

39822

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

39822

VERİTABANI SİSTEMLERİNDE SORGULAMA OPTİMİZASYONU VE

SQL STANDARDLARI

YÜKSEK LİSANS TEZİ

Mat. Müh. Güler ÇELİK

Tezin Enstitüye Verildiği Tarih : 24 Ocak 1994

Tezin Savunulduğu Tarih : 8 Şubat 1994

Tez Danışmanı : Prof. Dr. Mithat Uysal

Diğer Jüri Üyeleri : Prof. Dr. Galip Tepehan
Doç. Dr. Gazanfer Ünal

ŞUBAT 1994

ÖNSÖZ

Bu tezin hazırlanmasında bana yön veren tez danışmanım Sayın Prof. Dr. Mithat Uysal'a katkılarından dolayı teşekkür ederim. Aynı zamanda öğrenim hayatım boyunca hiçbir fedakarlıktan kaçınmayan aileme en içten şükranlarımı sunarım. Ayrıca bu tezi gerçekleştirmek için kullandığım tüm olanakları sağlayan RENAULT MAİS A.Ş.' ye ve tezin yazım ve basımında bana destek olan arkadaşlarıma teşekkür ederim.

İSTANBUL, 1994

Güler ÇELİK

İÇİNDEKİLER

ŞEKİL LİSTESİ	v
ÖZET	vi
SUMMARY	vii
BÖLÜM 1. İNCELENEN ÖRNEK SİSTEM	1
1.1 Giriş	1
1.2 Sistem Tanımı	2
1.3 Gereksinim Formülasyonu	3
1.4 Mantıksal Tasarım	8
1.5 Yeniden Düzenleme	14
BÖLÜM 2. SQL NEDİR ?	22
2.1 Neden Bir Standart ?	23
2.2 Bazı Temel Kavramlar	25
BÖLÜM 3. TEMEL DİL ELEMANLARI	34
3.1 Karakterler	34
3.2 Simgeler.....	34
3.3 Tanımlayıcılar	35
3.4 İsimler	36
3.5 Sözcükler	39
3.6 Değerler ve Hedefler	39
3.7 Data Tipleri	40
BÖLÜM 4. VERİ TANIMLAMA	52
4.1 Create Shema	52
4.2 Create Table	54
4.3 Create Domain	70
4.4 Create View	72
4.5 Create Assertion	73
4.6 Create Character Set	75
4.7 Create Collation	75
4.8 Create Translation	77
BÖLÜM 5. ŞEMA İŞLEMLERİ	78
5.1 Drop Komutları	78
5.2 Alter Komutları	85
BÖLÜM 6. İMLEÇSİZ İŞLEMLER	92
6.1 Insert	92
6.2 Tek Satır Seçimi 'Select'	94
6.3 Update	95

6.4 Delete	97
6.5 Veri Atama Kuralları	98
BÖLÜM 7. İMLEÇ İSİMLERİ	101
7.1 Declare Cursor	103
7.2 Open	106
7.3 Fetch	107
7.4 Konumlanmış Update	109
7.5 Konumlanmış Delete	111
7.6 Close	112
BÖLÜM 8. ERİŞİM KONTROL	113
8.1 Grant	115
8.2 Revoke	116
8.3 Hakların Kontrolü	118
BÖLÜM 9. TÜREMİŞ TABLOLAR	120
9.1 Satır Değeri Yapıcıları	120
9.2 Tablo Değeri Yapıcısı	121
9.3 Sorgu İfadesi	121
9.4 Tablo İfadesi	137
9.5 Sorgu Özelliği	141
9.6 Alt Sorgular	143
BÖLÜM 10. ÖNERMELER VE ARAMA ŞARTLARI	145
10.1 Arama Şartları	145
10.2 Önergeler	147
BÖLÜM 11. FONKSİYONLAR VE İFADELER	155
11.1 Değer İfadesi	155
11.2 Skaler Fonksiyonlar	159
11.3 Şartlı Değer İfadesi	164
11.4 Küme Fonksiyonları	166
BÖLÜM 12. OTURUM YÖNETİMİ	168
12.1 Connect	169
12.2 Set Connection	169
12.3 Disconnect	170
BÖLÜM 13. HATA BULMA VE DÜZELTME	171
13.1 Get Diagnostics	171
SONUÇLAR	172
KAYNAKLAR	174
EKLER	175
ÖZGEÇMİŞ	188

ŞEKİL LİSTESİ

Şekil 1.1 Mantıksal Tasarım (1 .Sistem)	8
Şekil 1.2 Mantıksal Kayıt Tasarımı (1. Sistem).....	9
Şekil 1.3 Mantıksal Kayıt Erişim Sayıları ve Taşıma Hacmi (1. Sistem)....	14
Şekil 1.4 Mantıksal Tasarım (2 .Sistem)	16
Şekil 1.5 Mantıksal Kayıt Tasarımı (2. Sistem).....	17
Şekil 1.6 Mantıksal Kayıt Erişim Sayıları ve Taşıma Hacmi (2. Sistem).....	21
Şekil 2.1 Örnek Tablo.....	29
Şekil 2.2 Katalog Veri Tabanı.....	32
Şekil 2.3 Katalog Veri Tabanında Şema Tanımı	32
Şekil 4.1 Bölüm Tablosu.....	56
Şekil 7.1 Örnek İmleç.....	110
Şekil 7.2 Örnek İmleçde Konumlanmış Update	110
Şekil 7.3 Örnek İmleç.....	111
Şekil 7.4 Örnek İmleçde Konumlanmış Delete	112
Şekil 8.1 Şema Nesnelerini Oluşturmak İçin Gerekli Haklar	117
Şekil 10.1 OR Doğruluk Tablosu.....	146
Şekil 10.2 AND Doğruluk Tablosu.....	146
Şekil 10.3 NOT Doğruluk Tablosu.....	146
Şekil 10.4 IS Doğruluk Tablosu.....	146
Şekil 10.5 NULL Doğruluk Tablosu.....	150
Şekil 11.1 Geçerli Veri Tipi Dönüşümü	164

ÖZET

Bu çalışmada sorgulama optimizasyonu ve 1987 yılından itibaren standart bir dil olarak tanımlanan SQL (Structured Query Language) sorgulama dilinin, sorgulama optimizasyonundaki özel faydaları dilin özellikleri ile birlikte incelenmiştir.

SQL, ilk kez 1987 yılında International Standard Organization (ISO) tarafından standard olarak tanımlanmıştır. Bu çalışmada, SQL dili incelenirken temel alınan standardın versiyonu 1992 tarihli uluslararası standarttır (ISO 1992). SQL dilinin neden önemli olduğu, veri tabanı dünyasında ne denli ciddiye alınması gerektiği ve neden standart bir dil olarak tanımlandığı yine bu çalışmanın kapsamında incelenmiştir. SQL dilinin özellikleri vurgulanırken, dilin özelliklerinin daha iyi kavranması amacıyla örneklere yer verilmiştir.

Bu çalışma kapsamında örnek bir sistem seçilmiş ve bu sistemin veri tabanı tasarımı yapılmıştır. Veri tabanı tasarımında, gerçek hayatta uygulanagelen yöntemlere yakın bir yöntem kullanılmıştır. Bu yöntemde veritabanı yaşam süreci olarak kabul edilmekte ve veri ortak bir kaynak olarak görülmektedir. Yöntem gereksinim formülasyonu, mantıksal tasarım ve gerçekleştirme gibi adımlardan sonra veri tabanının bilgi işlem gereksinimleri altında verimliliğini hedefleyen bir yöntemdir. Mantıksal kayıt erişim sayıları ve taşıma hacmi değerleri yaklaşımı ile yapılan tasarımın performansı incelenir ve elde edilen bu değerleri en aza indirerek performansı artıracak yeni bir tasarım daha yapılır.

Bu tez kapsamında incelenen sistem, otomobil imalat ve satışını yapan firmaya bağlı yetkili servis ve bakım istasyonundaki onarım ve yedek parça stok hareketlerinin otomasyonunu amaçlamaktadır. Bu sistemin, bahsedilen yöntem ile veri tabanının tasarımı yapılmış ve bu elde edilen tasarım yeniden düzenlenmiştir. Yapılan her iki veri tabanı tasarımı bilgisayar ortamında denenmiş ve sonuçlar karşılaştırılmıştır.

QUERY OPTIMIZATION IN DATABASE SYSTEMS AND SQL STANDARDS

S U M M A R Y

The structure of the database management systems (DBMS) are composed of two basic parts: The data definition part and data manipulation part. Data definition refers to how the structure of the database is communicated to the DBMS.

The overall structure of the database describes all the records, all the data items, all the files, all the sets, used in the database. This is usually specified through some type of special language called the data definition language (DDL). Data manipulation refers to methods used to retrieve, add, modify, and delete data in a database. There are four basic methods of data manipulation in databases: programming language interface, query, report writers, and system utilities. Query is used to inquire into database.

Selecting the best possible solution to solving a query is called optimization. Actually, the system might not recognize all possible solutions to the query solution or, it might take too long to evaluate all possible solutions, so the system might just try to improve on the current solution.

There are two factors to consider when attempting to optimize a query: time and space. Time refers to the time that it takes to carry out the operations to solve the query. Space refers to how much space is needed to construct the intermediate tables to solve the query.

In general, the JOIN-type queries are the ones that the system has to try its best to optimize, for they are the most time consuming to execute and generate the most space for resultant tables. This means that JOIN's should be placed as late as possible in the execute queue. However, it must be the job of the relational DBMS to optimize queries, and this should be transparent. From the programmer's and the

system's query optimization point of view, it is generally considered better to write a query with as little nesting and implied ordering as possible.

Query languages are stand-alone languages that allow rapid access to the database. These languages do not have to be compiled/translated or linked before they are used. They are designed primarily for retrieval of data, although data maintenance use is usually allowed. Query languages are two category. The first category is called command-oriented-query languages. This category encompasses all query languages in which the commands are specified in English-like text. The second category is called screen-oriented languages. In this category, the user enters commands through a fill-in-the-blank mechanism.

This thesis presents the importance of the SQL in the query optimization. SQL is currently without rival as a standard database language. Every major supplier currently provides database products that either based on SQL or have an SQL interface. Besides being a standard in its own right, SQL also provides an important tool for defining other key standards. Most important research on database technology has been centred on SQL since the mid seventies in both academic community and industry research labs. In that context, it has been a focal point for investigating such crucial technological issues as query optimization, data distribution and distributed processing.

SQL, whose name is derived from Structured Query Language, is in the command-oriented-query languages category. SQL is a relational database language. SQL is a comprehensive database language; it has statements for data definition, query, and update. Hence, it is both DDL and DML. In addition, it has facilities for defining views on the database, for creating and dropping indexes on the files that represent relations, and for embedding SQL statements into a general-purpose programming language such as PL/1, COBOL or PASCAL. Each of this is discussed on this thesis.

SQL as such is a database sub-language, which means that it is not a complete language in its own right since it contains no control commands. SQL provides definitional language facilities for data definition, for integrity constraint definition and, to some extent, for access control definition. For data manipulation, it provides a module language that consists of procedures containing statements that can retrieve, store and modify various ways.

SQL is not a database management system, nor is it a stand-alone product. SQL is an integral part of database management system, a language and a

tool for communicating with the database management system. SQL links the components of the typical DBMS, such as forms facility, report writer, interactive query facility, together.

The database engine is the heart of the DBMS, responsible for actually structring, storing, and recieving the data on the disk. It accepts requests from other DBMS components, from user-written application programs, and even from other computer systems. SQL plays many different roles:

SQL is an interactive query language. Users type SQL commands into an interactive SQL program to retrieve data and display it on the screen. In addition to this, SQL is a database programming language. Programmers embed SQL commands into their application programs to acces the data in a database. The database administrator responsaible for managing a minicomputer or mainframe database uses SQL to define the database structure and control acces to the stored data. Personal computer programs use SQL to communicate over a local area network with database servers that stor shared data. Many new applications are using this client/server architecture, which minimizes network traffic and allows both PCs and servers to do what they do best.

SQL is also a distributed database language. Distributed database management systems use SQL to help distribute data across many connected computer systems. The DBMS software on each system uses SQL to communicate with the other systems, sending requests for data access. Finaly, SQL is a database gateway language. In a computer network with a mix of differnt DBMS products, SQL is often used in a gateway that allows one brand of DBMS to communicate with another brand.

SQL is an important language as it is the first and so far only standard database language to gain wide acceptance. The SQL standard language is used in other standards or devoloping standards as a definitional tool. The development of the language is supported by considerable academic interest, providing both a theoretical basis for the language and the techniques needed the implement it succesfully. This is true in the areas of query optimization, distrubition of data, privacy and security.

This thesis includes a sample database design. Although the methodology emphasizes the logical design step, the activities of requirements analysis and physical design are also addressed. The methos accepts database life

cycle concept. The design steps are requirement formulation and analysis taking place. In the logical design step address the design and refinement of an information structure through the consolidation of the user information requirements specifications.

In order to measure of project the performance of the design logical record access approach is used. Logical record access counts are made to estimate the expected number of occurrences of each record type retrieved explicitly in the course of executing an application. When weighted by processing frequencies, they show which application will probably require the most I/O accesses to and from the database.

If logical record access counts can be specified for each application by record type, it is easy to use their results to calculate transport volume, or the total amount of the data flow between the application programs and the database management system.

The transport volume due to one record type in one application is just the product of that record type's size and its number of logical record accesses in that application. This may be summed over all record types to calculate the transport volume in bytes for the application. Finally the various application frequencies and summed to obtain the global transport volume.

By noting which applications are dominant in terms of frequency of execution, logical record accesses per unit time and transport volume, one can determine where design improvements will be most effective.

Using this database design method, a sample database designed in this thesis. The system, aims to operate maintenance, restoration of automobile and automation of spare part and stock in the authorized service and repairs stations of a automobile company.

Service objects to use computer in each step of repairing of a automobile and in the automation of reserve piece and stock. For this reason, a data base is needed to bring into existence.

First a data base design had done and implemented, then examined in computer environment, for data base performance. The IBM AS/400 consists of a environment is used for this study. Then logical record access counts and transport volumes calculated. Improvements had done on the design and new design re-implemented and examined. The comparison charts and SQL programs for both systems are given in appendixes.

The main objective of this study is to optimize queries in database systems and express the advantages of using SQL standards in query optimization.



BÖLÜM 1. İNCELENEN ÖRNEK SİSTEM

1.1 Giriş

Bu bölümde, örnek bir sistem ele alınmış ve bu sistemin veri tabanı tasarımı yapılmıştır. Veri tabanı tasarımı adımları, gerçekleştirme ve tekrar düzenlemeye dayalı bir metodoloji incelenmiştir. Bu metodoloji veritabanı yaşam çevrimi olarak nitelenmiştir. Öncelikle veri tabanı tasarımı sırasında işlemden bağımsız gereksinimlerin belirlenmesi gereklidir. Bu şekilde yapılan bir tasarım sonucunda veri tabanı değişken işlem gereksinimlerine uyartılabilir olacaktır.

İşlem gereksinimleri üç bileşenden oluşmaktadır. Bunlar her uygulama için gerekli özel veri gereksinimleri, veri hacmi ve uygulamanın birim zamanda kullanılma sayısı cinsinden işlem frekansıdır. Performans gereklilikleri de unutulmamalıdır. Veri tabanı tasarım işlemi sonucunda, mantıksal veri tabanı yapısı, fiziksel depolama yapısı ve uygulama programları karakteristikleri oluşmaktadır.

Gereksinim formülasyonu aşamasında, organizasyonun hedefleri, amaçları, kişilerin veritabanından beklentileri ortaya çıkarılır ve bu bilgiler ile gerekli verilerin tanımı yapılır.

Mantıksal tasarım aşamasında, kullanıcı gereksinimleri ilk veritabanı tasarımını gerçekleştirmek üzere birleştirilir. Bunun için genelde varlık diyagramları kullanılmaktadır. Sonuç olarak ilk bilgi yapısı oluşturulmaya başlanır.

Artık elde edilen mantıksal veri tabanı yapısı sayısal bilgilere ve performans kısıtlarına göre incelenebilir haldedir. İşlem hacmi, işlem frekansı ve veri hacmi cinsinden belirlenir. İşlem frekansı, her işlemin seçilen bir birim zamanda kaç kez kullanılacağıdır. Veri hacmi ise oluşturulan kayıt tiplerinden kaç kaydın veri tabanı sistemi içinde bulunacağıdır.

Performans ölçümleri ise mantıksal kayıt erişim sayıları ve herhangi bir işlemi gerçekleştirmek için transfer edilmesi gerekli toplam veri miktarı ve veri tabanı içerisinde bulundurulacak veri miktarı değerleriyle kısıtlıdır. Mantıksal kayıt erişim sayıları, herhangi bir işlemin çalışması sırasında tanımlı her kayıt tipinden kullanılacak kayıtların sayısıdır. Buna göre taşıma hacmi, herbir kaydın uzunluğu ile o kayıt tipine ait o işlem için gerekli mantıksal kayıt erişim sayısının çarpımına eşittir. Buna göre de işlem için gerekli tüm kayıt tipleri için bu iş tekrarlanarak işlemin taşıma hacmi elde edilir. Taşıma hacimleri veri tabanı yapısında düzenleme yapılırken baz alınmalıdır. Mantıksal kayıt erişim sayılarının ve taşıma hacimlerinin en aza indirilmesi ile veri tabanı tasarımı yeniden düzenlenir.

Elde edilen ve mevcut yapı bilgisayar ortamında gerçekleştirilmiş ve işlemlerin yeni yapı üzerindeki işlem süreleri incelenmiştir. Bilgisayar ortamında denenen işlemlerin sonuçları ve denenen her iki yapının karşılaştırılması eklerde bulunmaktadır.

1.2 Sistem Tanımı

İncelenen sistem, otomobil imalat ve satışını yapan firmaya bağlı yetkili servis ve bakım istasyonundaki onarım işlemlerinin ve yedek parça stok hareketlerinin otomasyonunu amaçlamaktadır. Servis tüm aşamalarda bilgisayar kullanmayı amaçlamakta ve bu amaçla bir veri tabanı oluşturulmasını istemektedir.

1.3 Gereksinim Formülasyonu

1.3.1 Bilgi gereksinimleri

Servis ve bakım istasyonunda ana firmanın sattığı araçların bakımı ve onarımı gerçekleşmekte bunun yanısıra yedek parça perakende veya toptan yedek parça satışı yapmaktadır.

Servis ve bakım istasyonuna gelen araç için öncelikle müşterinin şikayetleri doğrultusunda arıza tespiti yapılmakta ve araç servise kabul edilmektedir. Servise giren araca gerekirse yedek parça takılmakta ve yapılan işe göre işçilik yapılmaktadır. Araç için yapılan işçilik mekanik, kaporta, boya ve elektrik olmak üzere 4 ana gruba ayrılmaktadır. Bu gruplara göre işçiliğin saat ücreti değişmektedir. Ayrıca ana firmanın ürettiği değişik tipte araçlar vardır ve bu araç tiplerine göre de işçilik kodları ve işçilik süresi değişmektedir.

Servis stoğundaki yedek parçaları ana firmadan veya yan sanayi kuruluşlarından temin etmektedir. Ana firma, servis ve bakım istasyonun gerek duyduğu işçilik kodlarının bulunduğu tamir kataloğunu ve yedek parça kataloğunu servise vermektedir.

Servis işlemlerinde ise araç servise girdiği anda o araca ve müşteriye ait bilgileri içeren bir onarım emri açılmaktadır. Onarım emri açılan araç servise kabul edildikten sonra arıza durumuna göre kaporta, mekanik, boya ve elektrik arızalarına göre servisin gerekli bölümlerinde onarıma alınır ve gerekirse parça takılır. Takılan parçalar için transfer fişi kesilerek stoktan parça çıkışı yapılmaktadır. Bu arada arızaya ilişkin bilgiler de saklanmaktadır. Onarımı tamamlanan araca takılan yedek

parçalar ve yapılan işçilikler fatura edilir. Aracın servise bir kez daha gelmesi durumunda araca ve müşteriye ilişkin bilgiler, takılan yedek parçalar ve yapılan işçilikler hatta müşteri şikayetleri saklanacak veriler arasındadır.

Servis ve bakım istasyonu aynı zamanda perakende ve toptan yedek parça satışı yapmaktadır. Bu durumda da müşterilere fatura kesmektedir.

Ana firma ayda bir yedek parçalara zam yapmaktadır. Yedek parça fiyat değişikliklerini yetkili servis ve bakım istasyonlarına yedek parça kataloğunu göndererek iletir. İşçilik saat ücretlerindeki değişiklikler için bir sirküler yayımlar ve bunun tamir kataloğuna güncellenmesi gerekmektedir.

Servis ve bakım istasyonuna ilişkin bazı sayısal bilgiler de şu şekildedir:

- Servis günde yaklaşık 30 onarım emri açmaktadır. Servise giren araçların yaklaşık 25 tanesi servise ilk kez gelmekte doğal olarak serviste müşteri ve araca ilişkin bilgiler bulunmamakta bu bilgilerin girilmesi gerekmektedir.

- Bir onarım için yaklaşık iki transfer fişi kesilmektedir ve bir transfer fişi ile yaklaşık iki farklı yedek parça çıkışı yapılmaktadır.

- Servise giren araç için bir onarımda yaklaşık 3 farklı işçilik olmaktadır.

- Ana firma yedek parça kataloğunda 5000 adet yedek parça mevcuttur. Servis stoğunda ise yaklaşık 2000 adet ana firmadan temin ettiği, 1500 adet yan sanayi kuruluşlarından temin ettiği yedek parça mevcuttur.

- Servis ve bakım istasyonu bir ayda yaklaşık 10 adet ana firma kataloğunda bulunan bir başka deyişle ana firmadan, 5 adet yan sanayiden yeni yedek parça tanımlar.

- Ayda bir kez ana firma yedek parça fiyatlarına zam yapmaktadır.
- 2 ayda bir kez tamir kataloğunda fiyat değişikliği olmaktadır.
- Günde yaklaşık 25 adet perakende veya toptan fatura kesilmektedir bir fatura ile de yaklaşık üç adet değişik yedek parça çıkışı olmaktadır.

1.3.2 İşlem Gereksinimleri

Bu bölümde servis ve bakım istasyonunun işlem gereksinimleri incelenmektedir. Aşağıdaki başlıklarda servisin işleme gereksinimleri için belirlenmiş hareketler listelenmiştir. Bu işlemlerin tanımlanmasın da öncelikle az öncelikli tanım hareketleri listelenmiştir. Daha sonra servisin onarım ve stok hareketleri ile ilgili hareketler sıralanmıştır. Daha sonraki hareketlerde ise fiyat değişiklikleri ile ilgili hareketler ve en sonunda servisin günlük ve aylık aldığı rapor örneklerine yer verilmiştir.

Bu hareketlere isimler verilmiş ve örnek olarak bilgisayar ortamında denenen veri tabanı yapısında bu hareket isimleri kullanılmıştır.

TT1: Ana firmadan temin edilen yedek parçaların tanımlanma hareketi

Yedek parçanın tipi, adı, satış fiyatı gibi kimlik bilgileri ana firma kataloğunda mevcut olduğundan bu yedek parçaları tanımlarken sadece yedek parçanın stoktaki yeri, miktarı ve maliyetinin girilmesi yeterlidir.

TT2: Yan sanayiden temin edilen parçaların tanımlanması hareketi.

Yedek parçanın numarası, parça adı, tipi, miktarı, maliyeti, satış fiyatı gibi kimlik bilgilerinin tanımlandığı harekettir. Bu hareket aynı zamanda yedek parça bilgilerini güncellemek içinde kullanılmaktadır.

TT3: Oto tiplerini tanımlama hareketi.

Her oto tipine ait oto katalog tipini ve mekanik, kaporta, elektrik ve boya işçilik saat ücretlerinin tanımını içeren harekettir. Bu hareket aynı zaman da oto tipine ait bilgileri güncelleştirmek için de kullanılır.

TS1: Müşteri tanımlanması hareketi.

Servise giren aracın sahibine ait bilgilerin sisteme girildiği harekettir. Müşterinin adı adresi gibi kimlik bilgilerinin yanısıra müşteriye uygulanacak işçilik ve parça indirim yüzdelerinin de girildiği harekettir.

