

YILDIZ TEKNİK ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

Birden Fazla Sita Üze, Ort, Çal,
Data, Man, Sıst, Tas,

YÜKSEK LİSANS TEZİ

Rahim Tezcan

1990

Yıldız Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Bilimleri Mühendisliği

Birden Fazla Site Üzerinde
Ortaklaşa Çalışabilecek
Database Management
Sisteminin Tasarımı

Yüksek Lisans Bitirme Tezi

Yöneten
Prof. M. Yahya Karşılıgil

Hazırlayan
Müh. Rahmi Tezcan
871806

Haziran 1990

YILDIZ TEKNİK ÜNİVERSİTESİ
KÜTÜPHANE DOKÜMANTASYON
DAİRE BAŞKANLIĞI

Kot	R 36856.....
Alındığı Yer	..Fan.Bil.,Tos,.....
Tarih	:13.1.95.....
Fatura	:.....
Fiyatı	:32.222.7.....
Ayniyat No	:1-1.....
Kayıt No	:51034.....
UDC	:.....
Ek	:.....

Bilgi sayı

BİRDEN FAZLA SITE ÜZERİNDE ORTAKLAŞA ÇALIŞABİLECEK
DATA BASE MANAGEMENT SİSTEMİ'NİN TASARIMI

30 Nisan 1990



Rahmi Tezcan

AMAC

Bu tezin amacı, bir network aracılığıyla birbirine bağlı IBM mainframe'lerde multisite çalışabilen bir data base yönetim sisteminin tasarlanması ve gerçekleştirilmesidir.

ÖZET

Modern computer system'lerinde data base'lerin giderek güçlendiği, ve buna bağlı olarak data base'lerden beklentilerin de arttığı bir gerçektir. Özellikle bilgi önem kazandıkça, computer'lerin performans ve kapasiteleri arttıkça data baselerin de hızlı gelişen bir süreç içine girmeleri kaçınılmaz olmuştur.

Enformatik ortamda - ki bundan haberleşme, endüstri, eğitim, ulaşım, vs ortamları kastedilmektedir- bilgisayar çoğunlukla data manipülasyonu, ve erişimi amacı içinde kullanılır. Esasında endüstri ortamında her ne kadar bilgisayar kontrol amacıyla da kullanılabilse de bu noktada da data erişimi ve işlenmesinin önemi büyüktür. Ancak endüstriyel data genel olarak diğer enformatik ortamlardakinden daha az çeşitli ve miktar olarak da azdır.

Data miktarı arttıkça, datanın işlenmesi, erişimi, transfer edilmesi de güçleşmekte ve bu konuda pekçok problem ortaya çıkmaktadır. Büyük bilgisayar sistemlerin her ne kadar büyük denilebilecek miktarda datanın işlenmesi mümkün olabiliyor ise de, yarattığı başka dezavantajlar, -örneğin Recovery-, nedeniyle bizi başka arayışlar içine sürüklemektedir.

Data'ya doğru ve zamanında erişim çok önemli olduğundan çok büyük bir miktar datayı aynı sistem içinde barındırmak, hiç de iyi bir strateji değildir. İşte burada dağıtılmış data base'ler kavramı ortaya çıkmıştır. Bu tezin ana fikri bu konudur.

ABSTRACT

The Data Base Systems has been rapidly growing up and getting strong in present computer systems, just because their responsibilities are being increased day by day. The importance of Informatics, by means of requirement of accessing the various information in many systems, and its requirements from the Data Bases, have been forcing those systems to be more effective, faster and cheaper.

Here, one way of performing the multiple and remote access of information from multiple sites, will be explained and one of the technics called Distributed Data Base Managements Systems, will be dealt in. Then, a small system will be designed and coded in the final chapter, using one of discussed methods.

In the Informatic environment, ie. industry, telecommunication, education, transportation etc., the computer are being used to access and manipulate large amount of information. The industry related data needs those facilities as well as its usage in control of mechanical systems. However, in usually the industry related data is smaller than the other enformatic environments in size.

As increasing of amount of information, the accessing, manipulating, transferring data between systems are being more difficult and due to these, several tens of problems are being encountered. Even it could be possible to serve those facilities by using the single stronger computer systems, some other additional problems, ie. recovery, concurrency, etc.. are pushing the system designers to find out alternative solutions.

Due to importance of correct and fast access of information on time, it is not a best solution to keep too much information in a single computer. Following of this idea, has figured out the concept of distributed data base. This is the focus of this text.

RSAVE

CONTENT

RLOAD

RCREAT

RADD

RREAD

RCMS

RSET

RHELP

RINIT

PRELOAD

İÇİNDEKİLER

KABULLER	1
COMPUTER SYSTEM ÖZELLİKLERİ	1
DBMS ÖZELLİKLERİ	1
DBMS'e GENEL BİR BAKIŞ	3
RELATIONAL DATA BASE KAVRAMI	3
INTERACTIVITY	3
RECOVERY	3
Sistem Problemleri	4
CONCURRENCY	4
INTEGRITY	5
SECURITY	5
DDBMS MİMARİSİ	7
DDBMS DESTEK YAZILIMI	11
SERVİS MAKİNESİ TEMEL YAZILIMI	11
Servis Makinesi Konfigürasyonu	12
KULLANICI-DDMBS ARAYUZU	12
XA İŞLEME	13
Xa Formatlama	13
Xa üretimi	14
Xa Yorumlama	15
Xa Uyuşumu	15
XA YAYILIMI	16
Orjinal Xa	16
DDBMS TASARIMI	17
PROCEDURE'LER	17
DEFFILE	17
READDEF	17
ALLHEAD	18
ALLBOT	18
WRDELIM	18
READSEC	18
WRUNDLN	18
RADDXA	19
RDEL	19
RCHA	19
RLIST	19
RSAVE	19
CONTENT	20
RLOAD	20
RCREAT	20
RADD	20
RREAD	21
RCMS	21
RSET	21
RHELP	21
RINIT	22
PRELOAD	22

RDROP	22
FUNCTION'LAR	22
UPPER	22
EK1 TDB	23
EK2 EXEC'LER	65
HUNTREQ	65
POPTDB EXEC	66
TDBJOBS EXEC	68
TDB EXEC	70
KAYNAKLAR	71

CPU noktaları arasında güvenilir bir data link, olmalıdır. Data base'in orta ve ortanın üzerinde bir data akışı yaratacağı düşünülerek bu linkin sürekli hizmette (dedicated) olması gerekmektedir. Datalink yönetimi IBM tarafından desteklenen RSCS software'i ile sağlanmaktadır.

Sistemde tape back-up olanağı olmalıdır. Özellikle diğer kullanımların yanı sıra, DBMS in gereksinimleri için sistematik bir tape back-up olanağı sağlanmalıdır. Örnek olarak yine IBM tarafından desteklenen TAPSYS software'i verilebilir.

DBMS ÖZELLİKLERİ

Data Base Management Sistem, CMS üzerinde çalışabilir, olacak yeni VM için hazırlanacaktır.

Data Base, bir mainframe üzerinde çalışacağı için, doğal olarak çok kullanıcı olmalıdır.

Data base management sistemi macro kullanımına elverişlidir. DBMS'in interactive komutları ile macro yazılıp, data base ortamında çalıştırılabilir.

DBMS Çok kullanıcı olmanın beraberinde getirdiği gizlilik kurallarını desteklemelidir. Belirli bilgilerin belirli kullanıcılar tarafından görülmesi ya da görülmemesi, belirli komutların yine belirli ve yetkili kullanıcılar (admin) tarafından kullanılabilmesi mümkün olmalıdır.

DBMS relational'dır. Data field-item (column,row) yapısında olup tam relational biçimde erişilebilir.

KABULLER

COMPUTER SYSTEM ÖZELLİKLERİ

Computer system, online ve çok kullanıcılıdır. IBM 43xx 30xx mainframe örnek olarak verilebilir. Sistem üzerinde, VM/SP'yi destekleyecek gerekli hardware bulunmaktadır.

Sistem üzerinde CP/CMS kullanılabilir haldedir ve CMS versionu 3 veya 3 den büyüktür.

CPU noktaları arasında güvenilir bir data link, olmalıdır. Data base'in orta ve ortanın üzerinde bir data akışı yaratacağı düşünülerek bu linkin sürekli hizmette (dedicated) olması gerekmektedir. Datalink yönetimi IBM tarafından desteklenen RSCS software'i ile sağlanmaktadır.

Sistemde tape back-up olanağı olmalıdır. Özellikle diğer kullanımların yanı sıra, DDBMS in gereksinimleri için sistematik bir tape back-up olanağı sağlanmalıdır. Örnek olarak yine IBM tarafından desteklenen TAPESYS software'i verilebilir.

DBMS ÖZELLİKLERİ

Data Base Management Sistem, CMS üzerinde koşabilir, olacak yani VM için hazırlanacaktır.

Data base, bir mainframe üzerinde koşacağı için, doğal olarak çok kullanıcı olmalıdır.

Data base management sistemi macro kullanımına elverişlidir. DBMS'in interactive komutları ile macro yazılıp, data base ortamında koşturulabilmelidir.

DBMS Çok kullanıcılı olmanın beraberinde getirdiği gizlilik kurallarını desteklemelidir. Belirli bilgilerin belirli kullanıcılar tarafından görülmesi ya da görülmemesi, belirli komutların yine belirli ve yetkili kullanıcılar (admin) tarafından kullanılabilmesi mümkün olmalıdır.

DBMS relational'dır. Data field-item (column,row) yapısında olup tam relational biçimde erişilebilir.

Integer, character ve string data yapısı desteklenmektedir.

Çıkışlar terminale ya da diske yönlendirilebilir.

RELATIONAL DATA BASE KAVRAMI

DBMS interactive'dir. Kullanıcı her bir entry'si için anında cevap alır. Batch çalışma söz konusu değildir.

Online HELP imkanı bulunmalıdır.

CMSSUBSET desteklenmeli, yani database içinde CMS Subset komutları çağrılabilir.

Relational Data Base kavramı geçtiğimiz 10 yıl içinde ortaya çıkmış ve oldukça yankı uyandırmıştır. Yine geçmişte relational data base sistemlerinin tamamı relational'dır. Bazı non-relational sistemler de relational data base sistemleri olarak çalışır. CMSSUBSET desteklenmeli, yani database içinde CMS Subset komutları çağrılabilir. Relational bir data base'de kullanıcı bütün her şeyi bir table olarak algılar, ve table dışında hiçbir şey yoktur. Gerçekten, OPERATIONAL DATA bile table olarak saklanır, ve değiştirilmesi adli datadan farkı olarak yapılır. Kullanıcı, table setleri yaratarak, bu setler üzerinde subsetler oluşturabilir. DBMS fonksiyonlarını bu subsetler üzerinde kullanabilir. Relational Data Base için en iyi örnek IBM'in SQL/DS'idir. Non-relational sistemler, Inverted List, Network, ya da Hierarchic yapıda olabilir.

INTERACTIVITY

Interactivity, sistemin kullanıcı ile daha iyi iletişim kurabilmesi için vazgeçilmez bir özelliktir. DBMS, kullanıcının terminalden girdiği her input için bir action yaratması ve bu kullanıcıya bir cevap vermesi gerekir. Bunun için kullanıcı öncelikle interactive bir komut yorumlayıcı ortamına girer. Interactive komutlar bu ortamda yorumlanır ve ilgili procedure'ler çağrılır. Hata belirleme, interactive sistemlerin bir parçasıdır ve kullanıcının sistemi daha verimli ve efektif kullanmasına olanak tanır. Ayrıca yapılan araştırmalar, interactive sistemlerin daha kısa sürede öğrenildiğini ve daha rahat kullanıldığını göstermektedir.

RECOVERY

Recovery, Data base sistemlerinde klasik ve önemli bir konudur. Önceden belirlenemeyen bazı problemlerin, sistemin çalışmasını kesikliğe uğratması şeklinde tanımlanabilir. Bu şekilde oluşan problemler, belirli aralıklarla alınan back-up'lerden yararlanılarak çözülür, ve sistem normal işlevine döndürülür.

DBMS'E GENEL BİR BAKIŞ

RELATIONAL DATA BASE KAVRAMI

Relational Data Base Kavramı geçtiğimiz 10 yıl içinde ortaya atılmış, ve oldukça yankı uyandırmıştır. Yine geçtiğimiz yıllarda yaratılan Data Base Sistemlerinin tamamı relational'dır. Eski, non-relational sistemlerin yaratıcıları da kendi sistemlerini relational hale getirmek için çaba harcamaktadırlar. Relational bir data base'de kullanıcı bütün her şeyi bir table olarak algılar, ve table dışında hiçbirsey yoktur. Gerçekten, OPERATIONAL DATA bile table olarak saklanır, ve değiştirilmesi adi datadan farksız olarak yapılır. Kullanıcı, table setleri yaratarak, bu setler üzerinde subsetler oluşturabilir. DBMS fonksiyonlarını bu subsetler üzerinde kullanabilir. Relational Data Base için en iyi örnek IBM'ın SQL/DS idir. Non-relational sistemler, Inverted List, Network, ya da Hierarchic yapıda olabilir.

INTERACTIVITY

Interactivity, sistemin kullanıcı ile daha iyi iletişim kurabilmesi için vazgeçilmez bir özelliktir. DBMS, kullanıcının terminalden girdiği her input için bir action yaratması ve bu kullanıcıya bir cevap vermesi gerekir. Bunun için kullanıcı öncelikle interactive bir komut yorumlayıcı ortamına alınır. Interactive komutlar bu ortamda yorumlanır ve ilgili procedure'ler çağrılır. Hata belirleme, interactive sistemlerin bir parçasıdır ve kullanıcının sistemi daha verimli ve efektif kullanmasına olanak tanır. Ayrıca yapılan araştırmalar, interactive sistemlerin daha kısa sürede öğrenildiğini ve daha rahat kullanıldığını göstermektedir.

RECOVERY

Recovery, Data base sistemlerinde klasik ve önemli bir konudur. Önceden belirlenemeyen bazı problemlerin, sistemin çalışmasını kesikliğe uğratması şeklinde tanımlanabilir. Bu şekilde oluşan problemler, belirli aralıklarla alınan back-up'lardan yararlanılarak çözülür, ve sistem normal işlerliğine döndürülür.

Sistem Problemleri

Enerji Kesintileri:

1. Enerji kesintileri (Power Failure) ya da uygunsuzlukları, sistemi zamansız bir biçimde kesikliğe uğratar ve sistem yeniden IPL edilir. Bu, bütün temporary yapının yeniden yaratılmasını gerektirir. Bu noktada, DBMS kesinti öncesi hangi noktada kaldığını belirlemek ve o noktadan devam etmek zorundadır.

