w_fatura.fkod
      no-lock no-error.
      if not available scode then do:
         run stgir.p.
         next.
      end.
      disp scode.sc-aciklama @ firm_ad with frame ext_frm.
   end.

   if frame-field = "kod" then do:
   if lastkey = keycode("enter") then do:
      if frame-value = "" then do:
       message "Bos gecemezsiniz." .
       next.
```

```
end.
found_flag = false.
find diskod where diskod.orkod = input w_fatura.kod[frame-index]
no-lock no-error.
if available diskod then do:
  disp diskod.daciklama @
  h_tanim[frame-index]
    with frame ext_frm.
  assign w_fatura.muh-kod[frame-index]  = firsabit.distali_kod
  w_fatura.falt_kod[frame-index] = firsabit.disalt_kod
  w_fatura.fana_kod[frame-index] = firsabit.disana_kod.
   found_flag = true.
end.
find extkod where extkod.ext_kod= input w_fatura.kod[frame-index]
no-lock no-error.
if available extkod then do:
  disp extkod.eaciklama @
  h_tanim[frame-index]
    with frame ext_frm.
  assign w_fatura.muh-kod[frame-index]  = extkod.muh-kod
  w_fatura.falt_kod[frame-index] = extkod.alt_kod.
if firma.f-yerli then
w_fatura.fana_kod[frame-index]  = "600".
else w_fatura.fana_kod[frame-index] = "601".
    found_flag = true.
 end.
 find reqfiyat where
 reqfiyat.reqkod = input w_fatura.kod[frame-index]
 no-lock no-error.
 if available reqfiyat then do:
  disp reqfiyat.reqtur @
  h_tanim[frame-index]
     with frame ext_frm.
  assign w_fatura.muh-kod[frame-index]  = reqfiyat.muh-kod
  w_fatura.falt_kod[frame-index] = reqfiyat.alt_kod.
   if firma.f-yerli then
w_fatura.fana_kod[frame-index]  = "600".
    else w_fatura.fana_kod[frame-index] = "601".
  found_flag = true.
 end.
 find shizmet where shizmet.skod = input w_fatura.kod[frame-index]
 no-lock no-error.
 if available shizmet then do:
   disp shizmet.saciklama @
     h_tanim[frame-index]
     with frame ext_frm.
   assign w_fatura.muh-kod[frame-index] = shizmet.muh-kod
   w_fatura.falt_kod[frame-index] = shizmet.alt_kod.
   if firma.f-yerli then
w_fatura.fana_kod[frame-index]  = "600".
    else w_fatura.fana_kod[frame-index] = "601".
    found_flag = true.
 end.
 if not found_flag then do:
    message "Bu kod mevcut degil , Yeniden giriniz...".
    next.
 end.
 assign w_fatura.kod[frame-index]
 h_tanim[frame-index].
end.
 if lastkey = keycode("F5") then do:
 {wfatref.i}
   ref = n.
   run oextw.p.
status input " F2 : Standard Hizmet F3 : Request F5 : Dışarıdan Sağlananlar".
   do i = 1 to n:
```

```
        disp w_fatura.kod[i] h_tanim[i] with frame ext_frm.
      end.
      next.
    end.
    if lastkey = keycode("F3") then do:
    {wfatref.i}
      run oextrw.p.
status input " F2 : Standard Hizmet F3 : Request F5 : Dışarıdan Sağlananlar".
      do i = 1 to n:
        disp w_fatura.kod[i] h_tanim[i] with frame ext_frm.
      end.
      next.
    end.

    if lastkey = keycode("F6") then do:
    {wfatref.i}
      ref = n.
      run oextew.p.
status input " F2 : Standard Hizmet F3 : Request F5 : Dışarıdan Sağlananlar".
      do i = 1 to n:
        disp w_fatura.kod[i] h_tanim[i] with frame ext_frm.
      end.
      next.
    end.

    if lastkey = keycode("F2") then do:
    {wfatref.i}
      run oextsw.p.
status input " F2 : Standard Hizmet F3 : Request F5 : Dışarıdan Sağlananlar".
      do i = 1 to n:
        disp w_fatura.kod[i] h_tanim[i] with frame ext_frm.
      end.
      next.
    end.
      end.
    apply lastkey.
        end.

        if lastkey = keycode("F4") then leave.
        w_fatura.toplam = 0. w_fatura.vat = 0.

        do i = 1 to 10:
     vat_tut[i] = (input w_fatura.tutar[i] - input indekr[i]) *
      (input frame ext_frm vat_yuz[i] / 100).
     w_fatura.toplam = w_fatura.toplam + input w_fatura.tutar[i] .
        end.
        ind_top = 0.
        do i = 1 to 10:
     w_fatura.vat = w_fatura.vat + vat_tut[i].
     ind_top      = ind_top      + input frame ext_frm indekr[i] .
        end.

        update w_fatura.komisyon with frame ext_frm .

        w_fatura.toplam = w_fatura.toplam + w_fatura.komisyon.
        komm = firma.komisyon.
        disp w_fatura.toplam with frame ext_frm.
        w_fatura.ind2_t = ind_top.
        disp  w_fatura.ind2_t with frame ext_frm.
        disp (w_fatura.toplam - w_fatura.ind2_t + w_fatura.vat) @ w_fatura.odenen
        with frame ext_frm.
        cnt = 1.
        do i = 1 to 10:
     if input frame ext_frm w_fatura.kod[i] <> "" then do:
       find diskod where
       diskod.orkod = input frame ext_frm w_fatura.kod[i]
```