TS2: Aracın tanımlanması hareketi.

Servise giren aracın kimlik bilgilerini içeren harekettir. Bu bilgiler aracın trafik plakası, oto tipi, üretim ve şasi numarası, aracın müşteriye satıldığı tarih ve aracın model yılı, aracın onarıma girdiği andaki km. bilgileridir.

TS3: Onarım emrinin açıldığı hareket.

Onarım emri açıldığında aracın servise girdiği tarih ve zaman, onarım emri numarası, aracın trafik plakası, müşteri numarası ve müşterinin şikayeti girilmektedir. Eğer araç servise daha evvel gelmişse veya müşterinin servis kayıtlarında girişi varsa bu bilgilerin girilmesine gerek yoktur.

TS4: Transfer fişinin kesildiği hareket.

Onarımdaki araca takılan parçaların stoktan çıkışını gösteren transfer fişinin kesildiği harekettir. Transfer fişi ile stoktan çıkışı yapılan yedek parçanın stok bilgileri güncellenir, stok hareketine ve onarım hareketine girişi yapılır.

TS5: İşçiliklerin girildiği hareket.

İşçilik kodları ve işçilik saat ücretleri aracın anafirma oto tip kataloğuna göre değişmektedir. Yapılan işçilikler ile onarım hareketine girişi yapılır.

TS6: Onarımı tamamlanmış araca fatura kesme hareketi.

Bu hareket vasıtasıyla onarımı tamamlanmış araca takılan yedek parça ve yapılan işçilikler müşteriye fatura edilir. Servis aynı zamanda cari hesap takibi yaptığından fatura kesilirken müşteri hareketi işlem görür.

TS7: Toptan veya perakende fatura kesme hareketi

Servis toptan veya perakende olarak yedek parça satışı yapmaktadır. Servis cari hesap takibi yaptığından cari hesap kaydı oluşturulur. Stok hareketi oluşturulurken stok miktarları servis stoğunda ilgili yedek parçalar için güncellenir.

TF1: Yedek parça fiyat değişikliği.

Ana firma tarafından yedek parçalara yapılan değişiklikler yedek parça kataloğu ile servis ve bakım istasyonlarına iletilmektedirler. Bu hareket vasıtasıyla fiyatlara yapılan bu değişiklikler servis stoğuna güncellenir.

TF2: Tamir kataloğu fiyat değişikliği hareketi.

Ana firma tarafından oto tiplerine ait mekanik, kaporta, boya ve elektrik işçilik saat ücretlerine yapılan değişiklikler güncellenir. Bundan sonra işçilik gruplarına göre işçilik sürelerine göre saat ücretleri ile doğru orantılı olarak tamir kataloğundaki fiyatların güncelendiği harekettir.

TR1: Günlük icmal raporu hareketi.

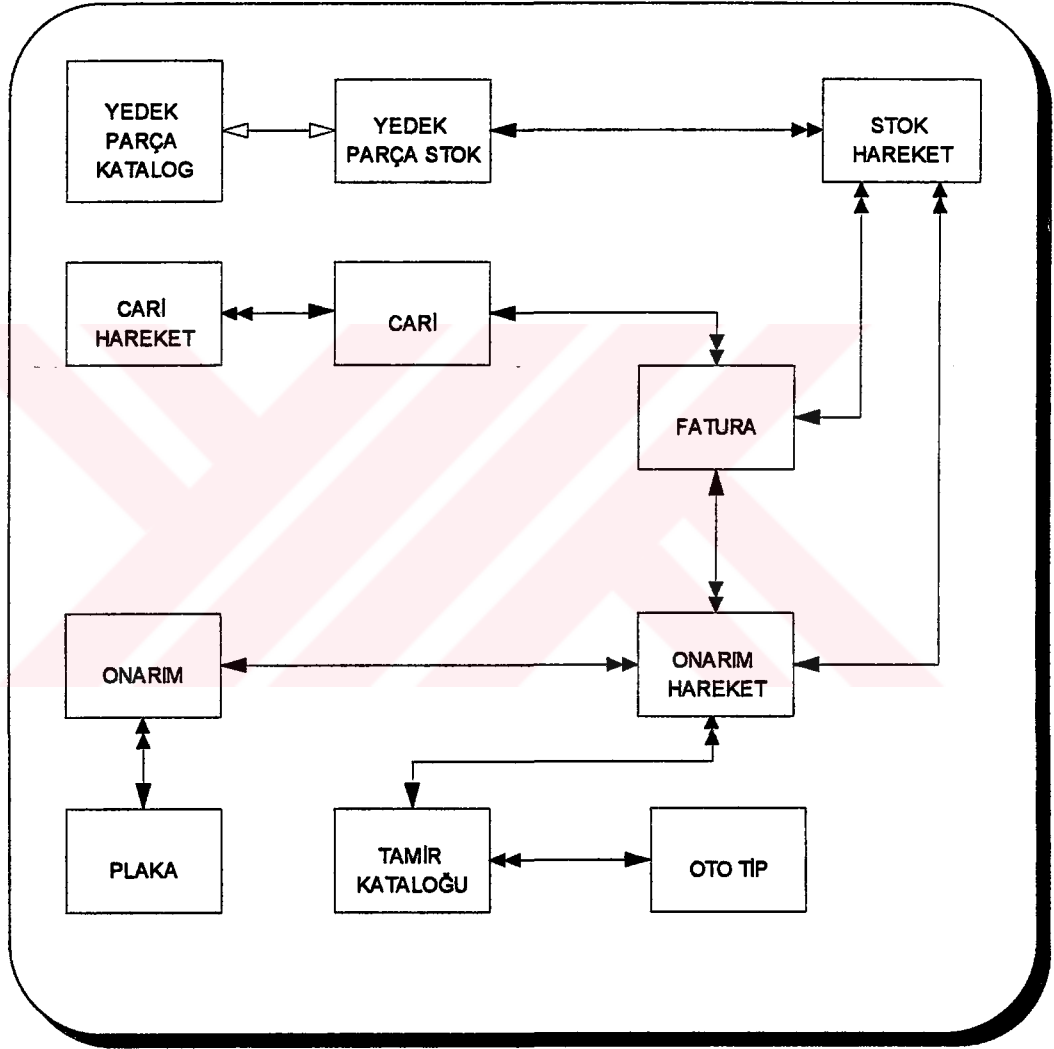
Serviste bir gün içinde kesilen faturaların listesini veren ve hergün mutlaka alınan bir rapordur.

TR2: Stok hareket raporu hareketi.

Servisin ay içindeki parça hareketlerinin listesini alındığı harekettir. Bu rapor sayesinde servisteki parça çıkışı kontrol edilmektedir.

1.4 Mantıksal Tasarım

Yukarıda belirtilen gereksinimleri karşılamak üzere tasarlanan bilgi yapısı aşağıdaki gibidir



Şekil 1.1 Mantıksal Tasarım (1. Sistem)

1.4.1 Depolama Alanı

Oluşturulan bu bilgi şemasına göre mantıksal kayıt tasarımı şu şekildedir:

Tablo Adı : YPK_STOK Yedek Parça Katalog			
Kol.No	Kolon Adı	Kolon Tipi	Uzunluk
1	YPK_PARNO	CHAR	10
2	YPK_PARAD	CHAR	20
3	YPK_PARGRB	CHAR	10
4	YPK_MLZTIP	CHAR	4
5	YPK_SATFYT	DECIMAL	13
Tablo Adı : YP_STOK Servis Stok			
1	YP_PARNO	CHAR	10
2	YP_PARAD	CHAR	20
3	YP_PARGRB	CHAR	10
4	YP_MLZTIP	CHAR	4
5	YP_SATFYT	DECIMAL	13
6	YP_YER0	CHAR	2
7	YP_YER1	CHAR	3
8	YP_YER2	CHAR	3
9	YP_SONFATNO	INTEGER	8
10	YP_SONBRMMLY	DECIMAL	11
11	YP_SONFATMIK	INTEGER	7
12	YP_YLBMLY	DECIMAL	11
13	YP_GIRMLY	DECIMAL	11
14	YP_CIKMLY	DECIMAL	11
15	YP_YLBMIK	INTEGER	7
16	YP_GIRMIK	INTEGER	7
17	YP_CIKMIK	INTEGER	7
Tablo Adı : STKHRK Stok Hareket			
1	HRKPAR	CHAR	10
2	HRKTRH	DATE	6
3	HRKKOD	DECIMAL	2
4	HRKCAR	DECIMAL	5
5	HRKFAT	INTEGER	8
8	HRKFIS	INTEGER	8
7	HRKSAT	DECIMAL	4
8	HRKMIK	INTEGER	7
9	HRKMMLY	DECIMAL	11

Şekil 1.2 Mantıksal Kayıt Tasarımı (1. Sistem)

10	HRKBSF	DECIMAL	11
Tablo Adı : ONARIM Onarım Bilgileri			
1	ONNO	DECIMAL	5
2	ONGIRNO	DECIMAL	4
3	ONTRH	DATE	6
4	ONPLK	CHAR	10
5	ONCAR	DECIMAL	5
6	ONPARSAT	DECIMAL	4
7	ONTOPSAT	DECIMAL	4
8	ONFATTRH	DATE	6
9	ONFATTUT	DECIMAL	11
10	ONFATIND	DECIMAL	11
11	ONPARTUT	DECIMAL	11
12	ONISCTUT	DECIMAL	11
Tablo Adı : ONRHRK Onarım Hareket			
1	ONRNO	DECIMAL	5
2	ONRPLK	CHAR	10
3	ONRTIP	CHAR	1
4	ONRFIS	INTEGER	8
5	ONRSAT	DECIMAL	4
6	ONRTRH	DATE	6
7	ONRTUT	DECIMAL	11
8	ONRMLY	DECIMAL	11
9	ONRPAR	CHAR	10
10	ONRADT	INTEGER	7
11	ONRZMN	DECIMAL	3.2
12	ONRACK	CHAR	20
Tablo Adı : PLAKA Araç Bilgileri			
1	PLKNO	CHAR	10
2	PLKOTOTIP	CHAR	15
3	PLKURET	DECIMAL	9
4	PLKSASI	DECIMAL	9
5	PLKSATTRH	DATE	6
6	PLKMODEL	DATE	4
7	PLKCAR	DECIMAL	5
8	PLKKM	INTEGER	6

Şekil 1.2 Mantıksal Kayıt Tasarımı (1. Sistem) (Devamı)

Tablo Adı : FATURA		Fatura Bilgileri	
1	FATCAR	DECIMAL	5
2	FATKOD	DECIMAL	2
3	FATNO	INTEGER	8
4	FATTRH	DATE	6
5	FATTAHD	CHAR	1
6	FATTUT	DECIMAL	12
7	FATIND	DECIMAL	11
8	FATKDV	DECIMAL	11
9	FATPARTUT	DECIMAL	11
10	FATISCTUT	DECIMAL	11
11	FATAACK	CHAR	20
Tablo Adı : OTOTIP		Oto Tipleri	
1	OTOTIP	CHAR	15
2	OTOADI	CHAR	25
3	OTOKATTIP	CHAR	15
4	ISSUCB	DECIMAL	11
5	ISSUCK	DECIMAL	11
6	ISSUCM	DECIMAL	11
7	ISSUCE	DECIMAL	11
Tablo Adı : TAMIRK		Tamir Kataloğu	
1	ISCOTOKATTIP	CHAR	15
2	ISCKOD	CHAR	4
3	ISCKODADI	CHAR	15
4	ISCFYT	DECIMAL	11
5	ISCZMN	DECIMAL	3.2
6	ISCTIP	CHAR	1
Tablo Adı : CARMAS		Müşteri Bilgileri	
1	CARNO	DECIMAL	5
2	CARAD	CHAR	30
3	CARADR	CHAR	30
4	CARTLF	CHAR	10
5	CARVD	CHAR	25
6	CARSIC	CHAR	10
7	CARBORTUT	DECIMAL	13
8	CARALATUT	DECIMAL	13

Şekil 1.2 Mantıksal Kayıt Tasarımı (1. Sistem) (Devamı)

9	CARBORDEV.	DECIMAL	13
10	CARALADEV	DECIMAL	13
11	CARPARIND	DECIMAL	2.2
12	CARISCIND	DECIMAL	2.2
Tablo Adı : CARHRK Müşteri Hareketleri			
1	CRXNO	DECIMAL	5
2	CRXTIP	CHAR	1
3	CRXFATNO	INTEGER	8
4	CRXTRH	DATE	6
5	CRXTUT	DECIMAL	13
6	CRXACK	CHAR	20

Şekil 1.2 Mantıksal Kayıt Tasarımı (1. Sistem) (Devamı)

Yapılan tasarıma için elde edilen kayıt yapılarına göre kayıt uzunlukları aşağıdaki gibidir;

Yedek Parça Katalog:	: 57
Yedek Parça Stok	:145
Stok Hareket	: 72
Plaka	: 64
Onarım	: 88
Onarım Hareket	: 99
Oto Tipleri	: 99
Tamir Kataloğu	: 52
Fatura	: 98
Müşteri	:172
Müşteri Hareket	: 53

Bu uzunluk değerleri ve bilgi gereksinimleri ile elde edilen bilgiler ışığında depolama alanını aşağıdaki şekilde belirleriz. Burada servis ve bakım istasyonunun 1 aylık bilgiyi güncel olarak saklayacağı varsayılmıştır.

Yedek Parça Katalog:	: 57*5000	= 278 Kb
Yedek Parça Stok	:145*3500	= 495 Kb
Stok Hareket	: 72*30 *(30*2*2 + 25* 3)	= 411 Kb
Plaka	: 64*25*30	= 46 Kb
Onarım	: 88*30*30	= 77Kb
Onarım Hareket	: 99*(30*2*2+30*3)	= 20Kb
Oto Tipleri	: 99*19	= 2Kb
Tamir Kataloğu	: 52*100	= 5Kb
Fatura	: 98*55	= 5Kb
Müşteri	:172*25*30	= 130Kb
Müşteri Hareket:	53*55*30	= 85Kb
Toplam 11 Tablo		Toplam 1554 Kb

1.4.2 Hareket Frekansları

Aşağıda incelenen işlemlerin frekansları listelenmiştir;

TT1:	Ayda 10 kere	TT2:	Ayda 5 kere
TT3:	Yılda 5 kere	TS1:	Günde 25 kere
TS2:	Günde 30 kere	TS3:	Günde 30 kere
TS4:	Günde 60 kere	TS5:	Günde 30 kere
TS6:	Günde 30 kere	TS7:	Günde 25 kere
TF1:	Ayda 1 kere	TF2:	2 Ayda 1 kere
TR1:	Günde 1 kere	TR2:	Ayda 1 kere

1.4.3 Mantıksal Kayıt Erişim Sayıları ve Taşıma Hacmi

İŞL	ERİŞİLEN KAYITLAR	MKE	MKE/G	FREK.	T.H.
TT1	1YP_STOK+1YPK_STOK	2	0.6	0.3	60.6
TT2	1YP_STOK	1	0.15	0.15	21.75
TT3	1OTOTIP	1	0.01	0.01	0.99
TS1	1CARMAS	1	25	25	4,300
TS2	1PLAKA	1	25	25	1,600
TS3	CARMAS+PLAKA+ONARIM +ONRHRK	4	30	120	12,690
TS4	ONARIM+2*(2YP_STOK+STKHRK+ ONRHRK)	9	60	540	60,600
TS5	ONARIM+PLAKA+OTOTIP +3*(TAMIRK+ONRHRK)	9	30	270	21,120
TS6	7ONRHRK+2CARMAS+1CARHRK+ 1ONARIM+1FATURA	12	30	360	38,280
TS7	2CARMAS+3*(STKHRK+2YP_STOK) +1FATURA+1CARHRK	13	25	325	39,525
TF1	2000*(YP_STOK+YPK_STOK)	4,000	0.03	120	12,120
TF2	1000*(OTOTIP+TAMIRK)	2,000	0.02	30	2,265
TR1	55*FATURA	55	1	55	5,390
TR2	30*(4*30+3*25)*STKHRK	5,850	0.03	175.5	12,630

Şekil 1.3 Mantıksal Kayıt Erişim Sayıları ve Taşıma Hacmi (1. Sistem)

1.5 Yeniden Düzenleme

Önceki tabloda görüldüğü gibi veri tabanı üzerinde çalışacak işlemlerin bazıları yüksek MKE sayılarına ve taşıma hacimlerine sahiptir. Bu tablo üzerinde çalışarak yüksek MKE sayılı ve taşıma hacimli işlemlerin bu değerlerini düşürerek yeni bir tasarım elde etmektir.

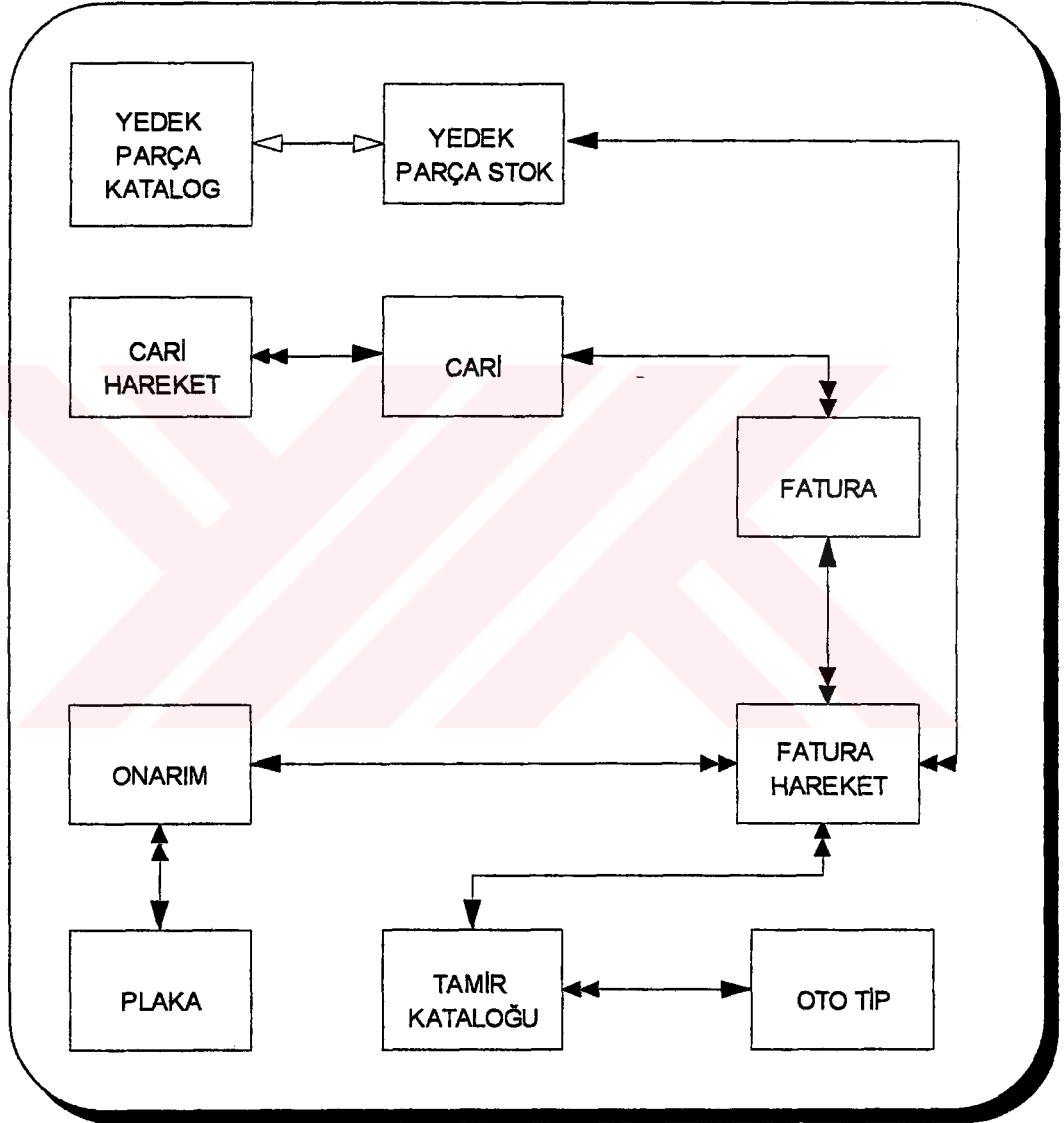
Buna göre öncelikli işlemler olarak TS1, TS2, TS3, TS4, TS5, TS6, TS7 ve TR1 olarak belirlenirse tabloya bakarak TS4, TS6 ve TS7 hareketlerini

düzenlemek gerekmektedir. Bu işlemler içinde en fazla stok ve onarım hareketleri kayıt tipine ulaşıldığı görülmektedir. Bu iki hareket, yeniden düzenleme yapılırken birleştirilerek taşıma hacmi en aza indirilebilir. Aynı zamanda stok giriş ve çıkışı olan hareketlerde stoğa anında güncelleme yapılması da MKE sayılarını artıran bir faktördür. Yine faturalama işlemlerinde müşteri hareketine gerekli bilgilerle birlikte müşteri bilgileri de anında güncellenmekte. Bu hareketlerin MKE sayılarını indirmek sisteme iki yeni hareket eklenerek bundan kurtarılabılır. Bu işlemler bir çeşit aysonu kapanışı hareketleri olacaktır. Çünkü gerek müşteri hareketleri gerekse stok hareketleri bir aylık olarak zaten sistemde saklanmaktadır. Aysonu kapanışı hareketi ile bu hareketlerden elde edilen bilgiler müşteri ve yedek parça stok bilgileri kayıt dosyalarına güncellenebilir.

Bu arada onarım kayıt tipinde fatura ile çıkan bilgiler mevcuttur. Gerektiği zaman bu bilgiler fatura kayıtlarında bulunacağından onarım bilgilerinin bulunduğu dosyada düzenlenebilir. TS5 hareketini düzeltmek içinde ototip dosyasında bulunan oto katalog tipi alanı plaka bilgilerine eklenerek kayıt erişim sayısı indirgenir.

1.5.1 Yeni Yapı

Yukarıda belirtilen düzenlemelerden sonra bilgi yapısı şu şekilde düzenlenmiştir.



Şekil 1.4 Mantıksal Tasarım (2. Sistem)

1.5.2 Yeni Depolama Alanı

Yapılan yeni tasarım için elde edilen kayıt yapılarına göre kayıt uzunlukları aşağıdaki gibidir.