Donanım Problemleri:

2. Donanımda zaman zaman fiziklssel sorunlar olabilir. Örneğin bir disk çeşitli nedenlerle bozularak okunamaz/yazılamaz hale gelebilir. Ya da elektriksel şoklar yüzünden bazı devre elemanları zarar görerek bir kısım üniteler devre dışı kalabilir. Bu konuda doğal olarak yetkili kişilerin teknik desteğine ihtiyaç duyulacaktır.

Yazılım Problemleri:

3. Yazılım, zaman zaman, kendi dinamik yapısından dolayı, beklenmedik ve önceden düşünülememiş hatalı sonuçlar uretebilir. Ya da insan mudahalesi gereken noktalarda yapılan dikkatsizlikler, benzer önemli sorunlara yolaçabilir. Bu durumda sistem kendi kendini recover edemez, yine back-up sistemi kullanılarak, sistem tekrar işler hale getirilir.

CONCURRENCY

Concurrency, çok kullanıcılı database sistemlerin en önemli konularından biridir. Herhangi bir kullanıcının, birden fazla başka kullanıcının o anda okumakta olduğu data'yı değiştirmesi bu konunun özünü oluşturur. Bir data değiştirilecekse, o datayı o anda herhangi bir başka kullanıcının okuyor olmaması gerekir. Bu problemin çözümünde en çok kullanılan yöntem, KİLİTLEME yöntemidir. Eğer bir kullanıcı bir datayı değiştirecekse, DBMS o datanın bulunduğu record'u kilitler. Kilitlenmiş bir record, değişiklik bitinceye kadar bir başka kullanıcı tarafından okunamaz. Ancak kilitleme, sistemin normal çalışma seyri içinde ek bir yük olarak karşımıza çıkar ve, DBMS her bir erişim isteği karşısında erişilecek datanın kilitli olup olmadığını kontrol etmek durumunda kalır. Ayrıca, kilitleme mekanizması beraberinde DEAD LOCK adı verilen bir problemi de

getirir, ki bu problemin çözümü de ilave monitor yazılımlar gerektirir.

İlerideki bölümlerde tasarımı yapılacak DDBMS için kilitleme mekanizması kullanılmayacak, ancak gizlilik kuralları içinde multiwrite özelliğın passwordler'le sağlanması mümkün olacaktır. Bununla beraber, multi-write özelliğı sağlandığı anda, concurrency'de problemlerin çıkması yine de olasıdır. Böyle bir durumda, ilgili transaction işlenirken uyumsuzluklar ortaya çıkar ve bu durum yetkili vmler tarafından manuel olarak düzeltilmek durumundadır.

INTEGRITY

Integrity, DBMS'in kullanıcı tarafından girilen birtakım komutların ve data'ların sistem topolojisine uygunluğunu kontrol etmek ve gerekirse düzeltmek, veya düzelttirmektir. Bazı durumlarda, kullanıcı hic syntax ve hatası yapmamış olabilir, ancak sözkonusu durumda verilere bağılı olarak, data type uyumsuzluğu, truncation (kırpılma), ya da benzer durumlar ortaya çıkabilir. Bu noktada integrity kuralları çerçevesinde, sistem gerekeni yapar, ve kullanıcı uyarılır. Örneğın, integer olarak önceden tanımlanmış bir field içine character data girilmeye çalışılırsa, sistem hata mesajı vererek, ilgili datayı 0 olarak değiştirir. Ya da, genişliğı 10 character olan bir field içine 15 character uzunluğunda bir string data eklenmeye çalışılırsa son 5 character kırpılır.

SECURITY

Security, hangi kullanıcının hangi data'yı, hangi şartlar altında görebileceğini, değiştirebileceğini saptayan kurallardır. Çok miktarda data'nın aynı ortamda saklanması, ve sistemin çok kullanıcılı olması, security gerekliliğini arttırmaktadır.

Security fonksiyonu kısaca aşağıdaki gibidir:

1. Sistem, kullanıcı tarafından yaratılan herbir table için bir security entry'si tutar ve bu entry daha sonra değiştirilemez. Kullanıcı isterse yarattığı table için WRITE ve READ password'ler verebilir. Eğer herhangi bir password verilmez ise, default olarak, herhangi bir vm tarafından okunabilen ancak sadece sahibi (owner) tarafından yazılabilen bir table oluşturulmuş olunur.

2. Sistem, vmid kontrolu de yapar. Bunun için DBMS ile CP/CMS interaction'u sözkonusudur. Bir table hangi vm kullanılarak yaratılmışsa, owner olarak o vm kabul edilir.

3. Security hiyerarşik bir yönetim mekanizması gibi, bazı kullanıcılara diğerlerini yönetme, hakkı verir. Bu tip kullanıcılar yönettikleri kullanıcıların bütün fonksiyonlarını denetleyebilirler.

4. Sistemde SERVVM olarak adlandırılan görüntü makinenin özel bir security üstünlüğü olmalıdır. Bu vm bütün data için okuma ve yazma hakkına password vermeksizin sahip olabilmelidir. SERVVM in tanımı TDB DEFDATA data file'ı içinde yapılır.

5. Sistemdeki bütün password'ler ve table owner'ların isimlerinin tutulduğu disk (DATA DISK) için sistemde mevcut olan herhangi bir security utility'si kullanılabilir.(Örnek. VMSECURE). Bu disk SERVVM update eder, ve diskin security'sinden sorumlu değildir.

Bilimsel çalışmalarda, araştırma ve geliştirme alanlarında doğal olarak daha çok miktarda data kullanılmakta ve gün geçtikçe artan bu miktar, sistemleri çok yüklemektedir. Büyük araştırma projeleri, çeşitli ülkelerde, birbirinden çok uzak noktalarda kurulan computer sistemleri kullanılarak yapılabilmektedir. Aynı anda, yüzlerce insan aynı amaç doğrultusunda, aynı bir ortak yapı içerisinde ancak bu şekilde çalışabilmektedir. Bu çalışmanın data trafiğini, güvenliğini ve gizliliğini ancak güçlü bir DBMS sağlayabilir.

Uzak dataya erişim için çeşitli yöntemler mevcuttur. Örneğin bankacılıkta, bir uzak terminal belirli bir hesap numarasındaki hesabı kontrol etmek için, ana bilgisayara telefon hattı üzerinden bağlanır ve sadece ilgili hesaba ait kayıt bilgisini alarak işlevini sürdürür. Bir kayıt ancak birkaç onbyte'lık bilgidir ve transferi fazla zaman almaz. Onun için terminalle bilgi saklanması da gerek yoktur.

Ancak, uzak bir data base içinde depolanan çok fazla sayıda bir data'ya örneğin büyük bir file'a, erişmek böylesine basit bir terminal ile imkansızdır. Ve kullanıcı sayısının artmasıyla terden çıkılmaz bir duruma gelinebilir. Bu durumda, bir mainframe içinden geçilerek, ve mainframe'in bilgi depolama mekanizmaları kullanılarak sorun çözülebilir. Bu noktada da iki ayrı yöntem üzerinde düşünmek mümkündür:

1. Aynı bir DBMS, herbir computerde kurulur ve local data base işleri hale getirilir. Bu data base'de

DDBMS MİMARİSİ

Dağıtılmış data base'ler, birbirlerinden uzak computer sistemleri üzerinde çalışan ve aralarında data akışı sağlayarak, aynı bir bilgiye farklı CPU noktalarından erişilmesine olanak veren sistemlerdir. Kavram olarak data base'lerin varolduğu günden bu yana sözü edilen, ancak, uzak data'ya hızlı ve yüksek sıklıkta erişim gereğinin artmasıyla adından daha çok bahsedilen bir sistem haline gelmiştir. Bazı yayınlarda 1970 li yılların sonlarına doğru Avrupa, Amerika ve bu konuyla ilgili bazı projeler çeşitli üniversitelerde ele alınmıştır.

Uzak bir data base içinde depolanan bilgiye çeşitli erişim yöntemleri mevcuttur. Özellikle ticari amaçlı pek çok sistem buna oldukça artan bir oranda ihtiyaç göstermekte ve haberleşme sistemlerinin bu denli gelişmesine ve yaygınlaşmasına itici güç teşkil etmektedir.

Bilimsel çalışmalarda, araştırma ve geliştirme alanlarında doğal olarak daha çok miktarda data kullanılmakta ve gün geçtikçe artan bu miktar, sistemleri çok yüklemektedir. Büyük araştırma projeleri, çeşitli ülkelerde, birbirinden çok uzak noktalarda kurulan computer sistemleri kullanılarak yapılabilmektedir. Aynı anda, yüzlerce insan aynı amaç doğrultusunda, aynı bir ortak yapı içerisinde ancak bu şekilde çalışabilmektedir. Bu olağanüstü data trafiğini, güvenliğini ve gizliliğini ancak güçlü bir DDBMS sağlayabilir.

Uzak data'ya erişim için çeşitli yöntemler mevcuttur. Örneğin bankacılıkta, bir uzak terminal belirli bir hesap numarasındaki hesabı kontrol etmek için, ana bilgisayara telefon hattı üzerinden bağlanır ve sadece ilgili hesaba ait kayıt bilgisini alarak işlevini sürdürür. Bir kayıt ancak birkaç onbyte'lık bilgidir ve transferi fazla zaman almaz. Onun için terminalde bilgi saklanmasına da gerek yoktur.

Ancak, uzak bir data base içinde depolanan çok fazla sayıda bir data'ya örneğin büyük bir file'a, erişmek böylesine basit bir terminal ile imkansızdır. Ve kullanıcı sayısının artmasıyla içinden çıkılmaz bir duruma gelinebilir. Bu durumda, bir mainframe içinden geçilerek, ve mainframe'in bilgi depolama imkanları kullanılarak sorun çözülebilir. Bu noktada da iki ayrı yöntem üzerinde düşünmek mümkündür:

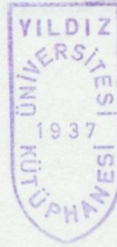
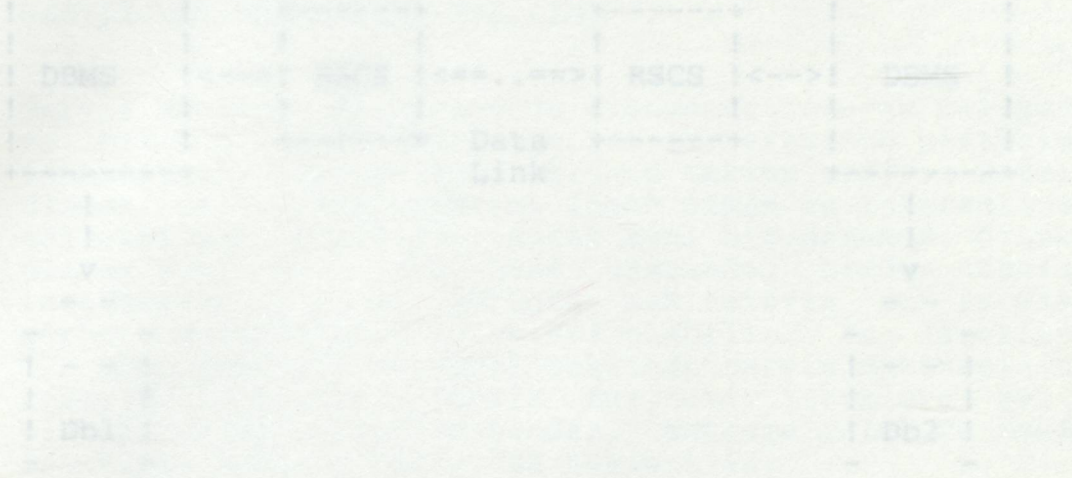
1. Aynı bir DBMS, her bir computerde kurulur ve local data base işler hale getirilir. Bu data base'de

sadece local data bulunmaktadır. Uzak data ya erişim isteği ortaya çıkınca DDBMS bu isteği, RSCS vm'leri üzerinden uzak sistemdeki DDBMS'e aktarır. Uzak DDBMS bu isteği (query) yerine getirir ve data'yı bir file halinde RSCS üzerinden istekte bulunan DDBMS'e ulaştırır. DDBMS bu datayı ilgili kullanıcıya yönlendirerek, kullanıcının bilgiye erişmesini sağlar. Kullanıcı local data ile uzak data arasında erişim yönünden sadece zaman farkı olduğunu hisseder. Bu yöntemde göre, uzak data base ile ilgili data, local diskler üzerinde saklanmadığından sadece local data için bellek gerekmektedir. Ancak kullanıcı aynı uzak data'yı, peşpeşe ve aralıklarla görmek isterse bu durumda gereksiz yere bir trafik yaratılmış olacaktır. Ayrıca fazla miktarda uzak data görme isteği, oldukça uzun bir süre beklemeye, dolaylı olarak zaman kaybına neden olacaktır. Data link'in efektif kullanılamaması da computer sistemindeki diğer kullanıcıların çalışmalarını da kötü yönde etkiler. Bu nedenlerden dolayı, uzak data'nın bu şekilde (contemporary) erişimi, bu şartlar altında verimli değildir.

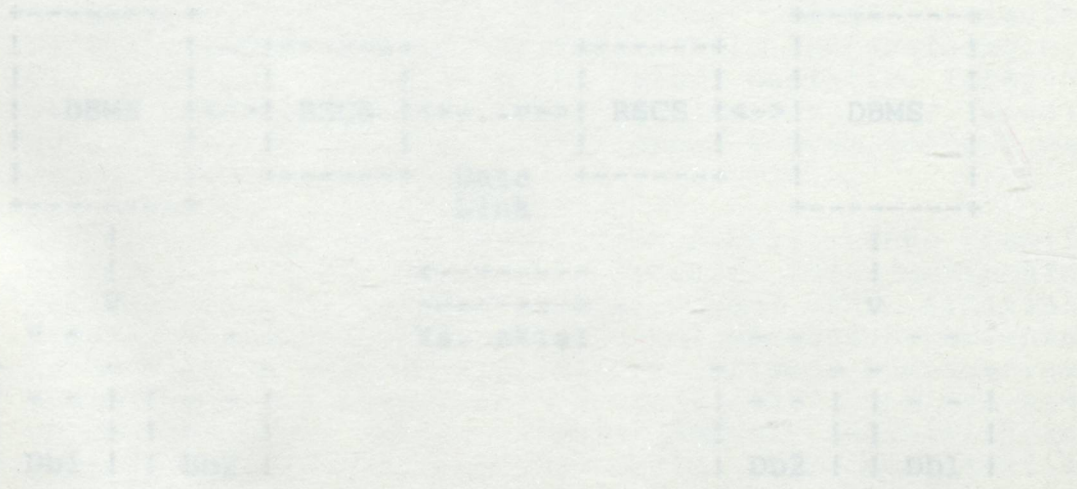
2. Bir diğer düşünce ise, local data gibi, uzak datanın da local diskler üzerinde depolanması ve uzak DBMS'in yaptığı her değişikliğin local DBMS tarafından da yapılmasının sağlanmasıdır. Uzak DBMS'de herhangi bir kullanıcının yarattığı değişiklik, bir DBMS komut/data grubu halinde harekete geçerek, network üzerindeki bütün DBMS'lerde aynı değişikliği yaratır. Bu komut/data paketine "TRANSACTION" denir ve kısaca XA ile gösterilecektir.