```
 no-lock no-error.
  if available diskod then  do:
    assign w_fatura.kod[cnt] = input frame ext_frm w_fatura.kod[i]
    w_fatura.muh-kod[cnt]  = firsabit.distali_kod
    w_fatura.falt_kod[cnt] = firsabit.disalt_kod
    w_fatura.fana_kod[cnt] = firsabit.disana_kod
    h_tanim[cnt] = input frame ext_frm h_tanim[i]
   w_fatura.tutar[cnt] = input frame ext_frm w_fatura.tutar[i].
    w_fatura.itotal[cnt] = input frame ext_frm indekr[i]  .
  end.
  find extkod where
  extkod.ext_kod = input frame ext_frm w_fatura.kod[i]
  no-lock no-error.
  if available extkod then  do:
    assign w_fatura.kod[cnt] = input frame ext_frm w_fatura.kod[i]
    w_fatura.muh-kod[cnt] = extkod.muh-kod
    w_fatura.falt_kod[cnt] = extkod.alt_kod
    h_tanim[cnt] = input frame ext_frm h_tanim[i]
   w_fatura.tutar[cnt] = input frame ext_frm w_fatura.tutar[i].
    w_fatura.itotal[cnt] = input frame ext_frm indekr[i]  .
    if firma.f-yerli then
w_fatura.fana_kod[cnt]  = "600".
    else w_fatura.fana_kod[cnt] = "601".
  end.
   find reqfiyat
   where reqfiyat.reqkod = input frame ext_frm w_fatura.kod[i]
   no-lock no-error.
   if available reqfiyat then do:
     assign w_fatura.kod[cnt]    = input frame ext_frm w_fatura.kod[i]
     w_fatura.muh-kod[cnt] = reqfiyat.muh-kod
     w_fatura.falt_kod[cnt] = reqfiyat.alt_kod
     h_tanim[cnt]         = input frame ext_frm h_tanim[i]
  w_fatura.tutar[cnt]   = input frame ext_frm w_fatura.tutar[i]
     w_fatura.itotal[cnt]  = input frame ext_frm indekr[i]  .
    if firma.f-yerli then
 w_fatura.fana_kod[cnt]  = "600".
    else w_fatura.fana_kod[cnt] = "601".
   end.
   find shizmet
   where shizmet.skod = input frame ext_frm w_fatura.kod[i]
   no-lock no-error.
   if available shizmet then do:
     assign w_fatura.kod[cnt]    = input frame ext_frm w_fatura.kod[i]
     w_fatura.muh-kod[cnt] = shizmet.muh-kod
     w_fatura.falt_kod[cnt] = shizmet.alt_kod
     h_tanim[cnt]         = input frame ext_frm h_tanim[i]
  w_fatura.tutar[cnt]   = input frame ext_frm w_fatura.tutar[i]
     w_fatura.itotal[cnt]  = input frame ext_frm indekr[i]  .
    if firma.f-yerli then
 w_fatura.fana_kod[cnt]  = "600".
    else w_fatura.fana_kod[cnt] = "601".
   end.
   cnt = cnt + 1.
/******* Eğer KDV Varsa  ***********/
    if input frame ext_frm vat_yuz[i] > 0 then do:
 find muhist where muhist.uh-istk = w_fatura.uh-istk no-lock no-error.

      find diskod where
      diskod.orkod = input frame ext_frm w_fatura.kod[i]
      no-lock no-error.
      if available diskod then do:
      assign w_fatura.kod[cnt]  = input frame ext_frm w_fatura.kod[i]
   w_fatura.muh-kod[cnt]  = muhist.vmuh-kod
   w_fatura.falt_kod[cnt] = muhist.valt_kod
   w_fatura.fana_kod[cnt] = muhist.vana_kod
   h_tanim[cnt] = "Vat" + string(input frame ext_frm vat_yuz[i])
```

```
  w_fatura.tutar[cnt] =
      (input frame ext_frm vat_yuz[i] / 100) *
      ((input frame ext_frm w_fatura.tutar[i]) -
      (input frame ext_frm indekr[i])).
      cnt = cnt + 1.
      end.
      find extkod where
      extkod.ext_kod = input frame ext_frm w_fatura.kod[i]
      no-lock no-error.
      if available extkod then do:
find muhist where muhist.uh-istk = w_fatura.uh-istk no-lock no-error.
      assign w_fatura.kod[cnt] = input frame ext_frm w_fatura.kod[i]
  w_fatura.muh-kod[cnt] = muhist.vmuh-kod
  w_fatura.falt_kod[cnt] = muhist.valt_kod
  w_fatura.fana_kod[cnt] = muhist.vana_kod
  h_tanim[cnt] = "Vat" + string(input frame ext_frm vat_yuz[i])
  w_fatura.tutar[cnt] =
      (input frame ext_frm vat_yuz[i] / 100) *
      (input frame ext_frm w_fatura.tutar[i]).
  kdv = input frame ext_frm vat_yuz[i].
      cnt = cnt + 1.
      end.

    end.
   /*****Eğer KDV varsa son ********/
 end.
end.
reff = 0.
repeat i = 1 to 10:
   if input frame ext_frm w_fatura.kod[i] <> "" then do:
     reff = reff + 1.
     leave .
end.
end.
if w_fatura.komisyon > 0 then reff = reff + 1.
      assign
      w_fatura.odenen.
      repeat i = 1 to 30:
  if w_fatura.kod[i] =  ""
   then leave.
   w_fatura.aciklama[i] = h_tanim[i].
      end.

pause 0.

find last fatura use-index fatno no-lock no-error.
if not available fatura then old_no = 0.
else old_no = fatura.fat_no.
      faturano = old_no + 1.
assign w_fatura.fat_no = faturano.
output to extw.dat.
   export w_fatura.
output close.
create fatura.
input from extw.dat.
   import fatura.
input close.
update son_tarih label "Son Ödeme Tarihi" help "Son ödeme tarihi verilsin mi?"
with side-labels overlay row 12 centered frame son.
/*** Kredili şirket ise mektup  ve muacederiyet tarihini hesapla***/
if son_tarih = true then do:
if firma.cash = false then do:
 if firma.adryer then assign fatura.fat_muc = fatura.mek_tar + 32.
 else assign fatura.fat_muc = fatura.mek_tar + 40.
 if weekday(fatura.fat_muc) = 1 then
   assign fatura.fat_muc = fatura.fat_muc + 1.
```

```
   else if weekday(fatura.fat_muc) = 7 then
        assign fatura.fat_muc = fatura.fat_muc + 2.
   repeat:
      find tatil where tatil.tarih = fatura.fat_muc no-lock no-error.
      if available tatil then do:
  fatura.fat_muc = fatura.fat_muc + 1.
  next.
      end.
     else leave.
   end.
  end.
  end.
  /*************************************************************************/

        run extcbulwl.p(fatura.fat_no,komm).

 if lastkey = keycode("F4") then undo basla, next basla.
 end.
```

```
/*******************GHCN den Kredili Faturalama******************/
/*********************wfatural.p*********************************/
{wfatura.i}/***Tanımlamaların Olduğu Dosya*******/
{wfatural.frm}
 FORM  fatura.aciklama AT 1 tut AT 40 WITH FRAME FATURA
 2 COLUMNS.
       create w_fatura.

       assign
       w_fatura.fkod    = "NORM"
       w_fatura.uh-istk = userid
       w_fatura.fat_tar = today
       indirim_flag     = true.
       update
     w_fatura.uh-kod  validate(can-find(firma
     where firma.uh-kod = input w_fatura.uh-kod),
     "Girilen firma mevcut degil")
     w_fatura.fkod auto-return
     indirim_flag auto-return
     w_fatura.tar1 auto-return w_fatura.tar2 auto-return
     w_fatura.fat_tar auto-return
     VOUC w_fatura.uh-istk validate(can-find(istasyon
     where istasyon.uh-istk = w_fatura.uh-istk),
     "Girilen istasyon mevcut degil.")
     w_fatura.uclu
     son_tarih help "Son ödeme tarihi verilsin mi? (Evet/Hayır)"
     WITH FRAME FATGIR.
    find firma where firma.uh-kod = w_fatura.uh-kod no-lock no-error.
    find firsabit where firsabit.uh-kod = w_fatura.uh-kod no-lock no-error.
    HIDE FRAME FATGIR.
 pp = 1. kl = 1. tvat = 0.