Tablo Adı : YPK_STOK Yedek Parça Katalog			
Kol.No	Kolon Adı	Kolon Tipi	Uzunluk
1	YPK_PARNO	CHAR	10
2	YPK_PARAD	CHAR	20
3	YPK_PARGRB	CHAR	10
4	YPK_MLZTIP	CHAR	4
5	YPK_SATFYT	DECIMAL	13
Tablo Adı : YP_STOK Servis Stok			
1	YP_PARNO	CHAR	10
2	YP_PARAD	CHAR	20
3	YP_PARGRB	CHAR	10
4	YP_MLZTIP	CHAR	4
5	YP_SATFYT	DECIMAL	13
6	YP_YER0	CHAR	2
7	YP_YER1	CHAR	3
8	YP_YER2	CHAR	3
9	YP_SFATNO	INTEGER	8
10	YP_SBRMMLY	DECIMAL	11
11	YP_SFATMIK	INTEGER	7
12	YP_YLBMLY	DECIMAL	11
13	YP_GIRMLY	DECIMAL	11
14	YP_CIKMLY	DECIMAL	11
15	YP_YLBMIK	INTEGER	7
16	YP_GIRMIK	INTEGER	7
17	YP_CIKMIK	INTEGER	7
Tablo Adı : FATHRK Stok ve Onarım Hareket			
1	HRKNUM	INTEGER	8
2	HRKKOD	DECIMAL	2
3	HRKTIP	CHAR	1
4	HRKCAR	DECIMAL	5
5	HRKPLK	CHAR	10
8	HRKFIS	INTEGER	8
7	HRKSAT	DECIMAL	4
8	HRKTRH	DATE	6
9	HRKPARISC	CHAR	10
10	HRKMIK	DECIMAL	7.2

Şekil 1.5 Mantıksal Kayıt Tasarımı (2. Sistem)

11	HRKMLY	DECIMAL	11
12	HRKBSF	DECIMAL	11
13	HRKACK	CHAR	20
Tablo Adı : ONARIM Onarım Bilgileri			
1	ONNO	DECIMAL	5
2	ONGIRNO	DECIMAL	4
3	ONTRH	DATE	6
4	ONPLK	CHAR	10
5	ONCAR	DECIMAL	5
6	ONPARSAT	DECIMAL	4
7	ONTOPSAT	DECIMAL	4
Tablo Adı : PLAKA Araç Bilgileri			
1	PLKNO	CHAR	10
2	PLKOTOTIP	CHAR	15
2	PLKOTOKTIP	CHAR	15
3	PLKURET	DECIMAL	9
4	PLKSASI	DECIMAL	9
5	PLKSATTRH	DATE	6
6	PLKMODEL	DATE	4
7	PLKCAR	DECIMAL	5
8	PLKKM	INTEGER	6
Tablo Adı : FATURA Fatura Bilgileri			
1	FATCAR	DECIMAL	5
2	FATKOD	DECIMAL	2
3	FATNO	INTEGER	8
4	FATTRH	DATE	6
5	FATTAHD	CHAR	1
6	FATTUT	DECIMAL	12
7	FATIND	DECIMAL	11
8	FATKDV	DECIMAL	11
9	FATPARTUT	DECIMAL	11
10	FATISCTUT	DECIMAL	11
11	FATAACK	CHAR	20
Tablo Adı : OTOTIP Oto Tipleri			
1	OTOTIP	CHAR	15
2	OTOADI	CHAR	25

Şekil 1.5 Mantıksal Kayıt Tasarımı (2. Sistem) (Devamı)

3	OTOKATTIP	CHAR	15
4	ISSUCB	DECIMAL	11
5	ISSUCK	DECIMAL	11
6	ISSUCM	DECIMAL	11
7	ISSUCE	DECIMAL	11
Tablo Adı : TAMIRK Tamir Kataloğu			
1	ISCOTOKATTIP	CHAR	15
2	ISCKOD	CHAR	4
3	ISCKODADI	CHAR	15
4	ISCFYT	DECIMAL	11
5	ISCZMN	DECIMAL	3.2
6	ISCTIP	CHAR	1
Tablo Adı : CARMAS Müşteri Bilgileri			
1	CARNO	DECIMAL	5
2	CARAD	CHAR	30
3	CARADR	CHAR	30
4	CARTLF	CHAR	10
5	CARVD	CHAR	25
6	CARSIC	CHAR	10
7	CARBORTUT	DECIMAL	13
8	CARALATUT	DECIMAL	13
9	CARBORDEV	DECIMAL	13
10	CARALADEV	DECIMAL	13
11	CARPARIND	DECIMAL	2.2
12	CARISCIND	DECIMAL	2.2
Tablo Adı : CARHRK Müşteri Hareketleri			
1	CRXNO	DECIMAL	5
2	CRXTIP	CHAR	1
3	CRXFATNO	INTEGER	8
4	CRXTRH	DATE	6
5	CRXTUT	DECIMAL	13
6	CRXACK	CHAR	20

Şekil 1.5 Mantıksal Kayıt Tasarımı (2. Sistem) (Devamı)

Yapılan tasarım için elde edilen yeni kayıt yapılarına göre kayıt uzunlukları aşağıdaki gibidir;

Yedek Parça Katalog:	: 57
Yedek Parça Stok	:145
Fatura Hareket	: 106
Plaka	: 79
Onarım	: 38
Oto Tipleri	: 99
Tamir Kataloğu	: 52
Fatura	: 98
Müşteri	:172
Müşteri Hareket	: 53

Bu uzunluk değerleri ve bilgi gereksinimleri ile elde edilen bilgiler ışığında depolama alanını aşağıdaki şekilde belirleriz. Burada servis ve bakım istasyonunun 1 aylık bilgiyi güncel olarak saklayacağı varsayılmıştır.

Yedek Parça Katalog:	: 57*5000	= 278 Kb
Yedek Parça Stok	:145*3500	= 495 Kb
Fatura Hareket	:106*30 *(30*2*2 + 25*3 +30*3)	= 543 Kb
Plaka	: 79*25*30	= 58 Kb
Onarım	: 38*30*30	= 33Kb
Oto Tipleri	: 99*19	= 2Kb
Tamir Kataloğu	: 52*100	= 5Kb
Fatura	: 98*55	= 5Kb
Müşteri	:172*25*30	= 130Kb
Müşteri Hareket	: 53*55*30	= 85Kb
Toplam 10 Tablo	Toplam	1634 Kb

1.5.2 Yeni Hareket Frekansları

Yeni veritabanı kayıt tasarımı ile aysonu kapanışı adı altında iki hareket daha belirlenmiştir. Aşağıda eklenen işlemlerin frekansları listelenmiştir;

TK1: Aysonu stok hareket kapanışı. Bu hareket yardımı ile stok giriş çıkışlarında anında güncelleme kaldırılıp sistemin MKE sayıları düzenlemesi ile ihtiyaç duyulan bir harekettir. Ayda bir kere bütün parça hareketlerine ulaşarak bunu stoğa günceller.

TK2: Aysonu müşteri hareket kapanışı. Bu hareket vasıtasıyla müşteriye ait fatura kesilirken oluşan borç ve alacak hareketleri müşteri bilgilerine güncellenir.

1.5.3 Yeni Mantıksal Kayıt Erişim Sayıları ve Taşıma Hacmi

İŞL	ERİŞİLEN KAYITLAR	MKE	MKE/G	FREK.	T.H.
TT1	1YP_STOK+1YPK_STOK	2	0.6	0.3	60.6
TT2	1YP_STOK	1	0.15	0.15	21.75
TT3	1OTOTIP	1	0.01	0.01	0.99
TS1	1CARMAS	1	25	25	4,300
TS2	1PLAKA + 1OTOTIP	2	25	50	4,450
TS3	CARMAS+PLAKA+ONARIM +FATHRK	4	30	120	11,850
TS4	ONARIM+2*(YP_STOK+FATHRK)	5	60	300	32,400
TS5	ONARIM+PLAKA+ 3*(TAMIRK+FATHRK)	8	30	240	17,730
TS6	7FATHRK+1CARMAS+1CARHRK+ 1FATURA	10	30	300	31,950
TS7	1CARMAS+3*(FATHRK+YP_STOK)+ 1FATURA+1CARHRK	9	25	225	31,250
TF1	2000*(YP_STOK+YPK_STOK)	4,000	0.03	120	12,120
TF2	1000*(OTOTIP+TAMIRK)	2,000	0.02	30	2,265
TR1	55*FATURA	55	1	55	5,390
TR2	30*(4*30+3*25)*FATHRK	5,850	0.03	175.5	18,603
TK1	5850*(FATHRK + YP_STOK)	11,700	0.03	351	44,050
TK2	1650*(CARHRK + CARMAS)	3,300	0.03	99	1,114

Şekil 1.6 Mantıksal Kayıt Erişim Sayıları ve Taşıma Hacmi (2. Sistem)

BÖLÜM 2. SQL NEDİR ?

SQL bir veri tabanı alt dilidir, kontrol komutlarına sahip olmaması sebebiyle tek başına bir programlama dili değildir.

SQL; veri tanımlaması, bütünlük sınırlama tanımlaması ve bazı durumlarda erişim kontrol tanımlaması için tanımsal dil özellikleri sunar. Verilerin idaresini sağlamak için, birçok şekilde verileri getirme, saklama ve güncelleme işlemlerini sağlayan ifadeleri içeren prosedürleri içeren modüler bir dildir. SQL ;mevcut standart programlama dilleri içine yerleştirilmiş bir dildir. Tek başına da kullanılabilir ama bunun için tanımlanmış bir standart değildir. SQL , kontrol komutlarını içermemekle beraber sadece tanımsal ve işlevsel komutları kapsar. Bunun yanında, kontrol akışını sağlamak için kullanılan IF THEN ELSE , GOTO , DO WHILE veya buna benzer komutlar yoktur. Bunlar zaten mevcut programlama dili veya doğrudan kullanıcının kararları doğrultusunda yerine getirilirler.

SQL'in tarihçesinin , E.F.CODD (1970) tarafından yazılan ' Geniş Paylaşımlı Veri Bankaları İçin İlişkisel Bir Model' isimli bir yayınlı başladığı söylenebilir. Daha sonra IBM 'in San Jose'deki Araştırma Laboratuvarındaki çalışmaları gelir. 1974 yılında D.D. Chamberlain yine IBM San Jose Laboratuvarında 'Yapısal İngilizce Sorgulama Dili' veya SEQUEL denilen bir dil tanımladı. Yeni Versiyonu SEQUEL/ olarak 1976-1977 de hazırlandı ama sonradan ismi yasal nedenlerden dolayı SQL olarak değiştirildi.

Bir IBM projesi olan 'System R' SEQUEL/ tabanına dayandırıldı ve 1977'de işlevsellik kazandı. Ayrıca 1970'lerin sonunda ORACLE Corporation firması

tarafından da ORACLE veri tabanı sistemi üretildi. Bu sistem muhtemelen bu firmanın ilk ticari çalışmasıydı. IBM System R'deki deneyimlerinden faydalanarak DOS/VSE ve VM/CMS çevre birimleri için 1983'de çıkan DB2 olarak bilinen yeni bir takım ürünler piyasaya sundu.

1982'de American National Standards Institute, ANSI, IBM'in çalışmalarını esas alarak RDL, İlişkisel Veri Tabanı Dili, üzerinde çalışmalara başladı. Milletlerarası Standartlaşma Organizasyonu, ISO, 1983'te ANSI tarafından başlatılan çalışmaları topladı ve birlikte bir SQL standardı tanımladılar. 1984 'te RDL isminden (İlişkisel Veri Tabanı Dili) vazgeçildi ve mevcut SQL çalışmalarına daha çok benzeyen bir hale getirildi. Bugün varolan veri tabanı sistemleri içinde düzinelerce SQL veya SQL benzeri arabirimler mevcuttur.

2.1 Neden Bir Standart ?

Bir standart bir çok fayda sağlayabilir. Standartın avantajları şunlardır:

AÇIK GELİŞME: Günümüzde sistemlere olan talep oldukça karmaşık bir hal almaya başladı. Öyleki, bu durum tüm ihtiyaçlara cevap verebilecek tek bir paket bulmayı oldukça zorlaştırıyor. Ayrıca arz olayını gerçekleştirenler içinde tüm bu ihtiyaçları karşılamak oldukça güçleşiyor. Başarılı olabilmek için bilinen durumlarla yeni bir takım yatırımlara girmeden eldekilerle yapılanmaya, hızlı bir şekilde arabirimler oluşturmaya ihtiyaç vardır. Mevcut standart arabirimler tedarikçilere, araştırmacılara daha az bir masrafla mevcut yazılımlar için yeni köprüler kurmaya yardım eder. CASE araçları alanı niçin standartlaşma gerektiğinin mükemmel bir örneğidir.

İNSAN ESNEKLİĞİ: İnsanlar gelişme hareketlerinin en pahalı kısmını teşkil ederler. Standartlar eğitim çabalarını düşürmeye yardımcı olurlar. Çünkü standartlara dayalı eğitim ona uyan tüm farklı ürünlerde etkin bir çalışma şansı tanır.

UYGULAMA ESNEKLİĞİ: Uygulamalar, standart araçlar ve diller kullanılarak veya belli bir takım araç ve dillerin standart durumları vasıtasıyla geliştirilebilirler. Bunun anlamı şudur, uygulamalar varolan standart araç ve dillerin benzer platformlarına sahip diğer çevrelere transfer edilebilirler. Hatta komple bir uygunluk yoksa bile standartların kullanımı gerekli çevirileri minimuma indirecektir. Şu anda bütün bir standart platformu olmadığı için tam bir esneklik sağlamak mümkün olmamaktadır. Ama yine de standartlar platformlar arasındaki farkı minimuma indirger ve muhtemel seçenek sayısını çoğaltırlar.

YÜKSEK ÖLÇÜDE YAYGIN KULLANIM: Veri dağıtımı ve işleme , teknik yapı tasarlanırken önemli bir ihtiyaçtır. Bunun sebebi ; ölçülebilirlik, esneklik, yerel özerklik gibi çeşitli şeyler olabilir. Oldukça sık görülen bir durum daha vardır ki, o da şudur;mevcut durumlar ilişkisel olmalıdır ve bu da farklı yerlerdeki farklı ihtiyaçlarla birlikte ilişkilendirilmiş heterojen donanım ve yazılıma sebebiyet vermektedir. Ortak dillerin ve düğümler arasındaki iletişim protokollerinin varlığı bilinen bir gerçektir ve tüm alanlarda standartlaşma şansını artırır.

REFERANS NOKTASI: Bir sürü seçeneğe sahip bir dünyada bir çok ürünün faydalarına ve zararlarına karar vermek oldukça güçtür. Bir standart ortak tabana sahip bütün durumları indirgemeye yardımcı olur.

YATIRIMI KORUMA: Çok karmaşık uygulamalar geliştirilir ve bu çalışmaların maliyeti yüksek olur. Dilin sürekliliği ve standardın sağladığı , sunduğu seçim esnekliğinin sürekliliği yazılım alanındaki yatırımların meyvelerini toplatacak,

masrafları çıkartacaktır. Dilin ne olduğunu anlatmak için onu markete terketmeyiz. Çünkü kullanıcılar onu sunanlar kadar gerekli ilgiyi göstermezler ve standartlar kullanıcıların bazı etkileri deneyebildiği tek yoldur. Ayrıca standartlar bir veya birkaç tedarikçinin kendi çıkarları doğrultusundaki market idarelerini azaltır.

Bu durum, standartlaşmanın mükemmel olduğu anlamına gelmez. Bir de madalyonun öteki yüzü vardır. Standartlar kesinlikle mükemmel değildirler ve de sabit kalamazlar. Yeryüzündeki diğer bütün şeyler gibi ya büyürler gelişirler yada yokolup giderler. Onlar da kararlılık derecesi sağlayabilirler ama durgunluğa kesinlikle sebebiyet vermemelidirler. Standart daima odadan dışarı çıkacak ve zeki insanlar onu yeni ürünlerde destekleyip geliştireceklerdir ve gelişmesi için hep en iyi fikirleri benimseyeceklerdir. Bu özellikler SQL standardının bu yeni versiyonunda açıkça görülebilir.

2.2 Bazı Temel Kavramlar

İlişkisel veri tabanı hakkındaki özellikleri tanımlayan SQL'in diğerlerinden farkı ilişki özelliklerine göre veri tabanı içindeki ilişkilerin tanımlanma özellikleridir. İlişki kuralları matematikten elde edilerek teori haline getirilmiş, ancak veri tabanı ile ilgili güncel istekleri de karşılayacak şekilde gelişimini sürdürmektedir. Tanımlamalardan biri BOŞ (NULL) tanımıdır. Çünkü bunun tanımı ve uygulaması akademisyenler arasında farklılık göstermektedir.

Uygulamacılar daha sonra ilişkisel modelde yer alan özellikleri taşıyan ürünlerini üretmişlerdi. Bu her uygulama verinin tanımlanmasını ve işlenmesini tanımlayan kurallar için bir dil kullanmıştır. SQL'de böyle bir veri tabanı alt dilidir. Bu dillerin tanımlarının uygulamacılar tarafından seçilmesi ve ihtiyaçlarını tatmin edici seviyede karşılaması gerekmektedir. İlişkisel model sadece bir teori olarak kalırken

uygulamacılar bunu ilerleterek veri tabanı uygulamacıları grubu tarafından kolayca anlaşılabilir ve uygulanabilir seviyeye getirmişlerdir. Bazı problemlerin dar çözümleri teorik olarak yetersiz ve ilişkisel olmamışlardır. Bu bakış açısıyla SQL tam olarak ilişkisel değildir.

Bu bölümde SQL'in uluslararası standardında yer alan ve SQL kullanımıyla ilgili değişik yönleri ilgilendiren bazı tanımlar üzerinde durulacaktır.

2.2.1 Çevre

Çevre, kısaca tamamıyla bağımsız olan yada bazı yönlerden diğerleriyle ilişkili olan işlem, işleme gücünün yada verinin gruplanmasıdır. Çevre izole edilmiş bir bilgisayar yada şartlara bağlı dağınık bilgisayar ağı olarak düşünülebilir.

2.2.2 Bilgi Sistemi

Bilgi sistemi kavramı, veri tabanı kombinasyonları, veri tabanı yönetim sistemi ve uygulama programlarını ifade etmek için kullanılır. Bazı açılardan sistemi kullanan insanlarda sistemin bir parçası olarak düşünülebilir.

2.2.3 Veri Tabanı Yönetimi Sistemi

Bilgisayardaki verinin değiştirilmesi sadece bir acentanın yani veri tabanı yönetim sistemi tarafından gerçekleştirilir (DBMS). SQL dili bu acentanın nasıl

kullanılacağını tanımlar, yani DBMS ile çalışırken kullanılacak dil özelliği, yada fonksiyonlarını tanımlar. Bir DBMS değişik şekillerde uygulanabilir, ancak SQL standardı belli bir konfigürasyona ihtiyaç duymaz. Bu yüzden uygulayıcı bilgisayar sisteminde istediği kullanıcı arayüzeyini yada dosya sistemini ve bilgisayar işletim sistemini kullanabilir.

2.2.4 Veri Tabanı

Veri tabanı yönetim sistemi tarafından oluşturulan nesneye Veri Tabanı denir. Bir veri tabanı iki parçadan oluşur; Tanım ve Veri. SQL tanımlama yapılması ve bilgisayardaki verinin değiştirilmesi için gerekli olan dil özelliklerini sunar. Bu iki bölüm içiçe girmiş iki parça olarak düşünülebilir. Tanım kısmı ayrıca şema, meta-veri, veri tabanı tanımı, veri tabanı açıklaması yada veri tabanı yapısı olarak da adlandırılabilir. SQL her zaman şema kelimesini tanım kısmı için kullanır. Tanım kısmı varolamayan veri kısmı, veriyi içerir ve genelde veri tabanı olarak yada fiziksel veri veri tabanı veya veri dosyaları olarak çağrılır.

Veri tabanlarının ilk günlerinde , veri tabanı sadece bir tanım veya bir şema olarak kabul edilirdi. Günümüzde ise bir ağdaki değişik bilgisayarlardaki verilere ulaşan programları da hesaba katmamız gerekmektedir. Bütün bilgisayarların tek bir merkezden yönetilmesi gerekmediği gibi bir veri tabanındaki bütün verilerin bir tanıma veya şemaya bağlı olması düşünülemez. Bir ağa bağlı binlerce bilgisayarda bulunan verilerin tamamının tanımının yapılması veya bir şemaya bağlanması mümkün olmadığı gibi bütün noktalarında aynı anda çalışır olması beklenemez. SQL bu yüzden bu teorik problemi çözmek için daha serbest bir tanım getirmiştir. SQL'de bir veri tabanı şu şekilde tanımlanır: "Bir SQL çalışmasında bütün kataloglardan tanımlanmış bütün şemaların içerdiği verinin tamamıdır" . Eğer şu an için çalışmayı uygulama programının çalışması olarak tanımlarsak uygulamanın tanımladığı ve eriştiği bütün tanımların içerdiği verilere Veri Tabanı diyebiliriz. Tabi ki SQL veri tabanı

uygulamanın çalışmasına bağılı olarak değişecektir. Bir SQL veri tabanı bütün insanlara ait her şey, en basit olarak bir şema tarafından tanımlanmış ve karmaşık olarak ise bir çok şema tarafından tanımlanarak bütün dünyaya yayılmış bilgisayarlardaki verilerin tamamı olarak düşünülebilir.

2.2.5 Veri Tabanı Tanımı

SQL'in bir kısmı, Veri Tanımlama Dili (VTD) yani veri yapısının tanımını yapan, erişim özelliklerini belirleyen bölümdür. Bu bölümle ilgili SQL nesneleri tablolar, satırlar, kolonlar ve diğerleridir.

2.2.5.1 Tablo

SQL veri tabanının temel taşı tablodur. Tablonun resmi tanımı "Bir tablo birden fazla satır kümesidir. Bir satır boş olmayan değerler dizisidir". Satır ve sütunlardan oluşan bir matrise benzediği için bu terimler kullanılacaktır. Bu çok satırlı küme tekrar eden değerleri dışlamaz. Bu teorik olarak ilişkisel modelin kabul etmediği bir durumdur. Ancak SQL bunu reddetmediği gibi tekrar eden değerlere izin verilip verilmemesi değiştirilebilmektedir. Ancak tekrarların kullanılması bilgi sisteminin gerçekten çalışması için bazı düzenlemelerin yapılmasına ihtiyaç gösterebilir.

Bir tablonun amacı, ister gerçek ister hayal olsun , geçmişte, bugünde ve gelecekte nesneler hakkında yapılandırılmış bilgilerin saklanmasıdır. Tablodaki her satır, herbiri bir kolonda olacak şekilde belli sayıda bilgiye sahiptir. Her satır aynı sayıda kolon içerir ve kolon sayısı da derece olarak adlandırılır. Tablodaki satır sayısı ise bazen kümenin eleman sayısı olarak adlandırılır. [1]

Verinin SQL'de tek gösterim yöntemi budur. SQL'deki bütün veriler bir yada birden fazla tabloda saklanır ve sorgulamalar bu tablolar arasında yapılır.

Satır, tabloya eklenebilecek veya tablodan silinebilecek en küçük veri birimidir. Çoğu tablonun farklı bir ismi vardır ve bütün tablolar ki veri tabanı yönetim sistemi tarafından üretilmiş olsalar bile bir isme sahip olmak zorundadır .

Tablonun elemanları hakkında değişik bilgileri tutmak için bir yada birden fazla kolonu vardır. Her kolonun, Veri Tabanı Yönetim Sistemi (VTYS) tarafından tanımlanabilecek bir ismi vardır.

Şekil 2.1'de de görüldüğü gibi 'Personel' tablonun ismi 'isim', 'adres', 'ev_tlf' ve 'blmkod' kolon isimleridir.

Personel			
isim	adres	ev_tlf	blmkod
Çiğdem	Küçükyalı	366 06 87	PZR
Ümit	Sarıyer	242 14 51	BİM
Erol	Bahçelievler	578 15 10	SAT
...	...	...	...
Ahmet	Erenköy	302 68 58	EĞT

Şekil 2.1 Örnek Tablo

Satır ve kolonun kesiştiği yerler bir saklama alanı, alan oluşturarak gerçek veriyi saklamaktadır. Burada 'Çiğdem' veri 'isim' ilgilenilen nesnedir. Tablo ismi ilgilenilen nesnenin ismidir. Ancak tekil nesneler ise satırlarla ifade edilmiştir. Alanlardaki değerler önemli gerekçelerle kısıtlanmıştır.

Doğum tarihi, maaş, yaş vb. gibi karakteristik özellikler özel kısıtlamalar içerebilir. SQL gibi diller bu gibi kısıtlamalar için bazı kolaylıklar sunar. İzin verilen değerler bir 'saha' da tanımlanır. 'SAHA' izin verilen değerler kümesidir. Teorik olarak her kolonun bir sahası vardır.

Örnek verecek olursak ev_tlf kolonunda verinin '+', '-', '(', ') ve 0-9 aralığındaki rakamları içermesini isterken harf içermesi istenmeyebilir. Her kolonun tanımlanmış bir maksimum uzunluğu, duyarlılığı vs. vardır. Böylece o kolona girebilecek veri sınırlandırılmış olur. SQL'de saha kelimesi Domain tanımı ile kısıtlanmıştır. İsimlendirilmiş tanım kümeleri kolonların tanımında kullanılabilir. Aksi takdirde gerçek özellikleri yansıtan domainin belli karakteristik özelliklerini ifade eden veri tipi kullanılır.