Bu yöntemin tek sakıncası, local data yanında uzak data'nın da aynı sistem üzerinde depolanmasının doğurduğu, fazla miktarda disk gereği ve yine data link üzerindeki trafiktir. Ancak, datalink üzerinde bir önceki yöntemin getirdiği kadar yoğun trafik olmaz, çünkü data erişim sıklığı her zaman data değişikliği yaratma sıklığından fazladır. Ancak, bu yöntemde, Xa yoğunluğu ve trafiği ölçüsünde DDBMS'in harcayacağı CPU time artacaktır. Bununla beraber, özellikle sık ve peşpeşe erişim gereken hallerde sadece local CPU içinde kalınacağından, data linkin efektif kullanılması sağlanmış olacaktır. Data link, sadece düzenli ve kontrollü olarak update yayılımı için kullanılacaktır. Bu yöntemin bir başka avantajı da data link üzerinde oluşabilecek kısa süreli kesintilerin hiçbir CPU noktasında DDBMS'in normal işlerliğini yitirmesine yol açmamasıdır. Data link'te uzun süreli (Bir günden çok) kesintiler, bu süre boyunca yaratılan değişikliklerin

sayısına bağılı olarak, DDBMS lerin farklılaşmasına neden olacaktır. Bu fark normal işlerliğe dönülmesiyle bir süre sonra ortadan kalkacak ve DDBMS bütün noktalarda aynı hale gelecektir.

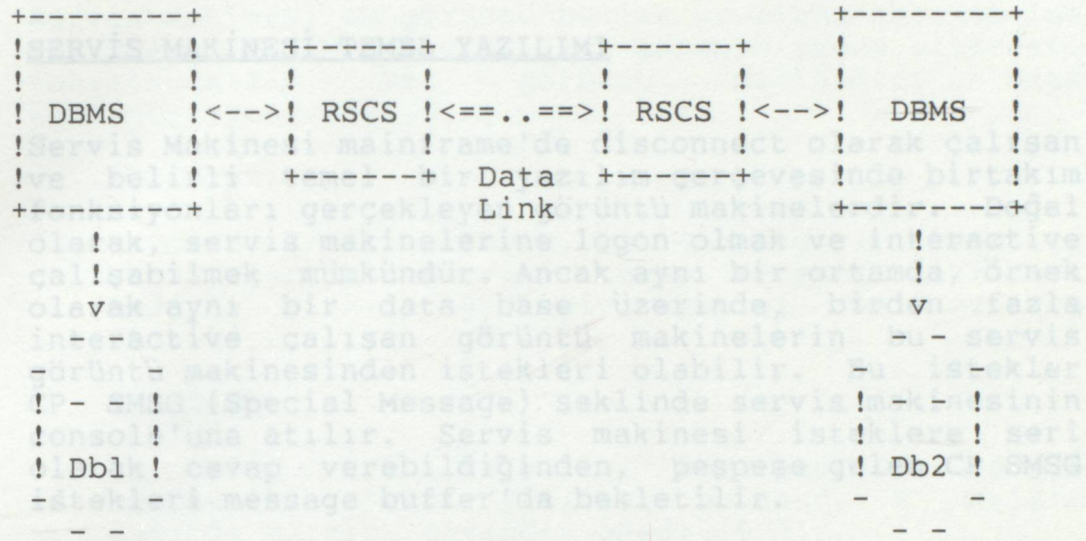


Sekil 2. Data Link Çekirdeğinin Tutulması



DBMS DESTEK YAZILIMI

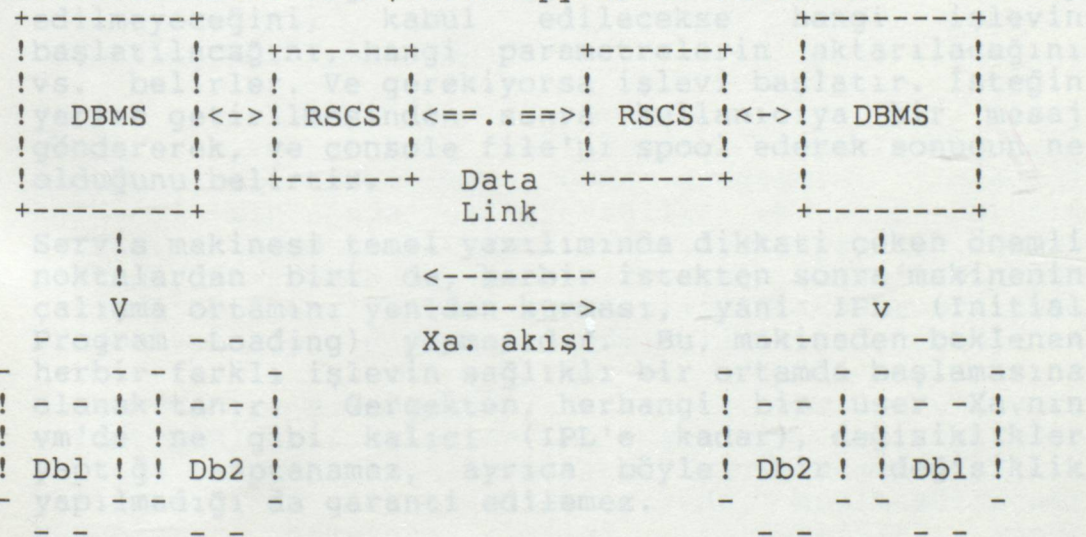
Şekil 1. Data'nın Birimler Arasında Dinamik Transferi



kapasitesiyle sınırlı olduğundan, bazı durumlarda servis makinesi herhangi bir isteğe cevap verirken SMSG OFF konumuna geçerek buffer taşması problemini temelinden halleder. Bu durumda, istekler birer spool file'ı biçiminde servis makinesinin reader'ına gönderilir. Bu, sisteme spooling ve paging yükü getirmekle beraber daha emin bir yoldur.

Servis makinesi, reader'ında herhangi bir istek bildiren spool file (JOB), veya console'una bir SMSG atılmadığı sürece bekleme konumundadır. Bekleme konumunda iken, servis makinesi kendine spool edilen bir reader file ile, harekete geçer ve belirli bir EXEC'i kosterur. Bu exec reader file'ı işleyerek kim

Şekil 2. Data'nın Çoklu Kopyasının Tutulması



DDBMS DESTEK YAZILIMI

SERVİS MAKİNESİ TEMEL YAZILIMI

Servis Makinesi mainframe'de disconnect olarak çalışan ve belirli temel bir yazılım çerçevesinde birtakım fonksiyonları gerçekleyen görüntü makinelerdir. Doğal olarak, servis makinelerine logon olmak ve interactive çalışabilmek mümkündür. Ancak aynı bir ortamda, örnek olarak aynı bir data base üzerinde, birden fazla interactive çalışan görüntü makinelerin bu servis görüntü makinesinden istekleri olabilir. Bu istekler CP SMSG (Special Message) şeklinde servis makinesinin console'una atılır. Servis makinesi isteklere seri olarak cevap verebildiğinden, peşpeşe gelen CP SMSG istekleri message buffer'da bekletilir.

Buffer'da bekletilebilecek SMSG miktarı buffer kapasitesiyle sınırlı olduğundan, bazı durumlarda servis makinesi herhangi bir isteğe cevap verirken SMSG OFF konumuna geçerek buffer taşması problemini temelinden halleder. Bu durumda, istekler birer spool file'ı biçiminde servis makinesinin reader'ına gönderilir. Bu, sisteme spooling ve paging yükü getirmekle beraber daha emin bir yoldur.

Servis makinesi, reader'ında herhangi bir istek bildiren spool file (JOB), veya console'una bir SMSG atılmadığı sürece bekleme konumundadır. Bekleme konumunda iken, servis makinesi kendine spool edilen bir reader file ile, harekete geçer ve belirli bir EXEC'i koşturur. Bu exec reader file'ını işleyerek kim tarafından gönderildiğini, kabul edilip edilmeyeceğini, kabul edilecekse hangi işlevin başlatılacağını, hangi parametrelerin aktarılacağını vs. belirler. Ve gerekiyorsa işlevi başlatır. İsteğin yerine getirilmesinden sonra kullanıcıya bir mesaj göndererek, ve console file'nı spool ederek sonucun ne olduğunu belirtir.

Servis makinesi temel yazılımında dikkati çeken önemli noktalardan biri de, herbir istekten sonra makinenin çalışma ortamını yeniden kurması, yani IPL (Initial Program Loading) yapmasıdır. Bu, makineden beklenen herbir farklı işlevin sağlıklı bir ortamda başlamasına olanak tanır. Gerçekten, herhangi bir user Xa'nın vm'de ne gibi kalıcı (IPL'e kadar), değişiklikler yaptığı saptanamaz, ayrıca böyle bir değişiklik yapılmadığı da garanti edilemez.

Servis Makinesi Konfigürasyonu

Servis makinesi 4M görüntü bellek üzerinde çalışabilen ve READER PUNCH, PRINTER gibi görüntü çevre birimleri tanımlanabilir. Bir görüntü makinedir. Disk konfigürasyonu şöyledir:

191 Disk:

Bu disk, R/W erişilerek, makinenin geçici yazma ve okuma gereksinmelerine cevap verecek şekilde A modunda alınır. Silindir sayısı en az 10 olmalıdır.

192 Disk:

Bu disk R/W erişilir ve üzerinde servis makinesinin temel yazılımı bulunur. B erişim modunda alınır. Silindir adedi 15 dir.

194 Disk:

Bu disk DATA disk olarak da adlandırılır ve üzerinde servis makinesi temel yazılımının kullanacağı tablolar ve data file'ları bulunur. R/O olarak, J erişim modunda erişilir.

196 Disk:

Bu disk I erişim modunda ve R/W olarak erişilir. Üzerinde servis makinesi temel yazılımının sürekli olarak kullandığı ve gerekirse değiştirdiği kalıcı data file'ları bulunur.

KULLANICI-DDMBS ARAYÜZÜ

Genel bir DDBMS kullanıcısı, vm ortamında, ulaşabildiği disklerden DDBMS programını istediği herhangi bir anda çalıştırabilir ve kendi görüntü belleğine istediği bir data base'i yükleyebilir. DDBMS programı ve DDBMS'in yönettiği data base'ler aynı ortamda bulunmalı, yeni DDBMS erişildikten sonra data base'lere erişim de garanti edilmelidir.

Kullanıcı ilk adımda interactive bir komut yorumlayıcı ortamına alınır. Bu ortam CMS üzerinde çalışmaktadır ve bu ortamdan çıkıldığı anda CMS'e tekrar dönülür. Kullanıcı, bu interactive ortamda, kullanabileceği komutları hakkında bilgi isteyebilir, komut formatlarını sorgulayabilir, hata yaptığı anda ikaz edilir.

Kullanıcı eğer yetkisi altında ise, herhangi bir data'yı değiştirebilir, silebilir, data ekleyebilir. Hatta data base yapısında değişiklik yapabilir. Ancak, kullanıcının kendi görüntü belleğinde yaptığı değişiklik hemen gerçek data base diskleri üzerine yansımaz. Yapılan değişikliklerin Xa haline getirilip ilgili servis makinesine gönderilmesi ve bu hatasız bir şekilde işlenmesi gerekmektedir. Servis makinesi data base disklerine R/W erişerek kullanıcı Xa larını seri olarak işler. Bu işleme, XA İŞLEME adı verilir ve daha sonraki bölümlerde ayrıntılı olarak üzerinde durulacaktır.

Kullanıcı isterse interactive çalışma sonunda yarattığı değişiklikleri iptal edip, CMS ortamına tekrar dönebilir. Bu durumda Xa aktif hale getirilmediği için, herşey kullanıcı görüntü belleğinde olup bitmiştir.

XA İŞLEME

Xa Formatlama

Xa, işlenme anında herhangi bir bilgi eksikliği yaratmayacak şekilde yeterli bilgiyle donatılmalı ve bu belirli, sabit bir formatta olmalıdır. Bu format çok karışık olmamalı, aksine DDBMS bakım ve sürekliliğini sağlayan insanların rahatça okuyabilmelerine olanak sağlamak amacıyla okunabilir olmalıdır. Söz konusu Xa formatı şu şekildedir:

```
*dbname
*vmid cpuid
-->cmd1
data1, data2,...
-->cmd2
-->data2
-->...
-->cmdn
-->datan
```

Xa'ın ilk olarak yaratıldığı site original site'dır ve Xa içinde ilk işlendiği DDBMS ortamının cevapları da eklenir. Bu işlem, normal bir kullanıcının interactive çalışması sırasında sistemin oluşturduğu console file oluşumuna benzer. Kullanıcının girdiği bir komuta karşılık -eğer varsa- sistemin cevabı da hemen alt satırda yer alır.

Sekil 3. DDBMS Servis Makinesi Konfigürasyonu.

+-----+(R/O Erişim)

! Vm1 !

! !----->

+-----+>

+-----+> +-----+

+-----+>

! Vm2 !

! !----->

+-----+>

+-----+>

+-----+>

! Vm3 !

! !----->

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

! Vmn !

! !----->

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

+-----+>

Bu yöntemin en önemli avantajı, original Xa ile birlikte yayılan sistem cevaplarının da her bir CPU üzerinde kontrol edilmesi, yani sistem güvenilirliğinin artmasıdır. Herhangi bir noktada sistem, işlenen Xa içinde gösterilen Xa'dan farklı bir cevap vermişse, DDBMS yetkili vm'leri uyarılır ve işlem askıya alınır. Bu noktada yetkili vm tarafından sorunun çözülmesi ve tekrar işleme devam edilmesi gerekir.

Xa üretimi

Xa, kullanıcının interactive çalışması sırasında üretilir. Kullanıcı kendi görüntü belleğinde çalışırken, data base de yarattığı değişikliklerin sonuçlarını hemen görür. Ve değişiklik eğer sorunsuz olarak sonlandırılmışsa, bu komut ve ilgili data bir Xa içine eklenir. Xa, kullanıcının kendi A diskinde yaratılır, ve interactive ortamdan çıkılınca kadar bu file değiştirilebilir. Kullanıcı, belli bir birim değişikliği tamamladıktan sonra bu Xa'ı hareket ettirebilir. İstenir ise Xa iptal edilerek, değişikliklerden vazgeçilir.