 assign w_fatura.komisyon = 0
 w_fatura.mek_tar = today.
 if son_tarih = true then do:
  if firma.adryer then assign w_fatura.fat_muc = w_fatura.mek_tar + 32.
  else assign w_fatura.fat_muc = w_fatura.mek_tar + 40.
  if weekday(w_fatura.fat_muc) = 1 then
     assign w_fatura.fat_muc = w_fatura.fat_muc + 1.
  else if weekday(w_fatura.fat_muc) = 7 then
     assign w_fatura.fat_muc = w_fatura.fat_muc + 2.
 end.
 repeat:
    find tatil where tatil.tarih = w_fatura.fat_muc no-lock no-error.
    if available tatil and son_tarih = true then do:
       w_fatura.fat_muc = w_fatura.fat_muc + 1.
       next.
    end.
    else leave.
 end.
  BOSTOP = 0. s_top = 0. xx = 1. k = 1. t = 0.

 for each shizmet where saciklama begins "traf":
  tno = integer(skod).
 end.
 for each shizmet where saciklama begins "ram":
  rno = integer(skod).
 end.
     {wfatural.i}
   /* gsonolarının workfile kayıt edilmesi*/
     create w_gsofat.
     assign w_gsofat.ghcno = gso.gsono.
 .....................................................................
```

```
/*****************************************************************************/
/*            Dışarıdan Sağlananların Hesaplatılması                         */
/*****************************************************************************/

  for each oreq where oreq.uh-kod    =  gso.uh-kod   and
       oreq.uh-tarih  =  gso.uh-tarih and
       oreq.uh-nol    =  gso.uh-nol   and
       oreq.uh-no2    =  gso.uh-no2   and
       oreq.uclu      =  gso.uclu     and
       oreq.uh-istk   =  gso.uh-istk no-lock :
    for  each diskod where diskod.orkod = oreq.orkod:
      okod[pp] = orkod.
      oT[pp] = oT[pp] + oreq.ortutar.
      vt[pp] = vt[pp] + oreq.orvat.
      if oreq.orkom then do:
   kt[pp] = kt[pp] + (oreq.ortutar + oreq.orvat) .
      end.

      tvat = tvat + oreq.orvat.
   B1:
     DO LL = 1 TO (K1 - 1):
       IF oKOD[k1] = oKOD[LL] THEN
   DO:
     oT[LL] = oT[LL] + oreq.ortutar.
     vt[ll] = vt[ll] + oreq.orvat.
     if oreq.orkom then
     kt[ll] = kt[ll] + oreq.ortutar .
     oKOD[K1] = "".
     oT[K1] = oT[K1] - oreq.ortutar.
     vt[k1] = vt[k1] - oreq.orvat.
     if oreq.orkom then
     kt[k1] = kt[k1] - oreq.ortutar .
     pp = pp - 1.
     LEAVE B1.
   END.
     END.
   pp = pp + 1. k1 = pp .
      END.
  end.
   IF gso.ucus_tipi = "f" THEN BOSTOP = BOSTOP + gso.stutar[tno].
    TOT = 0. EXT = 0. tut = 0.
    DO TT = 1 TO 15 :
    find request where request.uh-kod = w_fatura.uh-kod and
         request.reqkod = gso.rkod[tt] no-error.
    if available request then do:
    if not request.indirim then indx_top = indx_top + gso.rtoplam[tt].
   /* IF gso.REQFLAG[TT] THEN w_fatura.komisyon = komisyon + gso.K_TOPLAM[TT].*/

    IF gso.RKOD[TT] <> "" THEN
      DO:
       KOD[XX] = gso.RKOD[TT].
       T[XX]   = T[XX] + gso.RTOPLAM[TT].
   B1:
     DO LL = 1 TO (K - 1):
        IF KOD[K] = KOD[LL] THEN
   DO:
     T[LL] = T[LL] + gso.RTOPLAM[TT].
     KOD[K] = "".
     T[K] = T[K] - gso.RTOPLAM[TT].
     XX = XX - 1.
     LEAVE B1.
   END.
     END.
   XX = XX + 1. K = XX .
   END.
  end.
```

```
  END.
    do h = 1 to 6:
     s_top[h]  =  s_top[h] + gso.stutar[h] .
    end.
END.
    I = 1. kno = 1.
h = 0.
for each shizmet by skod:
 h = h + 1.
 if  s_top[h] > 0 then do:
  w_fatura.aciklama[i]  = saciklama.
  tut[i]        = s_top[h].
  w_fatura.kod[kno]        = shizmet.skod.
  w_fatura.muh-kod[kno]    = shizmet.muh-kod.
  if firma.f-yerli then
  w_fatura.fana_kod[kno]  = "600".
  else w_fatura.fana_kod[kno] = "601".
  w_fatura.falt_kod[kno]  = shizmet.alt_kod.
  w_fatura.tutar[kno] = s_top[h].
  /* tnot trafic indis göstergesi */
  /* rnot ramp   indis göstergesi */
  if integer(w_fatura.kod[kno]) = tno then tnot = kno.
  else if integer(w_fatura.kod[kno]) = rno then rnot = kno.
  else
w_fatura.itotal[kno] = firma.f-ind2 * w_fatura.tutar[kno] / 100.
  kno = kno + 1.
  tot = tot + tut[i].
  i   = i + 1.
     end.
    end.

 REPEAT LL = 1 TO (k + 2):
      FIND REQfiyat WHERE REQfiyat.REQKOD = KOD[LL] no-error.
      IF AVAILABLE reqfiyat and t[ll] > 0 THEN DO:
w_fatura.kod[kno]    = reqfiyat.reqkod.
w_fatura.muh-kod[kno]    = reqfiyat.muh-kod.
   if firma.f-yerli then
   w_fatura.fana_kod[kno]  = "600".
   else w_fatura.fana_kod[kno] = "601".
w_fatura.falt_kod[kno]  = reqfiyat.alt_kod.
w_fatura.tutar[kno] = t[ll].
   find request where request.uh-kod = w_fatura.uh-kod and
        request.reqkod = reqfiyat.reqkod no-lock no-error.
   if  input indirim_flag and request.indirim then
w_fatura.itotal[kno] =  (firma.f-ind2 / 100) * w_fatura.tutar[kno].

kno = kno + 1.
w_fatura.aciklama[I] = reqfiyat.REQTUR.
tut[I] = T[LL].
TOT = tut[I] + TOT.
I = I + 1.
      END.
 END.
 REPEAT LL = 1 TO (pp + 2):
      FIND diskod WHERE diskod.orkod = oKOD[LL] no-error.
      IF AVAILABLE diskod THEN DO:
 if ot[ll] > 0 then do:
 w_fatura.kod[kno]       = diskod.orkod.
 w_fatura.muh-kod[kno]   = firsabit.distali_kod.
 w_fatura.falt_kod[kno]  = firsabit.disalt_kod.
 w_fatura.fana_kod[kno]  = firsabit.disana_kod.
 w_fatura.tutar[kno]     = ot[ll].
 indx_top = indx_top + w_fatura.tutar[kno].
 kno = kno + 1.
 w_fatura.aciklama[I]   = diskod.daciklama.
 tut[I] = oT[LL].
```