Daha fazla kısıtlama tablolar arasındaki İlişkisel Bütünlük tanımlamalarıyla sağlanabilir. İlişkisel Bütünlük kavramı nesneler arasındaki bağımlılıkla ilgilenir. Örneğin Personel tablosunda bölümü olmayan hiçbir personel olamaz. Bu bağımlılık SQL'de sembolik anahtar ile ifade edilir. Her bölümün bölümün tekil bir tanımlayıcısı vardır. Buna 'ana anahtar' denir. Örnek tabloda bu blmkod isimli sahadır. Bu bağlı olan personelin bağımlılığını göstermek için kullanılır. Yani her personelin blmkod kolonu Bölüm tablosundaki blmkod da saklanan bilgilerdir.

Anahtar, bir tabloda bir satırı ifade edebilecek bir yada birden fazla kolona verilen addır. Birden fazla anahtar tipi vardır.

Aday Anahtar: Aday anahtar o tablonun satırlarını ifade edebilen kolon yada kolon topluluğudur. Bir tabloda birden fazla aday anahtar olabilir. Örneğin kişi tablosunda aşağıdaki kolonlar aday anahtar olabilir.

Vergi numarası	Sicil Numarası
Ad + Doğum tarihi	Sosyal Güvenlik Numarası

Ana Anahtar: Ana anahtar aday anahtarlardan biridir ve tablonun satırlarının ana tanımıdır. Bir tablonun birden fazla aday anahtarı olabileceken ancak bir tane ana anahtarı olabilir.

Yabancı Anahtar: Bir tablonun bir yada birden fazla kolonda bulunan verilerinin başka bir tablonun satırında tanımlanmış olmasıdır. Bu ancak değer boş değilse gerçekleşir.

2.2.5.2 Şema

Şemanın amacı, SQL nesnelerinin , Tablo , görüntü ve sahalarının bir şekilde diğerleriyle bağlantılı şekilde bir arada toplanmasıdır. Bir veri tabanındaki bütün nesneler bir şemada yada diğerinde tanımlanmıştır. Şemayı bir kimsenin sahip olduğu nesnelerin tamamı olarak kabul edebiliriz. Bütün nesneler arasındaki bağlantıları isimlendirerek gösterir. Herbiri örnek olarak onu alır ve böylece bir nesnenin sahibi kolayca görülür.

2.2.5.3 Katalog

Katalog, şemaların bir grubunun isimlendirilmesidir. Böylece bir katalog içindeki bütün şemaların önüne katalog ismi gelir. Şemaların bu şekilde gruplanması, yönetim amaçları için yada şemaların bulundukları fiziksel yer vb. sebepler için yapılır. Bir katalogdaki şemalar veri tabanı diye adlandırılacak en küçük birimi oluşturur.

Veri tabanının bir kısmını tanımlayan şemanın meta-veri kısmı veri olarak da kabuledilebilir. SQL de bu anlayışa katılarak her katalog için Bilgi Şeması adı verilen özel bir şema tutar. Bu şema katalogda yeralan bütün şemaların meta-veri kısımlarını içerir. Bu sanki veri tabanı içinde veri tabanı gibidir. Bazı açılardan katalog, veri sözlük sistemi veya bilgi kaynak sözlük sistemi ile kıyaslanabilir. Ancak bu sistemler katalogdan daha fazla bilgi tutarlar. Katalog veri tabanı şu şekilde gösterilebilir:

Katalog Veri Tanımı
Katalog Veri Tabanı Verisi

Şekil 2.2 Katalog Veri Tabanı

Ancak, katalog veri tabanı şemaları tanımlarken uygulamanın verilerini de tanımladığı için aşağıdaki şekli çizebiliriz.

	Katalog Veri tanımı
Uygulama Veri Tanımı	Katalog Veri Tabanı Verisi
Uygulama Veri Tabanı Verisi	

Şekil 2.3 Katalog Veri Tabanında Şema Tanımı

Her veri tabanının birey çiftleri birbirine bağlanmıştır. Bunun sebebi bir veri tabanının veri tabanı yapısının tanımının katalog veri tabanı verileri arasında uygulama verisi olarak bulunmasıdır. Katalog veri tabanı uygulama veri tabanını kontrol eder. Üç veri seviyesi ifade eden iki bağlı çift vardır: uygulama verileri , sözlük veya katalog verileri ve katalog tanım verileri. Bir uygulama veri tabanının işlevselliği bir sonraki seviye katalogda belirlenen uygulam veri tabanı yapısıyla tanımlanır. Kataloğun işlevselliği ise üst seviyedeki katalog veri tabanı yapısında belirlenir. Bu bir yada daha fazla seviye için devam eder.

2.2.6 Veri Tabanı Deęiřiklięi

SQL'in dięer önemli bölümü veri deęiřtirme dilidir (VDD). Bu dil Veri tabanı Yönetim Sisteminin olan verilerin yapılan tanımlara göre yönlendirileceęini belirler. Bu dil yardımıyla veri tabanına bilgiler eklenebilir, silinebilir, güncellenebilir ya da erişilebilir.



BÖLÜM 3. TEMEL DİL ELEMANLARI

Bu bölümde sıkça kullanılan temel dil elemanlarının tanımı yapılacaktır. Bunlar, geçerli karakterler, simgeler, tanıtıcılar, veri tipleri ve sabitlerdir.

3.1 Karakterler

En basit ve en temel dil elemanı karakterdir. Her SQL gösterimi için SQL-TEXT olarak adlandırılan bir karakter kümesi vardır ve ancak bu kümenin elemanları SQL gösteriminde kullanılır. Bilgi şeması, bu kümenin tanımını içermektedir. SQL standardı, SQL-TEXT'in uygun gösteriminde bulunması gereken karakterleri belirlemiştir. Bu minimum küme dilin bütün komutlarını yazmak için gereken karakterlerden oluşur.

Temel küme, büyük ve küçük Latin harfleri; A, B, C, ..., Z, a, b, c, ..., z, sayılar; 0,1, 2, ..., 9 ve bazı özel karakterler; . ; () : % _ ? ' " + - * / < > = & > | ve bunlara ilave olarak boşluk karakterinden oluşur. Bunlar SQL dilinin karakter kümesi olarak tanımlandığından 'TEMEL SQL' kümesi olarak kullanılır.

3.2 Simgeler

Simgeler, dilin birimleridir. Bu birimler, dildeki karakter düzeyinden bir adım daha ileridedir ve dilin yapısını kurmak için bloklar oluşturma imkanını verirler. Simgelerle, bir cümleyi oluşturan karakterler dizisinin nasıl parçalanacağı belirlenir.

İki çeşit simge vardır; sınırlı ve serbest. Simgelere ilave olarak ayraçlar bulunmaktadır. Sınırlı bir simgeden sonra ; herhangi sayıda ayraçlar veya başka sınırlı veya serbest simgeler bulunabilir. Serbest simgeden sonra ise herhangi sayıda ayraçlar veya sınırlı simge gelebilir. Eğer sözdizimi bir serbest simgeden sonra sınırlı simge gelmesine izin vermiyorsa, serbest simgeden sonra en az bir ayraç bulunmalıdır.

Ayraç; boşluk karakteri, yeni satır işareti veya bir açıklamadan oluşur. Yeni satır işareti; açıklama ise 2 veya daha fazla (-) eksi işareti ile başlayan , herhangi sayıda karakterlerle devam eder ve ilk yeni satır işareti ile sona erer. Dil yapısının geri kalanına gelince, her açıklama ve her yeni satır işareti (açıklamanın parçası olmayan) sanki yeni bir satır işareti varmış gibi algılanır. [1]

Sınırlı simgeler aşağıdakilerden biri veya sınırlı tanımlayıcılar ve karakter dizisi olabilir ; . ; () , : % _ ? ' " + - * / < > = & | < > = < = || []

Serbest bir simge ise aşağıdakilerden biri olabilir;

Düzenli bir tanımlayıcı	Bir anahtar kelime
İşaretsiz nümerik bir değer	Tarih
Zaman	Time stamp dizisi
Aralık	Ulusal karakter
İkili	Sekizli

3.3 Tanımlayıcılar

Tanımlayıcı, bir isim yada ismin bir bölümüdür. Bu isimler; SQL veri tabanında tablolar, kolonlar, görünüm gibi nesneleri tanımlamak için kullanılır. SQL dilinde çok çeşitli isimler vardır; fakat bunların tamamı bir veya birden fazla aralıklarla birleştirilmiş tanımlayıcılardan oluşmuştur. Her tanımlayıcı için şöyle bir yapıya sahiptir; [<karakter küme ismi>] <gerçek tanımlayıcı>

Karakter küme ismi, gerçek tanımlayıcıdan alınan alternatif karakter kümesini seçmek için kullanılır. <Karakter küme ismi> tanımlandığı takdirde, <gerçek tanımlayıcı> daki tüm karakterler mutlak bu kümeden alınmalıdır. Eğer küme tanımlanmazsa, Temel SQL dilindeki karakterler kullanılır. Karakterler; tanıtıcıları içeren modül yada şemada belirtilen kümeden de alınabilir. Yani gösterim şekli çeşitli karakter kümelerini desteklediğinde, kümeleri tam olarak tanımlama gereği olmadan her ikisine birden ulaşma imkanı vardır. Ayrıca tanımlayıcının önünde bulunan karakter kümesi de kullanılabilir.

<Gerçek tanımlayıcı> düzgün bir tanımlayıcı yada tırnaklarla ayrılmış bir tanımlayıcıdır. Düzgün bir tanımlayıcıda en fazla 128 karakter bulunur. Bu karakterler şunlar içinden seçilebilir: Alfabe harfleri, sayı (tanımlayıcının ilk karakteri hariç), alt-çizgi (ilk karakter olarak kullanılmaz), syllable veya ideograph (boşluk karakteri ve noktalama işaretleri hariç). Düzgün tanımlayıcıdaki her küçük harf, büyük harf olarak değerlendirilir. Büyük harflere çevrilmiş bir tanımlayıcı, ayrılmış kelimelerle eşitlik açısından karşılaştırılmaz. Tırnaklarla ayrılmış bir tanımlayıcı da belirtilen kümeden en fazla 128 karakter bulunabilir. Bu tanımlayıcı içinde tırnak işareti kullanılmaz (Ancak ardışık iki tane yazıldığı takdirde kullanılır).

Ayırılanmış kullanıcı kullanılmasının nedeni, ayrılmış sözcüklerle karışmayı önlemektir. "TABLE" yazmakla bir tanımlayıcı belirtmiş oluruz ve TABLE ayrılmış kelimesiyle karışmayı engelleriz. Karşılaştırma durumunda ALİ ve Ali aynı kabul edilir. Fakat "ALİ" sadece "Ali" ye eşittir.

3.4. İsimler

İsimler, SQL dilinin kavramlarını ve yapılarını anlatmak için kullanılır. SQL dilinde bulunan çeşitli isimlerin listeri aşağıda verilmiştir:

1. Nitelenmemiş İsimler

-katolog ismi

- yetki tanımlayıcısı
- ilişki kuran isim
- modül ismi
- prosedür ismi
- imleç ismi
- dinamik imleç ismi
- SQL dili komutu tanımlayıcısı

2. Katolog ismiyle nitelenmiş isimler

- şema ismi

3. Şema ismiyle nitelenmiş isimler

- saha ismi
- tablo ismi
- kolon ismi
- kısıt ismi
- karakter küme ismi
- karşılaştırma ismi
- tercüme (çevirme) ismi
- değiştirme ismi

Niteleme, bir ismin öncesine bir nokta ve bir niteleyici ilave etmekle yapılır. Örneğin nitelenmiş şema ismi Şirket, şöyle nitelenebilir: Varsayılan katolog ismi KATALOG ise,

KATALOG.Şirket

nitelenmiş bir isim haline gelir. Yine aynı şekilde, bir tablo ismi olan Bölüm şöyle yazılabilir:

KATALOG.Şirket.Bölüm

Bu son isim 3-seviyeli olarak bilinir. Çoğu durumlarda niteleyiciler belli bir düzen içinde varsayılan kabul edilir. Bu varsayılan isim şöyle belirlenir: Eğer isim şema tanımı içinde geçiyorsa, şemanın ismi kullanılır. Eğer bir SQL komutu dinamik veya doğrudan işletiliyorsa "oturum" varsayılanları kullanılır. Diğer durumlarda varsayılan şema ismi olarak ilgili modül ismi kullanılır.

Nitelenmiş bir şema için varsayılan katalog ismi de şöyle belirlenir; Eğer isim modül başlığı içinde bulunuyorsa "taahhüt" tanımlıdır. Eğer şema tanımında bulunuyor ama şemayı isimlendiren kısımda bulunuyorsa şema isminin "kesin" yada "içerik" bölümü kullanılır. Eğer isim dinamik olarak çalıştırılan bir SQL komutu içinde ise, bu takdirde o oturumun varsayılanları kullanılır. Diğer durumlarda modülle ilişkili olan katalog ismi kullanılır.

İsimlere ilişkin bazı ilginç özellikler şöyledir:

-Bir tablo ismi, bir şema ismi ile nitelendirilme yerine, MODÜL sözcüğü ile de nitelenebilir,-ki bu durumda belirli bir tabloyu tanıtır.

-Bir SQL cümlesi tanımlayıcısı ve dinamik imleç ismi ; tanımlayıcı olmalarının yanında, genişletilmiş bir isim de olabilirler. Bu durumda bu isimler parametre veya bağlanan terminale bir tanımlama formatında değişken olabilirler. Buna ilave olarak genişletilmiş isimler , global ve lokal bir alan tanımlayıcısı olabilirler. Lokal olan; içinde genişletilmiş ismin tanımlandığı modüldür. Global olan ise, bir SQL oturumunda kullanılan tüm modüllerdir (varsayılanı lokaldır). Tarif edici isimlerde olan içinde alan içinde genişletilmiş sayılırlar ama tanıtıcı olarak kullanılmazlar.

-Bir karakter kümesi basit Latin harflerini, sayıları ve alt-çizgiyi içerebilir.

-Bir yetki tanıtıcısının izin verilen değerleri, gösterim (taahhüt) çevresine bağlıdır; çünkü yetkili kullanıcı SQL in bir parçası değildir.

-Bir şema ismi tarafından nitelenmemiş karakter küme isimleri, karşılaştırma isimleri,tip dönüştürme ve çevirme isimleri "BİLGİ-ŞEMASI" tanımlayıcısının şemasını alıyor kabul edilirler.Yani kullanıcının tanımladığı karakter kümeleri, çeviriler ve karşılaştırmalar mutlaka nitelenmiş bir isim tarafından çağrılmalıdır (dönüşümler kullanıcı tarafından tanımlanamazlar).

Bu sayede daha önceden tanımlanmış bir ismin yeniden tanımlanması engellenmiş olur. "BİLGİ-ŞEMASI" içinde geçen tüm standart ve gösterim tanımlı

nesnelerin hakları "ÖZEL HAK" seçeneği ile "GENEL" leştirilmiştir, böylece bu nesneler her yerde rahatlıkla kullanılabilir. [1]

3.5. Sözcükler

Sözcükler, SQL dilindeki sabit değerleri ifade etmek içindir. Bu sözcükler, veri tabanına bilgi almak, değerleri başka değerlerle karşılaştırmak v.s işlerde kullanılır. SQL dilinin desteklediği her data tipi için çeşitli şekillerde sözcükler vardır. Her sözcük ilişkili olduğu data tipinin normal karakteristiklerine sahiptir.

3.6. Değerler ve Hedefler

Değerler, SQL dili içinde birçok yerde kullanılır: Veri tabanındaki bir kolona bilgi aktarmak, imlecin yerini belirtmek, veri tabanındaki kolonda bulunanlarla karşılaştırma yapmak, v.s:

Değerler; sözcük, değişken tanımı, dinamik tanım, bağlantı yapılan dilin değişken tanımı, veya sistemden gelen değerler olabilir (örneğin: USER, SESSION USER, CURRENT USER, ve SYSTEM USER). Parametre tanımı şu şekildedir:

<parametre ismi> [[INDICATOR] <parametre ismi>]

Bağlanan dildeki değişken tanımı ise şöyledir:

<gömülmüş değişken ismi>[[INDICATOR]<gömülmüş değişkenismi>]

Her iki tanımda da <parametre ismi> ve <gömülmüş değişken ismi> tamsayı olmalıdır. Bunlar NULL durumunu sağlamada kullanılır. Bu parametrenin

veya bağlanan dildeki değişkenin negatif değeri birinci değişkenin kullanılmayacağını, bunun yerine NULL durumunun alınacağını gösterir. Pozitif veya sıfır olduğunda ise birinci parametre kullanılacak anlamına gelir. Bu işlem gösterici parametreler veya değişkenler olarak adlandırılır.

Sistemden gelen USER, CURRENT USER değerleri ise güncel yetki tanımlayıcının değerlerini belirtir. SESSION USER, açık modül yetkilendirmesine bakmaksızın, güncel yetki tanımlayıcısının değerlerini verir. SYSTEM USER, ise oturumu çalıştıran kullanıcı için sistem tanımlarını verir. her iki değer de, sabit yada değişken karakter dizileri olabilir. Burada (<işaretsiz değer tanımı>) işaretin gelemediği durumlarda kullanılır. Belirtilmediği takdirde (<değer tanımı>) varsayılır.

SQL dilinde hedefler, çeşitli kaynaklardan bilgi almak için kullanılır (veritabanı, açıklama alanı, bilinen alan gibi gibi).

Hedefin formatı da, değer formatı gibidir; yani bir parametre tanımı ve bağlanan dilin değişken tanımı olabilir. Gösterici değişken yada parametre de bir hedef olarak kullanılabilir. BNF' de <basit değer tanımı> ve <basit hedef tanımı> , hedef veya değer NULL olmadığı durumlarda kullanılabilir. Bu durumda ikinci isim ve anahtar kelime GÖSTERGE yazılabilir.

3.7. Data Tipleri

SQL dilinde 6 çeşit data tipi vardır:

- | | |
|--------------|-------------------|
| -karakter | -ikili |
| -tam sayısal | -yaklaşık sayısal |
| -tarih-zaman | -aralık |

İşlemlerde ve tip değiştirme işlerinde, 'karakter' ve 'ikili' beraberce 'karakter dizisi' olarak; 'tam sayısal' ve 'yaklaşık sayısal' ise 'sayısal' olarak çağrılır. Çünkü bunlar aynı özelliklere sahiptir.

3.7.1.Karakter Data Tipi

Bu tip, karakter dizilerini tanımlamada kullanılır. Karakter dizisi, tanımlanan kümedeki karakterlerden oluşur. Bu kümeye karakter repertuarı da denir. Her repertuarın bir ismi vardır. Örneğin, BASIC SQL karakter repertuarı.

Her repertuvarı, çok sayıda karakterlerin bulunması sebebiyle, her repertuvar için aynı kodlama kullanılması uygun değildir. Eğer, gösterimde hangi repertuarın kullanıldığı belirtilirse, bu kodun kaplayacağı alanın daha az olmasını sağlar. Böyle çeşitli kodlama yöntemlerine 'kullanım şekli' denir. Her karakter kümesinden fazla karşılaştırma dizisi bulunabilir. Her karakter kümesi için belirli bir karşılaştırma dizisi vardır.

Karakter dizisi tanımda basit veya değişken uzunlukta belirtilebilir. Bu dizinin uzunluğu, içerdiği karakterlerin sayısı kadardır. Eğer dizi sabit uzunlukta bu sayı onun uzunluğudur. Değişken olarak tanımlanmışsa bu sayı onun uzunluğudur. Değişken olarak belirtilmişse, uzunluk sıfır veya tanımlanan maksimum uzunluk olacaktır. Uzunluk mantıksaldır ve kodlama sisteminden bağımsızdır. Tüm dizilerin fiziksel karakteristiklerini belirlemek için fonksiyonları vardır; fakat bu karakteristiklerin dizilerin tanımlanmasında bir önemi yoktur.

Data tipleri şöyle tanımlanır:

CHARACTER [VARYING] [(<LENGTH>)]

[CHARACTER SET <karakter küme ismi>]

[VARYING] kullanıldığında (uzunluk) mutlaka belirtilmelidir. [değişken] yazılmadığında, dizinin uzunluğu 1 kabul edilir. "CHARACTER SET" tanımlaması da isteğe bağlıdır. Belirtilmediği takdirde şemada geçen kullanılır.

Örnekler;

CHARACTER

CHARACTER VARYING (200)

CHARACTER VARYING (10) CHARACTER SET Latin_1

Latin_1 karakter kümesinden alınan karakterlerden oluşan en fazla 1000 değişken uzunluğunda olan dizi.

Bu tanımlamalar için kullanılan kısaltmalar şunlardır;

CHAR (X)	CHARACTER (X)
VARCHAR (X)	CHARACTER VARYING (X)
NCHAR (X)	NATIONAL CHARACTER (X)

3.7.2. Karakter Sözcükleri

Karakter sözcükleri, karakter dizilerini göstermek amacıyla kullanılır. Her sözcük aşağıdaki yapıya sahiptir:

[_<karakter küme ismi>][<karakter>]
[{{<ayraç>..'<karakter>...'}}...]

Karakter ismi istenildiği gibi olabilir, bundan sonra ise turnakla kapatılmış olarak karakter dizisi gelir. Sözcük uzun olduğu zaman karakter dizisi ayraçlarla satırlara bölünebilir. Ayraçlar boşluk, açıklama yazısı, yeni satır işareti olmalıdır. Karakter kümesi tanımlanmışsa, sözcükte kullanılan tüm harfler bu kümenin elemanı olmalıdır. Tanımlanmadığı zaman sözcük o sözcüğü içeren modül yada şemanın , TEMEL SQL kümesinden alınacak karakterlerden oluşabilir.

Sözcükte geçecek olan (') ise (") olarak yazılmalıdır. Türkçe Karakter kümesini varsayılan küme olduğunu düşünürsek karakter sözcük örnekleri şöyle olabilir:

'Çağatay'
 _Turk 'İıĞğŞşÜüÇçÖö'
 ""ŞİŞE""
 ""Ahmet""
 '+44(0)71-549-9925"
 ' Oldukça uzun bir karakter sözcük dizisi' -- Açıklama
 'birden fazla satıra'
 'bu örnekte olduğu gibi yazılabilir.'

Karakter küme ismini belirtirken özel bir durum vardır: Eğer milli karakter kümesi olarak TÜRKÇE belirtilmişse aşağıdaki iki gösterim de aynıdır.

_Turk 'İıĞğŞşÜüÇçÖö'
 _N 'İıĞğŞşÜüÇçÖö'

3.7.3. İkili Veri Tipi

İkili veri tipi ikili dizileri oluşturmak için kullanılır; bu dizilerin her karakteri 1 veya 0 dır. Bazı yapısı bilinmeyen veriler SQL tarafından tanınmayabilir. İkili veri yapısı kullanılarak bu gibi durumlar ortadan kaldırılır. Çünkü karakter dizisi bir kümeden bir başka kümeye çevrilirken veya iki farklı veri tabanı sisteminin bilgi alışverişi sırasında bozulma ihtimaline sahiptir. İkili veri tipi ile böyle bir ihtimal kalmayacaktır. Çünkü ikili veya onaltılı bilgi tüm sistemlerde aynıdır.

İkili dizi sabit yada değişken uzunlukta olabilir. Dizinin uzunluğu içerdiği 1 ve 0 ların sayısıdır. İkili veri tipinin tanımı karakter tanımına benzerdir;

BIT	1 uzunluğunda
BIT 10	sabit 10 uzunluğunda
BIT VARYING (3000)	en çok 3000 olabilen değişken uzunlukta.

3.7.4. İkili Veri Sözcükleri

İkili dizi sözcükleri, ikili dizileri ifade için kullanılır. İkili dizi sözcükleri iki şekilde yazılabilir;

B' [<bit>...] ' [{ <ayraç>... ' [<bit>...] ' } ...] bit:0 veya 1
 X' [<hexsayı>...] ' [{ <ayraç>... ' [<hexsayı>...] ' } ...]
 hexsayı:0 1 2 3 4 5 6 7 8 A B C D E F a b c d e f

Örnekler;

B'001001101' X'c4' X'1FFfe'
 B'010101010101010101010' -- Birden fazla satıra
 '11110101110110000000' -- yazılmış ikili dizi sözcüğü

3.7.5. Tam Sayısal Veri Tipi

Bu tip, tamsayıları milli para değerlerini (\$, TL, p vb.) tanımlamak için kullanılır. Tanım şu şekilde yapılır;

<Tam Sayısal Tip>::=
 NUMERIC[(<Hane Sayısı>[,<hassasiyet>])]
 DECIMAL[(<Hane Sayısı>[,<hassasiyet>])]
 INTEGER
 SMALLINT

Hassasiyet verilmediği zaman 0 kabul edilir. SMALLINT ve INTEGER için hane sayısı gösterimde tanımlıdır. SMALLINT'in hane sayısı INTEGER hane sayısından büyük değildir. Bunların hassasiyetleri ise tanım gereği sıfırdır. [1]

INT = INTEGER

DEC(p,s) = DECIMAL(p,s) kısa yazım şekilleridir.