Xa, üretilirken kullanıcının terminalden (veya bir macro içinden) çalıştırdığı bütün komutlar Xa içine

alınır. Kısaca, sistemin yapısını değiştiren komutlar, eğer RC=0 ise, Xa'da yer alır. İşleme tekniği seri olarak ele alındığından, paralel Xa işleme ile ilgili problemlere burada değinilmeyecektir.

Xa Yorumlama

Xa yorumu, Data base üzerinde bütün yazma hakkına sahip olan servis görüntü makinesi tarafından yapılır. Normal olarak sonuçlandırılan bir interactive DBMS çalışması, en az bir Xa üretir ve bu Xa servis makinesinin reader'ına spool edilir. Servis makinesi readerı'nda biriken bu Xa'ları tek tek receive ederek işleme koyar.

Servis makinesi her bir Xa işlendikten sonra reader'ını kontrol eder, bir veya daha fazla reader file varsa, reader'ını belirli bir class'ı dikkate alarak sort eder. Ve ilk reader file'ni receive eder. Receive edilen Xa, DBMS komutlarından biri olan READ ile okunur. READ procedure içinde Xa önce stack bölgesine yüklenir, sonra satır satır okunarak, yorumlanır. Read işlemi sonrası bir sonlandırma dönüş kodu elde edilir. Bu kod eğer sıfır ise, işlenen Xa geçerlidir, değişiklik korunur. Eğer sıfırdan farklı ise hata routine'i çağrılarak hata belirlenmeye çalışılır, ve yetkili kullanıcılara mesaj gönderilir.

Xa Uyusumu

Xa işlenmesi sırasında sistem dengesinin korunması gerekir. Yani bir Xa işleniyorken, değişikliğin doğru olarak yaratılması için geçici olarak değiştirilen değerler tekrar eski haline getirilmelidir. Bu switch modda çalışmaya benzer. Örneğin bir Xa'nın işlenmesi tamamlanmadan yorumlama kesilmişse, ve sistem recovery'si tam olarak yapılamamışsa, sistem dengesi bozulur.

İşin kötü tarafı, bu durum belki o anda farkedilmeyebilir. Daha sonra hatasız gibi görünen bir Xa'nın işlenmesi sırasında, hiç de olmayacak bir noktada hata ortaya çıkar.

Bazı durumlarda Xa kaybolur. Xa'lar belirli bir sırada üretildiği ve üretildiği andaki sistemin yapısına bağlı oldukları için, aynı sırada işlenmeleri gerekir. Arada kaybolan bir Xa varsa, daha sonra kaybolan Xa ile ilgili bir data üzerinde değişimi amaçlayan başka bir Xa yorumlanırken problem ortaya çıkar.

Benzer şekilde, Xa sırasının karışması da yine benzer problemlere yol açar. Xa işleme tekniği seri olarak ele alındığından, paralel Xa işleme ile ilgili problemlere burada değinilmeyecektir.

Örnek bir DDBMS tasarımı, öncelikle küçük çaplı birtakım DBMS fonksiyonlarını yerine getiren, ve distribution gereklilikleri sağlayan bir DBMS'in yazılmasıdır. Bu bölümde, örnek olması bakımından ta-
XA YAYILIMI bir DBMS'in PASCAL programlama dilinde yazılması gerçekleştirilecektir.

Original Xa yonları bazen olarak link edilerek bir CMS module haline getirilmiş programda gerçekleştirilmiştir. Bu program procedure, function

Bir DDBMS servis görüntü makinesi için, başka bir DDBMS görüntü servis makinesinden gelmeyen, yani kendi CPU'su üzerindeki bir kullanıcıdan gelen her bir Xa, ORIGINAL XA olmaktadır. İlk defa işlenen bir Xa. eğer sıfır sonlandırma kodu ile işlenmiş ise, sistemin cevaplarını da içeren yeni bir Xa oluştururlur. Bir komut, ve bu komuta ait sistem cevabını şeklinde oluşan bu yeni Xa, REAPPLY XA olarak adlandırılır. İşte, network üzerindeki diğer birimlere bu Xa gönderilir.

Belirli bir birimden kendisinden sonra network üzerinde bulunan ilk birime transfer edilecek bir re-apply Xa için, belirli bir sıra numarası verilir. Bu sıra numarası Xa file'nın filename; birimdeki computer id'si de filetype olacak şekilde atanır. Reapply Xa numaralandırılması artan bir sıra takip eder. Ve 8 rakkamlı bir sayıdan oluşur. Bu sayı 99999999'a ulaştığında tekrar sıfırlanır.

README

Program ilk olarak çalıştırıldığında, local service vm, rscs linkleri, ve logical networkle ilgili tanımlamaları okur. Konusunu tanımlar DATA diskte üzerinde bulunan TDS BDFDATA file'ından okunur. File formatı şöyledir:

```
***** Top of file *****  
SITE      : TRITUI  
SERVICE  : RILY  
FROMSITE  : vmserv1  
FROMSITE  : TRITUI  
TOSITE    : vmserv2  
TOSITE    : TREARN  
VIA       : RSCS  
***** End of file *****
```


DDBMS TASARIMI

Örnek bir DDBMS tasarımı, öncelikle küçük çaplı birtakım DBMS fonksiyonlarını yerine getiren, ve distribution gereklerini sağlayan bir DBMS'in yazılması gerekmektedir. Bu bölümde, örnek olması bakımından table maintain eden bir DBMS'in PASCAL programlama dilinde yazılması gerçekleştirilecektir.

DBMS fonksiyonları bütün olarak link edilerek bir CMS module haline getirilmiş programda gerçekleştirilmiştir. Bu program procedure, function ve main routine'den oluşur. Anlaşılabilir olması bakımından herbir interactive komuta karşılık olarak bir MAIN ROUTINE'den bir procedure ve/veya function'lara dallanılır. Ancak bazı durumlarda bir interactive komut birden fazla procedure ve/veya function çağırabilir.

PROCEDURE'LER

DEFFILE

CMS FILEDEF komutu kullanılarak, program içinde kullanılacak file'lar tanımlanır.

READDEF

Program ilk olarak çağırıldığında, local service vm, rscs linkleri, ve logical networkle ilgili tanımlamaları okur. Soz konusu tanımlar DATA diski üzerinde bulunan TDB DEFDATA file'ından okunur. File formatı şöyledir:

```
===== * * * Top of file * * *
===== SITE      : TRYILDIZ
===== SERVVM    : BILY4
===== FROMSRV   : vmserv1
===== FROMSITE  : TRITUL
===== TOSERV    : vmserv2
===== TOSITE    : TREARN
===== VIA       : RSCS
===== * * * End of file * * *
```


ALLHEAD

Programda kullanılan bütün pointer'ları başlangıç noktalarına set eder. *ssa ve transactional modda çalışılıyorsa, başarıyla işlenmiş her bir komut ve data, transaction file'i içine eklenir. Service ve transaction file'i oluşturmaz.*

ALLBOT

Programda kullanılan bütün pointer'ları en son noktalarına set eder.

DEL interactive komutu tarafından çağrılır. Table içinde verilen şartları sağlayan data satırlarını
WRDELIM *komut formatı:*

DELETE ITEM WHERE field_name1 data1
Kullanıcının, hangi ortamda olduğunu anlaması bakımından, bir prompt'un kullanılması gerekmektedir. Bu procedure, uygun prompt'u ekrana basar.

RCHA

READSEC

CHANGE interactive komutu tarafından çağrılır. Table içinde verilen şartları sağlayan data satırlarını
Data *stir* base *komut* birtakım gizlilik kurallarını desteklemektedir. Örneğin bir kullanıcı başka bir kullanıcıya ait bir datayı değiştirebilmesi için yazma hakkına (write permit) sahip olmalıdır. Bunu da ancak o dataya erişirken girdiği *write-password'u* ile sağlar. Yanlış password girilmişse, write permit verilmez. Aynı şey, okuma için de geçerlidir. yani bir kullanıcı kendi datasına read password atayabilir ve, diğer bir kullanıcı bu passwordu girmedikçe o datayı okuyamaz.

Bir table'ın (field ve item'lardan oluşan data type) bütünüyle silinmesi ancak o table'ın sahibi (owner) tarafından yapılabilir.

LIST ITEM WHERE field_name1 data1
ALL

WRUNDLN

Table içindeki bilgileri ekrana yazarken field isimlerinin altını çizer.

SAVE interactive komutu tarafından çağrılır. Program bir service vm tarafından çağrılmışsa, transaction file service vm reader'ına spool edilir. Eğer, program service makinesi üzerinde çalışmıyorsa o zaman process

RADDXA data gerçek anlamda DATA disk üzerine save edilir. Komut formatı:

Eger program service vm dışında bir kullanıcı tarafından çağrılmışsa ve transactional modda çalışılıyorsa, başarıyla işlenmiş herbir komut ve data, transaction file'ı içine eklenir. Service vm transaction file'ı oluşturmaz.

RDEL interactive komutu tarafından çağrılır. Kullanıcının daha önceden yapılmış field tanımlarını görmesine olanak tanır. Ancak bu tanımların değiştirilmesi olanaksızdır. Komut formatı:

DEL interactive komutu tarafından çağrılır. Table içinde verilen şartları sağlayan data satırlarını siler. Komut formatı:

```
DELeTe ITEM WHERE field_name1 data1
ALL
```

RCHA Kullanıcının adını ve passwordlerini girdiği bir RCHA's alt datayı virtual memory'sine yükler. Ancak kullanıcının table'ın sahibi ya da doğru passwordleri olması gerekmektedir. Komut formatı:

CHANGE interactive komutu tarafından çağrılır. Table içinde verilen şartları sağlayan data satırlarını değiştirir. komut formatı:

```
CHAnge ITEM WHERE field_name1 data1 AS
field_name2 data2
ALL AS field_name3 data3
```

RLIST Kullanıcının field tanımlamalarını yapabildiğini sağlar. Kullanıcı bu procedure içinde prompt edilerek, belirli field özellikleri sorulur ve field paratılır. ADD komutu tarafından çağrılır.

LIST interactive komutu tarafından çağrılır. Table içinde verilen şartları sağlayan data satırlarını listeler. komut formatı:

```
ADD LIST ITEM WHERE field_name1 data1
ALL data2
```

Kullanıcının data eklemesini sağlar. Bu data field ya da item olabilir. ADD komutu kullanıcıyı gerektiği kadar prompt ederek datayı doğru bir şekilde alır. Komut formatı:

RSAVE

```
ADD FIELD
ITEM
```

SAVE interactive komutu tarafından çağrılır. Program bir service vm tarafından çağrılmamışsa, transaction file service vm reader'ına spool edilir. Eğer, program service makinesi üzerinde koştakiysa o zaman process

edilen data gerçek anlamda DATA disk üzerine save edilir. Komut formatı:

READ SAVE interactive komutu tarafından çağrılır. Kullanıcının herhangi bir file içinden interactive komutları pespe çalıştırmasını sağlar. Service vm transaction file'ni okurken bu proceduru çağırır. CONTENT formatı:

CONTENT interactive komutu tarafından çağrılır. Kullanıcının daha önceden yapılmış field tanımlarını görmesine olanak tanır. Ancak bu tanımların değiştirilmesi olanaksızdır. Komut formatı:

CONTENT

CMS interactive komutu tarafından çağrılır. CMS SUBSET çağrılarak kullanıcının TDB ortamından çıkmadan cms subset içinde çalışmasına olanak tanır. Komut formatı:

RLOAD

Kullanıcının adını ve passwordlerini girdiği bir table'a ait datayı virtual memory'sine yükler. Ancak kullanıcının table'ın sahibi yada doğru passwordleri girmiş olması gerekmektedir. Komut formatı:

RSET interactive komutu tarafından çağrılır. Kullanıcının bazı setting'leri değiştirmesini sağlar. RCREAT formatı:

Kullanıcının yeni bir table yaratırken FIELD tanımlamalarını yapabilmesini sağlar. Kullanıcı bu procedure içinde prompt edilerek, belirli field özellikleri sorulur ve field yaratılır. ADD komutu tarafından çağrılır.

RADD

ADD interactive komutu tarafından çağrılır. Kullanıcının yeni data eklemesini sağlar. Bu data field ya da item olabilir. ADD komutu kullanıcıyı gerektiği kadar prompt ederek datayı doğru bir şekilde okur. Komut formatı:

HELP interactive komutu tarafından çağrılır. TDB CMDHELP ADD FIELD ITEM HELP

RREAD

READ interactive komutu tarafından çağrılır. Kullanıcının herhangi bir file içinden interactive komutları peşpeşe çalıştırmasını sağlar. Service vm transaction file'ni okurken bu proceduru çağırır. Komut formatı:

READ fn ft fm

PRELOAD

RCMS

LOAD interactive komutu tarafından çağrılır. Birtakım on hazırlıklardan sonra RCMSD prosedur'u çağrılır. CMS interactive komutu tarafından çağrılır. CMS SUBSET çağrılarak kullanıcının TDB ortamından çıkmadan cms subset içinde çalışmasına olanak tanır. Komut formatı:

CMS

RDROP

RSET

table'i butünüyle ortadan kaldırır. DROP interactive komutu ile çağrılır. Bir table ancak sahibi tarafından DROP edilebilir. SET interactive komutu tarafından çağrılır. Kullanıcının bazı setting'leri değiştirmesini sağlar. Komut formatı:

SET DELIM <new_delim>

FUNCTION'LAR

ECHO <on!off>

PASSWORD W!R <pw>

UPPER

XAMODE (on!off>

Parametre OUTPUT TERM maksimum 80 karakterden oluşan bir string olabilir. bütün elemanlarını küçükse büyük harf FILE <fn ft fm> harf ya da başka bir karakterse, herhangi bir değişiklik yapmaz.

RHELP

HELP interactive komutu tarafından çağrılır. TDB CMDHELP file'ı içindeki açıklamaları ekrana yazar.

HELP

RINIT

Herbir table'a erişirken bu procedure çağrılır ve, data erişimi ile ilgili bütün başlangıç işleri yaptırılır. Bu işler arasında data diski üzerinde table ile ilgili file'i aramak, pointer'ların resetlenmesi passwordlerin kontrolü, vb. vardır.

PRELOAD

LOAD interactive komutu tarafından çağrılır. Birtakım on hazırlıklardan sonra RLOAD procedur'u çağrılır.

LOAD tablename <readpassword> <writepassword>

RDROP

Bir table'i bütünüyle ortadan kaldırır. DROP interactive komutu ile çağrılır. Bir table ancak sahibi tarafından DROP edilebilir.