```
 TOT     = tut[I] + TOT.
 I = I + 1.
      end.
if vt[11] > 0 then do:    /* Eğer KDV si varsa */
 w_fatura.kod[kno]      = diskod.orkod.
 find muhist where muhist.uh-istk = w_fatura.uh-istk no-lock no-error.
 w_fatura.muh-kod[kno]    = muhist.vmuh-kod.
 w_fatura.fana_kod[kno]   = muhist.vana_kod.
 w_fatura.falt_kod[kno]   = muhist.valt_kod.
 w_fatura.tutar[kno] = vt[11].
indx_top = indx_top + w_fatura.tutar[kno].
 kno = kno + 1.
 w_fatura.aciklama[i] = diskod.daciklama + " (VAT)".
 tut[i] = vt[11].
 tot    = tut[i] + tot.
 i = i + 1.
end.

if kt[11] > 0 then do:
 find muhist where muhist.uh-istk = w_fatura.uh-istk no-lock no-error.
   w_fatura.kod[kno]    = diskod.orkod.
   w_fatura.muh-kod[kno]    = muhist.kom-kod.
   if firma.f-yerli then
   w_fatura.fana_kod[kno]        = "600".
   else w_fatura.fana_kod[kno]   = "601".
   w_fatura.falt_kod[kno]   = muhist.kalt_kod.
   w_fatura.tutar[kno]   = kt[11] * firma.komisyon / 100.
   indx_top = indx_top +  w_fatura.tutar[kno].
   w_fatura.komisyon = kt[11] + w_fatura.komisyon.
   kno = kno + 1.
  /*w_fatura.aciklama[i] = diskod.daciklama + " (KOM.)".*/
   tut[i] = kt[11] * firma.komisyon / 100.
   tot    = tut[i] + tot.
   i = i + 1.
end.
      END.
 END.


     TOP = 0.
     DO LL = 1 TO I - 1:
      TOP = tut[LL] + TOP.
     END.
     w_fatura.komisyon = firma.komisyon * w_fatura.komisyon / 100.
     FIND FIRMA WHERE FIRMA.uh-kod = w_fatura.uh-kod no-lock NO-ERROR.
     IF AVAILABLE FIRMA and w_fatura.komisyon > 0 then
     DO:
      KOMCH = STRING(FIRMA.KOMISYON).
      w_fatura.ACIKLAMA[1] = "COM. (%" + KOMCH + ")".
      w_fatura.tutar[i] = w_fatura.komisyon.
      tut[I] = w_fatura.komisyon .
      I = I + 1.
     END.
     w_fatura.aciklama[I] = "T O T A L".
     tut[I] = TOP .
     xx = i.
     I = I + 1.

      TR = s_top[tno] - BOSTOP.
      RA = s_top[rno].

     if input indirim_flag = true then do:
     FIND FIRMA WHERE FIRMA.uh-kod =  w_fatura.uh-kod NO-ERROR.
  IF  w_fatura.uclu THEN
   DO:
     if tr > 0 then
```

```
    w_fatura.itotal[tnot] = firma.f-ind3 * (tr / 100).
    if ra > 0 then
    w_fatura.itotal[rnot] = firma.f-ind3 * (ra / 100).
    w_fatura.IND3_T =  (TR + RA) * (FIRMA.f-IND3 / 100).
    TR = TR - (TR * FIRMA.f-IND3 / 100).
    RA = RA - FIRMA.f-IND3 * RA / 100.
  END.

IF FIRMA.f-IND1 > 0 THEN DO:
    if tr > 0 then
    w_fatura.itotal[tnot] = w_fatura.itotal[tnot] +
       (firma.f-ind1 * tr / 100).
    w_fatura.IND1_T = firma.f-ind1 * tr / 100.
    tr = tr - w_fatura.ind1_t.
END.

      IF FIRMA.f-IND2 > 0 THEN DO:
    if tr > 0 then
    w_fatura.itotal[tnot] = w_fatura.itotal[tnot] +
       ((firma.f-ind2 / 100) * tr).
    if ra > 0 then
    w_fatura.itotal[rnot] = w_fatura.itotal[rnot] +
       (firma.f-ind2 * ra / 100).
 w_fatura.IND2_T = ( TOP - (w_fatura.ind1_t + w_fatura.ind3_t + indx_top
     + bostop)) *
     FIRMA.f-IND2 / 100.
      END.

   IND1ST = STRING(FIRMA.f-IND1).
   IND2ST = STRING(FIRMA.f-IND2).
   IND3ST = STRING(FIRMA.f-IND3).
   disc = w_fatura.ind1_t + w_fatura.ind2_t + w_fatura.ind3_t .
 end. /* indirim_flag */
if disc > 0 then do:
   w_fatura.aciklama[I] = "DISCOUNT".
   tut[I] = disc.
   DISC = tut[I].
   I = I + 1.
end.

      ASSIGN w_fatura.ODENEN = top - disc
      w_fatura.toplam = top.

run wcekranyaz.p.

form "Bu faturayı kayıt edip, yazıcıya göndermek istiyor musunuz?"
       anss with frame d_frm overlay no-label row 9 centered.
disp false @ anss with frame d_frm .
prompt-for anss with frame d_frm.
if input anss = false then undo,retry.
form "USD Fatura : " anss1 with frame d1_frm overlay no-label row 12 centered.
disp true @ anss1 with frame d1_frm.
prompt-for anss1 with frame d1_frm.

 find last fatura use-index fatno no-lock no-error.
 if not available fatura then old_no = 0.
 else old_no = fatura.fat_no.
       faturano = old_no + 1.
 assign w_fatura.fat_no = faturano.
 output to wfatura.dat.
    export w_fatura.
 output close.
 if input anss1 = true  then
 run wfatusdx.p.
 else run wfattlx.p.
```

```
for each w_gsofat:
    find gsofat where gsofat.ghcno = w_gsofat.ghcno no-error.
    if not available gsofat then create gsofat.
    assign gsofat.ghcno    = w_gsofat.ghcno
    gsofat.fat_num  = faturano.
end.
```

```
/************Firma sabit Bilgileri Girişi programı************/
/*******firgir4.p**********/
def variable sec as char extent 2 format "x(5)"
    initial ["EVET","HAYIR"].
def new shared frame firfrm.
def var yeni_flag as logical .
def new shared variable frec as recid.
form {firfrm3.f} with frame firfrm centered.
form " F4 :IPTAL    F8 :KAYIT SILME   F1 :IŞLEM KABUL F5 :AÇIKLAMALAR"
     WITH FRAME
     dip centered row 20 no-box overlay.
form firma.aciklama with title " AÇIKLAMALAR " no-labels centered
     row 8 frame ac_frm overlay.
view frame firfrm.
view frame dip.
pause 0.
status input "Bilgi giriniz ya da cikmak icin "
       + kblabel("end-error")
       + " tusuna basiniz".
basla:
repeat :
clear frame firfrm.
yeni_flag = false.
prompt-for firma.uh-kod with frame firfrm.
 find firma where firma.uh-kod = input uh-kod no-error.
   if available firma then display firma except aciklama  with frame firfrm.
   else do:
 message " Yeni Kayit".
 create firma.
 yeni_flag = true.
 assign firma.uh-kod.
   end.
   frec = recid(firma).
 update firma.firma_ad auto-return adres auto-return firma.ulke_kod
 f-ind1 auto-return
 f-ind2 auto-return f-ind3 auto-return
 firma.ozel auto-return firma.adryer auto-return
 kargo_fiy auto-return
 cash auto-return f-yerli auto-return komisyon auto-return
 anlasmali auto-return uclu auto-return
 go-on(f4 F1) with frame firfrm
 editing:
  readkey.
  if lastkey = keycode("F1")  then do:
     assign firma except aciklama.
    /* clear frame firfrm.*/
     leave .
  end.