Örnekler;

INTEGER	DECIMAL(6,2)	NUMERIC(5)
SMALLINT	NUMERIC(5,2)	DECIMAL(11)

3.7.6. Tam Sayısal Sözcükler

Tam sayısal sözcükler ; tamsayılar, para birimleri gibi sayıları göstermek için kullanılır. Bunlar işaretli veya işaretsiz olabilir. İşaretli değer; işaretsiz sayısal değerın soluna (+) veya (-) gelmesi ile olur. Tam sayısal sözcük, ondalık haneli yada hanesiz bir sayıdır. Standart SQL ondalık gösterimi desteklemez 100,00 gibi. Yine de çoğu gösterim, SQL kurallarının üzerine yenilerini koyarak böyle bir notasyona izin verirler.

Geçerli Tam Sayısal Sözcük örnekleri;

12	06	100.3	.003
+0.7	-.2	-200.75	7.

3.7.7. Yaklaşık Sayısal Veri Tipi

Bu veri tipi, gerçel sayılar gibi, tam gösterime sahip olamayan sayıları tanımlamak için kullanılır. Her yaklaşık sayısal veri tipi mantis ve üst kısımdan oluşur. Üs, işaretli bir tamsayıdır. Mantis ise hassasiyeti olan bir sayıdır. M bir mantis e de bir üs ise , yaklaşık sayısal bir değer $M \cdot 10^e$ olacaktır.

Bu veri tipi şöyle tanımlanır;

<yaklaşık sayısal tip> ::= FLOAT[(**<hassasiyet>**)]

REAL

DOUBLE PRECISION

<hassasiyet> mantisin hane sayısını kontrol eden bir sayıdır.

Desteklenen maksimum hassasiyet gösterimdeki tanıma bağlıdır. DOUBLE PRECISION, REAL'den daha fazla hassasiyete sahiptir. Üs için de izin verilen değerler de tanıma bağlıdır, mesela -127 ve +128 arasında olabilir.

Örnekler;

DOUBLE PRECISION	FLOAT
FLOAT(24)	REAL

3.7.8. Yaklaşık sayısal sözcükler

Gerçek sayıları ifade etmek için kullanılan bu sözcükler işaretli yada işaretsiz olabilirler ve şu formata sahiptirler: **<mantis>E<üs>**

Bu gösterimde mantis ve üs tam sayısal sözcüklerdir. Bu gösterime bilimsel gösterim de denmektedir.

Örnekler; 10E3 +57.4E5 -.2E-9 -7E+16

3.7.9 Tarih-Zaman Veri Tipi

Bu veri tipi, belli bir doğrulukla zamanı belirtmek için kullanılır (tarih, saat, saat vb.). her tarih-zaman veri tipi aşağıdaki alanların alt kümelerinden oluşur.

YEAR	YIL
MONTH	AY

DAY	GÜN
HOUR	SAAT
MINUTE	DAKİKA
SECOND	SANİYE
TIMEZONE-HOUR	Yerel Saat Farkının SAAT bölümü
TIMEZONE-MINUTE	Yerel Saat Farkının DAKİKA bölümü
3 çeşit tarih-zaman veri tipi vardır;	
DATE	TIME TIMESTAMP

Bu veri tiplerinin gösterimi ise şöyledir;

DATE | TIME [(**<zaman hassasiyeti>**)] [WITH TIME ZONE]
 | TIMESTAMP [(**<zaman hassasiyeti>**)] [WITH TIME ZONE]

DATE tipi YEAR, MONTH ve DAY alanlarını , TIME tipi ise HOUR, MINUTE, ve SECOND alanlarını destekler. TIMESTAMP ise YEAR'dan SECOND'a kadar olan 6 alanı birden destekler. Böylece DATE ve TIMESTAMP zaman içindeki tek noktaları belirtir, TIME ise zaman içindeki tekrarlı noktaları belirtir. TIME ve TIMESTAMP için, hassasiyet belirtilebilir, belirtilmediği takdirde TIME için 0 TIME STAMP için 6 değeri alınır.

TIME ve TIMESTAMP için WITH TIME ZONE seçeneği vardır. Eğer bu seçenek belirtilmezse, SQL_user oturumu için varsayılan zaman fark dilimi alınır. Eğer bu seçenek belirtilirse, TIMEZONE-HOUR ve TIMEZONE-MINUTE alanlarındaki değerler kullanılacaktır, yani 20:07:00-5:00 ile 01:07:00+0:00 aynı zamanı göstermektedir.

Örnekler;

DATE Yıl, ay, gün belirtir

TIME(3) WITH TIME ZONE	Zaman farkını da içeren bir zaman tanımı (saat, dakika, saniye ve milisaniye ile birlikte zamandilimi-saat ve zamandilimi-dakika)
TIMESTAMP(6)	Tarih ve zaman belirtir (yıl, ay, gün, saat, dakika, saniye, mikrosaniye mevcut oturumun zaman dilimi ile birlikte)
TIME	Zaman belirtir (saat, dakika, saniye mevcut oturumun zaman dilimi ile birlikte)

İstenen hassasiyette oturumun tarihini ve zamanını veren çeşitli fonksiyonlar vardır. Ayrıca bir tek komutla da oturumun zaman dilimi değiştirilebilir.

3.7.10. Tarih-Zaman Sözcükleri

Tarih-zaman sözcükleri zaman içindeki bir noktayı istenen hassasiyette ifade etmeye yararlar. Aşağıdaki şekiller de olabilirler;

DATE '<yıl>-<ay>-<gün>'

TIME '<saat>:<dakika>:<saniye>[<zaman diliimi>]'

TIMESTAMP'<yıl>-<ay>-<gün><saat>:<dakika>:<saniye>

[<zaman dilimi>]

Örnekler;

DATE '1900-01-01'

TIME '23:59:59.999999-05:00'

TIMESTAMP '1949-05-16 22:00:00+11:30'

Tarih için miladi takvim kuralları; saat için de 24-saat kuralı geçerlidir. Bu kurala uymayan tarih-zaman sözcükleri geçersiz sayılacaktır.

3.7.11. Aralık Veri Tipi

Zamanın periyotlarını belirtmek için aralık veri tipi kullanılır, bir başka deyişle zamandaki iki nokta arasındaki uzaklığı belirtir. Aralık belirten veri tipleri aşağıdaki alanların alt kümesi olabilir;

YEAR (yıllar), MONTH (aylar), DAY (günler), HOUR (saatler), MINUTE (dakikalar), SECOND (saniyeler ve onun alt birimleridir).

Yıl-Ay aralıkları ve Gün-Zaman aralıkları olmak üzere iki çeşit aralık veri tipi mevcuttur. Yıl-Ay aralık veri tipi sadece YEAR ve MONTH , Gün-Zaman aralık veri tipi ise sadece DAY, HOUR, MINUTE ve SECOND alanlarından seçilebilir.

Aralık veri tipinin gösterimi şöyledir;

<aralık tipi>::=INTERVAL

{ { <başlangıç alanı> to <bitiş alanı> } | <tek tarih-zaman alanı> }

<başlangıç alanı>::=<saniye içermeyen tarih-zaman alanı>

[(<alan hassasiyetini belirten aralık>)]

<bitiş alanı>::=<saniye içermeyen tarih-zaman alanı>

| SECOND [(<kesirli saniye hassasiyeti>)]

<tek tarih-zaman alanı>::=<saniye içermeyen tarih-zaman alanı>

[(<alan hassasiyetini belirten aralık>)]
 | SECOND [(<alan hassasiyetini belirten aralık> [, <kesirli saniye hassasiyeti>])]

<saniye içermeyen tarih-zaman alanı> : :=YEAR | MONTH | DAY | HOUR |
 MINUTE

Örnekler;

INTERVAL YEAR(4) 0-1999 arasındaki yıllar (hassasiyet 4)

INTERVAL DAY 0-99 arasındaki günler (hassasiyet 2)

INTERVAL SECOND (6 , 0) 0-999999 arasındaki saniye
 (hassasiyet 2; ondalık kısım, hassasiyet 0)

INTERVAL DAY TO HOUR 0 gün , 0 saat , ile 99 gün 23 saat
 arasında bir değer.

3.7.12 Aralık Sözcükleri

Aralık sözcükleri , bir zaman periyodunu belirtmek için kullanılır. Eğer zaman bir doğru olarak düşünülürse , aralık bu doğrudan bir parçadır. Aynı zamanda tarih zaman değeri bu doğru üzerinde bir nokta veya noktalara karşılık gelir. Aralık sözcüğünün gösterimi şöyledir:

INTERVAL [+ | -] '<değer dizisi>' <aralık dizisi>

<aralık niteleyicisi> nin gösterimi de şöyledir:

<başlangıç alanı> [(<hassasiyet>)] [TO <bitiş alanı>]

<başlangıç alanı> ve <bitiş alanı> şunlardan biridir:

"YEAR" , "MONTH" , "DAY" , "HOUR" , "MINUTE" veya

"SECOND [(<ondalık hassasiyet>)]"

<ondalık hassasiyet> tanımı , saniye alanın doğruluğunu belirler.

Örneğin , SECOND(6) mikrosaniyeyi belirtir. Eğer "SECOND" aynı zamanda

<başlangıç alanı> ise , <ondalık hassasiyet> ve <hassasiyet> SECOND(4,6) da olduğu gibi beraberce belirtilir. <ondalık hassasiyet için 6 hane varsayılan değerdir.

<hassasiyet> ilk alan için tutulan rakamlar sayısıdır. Varsayılan değeri ise 2 dir.

<bitiş alanı>, eğer varsa, <bitiş alanı> ndan farklı olmalı ve onun sağ tarafında bulunmalıdır.

<değer dizisi> , timestamp sözcüğüne benzer. Sadece aralık niteliciyle olan ilişkisi hariçtir.

Veri tiplerinin özellikleri, aralık niteleyicisi ile tanımlanır. Aşağıda geçerli aralık sözcükleri örneklenmiştir.

INTERVAL '1990' YEAR(4)

1990 yılının bir aralığı; tarih değil bir zamanın bir periyodudur.

INTERVAL '-1-1' YEAR TO MONTH

Eksi bir yıl ve bir ay aralığı.

INTERVAL '+100:0:0:0.1' DAY(3) TO SECOND

100 gün ve saniyenin onda birini belirten aralık.

INTERVAL '0:0:0:0' HOUR TO SECOND(9)

0(sıfır) nanosaniyelik bir aralık.

INTERVAL '100000.001' SECOND(6,3)

100000 saniye 1 milisaniyelik aralık.

INTERVAL '-10' MINUTE

10 dakikalık aralık.

INTERVAL '1:1' HOUR(6) TO MINUTE

61 dakikalık bir aralık.

INTERVAL '1-0-0' YEAR TO DAY

Başlangıç ve bitiş alanları hem MONTH hem de DAY içeriyor.

INTERVAL '1' DAY TO DAY

Başlangıç ve bitiş alanları aynı.

INTERVAL '1000' MINUTE

Değer hassasiyetten (2'den) büyük.

INTERVAL '+100' DAY(3) TO SECOND

Yanlış alt-alan tanımlanmış.

BÖLÜM 4. VERİ TANIMLAMA

SQL dilinin veri tanımlama komutları; şema oluşturma ve onu oluşturan tablolar, görünüşler, tanım kümeleri, karşılaştırma tabloları gibi nesneleri oluşturma imkanını verir [2].

4.1 CREATE SCHEMA

Şema veri tabanı nesnelerinin tümüne birden verilen isimdir. Aşağıdaki gibi tanımlanır:

```
CREATE SCHEMA <şema yetkilendirme cümlesi>  
[<şema karakter küme tanımı>]  
[<şema elemanı>...]
```

Şema tanımının bu üç parçasının sadece ilki gereklidir; çünkü aynı anda şemanın ismi, şemanın sahibi ve kendisinin bulunacağı kataloğu tanımlamaktadır. Geri kalan elemanlar seçime bağlıdır. İkinci kısım, şemada açık olarak tanımlanmayan kolonlar tanım sahaları için karakter kümesini belirtir. Üçüncü kısım ise CREATE cümlesindekiler kadar çok eleman içerir. Komutlar buraya sonradan ilave edilebilir, silinebilir veya değiştirilebilir.

4.1.1 Yetkilendirme

<şema yetkilendirme cümlesi> şu şekilde yazılır:

<şema yetki cümlesi> ::= <şema ismi>

|AUTHORIZATION <yetki tanımlayıcısı>

|<şema ismi> AUTHORIZATION <yetki tanımlayıcısı>

<şema ismi> nitelenmiş bir isimdir. Katalog ismi şemada belirtilmezse, <yetki tanımlayıcısı> ile aynı kabul edilir. Eğer yetki tanımlayıcısı belirtilmezse, o anda işlem yapan kullanıcının kabul edilir.

Örnek:

CREATE SCHEMA ahmet

Bu tanım , 'ahmet' isminde bir boş şema yaratır. Sahibi ise komutu veren kullanıcıdır. Şema ismi nitelenmiş olduğundan, şema gösterim tanımlı katalogda yer alır.

Gösterim tanımlı katalog isminin 'CATALOG' olduğunu ve kullanıcının isminin 'ahmet' olduğunu kabul edersek, aşağıdaki tanımların hepsi birbirine eş olacaktır:

CREATE SCHEMA ahmet

CREATE SCHEMA AUTHORIZATION ahmet

CREATE SCHEMA ahmet AUTHORIZATION ahmet

CREATE SCHEMA CATALOG.ahmet

CREATE SCHEMA AUTHORIZATION ahmet

CREATE SCHEMA CATALOG.ahmet AUTHORIZATION ahmet

4.1.2 Şemanın Varsayılan Karakter Kümesi

<şema karakter küme tanımı> aşağıdaki gibidir:

DEFAULT CHARACTER SET <karakter küme ismi>

<karakter küme ismi> gösterim tanımlı olmalı veya kullanıcı tarafından tanımlanmalıdır.

Bu gösterim belirtilmezse, 'DEFAULT CHARACTER SET implementation_defined_character_set' yazılı olduğu varsayılır. Buradaki 'implementation_defined_character_set' gösterimde tanımlanmış karakter kümesidir.

4.1.3 Şema Elemanları

'CREATE SCHEMA' da yer alan son cümle <şema alanı> şema içindeki elemanları tanımlar. Bunlar birden fazla istenen sayıda yazılabilir. Şema elemanları olarak tanımlanabilen nesneler şunlardır:

tablolar	tanım kümeleri	görünümler
iddialar	karakter kümeleri	
karşılaştırma tabloları	çevirmeler	yetkiler

Bu objelerin yazımı gereği, CREATE SCHEMA cümlesi tüm diğer tanım cümlelerini kapsar. Başlangıçta şema tanımı boş olarak yapılabilir ve gerektiği zaman elemanlar ilave edilebilir. Fakat şema tanımının tutarlı olmasına dikkat edilmelidir. Tam bir şema tanımını şöyle yazabiliriz:

```
CREATE SCHEMA örnek AUTHORIZATION ahmet
      DEFAULT CHARACTER SET BASIC
(Bu tanımda sadece şema elemanları yoktur.)
```

4.2 CREATE TABLE :

Tablo tanım cümlesi olan CREATE TABLE, geçici yada kalıcı veri tabanı cümlelerini oluşturmak için kullanılır. Geçici tablolar da kalıcı tablolarla aynı özelliğe sahiptir; fakat içerikleri kalıcı değildirler. Kalıcı tablolar SQL dilinin temelini

oluştururlar. Tablo, bir veya daha fazla kolanların sıralanmış bir topluluğudur. Yazım şekli aşağıdaki gibidir:

```
CREATE [{GLOBAL| LOCAL}TEMPORARY] TABLE <tablo ismi>
      (<tablo elemanı>[,<tablo elemanı>]...))
      [ ON COMMIT { PRESERVE| DELETE } ROWS]
```

```
<tablo elemanı>::=<kolon tanımı> | <tablo kısıt tanımı>
```

Buradaki iki seçenekli tanım beraberce kullanılır. Birincisi olmadan ikinci tanım yapılamaz. Eğer GLOBAL ve LOCAL belirtilmiyorsa, tablo kalıcı olarak yaratılır. Diğer durumlarda kalıcı değildir. GLOBAL geçici tablonun bir kopyası bulunur ve bu programdaki bütün modüller tarafından kullanılabilir; LOCAL tablonun ise programın her modülü için ayrı ayrı kopyaları vardır.

"ON COMMIT" cümlesi işlem hatasız olarak bittiği zaman, geçici satırlara ne yapılacağını belirler. 'PRESERVE' verilirse, satırlar kalır; 'DELETE' verilirse her işlemten sonra tablo boşaltılır.

Tüm tablolar boş olarak yaratılır. Tablo ile beraber, her tablonun sahibi için yetki tanımlayıcıları da oluşturulur. Bu tanımlayıcılar işlem sırasında tabloya ulaşma haklarını belirler. Örneğimizdeki 'Bölüm' tablosunun iki özelliği vardır: İsim ve kod. Tabular formda veriler şöyle olabilir:

SAT	Satış
BİM	Bilgi işlem
...	...
EĞT	Eğitim

'Bölüm' tablosu şöyle tanımlanır:

CREATE TABLE Bölüm

```
( blmkod CHARACTER(3)
  blmadı CHARACTER(12) )
```

Bu tablo iki kolon içerir: blmkod ve blmadı. Kolonların tanım sırası önemli değildir. Bazı tanımlarda ise kolon sırası çok önemlidir. Birinci kolon, 'blmkod' deki veriler en fazla üç karakter; ikinci kolonda ise en fazla 12 karakter uzunluğunda olabilir. Sabit uzunluklu karakter kolonları boşluklarla doldurulur; yada yükleme, karşılaştırma ve bilgi alma sırasında takipli boşluklar yuvarlanır. Bu nedenle 'blmadı' nın 12 'den kısa olmasının önemi yoktur. Tabloyu yaratan kişinin, o tablo için tüm yetkileri vardır. Tablonun yapısı şöyledir:

Bölüm	
Kod	Adı
SAT	Satış
PER	Personel
...	...
BİM	Bilgi işlem

Sekil 4.1 Bölüm Tablosu

CREATE TABLE Personel

```
(
perno          NUMERIC (5),
ad             CHARACTER (40) NOT NULL,
adres          CHARACTER VARYING (200),
cins           CHARACTER NOT NULL,
ev_tlf         tlfno,
...
is_tlf         tlfno UNIQUE,
```

maas NUMERIC (6) NOT NULL,
 prim NUMERIC (6,2),
 arabalı CHARACTER (7),
 secmenler CHARACTER (4),
 CHECK (cins IN ('E' , 'K')),
 PRIMARY KEY (perno),
 CHECK ((arabalı, perno) MATCH
 (SELECT arabalı, perno FROM TASİT)),
 FOREIGN KEY (perno) REFERENCES Per_Ext,
 FOREIGN KEY (blmkod) REFERENCES Bölüm,
 FOREIGN KEY (secmenler) REFERENCES Secmenler)

4.2.1 Kolon Tanımı

Kolon tanım cümleleri ile tablonun kolonları tanımlanır. Kolonların yazıldıkları sıra temel sıralarıdır. Kolon tanımı şu şekilde yapılır;

<kolon adı> { <veri tipi> | <saha adı> }
 [<varsayılan cümle>]
 [<kolon kısıt tanımları> ...]
 [<karşılaştırma cümlesi>]

<saha adı>; <veri tipinin> ve <varsayılan cümle>, <kolon kısıt tanımları>, <karşılaştırma cümlesi> nin dolaylı olarak tanımı olarak düşünülebilir. Eğer kolon tanımı <varsayılan cümle> ve/veya <karşılaştırma cümlesi> içeriyorsa ve de <saha adı> belirtilmişse <kolon tanımı> geçerlidir. <karşılaştırma cümlesi> karakter tipli kolonlar için geçerlidir. [3]

4.2.1.1 Varsayılanlar

<varsayılan cümle>si , bir tabloya aktarılan satırda bir değer bulunmadığı takdirde, hangideğerleri alacağını belirtir. Yazım şekli şu şekildedir;

```
DEFAULT{
  <sözcük>
  |<tarikh-zaman değır fonksiyonu>
  | USER
  | CURRENT-USER
  | SESSION-USER
  | SYSTEM-USER
  | NULL }
```

NULL seçeneđi DEFAULT için varsayılan değerdır. USER, CURRENT-USER o anki yetki tanıtıcısının değırlerini içerir. SESSION-USER, oturum için olan yetki tanıtıcısıdır. <tarikh-zaman değır fonksiyonu> cümlesi sistemden gelen saat-tarih bilgisini içerir. <sözcük> ise sabit değırleri içerir.

Değırlerin yerine konullacak olan veri tipleri, kolonlar için tanımlanan veri tiplerine uyumlu olmalıdır. Sabit uzunluktaki dizilerde veya ikili veri sözcüklerde, değırlerin sađındaki boş kısımlar; 'boşluk' veya '0' larla doldurulur.

4.2.1.2 Kolon Kısıtları

Kolon kısıtları bir kolon içine gelebilecek değırleri kontrol etmek için kullanılır. Birden fazla kolonu kısıtlayarak kontrol etmek için tablo kısıtları kullanılmalıdır. 4 çeşit kolon kısıtları vardır;

NOT NULL	'boş olamaz' kısıtı
UNIQUE	'tek olmalı' kısıtı
FOREIGN KEY	'yabancı anahtar' kısıtı
CHECK	'kontrol' kısıtı

Bu kısıtlar şu şekilde belirtilir;

```
<kolon kısıtı> ::= [ CONSTRAINT <kısıt adı> ]
{ NOT NULL
| <tek olmalı tanımlaması>
| <referans kısıtı>
| <kontrol kısıt tanımı> }
[ <kısıt özellikleri>]
```

```
<kısıt özellikleri> ::=
[NOT] DEFERRABLE [ INITIALLY IMMEDIATE |
INITIALLY DEFERRED ]
| { INITIALLY IMMEDIATE | INITIALLY DEFERRED }
[ [ NOT ] DEFERRABLE ]
```

Burada seçime bağlı olarak kullanılan <kısıt adı> daha sonra kullanılacak kısıtlara referans olması için tanım bilgilerinde mevcut olan bir tekrar etmeyen kimlik tamını sağlamak içindir. Eğer bu belirtilmezse VTYS tarafından buna bir kısıt ismi verilecektir. Aksi belirtilmedikçe, kısıtlar icra edilen her SQL komutundan sonra kontrol edilirler. Kısıt INITIALLY IMMEDIATE (belirtilmezse ilk değer bu alınır) , yada INITIALLY DEFERRED olarak tanımlanabilir. Bu her işlem başlangıcında, hangi modun alınacağını belirler. Belirlenen bu mod, <set constraints mode komutu> yani kısıt modunu değiştirme komutu ile değiştirilebilir. Eğer herhangi bir mod belirtilmediyse ve INITIALLY DEFERRED durumu belirtilmediyse, NOT DEFERRABLE kabul edilir. [1]

NOT NULL (Boş Olamaz Kısıtı)

'Boş olamaz' kısıtı, NOT NULL anahtar kelimesiyle, kolon tanımında yapılır. Bu kısıtın belirtilmesi ile birlikte o alan mutlaka geçerli bir değer almak zorundadır. Buna örnek verecek olursak;

Örnek 1;

```
CREATE TABLE Personel
(
perno          NUMERIC (5) Personel_PK PRIMARY KEY,
ad             CHARACTER (40) NOT NULL,
adres         CHARACTER VARYING (200),
cins           CHARACTER NOT NULL,
ev_tlf        tlfno,
is_tlf        tlfno CONSTRAINT Per_is_tlfno_U UNIQUE,
maas          NUMERIC (6) NOT NULL,
prim          NUMERIC (6, 2),
)
```

Personel örneğinde olduğu gibi, ad, cins ve maas kolonları bir değer almadıkça herhangi bir kayıt depolanamaz. Diğer kolonlaradeğer girilip girilmediği önemli değildir. Kolonun NOT NULL ve NULL değer alıp almayacağına karar vermek, verinin mantıksal karakteristiğine bağlıdır. Fakat bazı durumlarda NULL kullanımı için ek bir depolama alanına ihtiyaç vardır ve bu yüzden mantıklı olarak NULL değerini kullanmak gerektiği bir yerde performansı düşürmemek için NOT NULL kullanmak da gerekebilir.