DROP tablename

FUNCTION' LAR

UPPER

Parametre olarak aktarılan maximum 80 karakterden oluşan bir string değişkenin bütün elemanlarını küçükse büyük harfe çevirir; büyük harf ya da başka bir karakterse, herhangi bir değişiklik yapmaz.


```

EKL TDB----- Variables -----*)
Var-----
(*----- Strings -----*)
parmline string(80);(*for initial arguments *)
PROGRAM TDB(INPUT,OUTPUT) ; (*to determine caller void*)
(*----- Constants ----- *)
const maxlin = 500; (*site var read from tob
maxflds = 10; (*def file
(*----- Types ----- *)
type line = alpha; (*previous servvm in the
ptr1 = @chrdat; network
ptr2 = @numdat; (*previous servvm's site
tosex chrdtyp = string(100); next servvm in the
network
userflds = record (*next servvm's site
name : string(30) ; (*the rscs link
ttyp : (CHR$,NUM) ; (*read password
long : integer ; (*write password
link : integer ; (*table name
end ;
outfn : string(4) ;
outft : integer ;
secflds = record (*Output file variables
tabid ,owner,wpass,rpass : string(8) ;
arg4,arg5,arg6,arg7,arg8,arg9 ;
chrdat = record (*arguments strings
val : chrdtyp ; (*User defined delimiter
next : ptr1
end;
(*----- Integer -----*)
fin,din : integer ; (*Field index, data index
numdat = record (*chr link count
val : integer ; (*integer link count
next : ptr2 (*token number
end; (*global count variable
Retc : integer ; (*Return code of called
CMS routine

```



```

(*----- Variables -----*)
var
(*---- Strings ----*)
parmline      : string(80);(*for initial arguments *)
self,         : chrtyp      (*to determine caller vmid*)
here,         : chrtyp      (*Local site *)
site,         : chrtyp      (*site var read from tdb
def file      : ptr2        (*def file *)
servvm,       : ptr2        (*Local service machine *)
fromsrv,      : ptr2        (*previous servvm in the
network      : ptr2        *)
fromsite,     : ARRAY(0..10) (*previous servvm's site *)
toserv,       : ptr2        (*next servvm in the
network      : ptr2        *)
tosite,       : ptr2        (*next servvm's site *)
via,          : ARRAY(0..maxfld) (*the rscs link *)
rpwstr,       : ptr2        (*read password *)
wpwstr,       : ptr2        (*write password *)
tabname,      : ptr2        (*table name *)
outfn,        : ptr2        (*bottom, head pointers *)
outft         : string(8) ;
outfm         : string(1) ;(*Output file variables *)
arg1,arg2,arg3,
arg4,arg5,arg6,
arg7,arg8,arg9,
arg10,        : ptr2        (*arguments strings *)
delimmit      : string(16);(*User defined delimiter *)
(*---- Integers ----*)
fin, din,     : integer      (*Field index, data index *)
chrlkcnt,     : integer      (*chr link count *)
intlkcnt,     : integer      (*integer link count *)
tokn1,        : integer      (*token number *)
c,            : integer      (*global count variable *)
Retc          : integer      (*Return code of called
CMS routine *)

```



```

(*---- Private ----*)
uftable      : userflds ;(*user fields table      *)
tabdir       : secflds  ;(*table directory      *)
              -security director-
cmdline      : chrdtyp  ;(*user entered command line*)
cmdstr       : chrdtyp  ;(*actual command string  *)
(*---- Pointers ----*)
strpo        : ptr1     ;(*string type data pointer *)
intpo        : ptr2     ;(*integer type data pointer*)

(*---- Arrays ----*)
ut           : ARRAY(.0..10.) of userflds ;
              (*UserTable array element*)
SPA, SPAH,
              (*Str.ptr. arrays *)
SPAB, SPAP : ARRAY(.0..maxflds.) of ptr1;
              (*bottom, head pointers *)
IPA, IPAH,
              (*integer ptr. arrays *)
IPAB, IPAP : ARRAY(.0..maxflds.) of ptr2;
              (*bottom, head pointers *)

(*----- Alpha -----*)
rpwalp,      (*read pw alpha variable *)
wpwalp,      (*write pw alpha variab. *)
cmd1,        (*tokenized actual command
              alpha *)
tabnalp,     (*tokenized table name *)
parm1, parm2, parm3,
parm4, parm5, parm6,
parm7, parm8, parm9,
parm10      : alpha; (*tokenized parameters *)

```



```

(*----- Procedures & Functions -----*)

(*----- Booleans -----*)
delim, (*If '---->' will be typed or not *)
rwflag, (*If actual table really rw or not *)
xaflag, (*if cmds will be add in xa. or not *)
wperm,rperm, (*read or write permit ok or not *)
newtflag, (*If actual table is a new table *)
xavisa, (*if last cmdstr ran ok, or not *)
servflag, (*if the self is servvm *)
termflag, (*if outputs will be typed on term. *)
dropflag : boolean ; (*if actual table dropped or not *)

(*----- Files -----*)
strf : file of chrdtyp ;(*string data file *)
intf : file of integer ;(*integer data file*)
CTRDATA : file of userflds;(*user fields control file *)
tabdirf : file of secflds ;(*table directory file *)
defdata,
xafile, (*transaction file *)
outfile : text ;(*output file *)

```



```

Procedure READDEF;forward;mstr,site);
procedure readdef;
(*Reads the information from TDB DEFDATA file
TDB DEFDATA file contains the information about
(servvm) local and and remote service machine, RSCS
links, and recent sites connected via the links.*)
var aline: array(.0..7.) of string(80);
    dumstr:string(10);
    dummychr : string(2);
    xaorigvm : string(8);
    k : integer;
begin
(*Default TDB definitions *)
site:='*';
servvm:='*';
fromsrv:='*';
fromsite:='*';
toserv:='*';
tosite:='*';
via:='*';
CMS('ESTATE TDB DEFDATA *',Retc);
If Retc=0 then
begin
    writeln('TDB DEFDATA file cannot be found.');
```

RSCS

```

    writeln('Default DEFDATA is valid.');
```

RSCS

```

    return;
end;
CMS('FILEDEF DEFDATA DISK TDB DEFDATA *',Retc) ;
reset(defdata);
k:=0;
while not eof(defdata) do
begin
    readln(defdata,aline(.k.));
    k:=k+1;
end;
If k>=6 then
begin
    outs:=;
    for j:=1 to upto do
    begin
        readstr(mixed,j);
        if li>=129 then
        begin
            if li<=169 then c2:=chr(li-64);
            end;
        outs:=outs||str(c2);mixed:=delete(mixed,1,1);
        end;
    upper:=outs;
end;

```



```

Proc readstr(aline(.0.),dumstr,site);
proc If site='' then site:='*';
readstr(aline(.1.),dumstr,servvm);
If servvm='' then servvm:='*';
readstr(aline(.2.),dumstr,fromsrv);
readstr(aline(.3.),dumstr,fromsite);
readstr(aline(.4.),dumstr,toserv);
If toserv='' then toserv:='*';
readstr(aline(.5.),dumstr,tosite);
If tosite='' then toserv:='*';
readstr(aline(.6.),dumstr,via);
If via='' then via:='*';
If self=servvm then
begin
  writeln(self,' is SERVVM. '); servflag:=TRUE;
  (* here, I am sure this vm is service vm,
  but not it should be the xa originator. *)
  CMS('FILEDEF XAFILE DISK TDB CMDS *',Retc);
  CMS('ESTATE TDB CMDS *',Retc);
  If Retc=0 then
    begin
      reset(xafile);readln(xafile,dummychr);
      readln(xafile,dummychr,xaorigvm);
      close(xafile);self:=xaorigvm;
    end
  else
    writeln('*** Error, cannot read the xafile. ');
  end;
end
else
  writeln('TDB DEFDATA file format invalid. ');
  write('Defaults in effect. ');
end;
Function UPPER( mixed : string(80)) : string(80);
var j,i1,upto : integer ;
    outs      :string(80); c1,c2      :char;
begin
  outs:='';upto:=length(mixed);
  for j:=1 to upto do
    begin
      readstr(mixed,c1); i1:=ord(c1);c2:=c1;
      if i1>=129 then
        begin
          if i1<=169 then c2:=chr(i1+64);
        end;
      outs:=outs||str(c2);mixed:=delete(mixed,1,1);
    end;
  upper:=outs;
end;

```



```
Procedure ALLHEAD; forward ;
  procedure allhead;
    (*Sets the all string and integer data pointers
    to headers. So when this procedure called all
    the used pointers for current table, shows
    the top element.*)
    var i : integer ;
    begin
      for i:=0 to maxflds do
        begin
          SPA(.i.):=SPAH(.i.);
          IPA(.i.):=IPAH(.i.);
        end;
      end;
    end;

Procedure ALLBOT; forward ;
  procedure allbot;
    (*Sets the all string and integer data pointers
    to bottom. So when this procedure called all the
    used pointers for current table, shows the
    last element.*)
    var i : integer ;
    begin
      for i:=0 to maxflds do
        begin
          SPA(.i.):=SPAB(.i.);
          IPA(.i.):=IPAB(.i.);
        end;
      end;
    end;

Procedure WRDELIM ; forward ;
  procedure wrdelim ;
    (*Writes the delimeter and echoes the last entered
    command, if specified, for user information.*)
    begin
      If delim then writeln('---> ',cmdline);
      If cmdline='' then WRITELN(delimit) ;
    end;
```



```

Procedure READSEC ; forward ;
procedure readsec ;
(*Reads the security lines from TABDIRF TDB file
for checking if caller vm is table owner and/or
entered passwords correct or not.*)
var secline: secflds ;
begin
  (*Scan the tabdirf file to assign security
variables.*)
  tabdir.owner:='' ;
  reset(tabdirf);
  while not eof(tabdirf) do
    begin
      read(tabdirf,secline);
      if secline.tabid=tabname then
        begin
          tabdir.owner:=secline.owner;
          tabdir.wpass:=secline.wpass;
          tabdir.rpass:=secline.rpass;
          If self=tabdir.owner then
            begin
              Writeln('You are the owner of:',tabname);
              wperm:=TRUE;rperm:=TRUE;
              close(tabdirf);
              return;
            end;
          Writeln('Password incorrect. ');
          Writeln('Read right false');
          rperm:=FALSE;
        end;
      end;
    close(tabdirf);
    return;
  end;
end;
close(tabdirf);
writeln('The security lines not found in the TABDIRF');
writeln('Default settings assumed. ');
tabdir.owner:=self;
tabdir.wpass:='';
tabdir.rpass:='';
rperm:=TRUE;wperm:=TRUE;
end;

```



```

(*---Now the write password reading *)
if tabdir.wpass='' then wperm:=FALSE
else
if tabdir.wpass='*' then wperm:=TRUE
else
begin
begin
if wpwstr=tabdir.wpass then wperm:=TRUE
else
begin
writeln('Password incorrect. ');
writeln('Write right false');
wperm:=FALSE;
end;
end;
end;
(*---Now the read password reading *)

if tabdir.rpass='' then rperm:=FALSE
else
if tabdir.rpass='*' then rperm:=TRUE
else
begin
if rpwstr=tabdir.rpass then rperm:=TRUE
else
begin
writeln('Password incorrect. ');
writeln('Read right false');
rperm:=FALSE;
end;
end;
close(tabdirf);
return;
end;
end;
close(tabdirf);
writeln('The security lines not found in the TABDIR!');
writeln('Default settings assumed. ');
tabdir.owner:=self;
tabdir.wpass:='';
tabdir.rpass:='*';
rperm:=TRUE;wperm:=TRUE;
end;

```



```
Procedure WRUNDLN(var chrcnt: integer) ; forward ;
procedure wrundln ;
(*Prints out the underlines of field names.*)
var i: integer ;
begin
  If termflag then
    begin
      for i:=0 to chrcnt-1 do
        write('-');
      end
    else
      begin
        for i:=0 to chrcnt-1 do
          write(outfile, '-');
        end;
      end;
    end;
```



```

Procedure RADDXA ; forward ;
procedure raddxa ;
(*Adds the proper command into the (TDB CMDS) for
further use of servvm. If xaflag and xavisa TRUE
at the same time then last entered command is
proper. *)
var c1,p1,p2,p3,p4,p5,p6,p7: string(16) ;
    xaline : string(80) ;
begin
    If xaflag and xavisa then
    begin
        xaline:=compress(cmdline);
        writeln(xafile,xaline);
    end;
end ;

ALLHEAD ;
strpo:=SPA(.0.); (* Not a good way *)
while strpo.next~nil do (*repeat*)
begin
    for g:=0 to flg-1 do
    begin
        uftable:=UP(.g.) ;
        ll:=uftable.link ;
        If uftable.ityp=CHR$ then
        begin
            strpo:=SPA(.ll.);
            strpo.val:=strpo;
            strpo:=strpo.next;
            strpo.val:=strpo.val;
            strpo:=strpo;
        end
        else
        begin
            intpo:=IPA(.ll.);
            intpo.val:=intpo;
            intpo:=intpo.next;
            intpo.val:=intpo.val;
            intpo:=intpo;
        end
    end;
end;

```



```

Procedure RDEL;forward;
procedure rdel ;
(*Deletes the items from the current table.*)
var ll,n,s,g,i,j,int4: integer ;
    strcurr: array (.0..maxflds.) of chrddtyp ;
    intcurr: array (.0..maxflds.) of integer ;
    fldprok: boolean ;
    sprevp : ptr1 ;
    iprevp : ptr2 ;
begin
arg1:=UPPER(arg1);arg2:=UPPER(arg2);arg3:=UPPER(arg3);
s:=0; (* How many items deleted*)

(* Parameter handling...*)

ALLHEAD ;
strpo:=SPA(.0.); (* Not a good way *)
while strpo@.next~=NIL do (*repeat*)
begin
for g:=0 to fin-1 do
begin
uftable:=UT(.g.) ;
ll:=uftable.link ;
If uftable.ttyp=CHR$ then
begin
strpo:=SPA(.ll.);
SPAP(.ll.):=strpo;
strpo:=strpo@.next;
strcurr(.ll.):=strpo@.val;
SPA(.ll.):=strpo;
end
else
begin
intpo:=IPA(.ll.);
IPAP(.ll.):=intpo;
intpo:=intpo@.next;
intcurr(.ll.):=intpo@.val;
IPA(.ll.):=intpo;
end;
end;
end;