  if frame-field = "cash" and (lastkey = keycode("c") or
  lastkey = keycode("k")) then do:
      apply lastkey.
      next-prompt firma.f-yerli with frame firfrm.
      next.
  end.
  if frame-field = "uclu" and (lastkey = keycode("e") or
  lastkey = keycode("h")) then do:
      apply lastkey.
      /*next-prompt with frame firfrm.*/
      next.
  end.

  if frame-field = "f-yerli" and (lastkey = keycode("e") or
```

```
 lastkey = keycode("h")) then do:
     apply lastkey.
     next-prompt firma.komisyon with frame firfrm.
     next.
 end.
if frame-field = "ulke_kod" and lastkey = keycode("enter") then do:
   find ulke where ulke.ulke_kod = input firma.ulke_kod no-lock no-error.
   if not available ulke then do:
      message "Bu ülke kodu girilmemiş!".
      next.
   end.
   else
      display ulke.ulke_ad with frame firfrm.
end.
 if frame-field = "anlasmali" and (lastkey = keycode("e") or
 lastkey = keycode("h")) then do:
     apply lastkey.
     next-prompt firma.uclu with frame firfrm.
     next.
 end.


 if lastkey = keycode("f5") then do:
    run aciklama.p.
    if lastkey = keycode("f4") then next.
    else if lastkey = keycode("F1") then do:
  assign firma.aciklama.
  next.
    end.

 end.
if frame-field = "uclu" then do:
      kabul:
      repeat:
message "F1 : Kabul   F4 : Iptal " .
readkey.
if lastkey = keycode("f1") or lastkey = keycode("F4")
or lastkey = keycode("f8")
then leave kabul.
else do:
 apply lastkey.
 next.
end.
    end.
    pause 0.
 end.

 if lastkey = keycode("F8") then do:
  display "Silmek istediginizden emin misiniz ? (E/H)" SKIP
 SEC[1] AT 10  SEC[2] AT 30 with frame
      sor_frm  centered row 13 overlay NO-LABEL.
      CHOOSE FIELD SEC WITH FRAME SOR_FRM.
      IF FRAME-VALUE = SEC[1] THEN DO:
  DELETE FIRMA.
  NEXT BASLA.
      END.
 end.
 apply lastkey.
end. /*end editing.*/
if lastkey = keycode("f4") then
   undo , retry basla.
find firsabit of firma no-error.
if not available firsabit then create firsabit.
assign firsabit.uh-kod = firma.uh-kod.
{firsabit.i firsabit firma}
```

```
 if yeni_flag then leave basla.
 next basla.
end. /* end repeat*/
if yeni_flag and lastkey = keycode("F1") then do:
 message "Fiyat skalası aktarılıyor.....".
 run ak3.p(firma.uh-kod).
 run akll3.p(firma.uh-kod).
 find request where request.uh-kod = firma.uh-kod and
      request.reqkod = "71" no-error.
 if available request then assign request.reqfiyat = firma.kargo_fiy.
 run firgir4.p.
end.
if lastkey = keycode("F1") then do:
 find request where request.uh-kod = firma.uh-kod and
      request.reqkod = "71" no-error.
 if available request then assign request.reqfiyat = firma.kargo_fiy.
 run firgir4.p.
end.
```

```
DEFINE VARIABLE menu AS CHARACTER EXTENT 15.
DO WHILE TRUE:
  HIDE ALL.
    DISPLAY
     SKIP(1)
     "    1. HAREKAT BİLGİ GİRİŞİ              " @ MENU[1] SKIP
     "    2. FIRMA-REQUEST BILGI GİRİŞİ        " @ MENU[2] SKIP
     "    3. FIRMA BİLGİLERİ                   " @ MENU[3] SKIP
     "    4. İSTASYON BİLGİ GİRİŞİ             " @ MENU[4] SKIP
     "    5. UÇAK BİLGİ GİRİŞI                 " @ MENU[5] SKIP
     "    6. STANDARD HİZMET FİYAT SKALASI     " @ MENU[6] SKIP
     "    7. REHBER ÜCRETLER                   " @ MENU[7] SKIP
     "    8. APRON  BİLGI GİRİŞİ               " @ MENU[8] SKIP
     "    9. DIŞARIDAN SAĞLANAN HİZMETLER      " @ MENU[9] SKIP
     "   10. FATURALAMA                        " @ MENU[10] SKIP
     "   11. HIZMET CETVELI                    " @ menu[11] skip
     "   12. MEKTUP                            " @ menu[12] skip
     "   13. TATIL GÜNLERİ GİRİŞİ              " @ MENU[13] SKIP
     "   14. RAPORLAMALAR                      " @ MENU[14] SKIP
     "   15. ÇIKIŞ                             " @ MENU[15] SKIP

      WITH FRAME choices NO-LABELS ROW 3.
    CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices
      TITLE "  GHCN-Faturalama Ana Menu  " CENTERED .
    HIDE FRAME choices.
    IF      FRAME-INDEX = 1 THEN RUN ygsogir7.p.
    ELSE IF FRAME-INDEX = 2 THEN RUN reqgir2.p.
    ELSE IF FRAME-INDEX = 3 THEN RUN firmamenu.p.
    ELSE IF FRAME-INDEX = 4 THEN RUN istgir.p.
    ELSE IF FRAME-INDEX = 5 THEN RUN ucakgir.p.
    ELSE IF FRAME-INDEX = 6 THEN RUN fgir1.p.
    ELSE IF FRAME-INDEX = 7 THEN RUN skala.p.
    ELSE IF FRAME-INDEX = 8 THEN RUN apreq.p.
    ELSE IF FRAME-INDEX = 9 THEN RUN  dismenu.p.
    ELSE IF FRAME-INDEX = 10 THEN RUN fatmenuw.p.
    ELSE IF FRAME-INDEX = 11 THEN RUN cetmenu.p.
    ELSE IF FRAME-INDEX = 12 THEN RUN mekmenu.p.
    ELSE IF FRAME-INDEX = 13 THEN RUN tatgir.p.
    else if frame-index = 14 then run istasyon.p.
    else if frame-index = 15 then quit.
END.
```

```
/***********Request Hizmetleri Bilgi Girişi******************/
/***************rgso5.p***************/
def new shared var n as integer format ">>".
def new shared var ret as integer format ">>".
def var i as integer.
def var bas as integer.
def shared var grec as recid.
def new shared var frec  as recid.
def var uflag as logical.
def var bul_flag as logical.
def var aflag as logical.
def new shared var radi as char format "x(16)" extent 30.
def new shared var rfiyat as decimal  extent 30.
def var x as int.
def var flag as logical.
def var disflag as logical.
def var dflag as logical.
def var k as int.
def var basflag as logical.
def var bas_flag as logical.