NOT NULL tanımı, tablo kısıtı olarak aşağıdaki ifadeye eşdeğerdir;
CHECK (kolon IS NOT NULL);
kolon ::= NOT NULL 'ın bulunduğu kolon ismi.

UNIQUE (Tek Olmalı Kısıtı)

'Tek olmalı' kısıtı, kolon sonuna ilave edilen PRIMARY KEY veya UNIQUE anahtar kelimesi vasıtasıyla yapılır. Örnek 1 de görüldüğü gibi perno, is_tlf kolonları bu tanım sayesinde başka satırlarda olmayan değer almak zorundadır. PRIMARY KEY, NOT NULL UNIQUE için bir kısa yazım şeklidir. UNIQUE ve NOT NULL UNIQUE tablo tanımında, gerekli oldukça istenen sayıda kullanılabilir. Dikkat edilirse, UNIQUE ve PRIMARY KEY tanımı aşağıdaki tablo kısıtına eşdeğerdir;

UNIQUE (kolon)

PRIMARY KEY (kolon)

Her iki durumda da , NOT DEFERRABLE veya DEFERRABLE kullanılabilir. Buna örnek verecek olursak;

blmkod CHARACTER(3) PRIMARY KEY şuna eşdeğerdir;

blmkod CHARACTER (3), PRIMARY KEY (blmkod)

is_tlf tlfno UNIQUE, ise tablo kısıtı olarak tanımlandığında

UNIQUE (is_tlf)

olarak tanımlanır.

REFERANSLAR

Referans cümlesi, iki tablonun, yabancı anahtar kısıtı vasıtasıyla mantık bağlantısına izin verir. Bu tanım kolon tanımının sonuna ilave edilen REFERENCES kelimesiyle yapılır. Yabancı anahtar kısıtı, boş olmayan bir kolondaki her değer referans gösterilen tablodaki başka bir kolondaki değerlere sahip olacağını

anlatmaktadır. Kendisine referans edilen tablonun ilgili kolonu 'tek olmalı' kısıtına sahip olmalıdır. Örnek 1 deki tabloya personelin hangi bölümde çalıştığına ait bilgiyi gösterecek olan 'blmkod' kolonu eklenecek olursa, bu kolon 'Bölüm' tablosunu referans olarak FOREIGN KEY olacaktır. Bu durumda Personel ve Bölüm tabloları şu hali alacaktır;

Örnek 2;

```
CREATE TABLE Bölüm
```

```
(
  blmkod      CHARACTER (3) PRIMARY KEY,
  blmno       CHARACTER (12)
)
```

```
CREATE TABLE Personel
```

```
(
  perno       NUMERIC (5) Personel_PK PRIMARY KEY,
  ad          CHARACTER (40) NOT NULL,
  adres       CHARACTER VARYING (200),
  cins        CHARACTER NOT NULL,
  ev_tlf      tlfno,
  blmkod      CHARACTER (3) REFERENCES Bölüm(blmkod)
  is_tlf      tlfno UNIQUE,
  maas        NUMERIC (6) NOT NULL,
  prim        NUMERIC (6, 2),
)
```

Bu kısıtın tam formatı ise şöyledir;

```
REFERENCES <tablo ismi> [ ( referans kolon listesi ) ]
[ MATCH { FULL | PARTIAL } ]
[ <yapılacak iş> ]
```

Buradaki <referans kolon listesi> örnek.2' deki 'blmkod' dur. Kullanımı kendisine referans yapılan tablodaki yeri hariç seçime bağlıdır. Bu kolonun tanımı şu şekilde de yapılabilir; blmkod CHARACTER (3) REFERENCES Bölüm.

Yabancı anahtar kısıtı tanımlaması ' REFERENCES ... ' tablo kısıt tanımı olarak aşağıdaki şekilde de ifade edilebilir;

FOREIGN KEY (kolon) REFERENCES

CHECK (Kontrol Kısıtı)

Bu kontrol kısıtı için kolon tanımının sonuna CHECK anahtar kelimesi ilave edilir. Örnek.2 deki Personel tablosundaki 'cins' kolonu peronelin cinsiyetini belirten değeri içerir. Cinsiyet kadı veya erkek olarak sınırlı değer içerdiğinden ve bundan başka değer alamayacağından bu kolona aşağıdaki şekilde bir kontrol ekleyebiliriz;

cins CHARACTER NOT NULL CHECK (cins IN ('K', 'E')),

Bu Kısıt tanımının tam olarak yazım biçimi şu şekildedir;

CHECK (<arama şartları>)

Bu gösterim tablo kısıt tanımında değiştirilebilirliğinin belirtecini de içererek aynı şekildedir.

4.2.1.3 Karşılaştırma

<karşılaştırma cümlesi>, kolondaki verilerin karşılaştırılması için, karakter karşılaştırma dizisinin kullanılacağını belirtir. Bu karşılaştırma, tahmin etmede, karakter ifadelerde, cümlelere göre gruplama da kullanılır.

Yazım şekli: COLLATE <karşılaştırma ismi>

<karşılaştırma ismi> ::= gösterimde belirtilen karşılaştırma ismi veya CREATE COLLATION komutu ile tanımlanan isim.

Eğer bir karakter tipli kolon için, karşılaştırma cümlesi tanımlanmazsa, o veri için varsayılan karşılaştırma dizisi kullanılır.

4.2.2 Tablo Kısıtları

Kolon kısıtlarının, kolon değerlerini kısıtlamalarına karşın, tablo kısıtları da çeşitli kolonların değerlerini beraberce kontrol ederler. Daha evvelki bölümde de görüldüğü gibi bütün kolon kısıtları aynı zamanda tablo kısıtı olarak da yazılabilmektedir. Tablo kısıtının yazım şekli aşağıdaki gibidir:

```
<tablo kısıt tanımı> ::= [ CONSTRAINT <kısıt adı> ]
{ <tek olma kısıt tanımı>
  | <referans kısıt tanımı>
  | <kontrol kısıt tanımı> }
[ <kısıt özellikleri> ]
```

Burada <kısıt adı> kısıtın ihlal edildiğini rapor etmek için tanım bilgisinde bulunan ve daha sonra kullanılacak tek olan kısıt kimliğidir.

Aksi belirtilmedikçe; kısıtlar her SQL komutu icra edildikten sonra kontrol edilirler. Tablo kısıtları özellikleri ile birlikte tanımlanabilirler. Bu da zaten kolon üzerindeki kısıtlarla aynı anlam ifade etmektedir.

CHECK ve FOREIGN KEY kısıtları, tanımlandıkları tablodan başka tabloya referansa izin verirler. Şayet kendisine referans yapılan tablo, 'ON COMMIT DELETE ROWS' ile tanımlanmışsa, referans yapan tablo da aynı tanımı içermelidir.

4.2.2.1 UNIQUE (Tek Olma Kısıtı)

'Tek olmalı' kısıtının yazım şekli aşağıdaki gibidir;

<tek olmalı kısıt tanımı> ::=

<tek olmalı tanımı> (<kolon adı> [{ , <kolon adı> } ...])

<tek olmalı tanımı> ::= UNIQUE | PRIMARY KEY.

Burada dikkat edilmesi gereken, kolon adları tekrar edilmemelidir.

Aynı kolon isimleri, başka bir 'tek olmalı' kısıtında kullanılamaz. 'tek olmalı' kısıtı, o satırın aynı tablodaki başka bir değerle aynı olmasını önlemektedir. UNIQUE tanımlı kolonlarda satırlar boş -NULL- değer alabilirler, çünkü NULL değer, gerçek bir değer değildir. Tam kontrol için yazım şekli şu şekilde olmalıdır;

CHECK (UNIQUE (SELECT unique_kolon_listesi FROM T)

AND

(unique_kolon_listesi) IS NOT NULL)

Örnek;

UNIQUE (plaka_no)

PRIMARY KEY (perno) (Bu iki örnek de Personel Tablosundan)

UNIQUE (perno, İstek_trh) (Guzergah tablosundan)

4.2.2.2 Referanslar

Referans tanımı, iki tablo arasında yabancı anahtar ilişkisiyle bir ilişki kurmaktır. Bir başka deyişle, bu tanımdaki kolonlar listesindeki her değer, referans yapılan kolonlarla (aynı tablo veya farklı tabloda olabilir) aynı değerlere sahip olmalıdır. Referans yapılan tablodaki kolon tek olma kısıtına sahip olmalıdır, bir başka deyişle referans yapılan tabloda, sadece bir ve yalnız bir tane ilgili satır bulunabilir. Tablo referanslarını tanımlarken aşağıdaki format kullanılabilir;

FOREIGN KEY (<referans yapan kolonlar>)

REFERENCES <referans yapılan tablo ve kolonlar>

[MATCH { FULL| PARTIAL }]

[<referans durumunda yapılacak iş>]

<referans yapılan kolonlar> ::= <kolonadı> [{ , <kolon adı> } ...]

<referans yapılan tablo ve kolonlar> ::= <tablo adı>

[(<referans yapılan kolonlar>)] <referans yapılan kolonlar>
<kolon adı> [{ , <kolon adı> ...]

<referans durumunda yapılacak iş> ::=

<güncelleme kuralı> [<silme kuralı>]
| <silme kuralı> [<güncelleme kuralı>]

<güncelleme kuralı> ::= ON UPDATE <referans durumundaki iş>

<silme kuralı> ::= ON DELETE <referans durumundaki iş>

<referans durumundaki iş> ::=

{ CASCADE | SET NULL | SET DEFAULT | NO ACTION }.

Referans yapılan kolonlar, NOT DEFERRABLE özelliğine sahip 'tek olmalı' kısıt olmalıdır. Eğer <referans yapılan kolonlar> ihmal edilirse, referans yapılan tablonun birincil anahtarı bulunmalıdır ve kolon isimleri de referans kolonlarından alınmalıdır. Kolon isimlerinin hepsi aynı veri tipine sahip olmalıdır.

MATCH ilave bir sınırlayıcıdır. MATCH FULL seçeneği kullanıldığında referans yapan kolonlarda tüm değerler 'null' yada gerçek olmalıdır. MATCH PARTIAL seçeneği ise eğer herhangi bir kolon boş değere sahip değilse, diğer boş olmayan değerler doğru olarak konmuş ise, referans yapılmış tablodaki en az bir satır, kısıtlamayı doğrulamalıdır.

ON DELETE : Referans yapan tablonun herbir satırındaki aynı değere sahip referans yapılan tablodan, karşı gelen satırın silinmesi durumunda ortaya çıkan

referans kısıtına ait bir işlemdir. ON DELETE sağ tarafına alacağı referans işlerine göre aşağıdaki işleri yapmaktadır;

CASCADE : Eşleşen satırlar silinir.

SET NULL : Eşleşen satırlar NULL değerine sahip olurlar.

SET DEFAULT : Eşleşen kolonlar, her kolon için tanımlanmış varsayılan değeri alırlar.

NO ACTION : Hiç bir işlem yapılmaz. Şayet ON DELETE tanımlanmamışsa, eşleşen değerler için hiç bir işlem yapılmaz yani varsayılandır.

ON UPDATE: Referans yapan tablonun herbir satırındaki aynı değere sahip referans yapılan tablodan, karşı gelen satırın farklı bir değer ile güncellenmesi sırasında ortaya çıkan referans kısıtına ait işlemdir. (İki tablodaki satırların eşleşmesi güncelleme işleminden önce yapılıyor.) ON UPDATE sağ tarafına alacağı referans işlerine göre aşağıdaki işlemleri yapmaktadır;

CASCADE : Eşleşen satırlar, referans yapan satıra karşı gelen referans yapılan tablonun ilgili kolonundaki değeri alırlar.

SET NULL : Referans yapan kolonların referans yapılan kolonlardaki eşleşen satırlar, NULL değerini alırlar. Eğer bunun yanısıra MATCH FULL verilmişse, tüm referans yapan kolonlar boşaltılır.

SET DEFAULT : Referans yapandaki eşleşen satırlar, her kolon için belirtilen varsayılan değeri alırlar.

NO ACTION : Hiç bir işlem yapılmaz. Şayet ON UPDATE tanımlanmamışsa, eşleşen değerler için hiç bir işlem yapılmaz yani varsayılandır.

Aşağıda , referans kısıtları ve yapılacak işler konusunda üç adet tablo tanım örneği verilmiştir. İkinci tabloda, diğer iki tablo üzerine referans kısıtları vardır. İlk tablo tanımında, referans yapılan kolonların PRIMARY KEY olarak referans yapılan kolonların listesinin kullanımı açık ve kesin olmaksızın anlatılmaya çalışılmıştır. İkinci tablo tanımında ise; referans yapılan kolonların listesi açık bir şekilde kullanılmıştır. İlk durumda yapılacak işler, bir sipariş silindiğinde ilgili bütün sipariş hattı silinmektedir, veya herhangi bir sipariş numarası güncellendiği zaman sipariş hattı güncelenen değere bağlı olarak değer almaktadır. İkinci durumda ise, hiç

bir işlem yapılması istenmiştir, dolayısıyla siparişte referans olarak gösterilen hiçbir ürün silinmeyecek, hiç bir ürün kodu değişmeyecektir.

Örnek 3.

```
CREATE TABLE Siparis
```

```
[
```

```
  sipno          NUMERIC (5),
```

```
  teminno        NUMERIC NOT NULL,
```

```
  cıks_trh       DATE NOT NULL,
```

```
  PRIMARY KEY (sipno)
```

```
)]
```

```
CREATE TABLE Siparis_Hattı
```

```
(
```

```
  sipno          NUMERIC (5),
```

```
  siphat_no       NUMERIC (5),
```

```
  urun_kod        NUMERIC NOT NULL,
```

```
  miktar          NUMERIC NOT NULL,
```

```
  PRIMARY KEY (sipno, siphat_no),
```

```
  FOREIGN KEY (sipno ) REFERENCES Siparis
```

```
    ON DELETE CASCADE
```

```
    ON UPDATE CASCADE,
```

```
  FOREIGN KEY (urun_kod) REFERENCES Urun (urun_kod)
```

```
)]
```

```
CREATE TABLE Urun
```

```
(
```

```
  urun_kod        NUMERIC (5) UNIQUE,
```

```
  ad              CHARACTER (20=,
```

```
  PRIMARY KEY (ad:)
```

```
).
```

CHECK (Kontrol Kısıtı)

Kontrol cümlesi, tablodaki kolonları bulunduracakları değerleri kontrol etmek için gerekli şartları içermektedir. Yazım şekli aşağıdaki gibidir;

CHECK (<arama şartları>)

<arama şartı> zamanla değişen herhangi birşey içeremez yada çeşitli zamanda tanımsız değerler olmamalıdır (USER, CURRENT_USER, SESSION_USER, SYSTEM_USER, Tarih-zaman fonksiyonları CURRENT_DATE gibi). Ayrıca herhangi bir sorgulama da içeremez (DISTINCT, GROUP BY, MIN, MAX, INTERSECT, EXCEPT, UNION vb.).

Örnek verilirse;

CHECK ((SELECT COUNT (*) FROM Siparis) < 100)

tablo kontrol kısıtı doğrudur, fakat;

CHECK (COUNT (*) < 100) geçersizdir.

Kontrol kısıtı , şartı yanlış yapan bir kayıt (satır) yok ise doğrulanır. Fakat her satır için, kısıtın doğru olmasına rağmen , doğru olmak zorunda değildir . Şayet tablo boş ise, bir kontrol kısıtı her zaman doğrulanır. Bu nedenle tablo üzerindeki minimum çokluk kısıtları sınıflandırma olarak tanımlanmalıdırlar.

CHECK ((SELECT COUNT(*) FROM BOLUM > 1)

şu şekilde tanımlanmaktadır,

CREATE ASSERTION C

CHECK (NOT EXISTS (SELECT * FROM T

WHERE NOT (SC)))

c ::= Kontrol sınırlayıcısı

T ::= Tablo

SC ::= Arama şartı

Varsayalım Tasıt tablosunda bir kontrol cümlecği olsun ;

```
CREATE TABLE Tasıt;
(
  plaka_no          NUMERIC (5),
  perno             NUMERIC (5),
  makbuz_trh       DATE NOT NULL,
  PRIMARY KEY (plaka_no),
  CHECK ( ( perno, arabalı) MATCH
           ( SELECT perno, arabalı FROM Personel ) )
).
```

4.3 CREATE DOMAIN

Tanım kümesi, izin verilen değerlerden oluşan kümedir. Bir kolondaki her değer, bir tanım kümesine ait olmak zorundadır. Tanım kümesi, veri tipleri olarak yeni tanım kümeleri oluşturulmasına izin verir. Yazım şekli şöyledir;

<tanım kümesi tanımlama> ::=

CREATE DOMAIN <tanım kümesi adı> [AS] <veri tipi>

[<varsayılan cümle>]

[<tanım kümesi kısıtı>]

[<karşılaştırma cümlesi>]

<tanım kümesi kısıtı> ::=

[CONSTRAINT <kısıt adı>]

<kontrol kısıt tanımı>

[<kısıt özellikleri>]

Tanım kümesi ismi, kolon tanımında veri tipinin yerinde kullanılabilir. Her tanım kümesinin kullanıcısı için bir USAGE yaratılır. Bu yetki, ancak ve ancak, eğer kullanıcının hak verilebilir yetkileri mevcut ise REFERENCES değiştirilebilir.

Örnek 4.

```
CREATE DOMAIN tlfn AS CHARACTER (14)
```

```
  CHECK (    VALUE LIKE '+%( )%-%%-%'
           OR VALUE LIKE '+%-%%-%%-% '
           OR VALUE LIKE '%-%%'
           OR VALUE LIKE '---' )
```

```
CREATE DOMAIN makbuz_trh AS DATE DEFAULT
                                CURRENT_DATE
```

4.3.1 Varsayılanlar

Tanım kümesi için varsayılan değer tanımı, kolonlarla eş değerdir. Eğer bir kolon için varsayılan tanımlanmışsa, bu tanım kümesi varsayılanın üzerinde geçerlidir.

4.3.2 Karşılaştırma

Tanım kümesi için karşılaştırma değer tanımı, kolonlarla aynıdır. Eğer bir kolon için karşılaştırma tanımlandıysa bu tanım kümesi karşılaştırılmasının üzerinde geçerlidir.

4.3.3 Tanım Kümesi Kısıtları

Tanım kümesi kısıtları, kolon kontrol kısıtlarına eşdeğerdir. ancak bazı durumlar dışında: Bir arama şartı, tanım kümesine referans içeren tanımların isimlerini

içeremez, bir başka deyişle dairesel tanım yapılamaz. anahtar kelime VALUE kolon ismini temsil etmek için kullanılır.

4.4 CREATE VIEW

Bu cümle bir tablo tanımlar. Fakat bu tablo fiziki değildir, sadece bir görünüştür ve kalıcı olan diğer nesnelerden oluşturulur. Bu cümlemin yazım formatı şu şekildedir;

```
CREATE VIEW <tablo adı> [ ( <görünüş kolon listesi> ) ]
    AS <sorgulama ifadesi>
    [ WITH [ LOCAL | CASCADED ] CHECK OPTION ]
```

<görünüş kolon listesi> ::= >kolon adı> [[, <kolon adı>] ...].

Görünüş tanımı asla kendine referans yapamaz. Görünüş kolon listesi; eğer sorgulama ifadesinden çıkan tüm kolonların ismi belirtilmişse bu isimler tek olmalıdır (bu isimler seçime bağlıdır). Aksi takdirde görünüş kolon listesi belirtilmeli ve sorgulama ifadesindeki her kolon isimlendirilmelidir.

WITH CHECK OPTION , şayet görünüş güncellenebilir ise tanımlanabilir. Bir görünüş ise, sorgulam ifadesi yenilenebilir olduğunda yenilenebilir. Eğer görünüş yenilenebilir ise, ve kullanıcı, aynı haklara görünüş içinde sahip olur.

Örnek 5.

```
CREATE VIEW Bim_tlf_lst ( ad, tlfno, gorev_no )
    AS
    SELECT ad, 1s_tlf, perno
    FROM Personel
    WHERE blmkod = 'BIM'
```

Eğer kullanıcının Personel tablosu üzerinde, 'açma' ve 'seçme' hakları, is_tlf ve perno kolonları üzerinde de 'güncelleme' hakları olduğunu varsayarsak; kullanıcı görünüm tablosu üzerinde de aynı haklara sahip olacaktır. Genel olarak WITH CHECK OPTION tanımlanmışsa eğer, sonuç tablosunda 'açma' ve 'güncelleme' yapılamaz.

Örnek 6.

```
CREATE VIEW Non_bim_tlf_lst (ad,tlf no, gorev_no, blmkod)
AS
SELECT ad,is_tlf, gorev_no
FROM Personel
WHERE blmkod <> 'BIM'
WITH CHECK OPTION.
```

Şayet bir görünüş başka bir görünüş tanımı içeriyorsa, alt seviyedeki görünüşte WITH CHECK OPTION varsa, üstteki tanım bu seçeneği içermese bile hiç bir işlem yapılmaz.

Bir görünüş kalıcı bir nesne olmasına rağmen, görünüşün yapısı , görünüş yaratıldığı zaman saptanır. Bunu örnekleyecek olursak, <sorgulama ifadesi>

' SELECT * FROM T ...' , bu sorgulama ifadesi T tablosundan bütün kolonları görünüş tablosuna alır. Eğer tabloyu sonradan kolon eklenmişse bunlar görünüşe, yeniden yaratılıncaya ,silininceye kadar yansımaz.

4.5 CREATE ASSERTION

'assertion', tablo tanımı ile doğrudan bağlanamayan bir bütünlük kısıtıdır. Bu kısıtın yazım formatı aşağıdaki şekildedir;

```
CREATE ASSERTION <assertion adı> CHECK ( <arama şartı >
[< kısıt özellikleri> ].
```

Assertion, özellikler beraber tanımlanabilir ve bu tanım kolon kısıtları ile aynı anlama gelmektedir. Bir assertion ile ana tablodaki kontrol kısıtı arasında küçük bir fark vardır; assertion ile geçici tablolara referans yapamazlar. Assertion kullanılması daha temiz ve daha ulaşılabilir bir tanım içindir. Ayrıca minimum çokluk kısıtlarını kullanmak gerektiğinde, bu tablo kısıtları ile kontrol edemeyecektir.

Örnek 6.

```
CREATE ASSERTION UNIQUE_CONSTRAINT_NAME
CHECK(1 =
    (SELECT MAX(OCCURANCES)
     FROM (SELECT COUNT(*) AS OCCURRENCES
           FROM (SELECT CONSTRAINT_CATALOG,
                     CONSTRAINT_SCHEMA,
                     CONSTRAINT_NAME
                  FROM DOMAIN_CONSTRAINTS
                 UNION ALL
                 SELECT CONSTRAINT_CATALOG,
                     CONSTRAINT_SCHEMA,
                     CONSTRAINT_NAME
                  FROM TABLE_CONSTRAINTS
                 UNION ALL
                 SELECT CONSTRAINT_CATALOG,
                     CONSTRAINT_SCHEMA,
                     CONSTRAINT_NAME
                  FROM ASSERTIONS)
          GROUP BY CONSTRAINT_CATALOG,
                   CONSTRAINT_SCHEMA,
                   CONSTRAINT_NAME ) ) ).
```

Buradaki assertion üç tablo üzerine kontrol sağlamaktadır. Assertion , bir ismin, herhangi bir tablo kısıtında , tanım kümesi kısıtında birden fazla kullanılmamasını sağlamaktadır.