```



```

if arg1='ALL' then
begin
  fldprok:=FALSE;
  for g:=0 to fin-1 do
  begin
    uftable:=UT(.g.);
    ll:=uftable.link;
    if uftable.name=arg3 then
    begin
      if uftable.ttyp=CHR$ then
      begin
        if strcurr(.ll.)=arg4 then
        fldprok:=TRUE;
        leave;
      end
    else
      begin
        readstr(arg4,int4:0);
        if intcurr(.ll.)=int4 then
        fldprok:=TRUE;
        leave;
      end;
    end
  end;
end;
end;
SPAB(.ll.):=sprevp;
strpo:=SPA(.ll.);
end;
begin
  iprevp:=IPAP(.ll.);
  strpo:=IPA(.ll.);
  iprevp.next:=intpo.next;
  intpo.next:=NIL;
  IPA(.ll.):=iprevp;
  intpo:=IPA(.ll.);
end;
s:=s+1;
end;
end;
writeln(s:0,' textual deleted. ');
if s>0 then xav:=TRUE;
for j:=0 to fin-1 do
begin
  SPAB(.j.):=SPAB(.j.); SPAB(.j.):=IPA(.j.);
end;
din:=din-s;
end;

```



```

else
begin
  fldprok:=FALSE;
  ALLHEAD;
  for i:=0 to maxflds do
  begin
    strpo:=SPA(.i.); intpo:=IPA(.i.);
    SPAB(.i.):=SPA(.i.); IPAB(.i.):=IPA(.i.);
    strpo@.next:=NIL; intpo@.next:=NIL;
  end;
  s:=din;
  end;
  if fldprok then
  begin
    for g:=0 to fin-1 do
    begin
      uftable:=UT(.g.);
      ll:=uftable.link ;
      if uftable.ttyp=CHR$ then
      begin
        sprevp:=SPAP(.ll.);
        strpo:=SPA(.ll.);
        sprevp@.next:=strpo@.next;
        strpo@.next:=NIL;
        SPA(.ll.):=sprevp;
        strpo:=SPA(.ll.);
      end
      else
      begin
        iprevp:=IPAP(.ll.);
        intpo:=IPA(.ll.);
        iprevp@.next:=intpo@.next;
        intpo@.next:=NIL;
        IPA(.ll.):=iprevp;
        intpo:=IPA(.ll.);
      end;
    end;
    s:=s+1;
  end;
end ;
writeln(s:0,' item(s) deleted. ');
If s>0 then xavisa:=TRUE ;
for j:=0 to fin-1 do
begin
  SPAB(.j.):=SPA(.j.); IPAB(.j.):=IPA(.j.);
end;
din:=din-s;
end;

```



```

Procedure RCHA;forward;
procedure rcha ;
(*Updates the specified items for current table.*)
var ll,n,s,g,i,j,int4,int7: integer ;
    newdat : string(67);
    strcurr: array (.0..maxflds.) of chrdrtyp ;
    intcurr: array (.0..maxflds.) of integer ;
    fldprok: boolean ;
    sprevp : ptr1 ;
    iprevp : ptr2 ;
begin
arg1:=UPPER(arg1);arg2:=UPPER(arg2);
arg3:=UPPER(arg3);
arg5:=UPPER(arg5);arg6:=UPPER(arg6);
s:=0; (* How many items deleted*)

(* Parameter handling...*)

ALLHEAD ;
strpo:=SPA(.0.); (* Not the best way *)
while strpo@.next~NIL do (*repeat*)
begin
for g:=0 to fin-1 do
begin
uftable:=UT(.g.) ;
ll:=uftable.link ;
If uftable.ttyp=CHR$ then
begin
strpo:=SPA(.ll.);
SPAP(.ll.):=strpo;
strpo:=strpo@.next;
strcurr(.ll.):=strpo@.val;
SPA(.ll.):=strpo;
end
else
begin
intpo:=IPA(.ll.);
IPAP(.ll.):=intpo;
intpo:=intpo@.next;
intcurr(.ll.):=intpo@.val;
IPA(.ll.):=intpo;
end;
end;
end;

```



```
if arg1~='ALL' then
  begin
    newdat:= arg7||arg8||arg9||arg10;
    newdat:= trim(newdat);
    fldprok:=FALSE;
    for g:=0 to fin-1 do
      begin
        uftable:=UT(.g.);
        ll:=uftable.link ;
        if uftable.name=arg3 then
          begin
            if uftable.ttyp=CHR$ then
              begin
                if strcurr(.ll.)=arg4 then
                  fldprok:=TRUE ;
                leave;
              end
            else
              begin
                readstr(arg4,int4:0);
                if intcurr(.ll.)=int4 then
                  fldprok:=TRUE ;
                leave;
              end;
            end
          end
        end;
      end
    end
  end
```



```

else
begin
fldprok:=FALSE;
newdat:= arg4||arg5||arg6||arg7;
newdat:= trim(newdat);
(* ALLHEAD; *)
for g:=0 to fin-1 do
begin
uftable:=UT(.g.);
ll:=uftable.link ;
if uftable.name=arg3 then
begin
s:=din;
if uftable.ttyp=CHR$ then
begin
strpo:=SPA(.ll.);
strpo@.val:=newdat;
end
else
begin
intpo:=IPA(.ll.);
readstr(arg4,int7:0);
intpo@.val:=int7;
end;
(* writeint(int7); *)
(* s:=s+1; *) leave;
end;
end;
end;
writein(s: table) changed;
if s>0 then xavial:=TRUE;
for j:=0 to fin-1 do
begin
SPAB(.j.):=SPA(.j.);
IPAB(.j.):=IPA(.j.);
end;
(* until strpo@.next=0; *)
end;

```



```

if fldprok then
begin
  for g:=0 to fin-1 do
  begin
    loc : ar uftable:=UT(.g.);
    strcurr: ll:=uftable.link ;
    intcurr: if uftable.name=arg6 then
    fldprok: boo begin
    outfid:string if uftable.ttyp=CHR$ then
    begin
      arg1:=UPPER(arg1); ar strpo:=SPA(.ll.);
      arg3:=UPPER(arg3); strpo@.val:=newdat;
      If fin=0 then end
      begin else
      writeln('Nothing' begin
      return; intpo:=IPA(.ll.);
      end; readstr(arg7,int7:0);
      If termflag then intpo@.val:=int7;
      begin end;
      for g:=0 to s:=s+1;
      begin leave;
      uftable: end;
      end;
      (* writeln('');*)
      (* s:=s+1; *)
      end;
      end ;
      writeln(s:0,' item(s) changed.');
```

If s>0 then xavisa:=TRUE ;

```

for j:=0 to fin-1 do
begin
  SPAB(.j.):=SPA(.j.);
  IPAB(.j.):=IPA(.j.);
end;
(* until strpo@.next=NIL ; *)
end;
  writeln(outfid);
  end;
  writeln(outfid);
  end;

```



```

Procedure RLIST;forward;
procedure rlist ;
  (*Lists the current table's items. *)
  var ll,n,s,g,i,j,int4: integer ;
      loc : array(.0..maxflds.) of integer ;
      strcurr: array (.0..maxflds.) of chrdtyp ;
      intcurr: array (.0..maxflds.) of integer ;
      fldprok: boolean ;
      outfid:string(20);
begin
  arg1:=UPPER(arg1); arg2:=UPPER(arg2);
  arg3:=UPPER(arg3);
  If fin=0 then
    begin
      writeln('Nothing to list. ');
      return;
    end;
  If termflag then
    begin
      for g:=0 to fin-1 do
        begin
          uftable:=UT(.g.);
          loc(.g.):=uftable.long ;
          write(uftable.name:loc(.g.),' ');
        end;
      writeln('');
    end
  else
    begin
      outfid:=outfn||' '||outft||' '||outfm;
      CMS('FILEDEF OUTFILE DISK '||outfid,retc);
      rewrite(outfile);
      for g:=0 to fin-1 do
        begin
          uftable:=UT(.g.);
          loc(.g.):=uftable.long ;
          write(outfile,uftable.name:loc(.g.),' ');
        end;
      writeln(outfile,'');
    end;
end;

```



```
for g:=0 to fin-1 do
begin
  uftable:=UT(.g.);
  wrundln(uftable.long) ;
  If termflag then write(' ')
  else write(outfile,' ');
end;
If termflag then writeln('')
else writeln(outfile,'');
ALLHEAD ;
strpo:=SPA(.0.); (*A bad temporary checking *)
while strpo@.next~NIL do
begin
  for g:=0 to fin-1 do
  begin
    uftable:=UT(.g.) ;
    ll:=uftable.link ;
    If uftable.ttyp=CHR$ then
    begin
      strpo:=SPA(.ll.);
      strpo:=strpo@.next ;
      strcurr(.ll.):=strpo@.val;
      SPA(.ll.):=strpo ;
    end
  else
  begin
    intpo:=IPA(.ll.);
    intpo:=intpo@.next ;
    intcurr(.ll.):=intpo@.val;
    IPA(.ll.):=intpo ;
  end;
end;
end;
```



```

Procedure if arg1='ALL' then
begin
  (*If the caller is not a servvm,
  as a result of saving the
  things on the caller is not a servvm,
  it sends the servvm to be
  permit, then *)
  fldprok:=FALSE;
  for g:=0 to fin-1 do
  begin
    uftable:=UT(.g.);ll:=uftable.link ;
    if uftable.name=arg3 then
    begin
      if uftable.ttyp=CHR$ then
      begin
        if strcurr(.ll.)=arg4 then
        fldprok:=TRUE ; leave;
        end
      else
      begin
        readstr(arg4,int4:0);
        if intcurr(.ll.)=int4 then
        fldprok:=TRUE ; leave;
        end;
      end;
    end;
  end;
end;
else fldprok:=TRUE ;
if fldprok then
begin
  If rwflag for g:=0 to fin-1 do
  begin
    uftable:=UT(.g.);ll:=uftable.link ;
    if uftable.ttyp=CHR$ then
    begin
      If termflag then
        write(strcurr(.ll.):loc(.g.),' ');
      else
        write(outfile,strcurr(.ll.):loc(.g.),' ');
      end
    else
    begin
      If termflag then
        write(intcurr(.ll.):loc(.g.),' ');
      else
        write(outfile,intcurr(.ll.):loc(.g.),' ');
      end;
    end;
  end;
  If termflag then writeln('')
  else writeln(outfile,'');
end;
end ;
end ;

```



```

Procedure RSAVE; forward ;
procedure rsave ;
(*If the caller is servvm, then SAVE command acts
as a really save procedure, by means of saving the
things on the disk. But if the caller is not a servvm,
it sends the transaction file (TDB CMDS) off to
servvm to be processed. If the user has not write
permit, then save command does not work *)

var m,n,ll: integer ;
    secline: secflds;
    temp : file of secflds ;
begin
If not wperm then
begin
writeln('No items can be saved due to the disk. ');
write right false. ');
return;
end;
If fin=0 then
begin
writeln('Nothing to save. ');
return ;
end;

If rwflag & servflag then (* Temporary check *)
begin
If newtflag=TRUE then
begin
reset(tabdirf);
rewrite(temp);
while not eof(tabdirf) do
begin
read(tabdirf,secline);
write(temp,secline);
end;
close(tabdirf);
write(temp,tabdir);
close(temp);
CMS('ERASE TABDIRF TDB A',Retc);
CMS('RENAME TEMP FILE A TABDIRF TDB A',Retc);
end;

```



```

If dropflag then
begin
  CMS('ERASE ' || tabname || ' CTRDATA A',Retc);
  CMS('ERASE ' || tabname || ' INTDATA A',Retc);
  CMS('ERASE ' || tabname || ' STRDATA A',Retc);
  din:=0;fin:=0;
  return;
end;
ALLHEAD;
rewrite(ctrdata) ;
for n:=0 to fin-1 do
begin
  (* write(CTRDATA,UT(.n.)) ;*)
  CTRDATA@:=UT(.n.) ; put(CTRDATA) ;
end;
writeln(fin:1,' Field(s) saved on the disk.');
```

close(CTRDATA) ;

If din>0 then

```

begin
  rewrite(STRF) ;rewrite(INTF) ;
  write(INTF,din) ;(*I need to learn din when
                    loading *)
  write(INTF,chrkcnt);
  write(INTF,intlkcnt);
  for m:=0 to fin-1 do
  begin
    uftable:=UT(.m.);
    ll:=uftable.link ;
    If uftable.ttyp=CHR$ then
    begin
      strpo:=SPA(.ll.) ;
      while strpo@.next<=NIL do
      begin
        strpo:=strpo@.next ;
        write(STRF,strpo@.val) ;
      end;
      (*until strpo=NIL;*)
    end
  else
    begin
      intpo:=IPA(.ll.);
      while intpo@.next<=NIL do
      begin
        intpo:=intpo@.next;
        write(INTF,intpo@.val);
      end
      (*until intpo=NIL;*)
    end;
  end;
end;
end;
```



```
Procedure COM close(STRF) ;
procedure COM close(INTF) ;
(*This procedure
just shows the
var s, r, l, t
begin
If servflag then
begin
If toserv='*' then
begin
CMS('CP TAG DEV PUNCH '||tosite||' '
||toserv,Retc);
CMS('CP SPOOL PUNCH TO '||via,Retc);
CMS('DISK DUMP TDB CMDS ',Retc) ;
CMS('CP TAG DEV PUNCH ',Retc);
CMS('CP SPOOL PUNCH TO ',Retc) ;
end;
end;
end;
else
begin
WRITELN(xafile,'save');
WRITELN(xafile,'quit');
CLOSE(xafile) ;
CMS('CP SPOOL PUNCH TO '||servvm,Retc) ;
CMS('DISK DUMP TDB CMDS A',Retc) ;
CMS('CP SPOOL PUNCH TO ',Retc) ;
CMS('ERASE TDB OLDCMDS A',Retc) ;
CMS('RENAME TDB CMDS A = OLDCMDS A',Retc) ;
end;
(* User may go on his interactive session in the same
table, so he may need a new xafile if he is allowed.*)
If dropflag then newtflag:=TRUE;
If rwflag then return;
If wperm then
begin
rewrite(xafile);
writeln(xafile,'* ',tabname);
writeln(xafile,'* ',self);
end;
end;
```



```

Procedure CONTENT ; forward ;
procedure content ;
  (*This procedure thought as a maintenance and integer
  procedure for use of MAINT VMS. This
  just shows the user table definitions for now.*)
var m,n,ll : integer ;
begin
  If fin>0 then
    begin
      writeln('-----The usertable UT:-----') ;
      for n:=0 to fin-1 do
        begin
          uftable:=UT(.n.) ;
          write(uftable.name) ;
          (*If uftable.ttyp=CHR$ then write('CHR$')
          else write('NUM') ;
          CMS('EST' write(uftable.long) ;
          if Retc=0 then writeln(uftable.link) ;
          end;
          writeln('-----') ;
          writeln('The fin=', fin, ' ', ' The din=', din) ;
          end ;
        end;
      end;
      reset(INTF);
      read(INTF,din);
      read(INTF,chr);
      read(INTF,int);
      CMS('ESTATE' write(int,chr,Retc) ;
      If Retc=0 then
        begin
          write('Data has been loaded');
          return;
        end;
      reset(INTF);
      read(INTF,din);
      read(INTF,chr);
      read(INTF,int);
      CMS('ESTATE' write(int,chr,Retc) ;