form n label "Sira" gso.rkod[n] label "Kod"
radi[n] label "Request Turu" gso.rbirim[n] label "Birim"
rfiyat[n] label "Fiyat" rtoplam[n] label "Toplam"
with frame rfrm 15 down centered  row 2 overlay
title "GSO-REQUEST BİLGİ GİRİŞİ".
view frame rfrm . pause 0.
find gso where recid(gso) = grec no-error.
/* disp grec label "rgso" with overlay.*/
n = 1. basflag = true. flag = true. bul_flag = false.
bas = 1.
k = 0  .
tl:
repeat transaction:
basla:
repeat:
     message "F4 : Çıkış   F1 : Kabul  F7: Apron Request  F10: Yardım  F5 : Dış.
S. H.".
 if basflag then do:
 clear frame rfrm.
 up frame-line(rfrm) - 1 with frame rfrm.
    do n = bas to bas + 14:
       if rkod[n] = "" then assign radi[n]       = ""
        gso.rbirim[n]  = 0
        rfiyat[n]  = 0
        gso.rtoplam[n] = 0.
       display gso.rkod[n]  with frame rfrm.
 find request where request.reqkod =  input gso.rkod[n]
       and   request.uh-kod = gso.uh-kod no-lock no-error .
 if available request then do:
 find reqfiyat where reqfiyat.reqkod = request.reqkod no-lock no-error.
 assign  radi[n]       = reqfiyat.reqtur
  rfiyat[n]     = request.reqfiyat.
 end.

       display n gso.rkod[n]  radi[n] gso.rbirim[n] rfiyat[n]
       rtoplam[n] with frame rfrm.
       if frame-line(rfrm) = frame-down(rfrm) then leave.
       down 1 with frame rfrm.
    end.
    up frame-line(rfrm) - 1 with frame rfrm.
    assign n = bas.
    basflag = false
```

```
 end.

if bul_flag then do:
[ref.i]
hide message no-pause.
n = ref.
for each apronl where apronl.uh-tarih  = gso.uh-tarih and
        apronl.uh-no1     = gso.uh-no1 and
        apronl.uh-no2     = gso.uh-no2 and
        apronl.uh-kod     = gso.uh-kod no-lock:
    flag = true.
 do k = 1 to 30:
    if apronl.reqkod = gso.rkod[k] then do:
       flag = false.
      gso.rbirim[k] = apronl.birim.
    end.
 end.
        if flag then do:
        assign
    gso.rkod[n] = apronl.reqkod
    gso.rbirim[n] = apronl.birim.
       n = n + 1.
        end.
 END.

 basflag  = true.
 bul_flag = false.
 next basla.
end.

update gso.rkod[n]
gso.rbirim[n] with frame rfrm
 editing :
  readkey.
  if frame-field = "rkod" then do:
      if lastkey = keycode("F10") then do:
       [ref.i]
       RUN fbrowsel.p  .
       basflag = true .
       next basla .
   end.

      if lastkey = keycode("enter") then do:
  find request where request.reqkod = input gso.rkod[n]
       and   request.uh-kod = gso.uh-kod no-lock no-error.
  if not available request then do:
     Message "Bu request tanımlı değil!".
     undo,retry.
  end.
  find reqfiyat where reqfiyat.reqkod = request.reqkod no-lock no-error.
  do k = 1 to 30:
     if frame-value =  gso.rkod[k] then do:
        if not k = n then do:
    display gso.rkod[n] with frame rfrm.
    message "Bu request daha evvel girilmiş!".
    pause 2.
    /* n = k. */
    pause 0.
    next basla.
        end.
     end.
  end.


  assign gso.rkod[n] gso.rbirim[n] radi[n] rfiyat[n] rtoplam[n].
  dflag = true.
```

```
      DISPLAY request.reqkod @ gso.rkod[n] reqfiyat.reqtur @  radi[n]
      WITH FRAME rfrm.
    end.
  end.
  if lastkey = keycode("F5") then do:
    run ogsol.p.
    next.
  end.
  if lastkey  = keycode("F7") then do:
    bul_flag = true.
    message "F4 : Çıkış   F1 : Kabul  F7: Apron Request  F10: Yardım  F5 : Dış.
S. H.".
    pause 0.
    next basla.
  end.

  if lastkey = keycode("cursor-down") and input gso.rkod[n] <> "" then do:
    dflag = true.
    assign gso.rkod[n] gso.rbirim[n] radi[n] rfiyat[n] rtoplam[n].
    leave.
  end.

  if lastkey = keycode("cursor-up") then do:
    uflag = true.
    assign gso.rkod[n] gso.rbirim[n] radi[n] rfiyat[n] rtoplam[n].
    leave.
  end.

  if frame-field = "rbirim" and lastkey = keycode("enter") then
    do:
      find request where request.reqkod = input gso.rkod[n] and
    request.uh-kod = gso.uh-kod no-lock no-error.
      if available request then do:
      display request.reqfiyat @ rfiyat[n] with frame rfrm.
      find reqfiyat where reqfiyat.reqkod = request.reqkod no-lock no-error.
      end.

      if request.reqkod = "11" or request.reqkod = "12"
      then do: /* transport hizmeti için hesaplama */
   if input gso.rbirim[n] = 1 then
   gso.rtoplam[n] = 25 * (request.reqfiyat / reqfiyat.reqfiyat).
   else if input gso.rbirim[n] = 2 then
   gso.rtoplam[n] = 40 * (request.reqfiyat / reqfiyat.reqfiyat).
   else if input gso.rbirim[n] = 3 then
   gso.rtoplam[n] = 50 * (request.reqfiyat / reqfiyat.reqfiyat).
   else if input gso.rbirim[n] = 4 then
   gso.rtoplam[n] = 70 * (request.reqfiyat / reqfiyat.reqfiyat).
   else if input gso.rbirim[n] > 4 then
   gso.rtoplam[n] = (70 + (10 * (input gso.rbirim[n] - 4))) *
   (request.reqfiyat / reqfiyat.reqfiyat).
       end.
      else
       assign  gso.rtoplam[n] = input rfiyat[n] * input gso.rbirim[n] /
       (request.reqsure).
       display gso.rtoplam[n] with frame rfrm.
       assign  gso.rkod[n] gso.rbirim[n] radi[n] rfiyat[n] rtoplam[n].
     end.

   if lastkey = keycode("F4") then do:
    hide message.
    undo , leave.

   end.

   if lastkey = keycode("F1") then do:
     hide message
```

```
    return.
 end.

 apply lastkey.
end. /* end editing */

if uflag then do:
  if n > 1 then do:
     assign n = n - 1.
    if frame-line(rfrm) > 1 then up 1 with frame rfrm.
    else
    do:
     scroll down with frame rfrm.
     display n gso.rkod[n] radi[n] gso.rbirim[n] rfiyat[n] rtoplam[n]
     with frame rfrm.
    end.
 end.
    uflag = false.
    next.
end.