4.6 CREATE CHARACTER SET

Bu cümle, karakterlerin mantık kümesini ve bu karakterlerin fiziki kodlama biçimini tanımlar. Bu tanım karakter repertuarı ve kullanım şeklinden oluşur. Bunun yazım formatı aşağıdaki gibidir;

```
CREATE CHARACTER SET <karakter küme adı>
[ AS ] GET <karakter küme adı>
[ COLLATE <karşılaştırma adı>
| COLLATION FROM <karşılaştırma kaynağı> ]
```

Buradaki ikinci karakter küme adı, bu küme için alınacak temel kümeyi belirtir. Karşılaştırma kaynağı , karşılaştırma tanımında olan tüm alternatifleri sağlar. İkinci kümede tanımlı karakterlerle yeni küme yaratılır. Şayet hiç birşey tanımlanmazsa, COLLATION FROM DEFAULT tanımlı sayılmaktadır.

```
CREATE CHARACTER SET Rev_ASCII AS GET ASCII
COLLATION FROM DESC(ASCII).
```

4.7 CREATE COLLATION

CREATE COLLATION cümlesi, karakter dizilerini karşılaştırmak için gerekli kuralları belirler. Bu kurallar iki dizinin eşit olup olmadığını belirlemeye, sayet değilse hangi dizinin önce geleceğini belirlemeye yarar. Bir karakter kümesi için, çeşitli karşılaştırma dizileri bulunabilir. Karşılaştırma tanımı şu şekilde yapılmaktadır;

```
CREATE COLLATION <karşılaştırma adı>FOR<karakter küme adı>
FROM <karşılaştırma kaynağı> [ NO PAD | PAD SPACE ]
<karşılaştırma kaynağı> ::= DEFAULT
| <karşılaştırma adı>
| DESC ( <karşılaştırma adı> )
| EXTERNAL ( ' <harici karşılaştırma adı> ' )
| TRANSLATION <çeviri adı>
[ THEN COLLATION <karşılaştırma adı> ]
```

Harici karşılaştırma adı, gösterim tarafından desteklenmelidir. FROM bölümünde belirtilen karşılaştırma adı, karakter kümesinde mutlaka tanımlanmalıdır. Prensip olarak, kullanıcının harici karşılaştırma adı üzerinde USAGE hakkı olmalıdır. Fakat gösterimden gelen tanımlarda karşılaştırma adları PUBLIC olduğundan bu pek önem taşımaz.

Karşılaştırma tanımının kullanıcısı için, USAGE yetki tanıtıcısı yaratılır. Bu tanıtıcı, ancak ve ancak kullanıcının ilgili karakter kümesi üzerinde USAGE hakkı olduğu durumlarda değişebilir.

FROM ile tanımlanan karşılaştırma kuralı şu şekildedir;

DEFAULT: Karşılaştırmak için karakter repertuvarındaki liste kullanılır.

<karşılaştırma adı> Karşılaştırma, <karşılaştırma adı> tarafından tanımlandığı gibi yapılır

DESC : Parantez içinde görünen <karşılaştırma adı> azalan sırada kullanılarak yapılır.

EXTERNAL: Karşılaştırma, başka bir karşılaştırma isminde olduğu gibi yapılır.

TRANSLATION: Karakter dizisi; belirtilen çeviri adı kullanılarak önce çevrilir, sonra (THEN COLLATION <karşılaştırma adı> kullanılmamışsa) varsayılan karşılaştırma kümesi kullanılarak sıralanır.

NO PAD | PAD SPACE seçenekleri ile, eşit olamayan dizilerin sağındaki boşlukların ne olacağına karar verilir. Varsayılan <karşılaştırma kaynağı>nın adının seçimi ile aynıdır; aksi halde PAD SPACE varsayılandır. Örnek;

CREATE COLLATION Reverse_ASCII for SQL_TEXT FROM

DESC (ASCII)

CREATE COLLATION Sıralı FOR Kume FROM

EXTERNAL (' EBCDIC')

CREATE COLLATION tan_sıralı FOR Kume FROM

DEFAULT NO PAD.

4.8 CREATE TRANSLATION

Bu tanım ile, bir karakter kümesine çeviri sağlanır. Bu tanımın yazım formatı verilecek olursa;

```
CREATE TRANSLATION <çeviri adı>  
FOR <kaynak karakter küme adı>  
TO <hedef karakter küme adı>  
FROM { IDENTITY  
| <çeviri adı>  
| EXTERNAL ( 'harici çeviri adı' ) }
```

IDENTITY tanımı; karakterlerin değişikliğe uğramadan çevirilmesini sağlamaktadır. Aksi takdirde; karakterler, gösterimde tanımlı kurallara göre çevrilirler. Örneklenecek olunursa;

```
CREATE TRANSLATION Invert FOR ASCII TO EBCDIC  
FROM EXTERNAL ( 'ASCII_EBCDIC' ).
```

BÖLÜM 5. ŞEMA İŞLEMLERİ

Şema, birkez tanımlandıktan sonra, ona çeşitli kısımlarını ilave etmek, silmek yada güncellemek mümkündür. Eklemeler CREATE TABLE; CREATE VIEW gibi komutlarla olmaktadır. Silmek için DROP , güncellemek için de ALTER komutları ile işlem yapılmaktadır.

Bir şemanın, herhangi bir bölümünü işleme sokmak o anki yetkinin; şemanın sahibi ile aynı olmasını gerektirir. Bir başka deyişle, CREATE SCHEMA komutundaki AUTHORIZATION <yetki tanımlayıcısı> satırı ile belirtilen yetkiye sahip olmalıdır. [2]

5.1 DROP Komutları

DROP komutlarının çoğunda işlemin CASCADE olup olmadığı açıkça belirtilmelidir. Şayet böyle bir belirteç kullanılmadığı zaman non-cascading varsayılandır. Non-cascading komutlarda 'RESTRICT' alternatifi belirtildiyse, silinmek istenen nesnelere bağlı başka nesneler varsa, komut çalışmaz ancak CASCADE belirtilirse nesneler silinir. BU nedenle CASCADE dikkatli kullanılmalıdır. RESTRICT alternatifi ise varsayılandır.

5.1.1 DROP SCHEMA

DROP SCHEMA komutu ile, tüm şema dolayısıyla ilgili bütün ver ve tanımlar silinmektedir. Bu komutun formatı şu şekildedir;

DROP SCHEMA <şema adı> {RESTRICT | CASCADE } .

Eğer RESTRICT belirtildiyse şema mutlaka boş olmalıdır, hiçbir şekilde herhangi bir tanım kümesi, tablo tanımları, görüntü tanımları, assertion tanımları, karakter küme tanımları , karşılaştırma veya tercüme tanımları içermemelidir. Eğer Sirket şemasına DROP SCHEMA Sirket RESTRICT komutu silemez, çünkü bu şema bir çok tanım, tablo vb. içermektedir. Eğer CASCADE belirtilmişse, aşağıdaki işlemler sırayla yapılmalıdır;

1. Şemadaki her t tablosu için DROP TABLE t cascade
2. Şemadaki her v örüntüsü için DROP VIEW v CASCADE
3. Şemadaki her d tanım kümesi için DROP DOMAIN d CASCADE
4. Şemadaki her a assertion için
DROP ASSERTION a CASCADE
5. Şemadaki her co karşılaştırması için
DROP COLLATION co CASCADE
6. Şemadaki her tr çevirisi için
DROP TRANSLATION tr CASCADE
7. Şemadaki her c karakter küme tanımı için
DROP CHARACTER c CASCADE

Eğer bu komutlardan herhangi bir tanesi icra edilemezse o zaman DROP SCHEMA işlemi de icra edilemez ve verilerde, tanımlarda herhangi bir değişiklik olmaz. Şayet hiçbir sorunla karşılaşılmadan komut icra edilirse, ilgili tüm veriler ve tanımlar silinecektir.

5.1.2 DROP DOMAIN

DROP DOMAIN komutu ile bir tanım kümesi silinmektedir. Yazım şekli şöyledir;

DROP DOMAIN <tanım kümesi adı> { RESTRICT | CASCADE }

Şayet RESTRICT kullanılırsa tanım kümesi herhangi bir tablo tanımında, görüntü tanımında vb. kullanılamaz, buna örnek verecek olursak;

DROP DOMAIN tlfno RESTRICT komutu başarısız olur , çünkü tlfno tanım kümesi Personel tablosunda daha evvel kullanılmıştı.

Eğer CASCADE belirtildiyse, aşağıdaki işlemler sırasıyla yapılmalıdır;

1. Tanım kümesi ile belirtilen her kolon için; kolon tanımı şu şekilde değiştirilir;

a-) Tanım kümesi ismi; tanım kümesinin veri tipi tanımıyla değiştirilir.

b-) Eğer tanım kümesi bir kısıt almışsa, buna eşdeğer bir kolon kısıtı ilave edilir. Bu kısıt ancak ve ancak kullanıcının böyle bir kısıt yaratma hakkının varlığıyla mümkündür.

c-) Şayet kolonun, varsayılan bir cümlesi yoksa, tanım kümesinde varsayılan cümle ilave edilir.

d-) Eğer kolonun, karşılaştırma cümlesi yoksa, tanım kümesindeki karşılaştırma cümlesi ilave edilir (kullanıcının yaratma yetkisi olmalıdır) .

2. REVOKE USAGE ON DOMAIN d FROM u CASCADE komutu icra edilmelidir (d: tanım kümesi u: yetki tanıtıcısı) . Son olarak, Bilgi Şemasındaki tanım kümesi ve veriler yok edilmelidir. Daha sonra aşağıdaki komut tanım kümesini siler;

DROP DOMAIN tlfno CASCADE

Bu durumda Personel tablosundaki ev_tlf ve is_tlf kolonlarının tanımı aşağıdaki şekilde olacaktır;

ev_tlf CHARACTER (14)

CHECK (ev_tlf LIKE '+%()%-%-%'

OR ev_tlf LIKE '+%-%-%-%')

OR ev_tlf LIKE '%-%'

OR ev_tlf LIKE '—'),

is_tlf CHARACTER (14)

CHECK (is_tlf LIKE '+%()%-%-%'

OR is_tlf LIKE '+%-%-%-%' ' .

OR is_tlf LIKE '%-%'

OR is_tlf LIKE '—'), .

5.1.3 DROP TABLE

Bu komut, bir tabloyu yok etmek için kullanılmaktadır. Bu komutun yazım biçimi aşağıdaki gibidir;

DROP TABLE <tablo adı> { RESTRICT | CASCADE } .

<tablo adı>Z CREATE TABLE komutu ile oluşturulmuş olmalıdır. Eğer RESTRICT belirtilmişse, tabloya diğer tablolardan referans yapılamaz . Böylece aşağıdaki komut icra edilemeyecektir;

DROP TABLE bolum RESTRICT .

Çünkü Bolum tablosuna Personel tablosundan referans yapılmaktaydı. Eğer CASCADE belirtilmişse aşağıdaki komut yapılmalıdır;

REVOKE ALL PRIVILEGES ON t FROM u CASCADE

(t: tablo ismi u: tablo tanıtıcısının yetki tanıtıcısı)

Sonuç olarak tablo, tablonun tanımı, verilerin hepsi silinir. Yukarıda RESTRICT seçeneği ile başarısız kalan örnek ;

DROP TABLE Bolum CASCADE

Komutu başarı ile Bolum tablosunu, Personel tablosundaki FOREIGN KEY kısıtını düşürerek icra edilir.

5.1.4 DROP ASSERTION

Bu komutla sınıflandırmalar silinir. Bu komutun yazım biçimi şu şekilde olmaktadır;

DROP ASSERTION <kısıt adı>

Burada RESTRICT ve CASCADE seçenekleri bulunmamaktadır. Çünkü bunlara bağlı nesneler bulunmamaktadır. Verilen komutla sınıflandırma, onun tanımı ve bilgi Şemasındaki veriler silinir. Aşağıdaki komutun icra edilmesiyle Max_Siparis_Hattı sınıflandırılması silinecektir;

DROP ASSERTION Max_Siparis_Hattı

5.1.5 DROP VIEW

DROP VIEW komutu ile , bir görüntü silinmektedir. Bu komutun yazım şekli şu şekilde olmaktadır;

DROP VIEW <görüntü adı> { RESTRICT < CASCADE }

<görüntü adı> CREATE VIEW komutu ile oluşturulmuş olmaktadır.

RESTRICT seçeneği belirtilmişse, tabloya herhangi bir yerden referans yapılamaz. CASCADE belirtilmişse, aşağıdaki komut icra edilmelidir;

REVOKE ALL PRIVILEGES ON TABLE v FROM u CASCADE

(v: görüntü u: yetki tanıtıcısının adı).

Sonuç olarak bu komuttan sonra görüntü, görüntünün tanımı ve Bilgi Şemasındaki veriler de dahil olmak üzere silinecektir.

DROP VIEW Bim_tlf_lst CASCADE komutu ile Bim_tlf_lst görüntüsünü ve ona bağlı olan tüm nesneler silinecektir.

5.1.6 DROP COLLATION

Bu komut, karşılaştırma tanımlamasını silmek için kullanılmaktadır. Bu komutun yazım şekli aşağıdaki gibi olmaktadır;

DROP COLLATION <karşılaştırma adı>.

Buradaki karşılaştırmaya herhangi bir mevcut tanım kümesi tanımlamasından, tablo tanımından, görüntü tanımından veya sınıflandırma tanımından referans yapılmamalıdır. Burada CASCADE seçeneği bulunmamaktadır ve karakter kümelerinin karşılaştırmalara referans olabilmelerine rağmen karakter kümesinin doğrudan silinebileceği bir yol yoktur. Eğer silinmiş bir karşılaştırma y; bir karakter küme tanımında kullanılırsa (TRANSLATION x THEN y, DESC(y), COLLATE y vb.) ; bu tanımdan doğrudan silinir, sadece çeviri yada varsayılan karşılaştırma da kalır. Bundan sonra aşağıdaki komut icra edilir;

REVOKE USAGE ON COLLATION c FROM u CASCADE

(c: karşılaştırma adı u: yetki tanıtıcı adı).

Örnek;

DROP COLLATION Durum_bağımlı

komutu ile Durum_bağımlı karşılaştırması silinir.

5.1.6 DROP TRANSLATION

Bu komut, çeviri tanımlamasını silmek için kullanılmaktadır. Bu komutun yazım şekli aşağıdaki gibi olmaktadır;

DROP TRANSLATION <çeviri adı>.

Buradaki çeviriye herhangi bir mevcut tanım kümesi tanımlamasından, tablo tanımından, görüntü tanımından veya sınıflandırma tanımından referans yapılmamalıdır. Burada CASCADE seçeneği bulunmamaktadır ve karakter kümelerinin çevirilere referans olabilmelerine rağmen karakter kümesinin doğrudan silinebileceği bir yol yoktur. Eğer silinmiş bir çeviri x; bir karakter küme tanımında kullanılırsa (TRANSLATION x vb.) ; bu tanımdan doğrudan silinir, sadece varsayılan karşılaştırma da kalır. Bundan sonra aşağıdaki komut icra edilir;

REVOKE USAGE ON TRANSLATION tr FROM u CASCADE
(tr: çeviri adı u: yetki tanıtıcı adı).

Örnek;

DROP TRANSLATION ASCII_EBCDIC
komutu ile ASCII_EBCDIC çevirisi silinir.

5.1.6 DROP CHARACTER SET

Bu komut, karakter küme tanımlamasını silmek için kullanılmaktadır. Bu komutun yazım şekli aşağıdaki gibi olmaktadır;

DROP CHARACTER SET <karakter küme adı>.

Buradaki karakter kümeye herhangi bir mevcut tanım kümesi tanımlamasından, tablo tanımından, görüntü tanımından veya sınıflandırma tanımından referans yapılmamalıdır. Burada CASCADE seçeneği bulunmamaktadır ve karşılaştırma dizilerinin karakter kümelerine referans olabilmelerine rağmen karşılaştırma dizilerinin doğrudan silinebileceği bir yol yoktur. Bundan sonra aşağıdaki komut icra edilir;

REVOKE USAGE ON CHARACTER SET *cs* FROM *u* CASCADE

(*cs*: karakter küme adı *u*: yetki tanıtıcı adı).

Örnek;

DROP CHARACTER SET EBCDIC

komutu ile EBCDIC karakter kümesi silinir.

5.2 ALTER KOMUTLARI

İki çeşit ALTER komutu mevcuttur: ALTER DOMAIN ve ALTER TABLE. Çeviriler, karşılaştırmalar, karakter kümeleri, görüntüler ve şemalar için ALTER komutu kullanılmaz. ALTER ve DROP cümleleri nesnelerde bu işlemi gerçekleştirdiklerinden; ALTER komutu şemalar için geçerli değildir.

5.2.1 ALTER DOMAIN

ALTER DOMAIN komutu, tanım kümesinin özelliklerini değiştirmek için kullanılır. Bu özellikler gerçek veriyi etkilemez. Bu komutun 4 seçeneği vardır;

- set domain default : tanım kümesi varsayılanını belirleme
- drop domain default : tanım kümesi varsayılanını silme
- add domain constraint : tanım kümesine kısıt ekleme
- drop domain constraint : tanım kümesinden kısıt silme

5.2.1.1. Set Domain Default

Bu komut, tanım kümesi üzerine varsayılan değer yerleştirmek için kullanılmaktadır. Bu komutun yazım biçimi şu şekildedir;

```
ALTER DOMAIN <tanım kümesi adı> SET <varsayılan cümle> .
ALTER DOMAIN tlfn SET DEFAULT '100'
ALTER DOMAIN makbuz_trh SET DEFAULT DATE '1990-01-01'
```

5.2.1.2 Drop Domain Default

DROP DOMAIN komut, bir tanım kümesinden varsayılan cümleyi silmek için kullanılır. Bu komutun yazım biçimi şu şekildedir;

```
ALTER DOMAIN <tanım küme adı> DROP DEFAULT.
```

ALTER DOMAIN makbuz_trh DROP DEFAULT komutu icra edildikten sonra makbuz_trh tanım kümesi için varsayılan değer düşer ve bu tanım kümesi için varsayılan değer NULL değerini alır.

5.2.2.3 Add Domain Constraint

Bu komutla, tanım kümesine bir kontrol kısıtı ilave edilir. Bu komutun yazım şekli şöyledir;

```
ALTER DOMAIN <tanım küme adı> ADD <tanım kümesi kısıtı>
```

<tanım kümesi kısıtı> nın yapısı CREATE DOMAIN komutu ile belirtilen kısıt ile eş olmalıdır. Bu kısıt , her kolon için kontrol edilir ve var olan verilerde uygulanır. Eğer yeni kısıta uymayan mevcut verilervarsa bu komut icra edilmeyebilir. Örnek verecek olursak;

```
ALTER DOMAIN makbuz_trh ADD CONSTRAINT
CHECK ( VALUE > DATE '1990-01-01' ) .
```

5.2.1.3 Drop Domain Constraint

Bu komutla, bir kısıt tanım kümesinden silinir. Bu komutun yazım şekli şöyledir;

ALTER DOMAIN <tanım küme adı>

DROP CONSTRAINT <kısıt adı>.

Tanım kümesi, <kısıt adı> belirtilmiş bir tanım kümesi kısıtı içermelidir. Bu komutla ilgili kısıt silinir ve artık kontrol yapılmaz.

ALTER DOMAIN tlfno DROP CONSTRAINT

komutu ile tlfno tanım kümesi üzerindeki kısıt kaldırılmış olur.

5.2.2 ALTER TABLE

ALTER TABLE komutu, tabloların yapısını ve özelliklerini değiştirmek için kullanılır. Bazı özelliklerin değişmesi tablodaki veriyi etkilemez, fakat bazı özellikler de veriyi etkileyebilir. Tablo değişikliğini yapmanın 6 seçeneği vardır; [2]

add column	: tabloya kolon eklemek için
drop column	: tablodan kolon silmek için
add table constraint	: tabloya kısıt eklemek için
drop table constraint	: tablodaki mevcut kısıtı kaldırmak için
set column default	: kolon varsayılan değerini belirlemek için
drop column default	: kolon varsayılan değerini silmek için

5.2.2.1 ADD COLUMN

Bu komut tabloya kolon ilave etmek için kullanılmaktadır. Bu komutun yazım şekli şöyledir;

ALTER TABLE <tablo adı> ADD [COLUMN] <kolon tanımı>

Buradaki <kolon tanımı>, CREATE TABLE komutundaki kolon tanımı ile aynı olmak zorundadır.

Yeni eklenen kolon, her zaman tablodaki diğer kolonların en sonuna eklenir ve varsayılan değerler de belirtilen uygun değerlerle doldurulur. Varsayılan değer belirtilmemiş ise, kolon değerleri NULL değerini alırlar. Şayet kolon tanımında (*) kullanılmışsa, bu yeni kolon görüntü tanımına ilave edilmez.

Bu tablo için -DELETE- komutu haricinde hakları olan her kullanıcının hakları, bu yeni kolona verilir. Örnek verecek olursak;

ALTER TABLE Tasıt ADD motor_hacmi NUMERIC (4)

komutu; Tasıt tablosuna, motor_hacmi adındaki yeni kolonu ilave eder ve kolon null değerlerle doldurulur.

5.2.2.2 DROP COLUMN

DROP COLUMN komutu, tablodan bir kolonu silmek için kullanılır. Bu komutu n yazım biçimi şu şekildedir;

**ALTER TABLE <tablo adı> DROP [COLUMN] <kolon tanımı>
{ RESTRICT | CASCADE }**

Tabloda kalan son ve tek kolon silinemez. Eğer RESTRICT belirtilmişse, kolona referans yapılmamalıdır. Daha sonra aşağıdaki komut yerine getirilir;

REVOKE INSERT(c), UPDATE(c), REFERENCES(c)

ON TABLE t FROM u CASCADE

(t: tablo ismi u: yetkili tanımlayıcısı c : colon ismi)

Bu komut tablodan bütün hakları almakla kalmaz, ona bağlı bütün nesneleri de kaldırır. Sonuç olarak; kolon, onun tanımı ve Bilgi Şeması içinde bulunan veri silinir.

```
ALTER TABLE Tasit DROP makbuz_trh RESTRICT
```

komutu, Tasit tablosundan makbuz_trh kolonunu kaldırır. Silinen kolonun altında kalan kolonlar tekrar numaralandırılır.

5.2.2.3 ADD TABLE CONSTRAINT

Bu komut, tabloya yeni bir kısıt ekler. Bu komutun yazı biçimi şu şekildedir;

```
ALTER TABLE <tablo adı> add <tablo kısıt tanımı>
```

<tablo kısıt tanımı> CREAT TABLE komutundaki tablo kısıt tanımına eşdeğer olmalıdır. Böylece bu yeni ekelenen kısıt ile tablo sanki orjinal tanımlanmış gibi olacaktır. Eklenen bu yeni kısıt ile, sadece yeni veriler etkilenmez aynı zamanda bütün varolan eki veriler de etkilenecektir. Bu yüzden eğer bu kısıta uymayacak bir veri mevcut ise, bu komut icra edilemeyecektir. Bu komutun kullanımına ilişkin örnek verecek olursak;

```
ALTER TABLE Per_Ext ADD
```

```
CHECK ( doğum_trh + INTERVAL '16'
```

```
YEAR < giriş_trh )
```

bu komutla, bu şirkete 16 yaşından küçük hiç kimse alınmayacaktır. Eğer halihazırda böyle bir personel mevcut ise zaten komutun icrası geri çevrilecektir.

5.2.2.4 Drop Table Constraint

Bu komutla, tablodan bir kısıt silinmektedir. Bu komutun yazım biçimi şu şekildedir;

ALTER TABLE <tablo adı>

DROP CONSTRAINT <kısıt adı> { RESTRICT | CASCADE }

Eğer RESTRICT belirtildiyse, <kısıt adı> UNIQUE kısıt olmamalıdır, bir başka deyişle UNIQUE kısıtlar tablodan silinemezler. Şayet CASCADE belirtilirse, ve herhangi bir referans kısıtı varsa, bağlı olan her kısıt için kullanıcı lehine aşağıdaki komut icra edilir;

ALTER TABLE t DROP CONSTRAINT c CASCADE

(t: tanımında bağımlı c referansa kısıtını içeren tablonun adı).