```



```
Procedure RLOAD; forward ;
procedure rload ;
(* Loads the things (control data, string and integer
type data from disk. *)
var n,ll,m : integer ;
begin

If not rperm then
begin
writeln('No items can be loaded due to read right
false.');
```

```
return;
end;
If fin>0 then ALLHEAD ;
(*This because if any data defined before
then the pointers should be set to head*)
CMS('ESTATE '||tabname||' CTRDATA *',Retc) ;
If Retc=0 then
begin
fin:=0 ;
reset(CTRDATA) ;
while not eof(ctrdata) do
begin
read(CTRDATA,UT(.fin.)) ;
fin:=fin+1 ;
end ;
close(CTRDATA) ;
writeln(fin:1,' Field(s) has been loaded');
```

```
end;

CMS('ESTATE '||tabname||' INTDATA *',Retc) ;
If Retc=0 then
begin
writeln('Loading cannot be performed');
```

```
return;
end;
reset(INTF);
read(INTF,din);
read(INTF,chr1kcnt);
read(INTF,int1kcnt);
CMS('ESTATE '||tabname||' STRDATA *',Retc) ;
```



```
If Retc=0 then
begin
  writeln('*Warning! Loading was completed unsane');
  (* There may be integer data to be read in but,
  I cannot go ahead, if the string data file
  not found.*)
  close(INTF); return;
end;
(* Here, everything is ok, loading is sane....*)
reset(STRF);
for m:=0 to fin-1 do
begin
  uftable:=UT(.m.);
  ll:=uftable.link ;
  If uftable.ttyp=CHR$ then
  begin
    strpo:=SPA(.ll.);
    for n:=0 to din-1 do
    begin
      new(strpo@.next);
      strpo:=strpo@.next;
      read(STRF,strpo@.val);
    end;
    strpo@.next:=NIL;
    SPAB(.ll.):=strpo;
  end
  else
  begin
    intpo:=IPA(.ll.);
    for n:=0 to din-1 do
    begin
      new(intpo@.next);
      intpo:=intpo@.next;
      read(INTF,intpo@.val);
    end;
    intpo@.next:=NIL;
    IPAB(.ll.):=intpo;
  end ;
end;
writeln(din:1,' Item(s) has been loaded');
(*xavisa:=TRUE ;
I am still thinking about that if the LOAD is necessary
for a user. IF not, then the Load should not be called
by the user. Ok, the LOAD command should not be a
transaction command. Because a new xafile is produced
for every loaded table with write permit.*)
close(STRF);close(INTF) ;
end;
```



```

Procedure RCREAT ; forward ;
procedure rcreat ;
  (*Creates a new user field. This procedure called by
  ADD FIELD command by user. *)
var ftype : string(4) ;
    i,ll : integer ;
begin
  writeln('Creating Fields.....') ;
  writeln('Please enter Field name.') ;
  readln(uftable.name) ;
  uftable.name:=UPPER(uftable.name);
  writeln('Please enter the type of field. (char or num)');
  readln(ftype) ;
  ftype:=UPPER(ftype);
  if ftype='CHAR' then uftable.ttyp:=CHR$
  else
    if ftype='NUM' then uftable.ttyp:=NUM
  else
    begin
      writeln('***Warning!, field type assumed as CHAR.') ;
      uftable.ttyp:=CHR$ ;
    end ;
  writeln('Please enter the maximum length of field.') ;
  readln(uftable.long) ;
  (*-----
  ... Thats ok for user line, lets determine the linkage
  -----*)
  If uftable.ttyp=CHR$ then
    begin
      uftable.link:=chrlkcnt ;
      chrlkcnt:=chrlkcnt+1 ;
    end
  else
    begin
      uftable.link:=intlkcnt ;
      intlkcnt:=intlkcnt+1 ;
    end;
  UT(.fin.):=uftable ;
  fin:=fin+1 ;
  writeln('The field has been created for new field. ');
end ;

```



```
Writeln(ufable.name,' has been created.');
```

```
  If xaflag then
    begin
      writeln(xafile,cmdstr,arg1);
      writeln(xafile,ufable.name);
      writeln(xafile,ftype);
      writeln(xafile,ufable.long);
    end;
```

```
  (* If the user entered more fields after he entered a
    bit of data then new added field should be filled
    up with a proper data: null if it was string,
                                0 if it was integer or real*)

  If din>1 then
    begin
      ufable:=UT(.fin-1.);
      ll:=ufable.link;
      ALLHEAD;
      strpo:=SPA(.ll.);
      intpo:=IPA(.ll.);
      If ufable.ttyp=CHR$ then
        begin
          for i:=0 to din-1 do
            begin
              new(strpo@.next);
              strpo:=strpo@.next;
              strpo@.val:=' ' ;
            end;
          strpo@.next:=NIL ;
          SPAB(.ll.):=strpo;
        end
      else
        begin
          for i:=0 to din-1 do
            begin
              new(intpo@.next);
              intpo:=intpo@.next;
              intpo@.val:=0 ;
            end;
          intpo@.next:=NIL ;
          IPAB(.ll.):=intpo;
        end;
      writeln('The default data entered for new field.');
```

```
    end;
  end ;
```



```
Procedure RADD; forward ;
procedure radd ;
(* Adds the new items into current table *)
var cn,ll: integer ;
begin
  arg1:=COMPRESS(UPPER(arg1));
  If arg1='FIELD' then
    begin
      RCREAT;
      return;
    end;
  If arg1~='ITEM' then
    begin
      Writeln(arg1,' cannot be added.');
```

return;

end;

If fin=0 then

begin

Writeln('No field has been created yet.') ;

return ;

end;

If xaflag then writeln(xafile,cmdstr,arg1);

ALLBOT ;


```

for cn:=0 to fin-1 do
begin
  uftable:=UT(.cn.);
  writeln(uftable.name,':':1);
  ll:=uftable.link;
  If uftable.ttyp=CHR$ then
  begin
    strpo:=SPA(.ll.);
    new(strpo@.next);
    strpo:=strpo@.next;
    readln(strpo@.val);
    strpo@.val:=compress(strpo@.val);
    if xaflag then writeln(xafile,strpo@.val);
    strpo@.next:=NIL;
    SPAB(.ll.):=strpo;
  end
  else
  begin
    intpo:=IPA(.ll.);
    new(intpo@.next);
    intpo:=intpo@.next;
    readln(intpo@.val);
    if xaflag then writeln(xafile,intpo@.val);
    intpo@.next:=NIL;
    IPAB(.ll.):=intpo;
  end
end;
din:=din+1;
writeln('1 item added.');
```



```

Procedure RREAD ; forward ;
procedure rread ;
(*Reads any specified cms file, and executes the
recognized command lines as those are entered at
the terminal.*)
var fileid : string(30) ;
fname,ftype : string(16) ;
begin
if arg1~='' then
if arg2~='' then
begin
fname:=arg1;ftype:=arg2;
fname:=compress(fname);
ftype:=compress(ftype);
fileid:=fname||' '||ftype||' *' ;
CMS('ESTATE '||fileid,retc) ;
If retc=0 then
begin
CMS('SENTRIES',retc);
writeln('Reading file:', fileid);
CMS('EXECIO * DISKR '||fileid,retc);
If retc=0 then xavisa:=FALSE
end
end
(* I have set the xavisa to false because the commands
that pulled from the stach already entered into the
xafile if it should be there.*)
else
begin
writeln('Reading error for file:',fileid) ;
CMS('DROPBUF',retc) ;
end;
CMS('FINIS '||fileid,retc);
end
else writeln(fileid,' not found.');
```



```
Procedure RCMS; forward ;
procedure rcms ;
  (*Calls the cms subset. (This procedure is available for all vms. This procedure
  var singcmd : string(102);
begin
singcmd:=arg1||' '||arg2||' '||arg3||' '||arg4||' '
||arg5||' '||arg6;
If singcmd='' Then
begin
  CMS('CMSCMD',retc) ;
  WRDELIM ;
end
else begin
  CMS(singcmd,retc) ;
  Writeln('Ready(',retc:1,')');
  WRDELIM ;
end;

(* 194 **OLD   cmscmd:='DUMMY' ;
195 **OLD   writeln('TDBCMS');
196 **OLD   while cmscmd <='return' do
197 **OLD     begin
198 **OLD       readln(cmscmd) ;
199 **OLD       If cmscmd<=' ' then
200 **OLD       If cmscmd<='return' then
201 **OLD       begin
202 **OLD         CMS(cmscmd,retc) ;
203 **OLD         Writeln('Ready(',retc:1,')');
204 **OLD       end
205 **OLD       else
206 **OLD       else writeln('TDBCMS');
207 **OLD     end;

The above line was removed from the code since
CMSCMD is available in the NUCEXT. But if the
target system has not provide this facility
**OLD lines should be inserted in. *)

end ;
```



```
Procedure RSET; forward ;
procedure rset;
(* Settings are available for all vms. This procedure
   allows the user set them to a proper value.*)
var setp3:string(8);
begin
  readstr(arg3,setp3);
  setp3:=UPPER(setp3);
  arg1:=UPPER(arg1);arg2:=UPPER(arg2);
  arg3:=UPPER(arg3);arg4:=UPPER(arg4);
  arg5:=UPPER(arg5);arg6:=UPPER(arg6);

  If arg1='' Then
    begin
      Writeln('Missing parametter');
      return;
    end;
  If arg2='' Then
    begin
      Writeln('Missing parametter');
      return;
    end;

  If arg1='ECHO' then
    begin
      If arg2='ON' then delim:=TRUE ;
      If arg2='OFF' then delim:=FALSE ;
    end
  else
```



```

If arg1='OUTPUT' then
  begin
    If arg2='FILE' then
      begin
        outfn:=substr(arg3,1,8);
        outft:=substr(arg4,1,8);
        outfm:=substr(arg5,1,1);
        If outfn='' Then outfn:='TDB';
        If outft='' Then outfn:='OUTPUT';
        If outfm='' Then outfm:='A';
        termflag:=FALSE;
      end;
    If arg2='TERM' then termflag:=TRUE;
  end
else
  If arg1='XAMODE' then
    begin
      If arg2='ON' then xaflag:=TRUE ;
      If arg2='OFF' then xaflag:=FALSE ;
    end
  else
    If arg1='PASSWORD' then
      begin
        If newtflag=TRUE then
          begin
            If arg2='W' then tabdir.wpass:=setp3;
            If arg2='R' then tabdir.rpass:=setp3;
            xavisa:=TRUE ;
          end
        else
          writeln('Passwords has already been set.');
```

```

        end
      end
    else
      If arg1='DELIM' then delimmit:=arg2
    else
      writeln(arg1,' is unknown setting.');
```

```

WRDELIM ;
end ;
```

```

Procedure RHELP; forward ;
procedure rhelp ;
begin
  writeln('All commands:');
  CMS('TYPE TDB CMDHELP *',Retc) ;
end ;
```



```

Procedure RINIT(initmode: string(8)) ; forward ;
procedure rinit ;
(*Initialization procedure is called for loading any
   table as it calls the LOAD procedure after it did
   some initial settings. *)
var i : integer ;
begin
  fin:=0 ; din:=0 ; chr1kcnt:=0 ; intlkcnt:=0;
  delim:=FALSE ;
  xavisa:=FALSE ;
  dropflag:=FALSE ;
  newtflag:=FALSE ;
  termflag:=TRUE ;
  delimmit:='TDB '||tabname||':' ;
  new(strpo) ;
  new(intpo) ;
  new(strpo@.next) ;
  new(intpo@.next) ;
  for i:=0 to maxflds do
    begin
      new(SPA(.i.)); new(IPA(.i.));
      new(SPAP(.i.)); new(IPAP(.i.));
      SPAH(.i.):=SPA(.i.); IPAH(.i.):=IPA(.i.);
      SPAB(.i.):=SPAH(.i.); IPAB(.i.):=IPAH(.i.);
    end;

    CMS('STATEW '||tabname||' CTRDATA *',Retc) ;
    If Retc=0 then
      begin
        writeln('R/W items connected.') ;
        wperm:=TRUE ; rperm:=TRUE ;
        RLOAD;
        rwflag:=TRUE ;
        xaflag:=FALSE ;
      end
    else begin
      CMS('STATE '||tabname||' CTRDATA *',Retc) ;
      If Retc=0 then
        begin
          writeln('R/O items connected.') ;
          CMS('FILEDEF TABDIRF DISK TABDIRF TDB *',Retc) ;
          READSEC;
          RLOAD;
          rwflag:=FALSE ;
          xaflag:=TRUE ;
        end
      end
    end
  end
  (*
  xaflag:=TRUE ; *)

```



```

Procedure P If wperm then
begin
  (* Load command *)
  xafLAG:=TRUE;
  CMS('FILEDEF XAFILE DISK TDB CMDS A',Retc) ;
  rewrite(xafile);
  If xafLAG then
  begin
    writeln(xafile,'* ',tabname);
    writeln(xafile,'* ',self);
  end;
end;
else
begin
  Writeln(tabname,' is a new table');
  (*Here, default tabdir line should be assigned *)
  CMS('FILEDEF TABDIRF DISK TABDIRF TDB *',Retc) ;
  CMS('FILEDEF TEMP DISK TEMP FILE A',Retc) ;
  newtFLAG:=TRUE ;
  tabdir.tabid:= tabname ;
  tabdir.owner:= self;
  tabdir.wpass:= ' ';
  tabdir.rpass:= '*';
  wperm:=TRUE ; rperm:=TRUE ;
  (*rwFLAG:=TRUE ; User not allowed save his table on
  his disk. *)
  If servFLAG then
  begin
    rwFLAG:=TRUE;
    xafLAG:=FALSE;
  end
  else
  begin
    rwFLAG:=FALSE ;
    xafLAG:=TRUE ;
    CMS('FILEDEF XAFILE DISK TDB CMDS A',Retc) ;
    rewrite(xafile);
    If xafLAG then
    begin
      writeln(xafile,'* ',tabname);
      writeln(xafile,'* ',self);
    end;
  end;
end;
end;
end;
```



```
Procedure PRELOAD;forward;
procedure preload;
(* Load command available through this procedure,
so its named as PRELOAD.
Please notice that initialization procedure is
being called here. *)
begin
  If arg1='' then arg1:=tabname ;
  If length(arg1)>8 then tabname:=upper(substr(arg1,1,8))
  else tabname:=upper(arg1);
  If length(arg2)>8 then rpwstr:=upper(substr(arg2,1,8))
  else rpwstr:=upper(arg2);
  If length(arg3)>8 then wpwstr:=upper(substr(arg3,1,8))
  else wpwstr:=upper(arg3);

  CMS('ESTATE '|| arg1 ||' CTRDATA *',Retc);
  If Retc<>0 then
    begin
      Writeln(arg1,' not found. ');
      return;
    end
  else
    begin
      DEFFILE;
      RINIT('ALL');
      end;
end;
```



```

Procedure RDROP;forward;
procedure rdrop;
(* Removes the data and control files for
current table from disk, if the command
issued by table owner. SERVVM issues CMS
ERASE command when it sees DROP command in
the xafile. *)
begin
If newtflag then
begin
writeln('Table cannot be dropped.Use QUIT');
return;
end;
If rwflag then
begin
dropflag:=TRUE;
end
else
begin
If tabdir.owner~=self then
begin
writeln('You are not the owner of:',tabname);
return;
end;
end;
writeln(tabname,' dropped. ');
writeln('****WARNING: Changes will be ignored from now. ');
xavisa:=TRUE;
end;

Procedure NOP;forward;
procedure nop;
begin
end;