if dflag and input gso.rkod[n] <> "" then do:
    assign n = n + 1.
    if frame-line(rfrm) < frame-down(rfrm) then do:
down 1 with frame rfrm.
display n gso.rkod[n] radi[n] gso.rbirim[n] rfiyat[n] rtoplam[n]
with frame rfrm.
    end.
    else
    do:
     scroll up with frame rfrm.
     display n gso.rkod[n] radi[n] gso.rbirim[n] rfiyat[n] rtoplam[n]
     with frame rfrm.
     bas = bas + 1 . bas_flag = true.
     dflag = false.
    end.
 end.
 if lastkey = keycode("F4") then  leave basla.
 next basla .
end.
if lastkey = keycode("F4") then undo tl, leave tl.
/* end repeat,basla*/
end. /* end transaction tl */
```

```
/***********Fiyat Skalası Bilgi Girişi Programı***********/
/***************fgirl.p***************/
def variable sec as char extent 2 format "x(5)"
    initial ["EVET","HAYIR"].
def var down-flag as logical initial false.
def var up-flag as logical initial false.
def var oflag as logical initial false.
form
      pozisyon.poz label "Kod" pozisyon.aciklama with 12 down row 5 column 50
      frame poz_frm title " Pozisyon Kodları ".
form SKIP(1)
"Skala Kodu: " at 4 fiyat.uh-kod fiyat.poz  fiyat.ton_kod skip
"-------------------------------------"
     SKIP(8)
     with frame fiyfrm  row 5  column 10 overlay no-label title
     "      FİYAT BİLGİ GİRİŞİ      ".
form shizmet.saciklama ": "  fiyat.tutar with 8 down with frame dfiyfrm
     overlay no-label NO-BOX ROW 9 column 12.

form " F4 : IPTAL    F8 : SİLME  F1 : KABUL" WITH FRAME
     dip centered row 20 no-box overlay.
view frame dip.    PAUSE 0.
pause 0.
for each pozisyon no-lock:
    disp pozisyon with frame poz_frm .
    down 1 with frame poz_frm.
end.
status input "Bilgi giriniz ya da cikmak icin "
       + kblabel("end-error")
       + " tusuna basiniz".
t1:
repeat transaction:
repeat :
 clear frame fiyfrm.
 clear frame dfiyfrm.
 prompt-for fiyat.uh-kod validate(can-find(firma where
         firma.uh-kod = input uh-kod),
         "Geçersiz firma kodu -- Lütfen yeniden giriniz..")
     auto-return
 fiyat.poz validate(can-find(pozisyon where
     pozisyon.poz = input poz),
     "Gecersiz pozisyon kodu -- Lutfen yeniden giriniz...")
    auto-return
 fiyat.ton_kod validate(fiyat.ton_kod <> "","Bos gecemezsiniz") auto-return
 with frame fiyfrm
 editing:
 readkey.
 if lastkey = keycode("f4") then undo,return.
 apply lastkey.
 end.
 for each shizmet no-lock:
 find fiyat where fiyat.uh-kod   = input fiyat.uh-kod  and
    fiyat.poz      = input fiyat.poz     and
    fiyat.ton_kod  = input fiyat.ton_kod and
    fiyat.skod     = shizmet.skod  no-error.
 if available fiyat then do:
    display shizmet.saciklama fiyat.tutar with frame dfiyfrm.
    down 1 with frame dfiyfrm.
 end.
 else do:
      create fiyat.
  assign   fiyat.uh-kod   = input fiyat.uh-kod
    fiyat.poz      = input fiyat.poz
```

```
   fiyat.ton_kod  = input fiyat.ton_kod
   fiyat.skod     = shizmet.skod.
     find tarife where tarife.ton_kod  = input fiyat.ton_kod and
         tarife.skod     = shizmet.skod no-lock no-error.
     if available tarife then do:
      if input fiyat.poz begins "0"  then assign fiyat.tutar  = tarife.ictut.
      else assign fiyat.tutar = tarife.distut.
     end.
   display shizmet.saciklama fiyat.tutar with frame dfiyfrm.
   down 1 with frame dfiyfrm.
end.
end.
up 6  with frame dfiyfrm.
find first shizmet .
find first fiyat where fiyat.uh-kod   = input fiyat.uh-kod and
   fiyat.poz       = input fiyat.poz and
   fiyat.ton_kod  = input fiyat.ton_kod and
   fiyat.skod     = shizmet.skod no-error.
display fiyat.tutar with frame dfiyfrm.
basla:
repeat:
update fiyat.tutar go-on(f4) with frame dfiyfrm
editing:
readkey.
if lastkey = keycode("cursor-down") then do:
   down-flag = true.
   leave.
end.
if lastkey = keycode("cursor-up") then do:
   up-flag = true.
   leave.
end.
if lastkey = keycode("enter") then do:
   down-flag = true.
   leave.
end.

/*if lastkey = keycode("f4") or lastkey = keycode("esc") then do:
 oflag = true.
 leave .
end.*/

if lastkey = keycode("f1") then do:
  oflag = true.
  apply keycode("f1").
  leave .
end.

   if lastkey = keycode("F8") then do:
    display "Silmek istediginizden emin misiniz ? (E/H)" SKIP
   SEC[1] AT 10  SEC[2] AT 30 with frame
        sor_frm  centered row 13 overlay NO-LABEL.
        CHOOSE FIELD SEC WITH FRAME SOR_FRM.
        IF FRAME-VALUE = SEC[1] THEN DO:
  for each fiyat  where fiyat.uh-kod   = input fiyat.uh-kod and
           fiyat.poz       = input fiyat.poz and
           fiyat.ton_kod  = input fiyat.ton_kod :

    DELETE fiyat.
  end.
    hide frame sor_frm.
    leave BASLA.
        END.
        hide frame sor_frm.
   end.
apply lastkey.
```

```
end. /* end editing*/
if down-flag then do:
   if frame-line(dfiyfrm) < 6 then do:
    down 1 with frame dfiyfrm.

find next fiyat where fiyat.uh-kod   = input frame fiyfrm fiyat.uh-kod and
   fiyat.poz      = input frame fiyfrm fiyat.poz and
   fiyat.ton_kod  = input frame fiyfrm fiyat.ton_kod no-error.
    display fiyat.tutar  with frame  dfiyfrm.
   end.
  down-flag = false.
end.
if oflag then do:
 up frame-line(dfiyfrm) - 1 with frame  dfiyfrm.
 oflag = false.
 leave basla.
end.

if up-flag then do:
   if frame-line(dfiyfrm) > 1 then do:
   up 1 with frame dfiyfrm.
find prev fiyat where fiyat.uh-kod   = input frame fiyfrm fiyat.uh-kod and
   fiyat.poz      = input frame fiyfrm fiyat.poz and
   fiyat.ton_kod  = input frame fiyfrm fiyat.ton_kod no-error.
    display fiyat.tutar  with frame  dfiyfrm.

   end.
   up-flag = false.
end.
if lastkey = keycode("f4") then do:
 up frame-line(dfiyfrm) - 1 with frame  dfiyfrm.
 oflag = false.
 leave basla.
end.
next basla.
end.  /*end basla*/
if lastkey = keycode("f1") then next t1.
if lastkey = keycode("f4") then undo t1,next t1.
end.
end.
```