```



```
(*----- Main Routine -----*)
begin
  Termin(input) ; Termout(output);
  parmline:=parms;CMS('IDENTIFY ( STACK LIFO ',Retc) ;
  readln(self) ; writeln('User:', self);
  servflag:=FALSE;
  READDEF;
  tokn1:=0;
  token(tokn1,parms,tabnalp);
  tabname:=SUBSTR(STR(tabnalp),1,8);
  token(tokn1,parms,rpwalp);
  rpwstr:=SUBSTR(STR(rpwalp),1,8);
  token(tokn1,parms,wpwalp);
  wpwstr:=SUBSTR(STR(wpwalp),1,8);
  tabname:=upper(tabname);
  wpwstr:=upper(wpwstr);rpwstr:=upper(rpwstr);
  if tabname=' ' then tabname:='TDB';
  DEFFILE;
  RINIT('ALL') ;
  ALLHEAD;
  WRDELIM ;
  cmdstr:=' ' ;
  WHILE cmdstr<='QUIT' do
    begin
      readln(cmdline) ;
      WRDELIM ;
      xavisa:=FALSE ;
      If cmdline<=' ' then
        begin
          tokn1:=0 ;
          token(tokn1,cmdline,cmd1) ;cmdstr:=STR(cmd1);
          token(tokn1,cmdline,parm1) ;arg1:=STR(parm1) ;
          token(tokn1,cmdline,parm2) ;arg2:=STR(parm2) ;
          token(tokn1,cmdline,parm3) ;arg3:=STR(parm3) ;
          token(tokn1,cmdline,parm4) ;arg4:=STR(parm4) ;
          token(tokn1,cmdline,parm5) ;arg5:=STR(parm5) ;
          token(tokn1,cmdline,parm6) ;arg6:=STR(parm6) ;
          token(tokn1,cmdline,parm7) ;arg7:=STR(parm7) ;
          token(tokn1,cmdline,parm8) ;arg8:=STR(parm8) ;
          token(tokn1,cmdline,parm9) ;arg9:=STR(parm9) ;
          token(tokn1,cmdline,parm10) ;arg10:=STR(parm10) ;
          cmdstr:=UPPER(cmdstr);
```



```

IF cmdstr='QUIT' then
  begin
    IF cmdstr='ADD' Then RADD
    else
    IF cmdstr='DEL' Then RDEL
    else
    IF cmdstr='DELETE' Then RDEL
    else
    IF cmdstr='HELP' then RHELP
    else
    IF cmdstr='LIST' then RLIST
    else
    IF cmdstr='LI' then RLIST
    else
    IF cmdstr='L' then RLIST
    else
    IF cmdstr='SAVE' then RSAVE
    else
    IF cmdstr='LOAD' then PRELOAD
    else
    IF cmdstr='READ' then RREAD
    else
    IF cmdstr='CMS' then RCMS
    else
    IF cmdstr='CHA' then RCHA
    else
    IF cmdstr='CHANGE' then RCHA
    else
    IF cmdstr='CONTENT' then CONTENT
    else
    IF cmdstr='DROP' then RDROP
    else
    IF cmdstr='*' then NOP
    else
    IF cmdstr='SET' then RSET
    else writeln('ignored')
    end;(*quit*)
  RADDXA ;
  end
else (* WRDELIM *) ;
end;
end. (*endall*)

```


EK2 EXEC'LER

HUNTREQ

```
/*Generated on 16/3/90 */
TRACE 0 ;
ADDRESS COMMAND;

'NUCXLOAD IUCVTRAP ( SYSTEM SERVICE'
'IUCVTRAP OFF'
'IUCVTRAP ON'
'SET CMSTYPE RT'
'CP SET SMSG IUCV'
'CP SET CPCONIO IUCV'
/*'CP DISC'*/

DO FOREVER
'EXECIO 2 CP ( STRING QUERY READER ALL'
pull title_line
If queued()>0 then
do
pull orig . . clas . . . . time .
'EXEC POPTDB' time orig clas
If RC=0 then call exit
end
else leave
END

DO FOREVER
IF QUEUED() = 0 THEN
DO
PULL . TIME . . USER . CMDL
'EXEC POPTDB' time user cmdl
If RC=0 then call EXIT
END;
ELSE 'IUCVTRAP GRAB (WAIT' ;
END;

EXIT:
'CP SPOOL CONSOLE TERM'
'IUCVTRAP OFF'
EXIT
```


POPTDB EXEC

```

/*Generated on 16/1/90 */
TRACE 0 ;
/* -----Defaults ----- */
privusers_file='POPTDB PRIVUSER *' ;
cmdusers_file='POPTDB CMDUSERS *' ;
users_file='POPTDB USERS *' ;
fileexec='TDBJOBS EXEC *' ;
cunsult='RAHMI'
/* -----*/
ADDRESS COMMAND;
arg TIME USER CMDL
'IDENTIFY ( STACK'
pull me .
cmd1=subword(cmdl,1,1) ;
/*If any file spooled someone by the servvm,
none of exec should be called and this exec
must be exited quickly*/
route_key=subword(cmdl,4,1);
If route_key='TO' then call exit 0
'SCANFILE' users_file USER '( NOTYPE'
If RC=0 then call noserv
SELECT ;
WHEN CMD1='PUN' THEN CALL RDRJOB CMDL
WHEN CMD1='RDR' THEN CALL RDRJOB CMDL
WHEN CMD1='PRT' THEN
CALL ERROR 'PRT file cannot be processed'
WHEN CMD1='CMD' THEN CALL CMD_HANDLE CMDL
WHEN CMD1='CMS' THEN CALL CMS_COMMAND
WHEN CMD1='CP' THEN CALL CP_COMMAND
WHEN CMD1='' THEN NOP
OTHERWISE CALL UNKNOWN
END
CALL EXIT 0
RDRJOB:
CALL INFORM RAHMI CMDL 'is acceptable.'
'EXEC' fileexec CMDL
RETURN
ERROR:
parse arg RetC errline
CALL INFORM user errline 'RC=' RetC
RETURN
UNKNOWN:
If user = me then
CALL INFORM user 'Invalid request.'
RETURN;
NOSERV:
'EXEC TELL' USER "Request ignored."
RETURN

```



```

CMD_HANDLE:
  CMD2=SUBWORD(CMDL,2,1)
  'SCANFILE' privusers_file  USER '( NOTYPE'
  IF RC = 0 Then
    do
      select;
        WHEN CMD2='END' THEN
          CALL EXIT 100 'POPTDB Normally ended.)
        WHEN CMD2='' THEN NOP
        OTHERWISE CALL INFORM USER 'Unknown JOB';
      end;
    end
    else CALL INFORM user 'You are not authorized
    to execute PRIV. commands.'
  RETURN;
CMS_COMMAND:
  'SCANFILE' cmdusers_file  USER '( NOTYPE'
  IF RC = 0 Then
    do
      ADDRESS CMS
      EXCMDL=DELWORD(CMDL,1,1);
      EXCMDL="" || EXCMDL || ""
      interpret EXCMDL;call inform USER RC
    end;
    else CALL INFORM user 'You are not authorized
    to execute CP/CMS commands'
  RETURN
CP_COMMAND:
  'SCANFILE' cmdusers_file  USER '( NOTYPE'
  IF RC = 0 Then
    do
      ADDRESS CP
      EXCMDL=DELWORD(CMDL,1,1);
      EXCMDL="" || EXCMDL || ""
      interpret EXCMDL; call inform USER RC
    end;
    else CALL INFORM user 'You are not authorized
    to execute CP/CMS commands'
  RETURN
INFORM:
  parse arg infovm infoline
  'EXEC TELL' infovm infoline
  return ;
EXIT:
  parse arg ret exitline
  If ret='' then ret=0;
  CALL INFORM user exitline
  EXIT(ret)
  RETURN

```


TDBJOBS EXEC

```
/* To run corresponding exec, specified by a
spooled file from user      R.T. 16/01/90 */
TRACE 0;
Arg argline
execlist='TDBJOBS EXECDATA *'
owner='RAHMI'

'VM DIAG LIFO QREADER'
If RC=0 then call exit RC 'ERROR encountered
from VM DIAG'
pull me origin spoolid crdate crtime
dist copies hold device class form fname ftype .

/*Here, if required, many of checkings can
be done on attributes of the spool file */
IF (origin='RSCSDMS' | origin='RSCS') then
ORIGIN='RAHMI' ;
/*ACCESS workdisk A ( ERASE */
IF hold='NONE' then call BADFILE
IF device='PUN' then call BADFILE
IF fname='' then call BADFILE
IF ftype='' then call BADFILE

'CP SPOOL CONSOLE PURGE'
'SCANFILE' execlist fname '( NOTYPE'
If RC=0 then call JOBERORR fname 'undefined JOB.'
'ESTATE' fname 'EXEC *'
If RC=0 then call JOBERORR fname 'has not been
implemented yet.'
'CP SPOOL READER HOLD'
'DISK LOAD'
If RC=0 then call PROBLEM 'Error during DISK LOAD.'
'CP SPOOL CONSOLE TO' origin 'PURGE'
'CP SPOOL PRINTER FOR' origin 'HOLD'
/* 'CP SPOOL PUNCH FOR' origin 'HOLD'*/
'CP SPOOL PUNCH FOR' origin
/*This line changed due to servvm could
not transfer the original xa to nextvm.*/
/* REstart on ensured from the PRIVUTIL */
'EXECIO 1 DISKR' fname ftype 'A ( FINIS'
pull operands
'EXEC' fname operands
CALL SENDCONSOLE
'CP SPOOL PRINTER CLOSE NOCONT'
'CP SPOOL PUNCH CLOSE NOCONT'
'CP PURGE READER' spoolid
CALL CLEANDIR
CALL EXIT RC
```



```
CLEANDIR:
'LISTFILE * * A ( E'
'EXEC CMS ERASE'
'ERASE CMS EXEC A'
RETURN
```

```
SENDCONSOLE:
'CP SPOOL CONSOLE TO' origin
'CP CLOSE CONSOLE NAME' fname 'CONSOLE'
'CP SPOOL CONSOLE TO' owner
RETURN;
```

```
PROBLEM:
arg argline
'CP MSG' origin argline
CALL SENDCONSOLE
CALL EXIT 100
```

```
BADFILE:
'CP MSG' origin spoolid 'is not acceptable.'
CALL PURGEFILE
CALL EXIT 200
```

```
JOBERROR:
parse arg errline
'CP MSG' origin errline
CALL PURGEFILE
CALL EXIT 300
```

```
PURGEFILE:
'CP PURGE READER' spoolid
RETURN
```

```
EXIT:
arg retc
EXIT(retc)
```


TDB EXEC

```
/* To handle tdb xas. */
TRACE ALL;
arg none .
ADDRESS COMMAND ;
'EXECIO 1 DISKR TDB CMDS * ( FINIS'
  pull . entry_args
'EXEC GU DBSSERV3 194 MR ( STACK'
  pull . acmode .
'EXEC SWAP A' acmode
PUSH 'set echo on' ;
PUSH 'set xamode off' ;
PUSH 'read TDB CMDS' acmode ;
'TDB3' entry_args
REtc=RC
'EXEC TELL RAHMI end of xa. with RC=' REtc
'EXEC SWAP A' acmode
'EXEC DT' acmode
EXIT(REtc);
```


SONUÇ

Dikkat edilirse, TDB olarak adlandırılan programın, bir database management sistemi haline getirildiği, yani belirli sınırlar içinde, computer üzerinde data organizasyonu sağladığı görülür. Bu noktada işaret edilmesi gereken nokta, bu programın tam bir DBMS olmadığıdır. Daha önce de belirtildiği gibi amaç, çok kopyalı, güvenilir, ve birden fazla birim üzerinde ortak olarak kullanılabilen bir DBMS'i incelemek ve, gerçekleştirmektir.

TDB, daha da geliştirilerek, standart DBMS'lere daha da yakınlaşması sağlanabilir. Ancak bu, belli bir gelişim süreci içinde yapılmak zorundadır.

TDB, Netaş AŞ. mainframe'i (IBM 3081) üzerinde gerçekleştirilmiş, ve aktif olarak Enformatik bir amaç için kullanılmaya başlanmıştır. Kısaca, Londra, Ottawa ve İstanbul'da bulunan departmanlar arasında teknik konulardaki takvimleme ve ortaklaşa çalışma zamanlamaları (schedule) konusunda kullanılmaktadır. Genel amaçlı olduğundan ileride ortaya çıkacak yeni data base gereksinmelerine de cevap verebilecek durumdadır.

KAYNAKLAR

1. Introduction to Data Base System.
C.J.Date
2. Distributed Computing Systems.
CORNOFION
3. Multisite Data Base Concept.
BNR(Bell Northern Research)
4. Data distribution & Transaction Processing
Technics.
BNR
5. Transaction Handling on Large Data Base Systems.
BNR
6. Principles of Data Base Management.
James Martin

ÖZGEÇMİŞ:

Yazar, 1966 yılında Konya'da doğdu. İlk ve orta tahsilini Konya'da tamamladı. 1987 yılında Yıldız Üniversitesi Elektronik ve Haberleşme bölümünden mühendis olarak mezun oldu. Aynı yıl, Netaş AŞ.'de Araştırma Geliştirme Destek Mühendisi olarak çalışmaya başladı.

1987-1988 döneminde Yıldız Üniversitesi Bilgisayar Bilimleri Mühendisliği bölümünde Yüksek Lisans öğrenimine başladı.