```
DEFINE VARIABLE menu AS CHARACTER EXTENT 6.
DO WHILE TRUE:
  HIDE ALL.
    DISPLAY
     SKIP(1)
     "    1. Firma Bilgi Girişi           " @ MENU[1] SKIP
     "    2. Özel Kod Bilgi Girişi        " @ MENU[2] SKIP
     "    3. Firma-Adres Bilgi Girişi     " @ MENU[3] SKIP
     "    4. Ülke Bilgi Girişi            " @ MENU[4] SKIP
     "    5. Muhasebe-Firma Bilgi Girişi " @ menu[5] skip
     "    6. Çıkış                        " @ MENU[6] SKIP

      WITH FRAME choices NO-LABELS ROW 5.
    CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices
      TITLE "  FİRMA BİLGİLERİ  " CENTERED .
    HIDE FRAME choices.
    IF      FRAME-INDEX = 1 THEN RUN firgir4.p.
    ELSE IF FRAME-INDEX = 2 THEN RUN stgir.p.
    ELSE IF FRAME-INDEX = 3 THEN RUN adres.p.
    else if frame-index = 4 then run ulke-gir.p.
    else if frame-index = 5 then run firsabgir.p.
    else if frame-index = 6 then return.
END.
```

HAVAŞ

AIRPORTS GROUND HANDLING CO.

GROUND HANDLING CHARGE NOTE

NO: A 109533

STATION:	TYPE OF A/C:	REG.MARKS:	FLIGHT NO.:
IST	B737	tcsun	BA 671 /672

ARRIVAL: SCHEDULE : DATE LOCAL TIME : 12/10/93	DEPARTURE: SCHEDULE : DATE LOCAL TIME : 12/10/93
ESB ACTUAL : 12.30/12.45	ADB ACTUAL : 22.15/22.10
Aircraft Owner: BEA	Charges payable by :
	Form of payment : KREDİ
	Full address :
Nationality :	MTOW : 63

SERVICES RENDERED		CHARGES $	CHARGES TL.
1. GROUND HANDLING			
a) Representation	XXX	60.00	
b) Flight Operation			
c) Aircraft Line Maintenance			
d) Passenger Traffic	XXX	600.00	
e) Ramp			
f) Cargo	XXX	90.00	
g) Catering - Uplift			
h) Surcharge %	XXX		
TOTAL 1.		750.00	
2. OTHER CHARGES (Services Upon Request)			
a) GPU 28VDC/2000 A		50.00	
b) P/B OVER 107 TONS		250.00	
c) TOWBAR (HAVAS)		100.00	
d)			
e)			
f)			
g)			
TOTAL 2.		400.00	
3. AIRPORT CHARGES			
TOTAL 3.			
GRAND TOTAL:		1,150.00	

I here by certify, as authorized representative of the aircraft owner, that I recognize and accept the liability conditions of the handling company (appearing overleaf) for the performance of the facilities required on behalf of the owner of the aircraft.

Signature for acceptance of conditions:
Name in block letters:

Signature for receipt of facilities as indicated above:
Name in block letters:

All charges shown on this form are subject to approval by the Handling Company. Over - collections Will be refunded, under - collections Will be invoiced

The above facilities have been rendered

Signature of Station Manager:
Name in block letters:

DISTRIBUTION :	COLOUR	Green	Brown	Blue	Black
	TO	Captain / Airline	Representative	Accounting	Station

7255

HAVAŞ

HAVAALANLARI YER HİZMETLERİ A.Ş.
AIRPORTS GROUND HANDLING CO.
ATATÜRK HAVALİMANI C TERMİNALİ
YEŞİLKÖY - İSTANBUL
BAKIRKÖY VERGİ DAİRESİ : 4599870010

FATURA / INVOICE : 059591
SERİ NO. / SERIAL NUMBER : 31/05/94
TARİH / DATE : MER/000966
SIRA NO. / RECORD NO. :
TİCARET SİCİL NO. : 231818 - 179384

SUN EXPRESS

GÜNEŞ EXPRESS HAVACILIK A.Ş.
FENER MAHALLESİ SİNANOĞLU CAD.
OKTAY APT.P.O.BOX 28 ANTALYA

Son Ödeme Tarihi:
Due To : 12/07/94
Telephone Number: 6635509

($)

AÇIKLAMA / DESCRIPTION	FİYAT / PRICE	TUTAR / AMOUNT
The Sum incurred for the handling Services rendered to your flights by our İSTANBUL station at ATATÜRK Airport for the month MAY 1994		44,628.35
REPRESENTATION.....	2,155.00	
TRAFFIC.....	21,550.00	
RAMP.....	43,100.00	
TRANSPORT DEP.BY.BUS.....	675.00	
TRANSPORT MINIBUS.....	400.00	
GPU 115/200 V 90 KVA.....	600.00	
P/B UPTO 107 TONS.....	3,200.00	
MINIBUS.....	20.00	
TRANSPORT ARR.BY.BUS.....	800.00	
TOWING UPTO 107 TONS.....	375.00	
ACOMP CUSTOMER.....	547.19	
CATERING(USAŞ).....	64.04	
THY TEC.SERVICE.....	850.00	
THY TEC.SERVICE (VAT).....	22.50	
COM. (%10).....	148.37	
T O T A L	74,507.10	
DISCOUNT	29,878.75	
	44,628.35	
FORTY FOUR THOUSAND SIX HUNDERED TWENTY EIGHT US $ AND % 35 ONLY.		
OVERALL %41	29,878.75	
D I S C O U N T	29,878.75	
YEKÜN / TOTAL :	44,628.35	

Vergi usul kanunu yönetmelik hükümlerine tabi değildir.
Ödeme yaparken lütfen fatura numarasını belirtiniz. / In making settlement please refer to the invoice number.

İş bu fatura mürderecatına yedi gün içinde itirazda bulunulmadığı takdirde tutarı kabul edilmiş sayılır.
If there is no objection to this invoice within seven days from its receipt, the amount of this invoice will be considered acce

Parayı şu adrese yatırınız:
Deposit to : Türkiye İş Bankası Atatürk Havalimanı Şb. Account Number 1108 (USD)
T. Vakıflar Bankası Atatürk Havalimanı Şb. Account Number 400037 (USD)

ÖZGEÇMİŞ:

10.07.1969. tarihinde İstanbul'da doğmuştur. 1986 yılında İ.T.Ü. Fen Edebiyat Fakültesi Matametik Mühendisliği Bölümü'ne girmiştir. Aynı Bölümden 1991 yılında mezun olmuştur. Yine 1991 yılında İ.T.Ü. Fen Bilimleri Enstitüsü , Mühendislik Bilimleri Anabilim Dalı Sistem Analizi Programında Yüksek lisans Eğitimi'ne başlamıştır.

1992 yılından beri Havaş (Hava Alanları Yer Hizmetleri Anonim Şirketi) 'da yazılım uzmanı olarak görev yapmaktadır. Truug (Türkiye Unix Kullanıcıları Derneği) üyesi bulunmaktadır.