Sonuç olarak tablo kısıtı silinir ve artık bu kısıta ilişkin kontroller yapılmaz. Bu kısıt hiç tanımlanmamış hale gelir. Buna örnek verecek olursak;

ALTER TABLE Siparis DROP CONSTRAINT

En_az_bir_siparis_iceren_siparis CASCADE

komutu ile

CHECK (sipno IN (SELECT sipno FROM siparis_hatti)) kısıtını iptal etmiş oluruz. Bu kontrolün kaldırılması ile artık siparişlerin en az bir tane sipariş kalemi içermesi koşulunu iptal etmiş oluruz.

5.2.2.5 Set Column Default

Bu komut ile, bir kolon üzerine varsayılan değer belirler. Bu komutun yazım şekli şu şekildedir;

ALTER TABLE <tablo adı> ALTER [COLUMN] <kolon adı>

SET <varsayılan cümle>

<varsayılan cümle>de belirtilen değer kolona uygulanır. Buna örnek verecek olursak ;

ALTER TABLE Personel ALTER COLUMN maas

SET DEFAULT 5000000

komutu ile Personel tablosundaki maas kolonunun varsayılan değeri 5000000 TL. olarak belirlenmiştir.

5.2.2.6 Drop Column Default

Bu komutla, bir tablodan varsayılan değer silinebilir. Bu komutun yazım biçimi şu şekildedir;

```
ALTER TABLE <tablo adı> ALTER [ COLUMN ] <kolon adı>  
DROP DEFAULT
```

burada <kolon adı> ile belirtilen kolon, bir varsayılan değer cümlesine sahip olmalıdır. Eğer kolon, tanım kümeleri ile belirlenmişse, buna uygun varsayılan tanım kümesi kullanılır. Aksi takdirde NULL kullanılacaktır. Örnek;

```
ALTER TABLE Personel ALTER COLUMN maas  
DROP DEFAULT
```

ile yukarıda ilave edilen varsayılan değer iptal edilmiş olur.

BÖLÜM 6. İMLEÇSİZ İŞLEMLER

İmleçsiz işlemler, SQL dilinin bünyesinde bulunan, temel işlemlerdir. Genellikle kümeye-yönelikdirler ve tablolardaki satır kümelerinin araya sokma, silme ve güncelleme işlemlerini sağlarlar (keyfi seçilmiş bir tablodan alınan tek bir satırın yeniden düzenlenmesi gibi). Yeniden düzenleme işlemi de küme-yöneliktir ama programlama dillerinin sınırlamalarından dolayı bu komut doğrudan kullanım durumlarında uygundur.

6.1 INSERT

Bu komut, bir tablo içine bir satır eklemek için kullanılmaktadır. Komutun gösterimi şöyledir;

```
INSERT INTO <tablo adı>  
[ ( <kolon adı> [ { , <kolon adı> } ... ] ) ] <sorgu ifadesi>  
| DEFAULT VALUES .
```

Tablo güncellenebilir olarak tanımlanmalıdır. Ayrıca tablo geçici olmadıkça işlem, oku-yaz işlemi olmalıdır. Hatta işlem eğer sadece okuma işlemi ise hala geçici tabloya satırlar eklenebilir.

Herbir kolon adı tablonun kolon adı olmalı ve bir kolon adı birden fazla olmamalıdır. Şayet kolon adları listesi yapılmamışsa, tablodaki tüm kolon adlarını

tanımlandığı sırada içeren bir liste kabul edilir. DEFAULT VALUES seçeneği, her bir kolon için varsayılan değerleri tümüyle içeren bir satır belirleyen özel bir seçenektir.

Söz konusu yetki tanımlayıcısı, listedeki her kolon üzerinde bir INSERT hakkına sahip olmalıdır. Araya eklenecek satırlar, sorgu ifadesi tarafından tanımlanırlar. Sorgu ifadesi türemiş bir tablo oluşturur, bu tablo kolon listesinde belirtildiği sayıda ve aynı düzende kolona sahip olmalıdır. Eğer tablonun tüm kolonları belirtilmemişse, eksik kolonlar türemiş tabloya uygun şekilde ve uygun değerlerle eklenir. Bu türemiş tablo, belirtilen tablo ile birleştirilir. Türemiş tablodaki satır sayısı, sıfır dahil negatif olmayan herhangi bir sayı olabilir. Sıfır satırlar ekleme teşebbüsü bir hata değildir ama bitirme durumu 'başarılı tamamlanmış' dan 'veri yok' durumu ile değiştirilir.

Örneğin Tasıt tablosuna bir satır eklemek için;

INSERT INTO Tasıt VALUES (94, NULL, '1991-07-12') komutunu kullanabiliriz. Bu türemiş tablo için bir satır tanımlar ve sonra bu tablo Tasıt tablosuna eklenir. Bu şekildeki kullanım, sabit bir tablo tanımlar ama sorgu ifadesi herhangi bir türemiş tablo alabilir. Eğer türemiş tablo herhangi bir şekilde hedef tabloya dayandırılmamışsa, o zaman araya eklenecek tüm satırların bütün kümesi, herhangi bir satır araya sokulmadan önce, bir değerlendirmeye tabi tutulur. Böylece tekrarlama ve kararsızlık problemlerinden kaçınılmış olur.

Şayet, minimum miktarı 50 olan ve ürün kodu 6000 den büyük tüm ürünleri temin edebilecek bir tedarikçi için Urun_arzı tablosunu oluşturacak olursak, bunu şu şekilde ifade edebiliriz;

```
INSERT INTO Urun_arzı
( SELECT urun_kod, teminci, 50,
  FROM Teminci, Urun
 WHERE temin_no = 12017 urun_kod > 6000 )
```

Eğer minimum siparişin varsayılan değeri 50 ise bu örnek şu şekilde de yazılabilir;

```
INSERT INTO Urun_arzi ( urun_kod, teminci_no)
( SELECT urun_kod, teminci_no,
FROM Teminci, Urun
WHERE teminci_no = 12017 urun_kod > 6000 )
```

Burada sadece iki kolon için bir kolon listesi tanımlandı ve sorgu ifadesi de bu iki kolon için yazıldı. Üçüncü kolon için varsayılan değer kullanılacaktır. Bu kolon tabloda INSERT işleminin uygun olmadığı kolonlar için kullanılmalıdır. Eğer kolonlar USER ve CURRENT_TIMESTAMP varsayılan değerleri ile tanımlanırsa; INSERT'ü kullanmamak, kullanıcının, varsayılanlar tarafından kusursuz olarak sağlanan ve gerçek kullanıcı veya program tarafından değiştirilmemesi gereken bir satır veya zamanı takip etmesini kolaylaştırır.

6.2 Tek Satır Seçimi SELECT

Bu komut, bir tablonun tek satırından gelen değerleri yeniden elde etmek için kullanılır. Bu komutun kullanım biçimi şu şekildedir;

```
SELECT [ ALL | DISTINCT ] <seçim listesi>
      INTO <hedef belirtme> [ { , <hedef belirtme> } ... ]
      <tablo belirleme>
```

Bu , INTO kalıbı hariç, <sorgu tanımlama> ile aynı formata sahiptir. Etkisi <sorgu tanımlama> daki gibidir. Değerlemesi yapılmış ve sonra türemiş türemiş tablodan gelen değerler, hedef tanımlamalarının yardımıyla belirlenmiş hedeflere transfer edilir.

Türemiş tablodaki kolon sayısı kadar hedef bulunmalı ve bunlar birebir ve aynı düzende olmalıdır. Sonuçlandırılmış türemiş tablo, bir satırdan fazla satır içermemelidir. Aksi takdirde, hata oluşur. Hata durumunda ise, hedefler bir takım muhtemel satır değerlerine sahip olur veya olmaz. Burada kullanıcı iyi bilgilenmiş olmalıdır ve gideceği yolu iyi belirlemelidir. Eğer hedefler, sıfır satırları içeriyorsa o zaman 'veri yok' uyarısı görünür. Komutu, tek satırla kısıtlamanın sebebi, söz konusu programlama dillerinin sınırlamalarından kaynaklanmaktadır. Çok satır seçimi ancak Doğrudan SQL ile sağlanmaktadır. [2]

Örneğin, personel numarası 52 olan personelin adını, maaş ve prim toplamını almak için ;

```
SELECT ad, maas + prim
      INTO: ad, : brut
      FROM Personel
      WHERE perno = 52
```

SQL komutu yazılabilir. Burada ad ve brut normal dil değişkenleridir. Bu durumda sadece bir satır oluşacak çünkü perno Personel tablosunun birincil anahtarıdır. Alternatif olarak, eğer tüm personelin ortalama maaşını istersek;

```
SELECT AVG(maas) into temp FROM Personel
```

bu işlem bilgiyi temp değişkeni içine alacaktır doğal olarak bu da farklı bir satır olacaktır.

6.3 UPDATE

Bu komut bir arama durumu tarafından belirlenmiş olan, güncellenecek satırları güncellemek için kullanılır. Bu komutun yazım biçimi şu şekildedir;

```
UPDATE <tablo adı>
      SET <kolon adı> = { değer ifadesi> | NULL | DEFAULT }
      [ { , <kolon adı> = { değer ifadesi> | NULL | DEFAULT } } ... ]
      [ WHERE <arama şartı> ]
```

<tablo adı> ile tanımlanmış tablo güncellenebilir olmalıdır. Bununla beraber, tablo geçici bir tablo ise, işlem oku-yaz işlemi olmalıdır. İşlem sadece okuma işlemiyse, geçici bir tabloda yine satırları güncelleme yapılabilir.

Her kolon ismi tabloya ait bir kolon ismi ve bir kolon ismi birden fazla olmamalıdır. Söz konusu yetki tanıtıcısı, güncelleme işleminde kolonlar üzerinde UPDATE hakkına sahip olmalıdır. Güncellenecek satırlar, arama şartları tarafından belirlenirler. Eğer WHERE ifadesi yoksa, o zaman tablonun tüm satırları tanımlanabilir. Tanımlanan satır sayısı sıfır ile tablodaki toplam satır sayısı arasında olabilir. Bir sıfır satırı güncelleme, hat vermez fakat işlemin tamamlanma durumu 'başarılı tamamlanıs' dan 'veri yok' ifadesine değiştirilir. Eğer güncellenecek satır hali hazırda silinmiş veya hala açık bir imleç tarafından güncellenmişse o zaman da bu uyarı ile karşılaşılmaktadır.

Örneğin, Urun tablosunda bulunan bir satırı güncellemek için ;

```
UPDATE Urun_Arzı SET min_siparis = 1000
```

```
WHERE urun_kod = 10564 AND teminci_no = 12017
```

bu ifade Urun_Arzı tablosundaki urun_kod ve teminci_no değişkenlerini birincil anahtar olarak atayıp, bir satır oluşturur.

Personel tablosundaki bir çok satırı güncellemek için;

```
UPDATE Personel SET maas = maas + 1000000
```

```
WHERE maas = ( SELECT MIN(maas) FROM Personel )
```

bu ifade minimum maaşa sahip tüm personelin maaşını 1000000 TL. kadar artırmaktadır. MIN fonksiyonu, maaşı minimum olanları seçer ve yükseltme işlemi devamlı olmaktadır. İşlem iki aşamada gerçekleşmektedir, önce WHERE ifadesiyle güncellenecek satırlar belirlenir ve daha sonra bu satırlar güncellenir.

Birden fazla maaş kolonunun güncellenmesi de istenebilir;

UPDATE Personel

SET maas = maas + 500, prim = prim + maas / 10

burada prim kolonunun güncellenmesinde kullanılan maaş değerinde bir karışıklık görülebilir, burada prim acaba güncellendikten sonraki maaş değeri ile mi hesaplanmaktadır. Böyle bir durumda, eşitliğin sağ tarafındaki değerler tamamen hiç bir değişikliğe uğramamış, ilk hallerindeki değerlere sahiptirler.

DEFAULT ifadesi, hedef kolon için hali hazırda varsayılan değerleri uygular. Eğer kolon NULL değeri için güncellenecekse, SET ifadesi NULL' da belirleyebilir. Buna örnek verecek olursak, bilgi işlem merkezinde çalışan personel için prim imkanı olmadığı biliniyorsa;

UPDATE Personel SET prim = NULL WHERE blmkod = 'BİM'
şeklinde yazılır.

6.4 DELETE

Bu komut, bir tabloda arama ile belirlenen satırların silinmesi için kullanılır. Bu komutun yazım biçimi şu şekildedir;

DELETE FROM <tablo adı> [WHERE <arama şartı>].

<tablo adı> ile belirlenen tablo güncellenebilir olmalıdır. Bununla birlikte, tablo geçici bir tabloysa işlem oku-yaz işlemi olmalıdır. Hatta işlem sadece okuma işlemi ise bile geçici tablodan gelen satırlar yine de silinebilir. Söz konusu yetki tanıtıcısı, <tablo adı> ile belirlenen tablo üzerinde DELETE hakkına sahip olmalıdır. Silinecek satırlar, <arama şartı> ile belirlenirler. Eğer WHERE ifadesi yoksa, tablonun tüm satırları tanımlanabilir. Tanımlanan satır sayısı sıfır ile tablodaki toplam satır sayısı arasında olabilir. Bir sıfır satırı silme, hata vermez fakat işlemin tamamlanma durumu 'başarılı tamamlanış' dan 'veri yok' ifadesine değiştirilir. Örnek olarak Bolum tablosundan bir satırı silmek için;

DELETE FROM Bolum WHERE blmkod = 'BIM'

bu ifade, Bolum tablosunda bölüm kodu 'BIM' olan bölümü silecektir.

Personel tablosundan birçok satırı silmek için;

DELETE FROM Personel

WHERE prim = (SELECT MIN(prim))

FROM Personel;

bu ifade ile primi en düşük olan elemanları siler. UPDATE işleminde olduğu gibi tüm satırların silinmesini önlemek için, burada MIN fonksiyonu kullanılmıştır. Bu işlem iki aşamada gerçekleşmektedir. Önce WHERE ifadesiyle silinecek tüm değerler belirlenir ve sonra bunlar silinir. Eğer belirlenmiş tablo bir görüntü ise, bu görüntü belirlenip silinen satırlar temel tablodan da silinirler.

6.5 Veri Atama Kuralları

Veri tabanından , prosedür parametrelerine veya kesin durumlara sahip programlama dili değişkenlerine transfer edilecek değerler tutulmalı ve hareketler belli olmalıdır. Bu kurallar ve hareketler transferi gerçekleştirecek komutlara bağlı olmayıp genel olmalı ve transfer gerçekleştirilirken birden fazla tekrarlanmamalıdır. Transferi gerçekleştirmek için bazı kurallar ve hareketler vardır.

6.5.1 Yeniden Kullanma Ataması

Verileri, veri tabanından, bir parametreye veya bir programlama dili değişkenine transfer edebilmek için kaynak ve hedef veri tipi uzunluğuna sahip olmalıdır. Eğer kaynak, karakter, ikili, sayısal (tam ve yaklaşık), tarih-zaman veya karşılaştırılabilir aralık veri tipine sahip olmalıdır.

Eğer transfer edilecek değer NULL değeri ise o zaman, gösterge parametresi -1 olmalıdır. Eğer birden fazla transfer yapılıyorsa, hangisinde problem olduğunu belirlemenin yolu yoktur. Eğer transfer edilecek değer NULL değilse, gösterge parametresi sıfır olur. Nümerik değerlerin transferi, eğer rakamlarda belirli bir kayıp yoksa basit bir şekilde tamamlanır. Aksi takdirde 'veri bekleme - nümerik değer dışı' hatası belirir.

Tarih-zaman ve aralık değerleri, hedefte de şekliyle temsil ediliyorlarsa, işlem yine basit bir şekilde tamamlanır. Aksi takdirde, aralık alanında 'veri bekleme-aralık alanı taşması' hatası çıkar. Tarih-zaman herhangi bir hataya sebebiyet vermez, çünkü kaynaktaki ve hedefteki veri tipleri aynıdır. Karakter ve ikili sözcüklerin transferi biraz karmaşıktır. Öncelikle transfer edilecek katarın uzunluğu, hedefin tutacağından daha büyükse gösterge parametresi, kaynaktaki katarla aynı uzunluğa ayarlanmalıdır, daha sonra da karakter ve ikililer transfer edilirler. Eğer hedef çok küçükse, ozman kaynak sağdan kesilir ve uyarı belirir. Eğer katar hedefteki sabit uzunluk için fazla kısaysa, duruma göre boşluk veya sıfır ile sağa genişletilir. ikili genişleme durumunda da bir uyarı çıkar. Eğer hedef değişken uzunlukta veriye sahipse, uzunluğu transfer edildiği katarın uzunluğuna ayarlanır.

6.5.2 Depo Ataması

Verileri bir parametreden veya bir programlama dili değişkeninden veri tabanına transfer edebilmek için, kaynak ve hedef, uyumlu veri tipine sahip olmalıdır. Eğer kaynak, karakter, ikili, sayısal (tam ve yaklaşık), tarih-zaman veya karşılaştırılabilir aralık veri tipine sahip olmalıdır.

Eğer transfer edilecek değer NULL değeri ise o zaman, gösterge parametresi -1 olmalıdır. Nümerik değerlerin transferi, eğer rakamlarda belirli bir

kayıp yoksa basit bir şekilde tamamlanır. Aksi takdirde 'veri bekleme - nümerik değer dışı' hatası belirir.

Tarih-zaman ve aralık değerlerinin transferinde eğer kaynak ile hedef arasında tam bir veri tipi uygunluğu varsa, işlem yine basit bir şekilde tamamlanır. Aksi takdirde, 'veri bekleme- tarih-zaman alan taşması ' hatası çıkar. Karakter ve ikili sözcüklerin transferi biraz karmaşıktır. Hedef eğer fazla küçükse katar sağdan kısaltılır ve 'veri bekleme-katar veri kısaltma' uyarısı çıkar. Eğer katar hedeftekinden fazla kısaysa, kataşa, duruma göre boş karakter ekleme yapılır. Ama ikili veri tiplerinde 'veri bekleme-katar veri, uzunluk uyumsuzluğu' uyarısı görünür. Eğer hedef değışken uzunlukta veriye sahipse, uzunluğu transfer edildiğı katarın uzunluğuna ayarlanır. [4]



BÖLÜM 7. İMLEÇ İŞLEMLERİ

Cursor, temel tablo, görüntü veya keyfi seçilmiş herhangi bir türemiş tablodaki satırlar vasıtasıyla programa adım sağlayan bir özelliktir. Bu SQL ile birlikte çalışılan mevcut programlama dillerinin kayıt tabanı veya kayıt tabanı olmayan amaçlarda içinde çalışmasından dolayı gereklidir. Cursorler, kümeye dayalı SQL ile kayıda dayalı programlama dilleri arasındaki bir arabirimdir. SQL, bir SELECT komutu ile araştırılmış satırları tutabilmek için bir sonuç tablosu inşa eder. SQL, satırları sonuç tablosunda tanımlamak ve bu satırları programa hazırlamak için imleç kullanmaktadır. Bu SELECT komutunun ile oluşturulmuş satırları kümesi, sıfır, bir veya birden fazla satırdan oluşmuş olabilir. Satırların sayısı, arama şartının gerçekleşmesine bağlıdır. [4] Cursorü açıklayan ve idare eden komutlar şunlardır;

DECLARE CURSOR : Bu komut, imlecin karakteristiklerini tanımlar. Aslında modülün nerede açıklandığı o kadar önemli değildir, fakat burada her zaman modülün her zaman başlangıcında gibi farzedilecektir.

OPEN : Türemiş tablonun tanımlamasında kullanılır. Tabloda form oluşturacak satırlar bu noktada tanımlanırlar. Türemiş tablo fiziki olarak maddeleştirilebilir veya yapılamaz.

FETCH : Bu komut OPEN'dan sonra ilk yerine getirilecek komut olmalıdır. CLOSE'dan önce birçok kez tekrarlanabilir. Türemiş tablodan bir satırın içeriğinin, bir program içine taşınmasına sebep olur ve imlece bu satırda bir pozisyon belirler.

UPDATE / DELETE : Bu komutlar seçime göredir ve FETCH'den sonra kullanılabilirler, bir başka deyişle imleç bir satır üzerinde bir pozisyona sahipse kullanılırlar.

CLOSE : Türemiş tablo tanımlamasını bırakır ve tekrar açılana kadar başka işleme izin vermez.

Bir imleç; bir tabloyu, bu tablodaki bir satırların sırasını ve tablodaki satıra göre pozisyonu dizayn eder. Bir sorgulama ifadesi ile tanımlanan tabloda sıralama ya kesindir, belirlidir yada çeşitli komutlar yardımıyla sıralanır ve pozisyon çeşitli komutlarla saptanır. İmlecin pozisyonunun, satıra karşılık açıklandığı zaman şöyledir;

belirlenmiş bir satırdan önce
belirli bir satır üzerinde
en son satırdan sonra.

Burada 'ilk satırdan önce' gibi bir durum yoktur. Sorgulama ifadesi sıfır satır olan bir tabloda, imleç 'ilk satırdan önce' veya 'son satırdan sonra' olabilmektedir.

İmleçler, işlemler sona erdiği zaman, otomatik olarak kapanır. Aynı isimle bir tek imleç olmasına rağmen bir işlem süresince, belirli bir imleç, birçok kez açılıp kapanabilir, çünkü OPEN komutunun parametreleri vardır ve herseferinde imleç ve satırlar farklı olabilmektedirler.

İmleçler ilave özelliklere de sahiptir. Bu çeşitli özellikler, belirli bir imleç üzerinde yürütme gücüne sahip komutları etkiler. İmleçler, özellikle güncellenirken, arama koşullu UPDATE ve SİLME komutlarının zaman içinde uzatılmış versiyonları olarak düşünülebilir, fakat bazı temel farklılıkları vardır. Bu

farklılıklardan ilki, aynı imleç üzerinde, iki işlem arasında, veri tabanına ek değişiklikler ilave etmek mümkündür. Örnek verecek olursak, iki UPDATE WHERE CURRENT imleç komutu arasında, imleç işlemi olmayan DELETE işlemi yapmak mümkündür. Ancak imleçsiz UPDATE işleminin ortasında DELETE işlemi eklemek mümkün değildir, çünkü SQL komutları, bir sonraki komut icra edilmeden, tamamlanarak çalışmaktadır. İkinci bir farklılık olarak, aynı bir imleç üzerinde, UPDATE ve DELETE komutlarını karıştırarak kullanmak mümkündür.

7.1 DECLARE CURSOR

Bu komut, imleç tanımlamak için kullanılmaktadır. İmleç tanımlama işlemi, bir OPEN komutu icra edilmeden yerine getirilemez. Bu komutun yazım biçimi şöyledir;

```
DECLARE <imleç adı> [INSENSITIVE] [SCROLL] CURSOR
FOR <sorgu ifadesi>
[ <sıralama cümlesi> ]
[ <güncellenebilirlik cümlesi> ] .
```

Burada en önemli komut, türemiş tabloyu, yani başka deyişle imlecin tablosunu tanımlayan <sorgu ifadesi>dir. <sorgu ifadesi>, 'SELECT * FROM t' kadar basit veya altsorgularla birleşmiş karmaşık bir ifade de olabilir. <sıralama cümlesi> satırda imlecin olup olmayacağını belirten bu cümleciğin gösterimi şu şekildedir;

```
ORDER BY <sıralama anahtarı> [ { ASC | DESC } ] [ ,<sıralama
anahtarı> [ { ASC | DESC } ] ] ...
```

<sıralama anahtarı> ya kolon ismidir, veya işaretli bir tamsayıdır. İşaretsiz tamsayı, kolon ismine karşılık gelen (i) nin kullanımı birçok esneklik sağlamaktadır.

ORDER BY ile, soldan sağa doğru, genelden özele kolon isimleri belirlenir. ASC ve DESC kelimeleri ise sıralamanın artan veya azalan sırada olacağını belirler. Sıralama anahtarları ile oluşturulmuş satır düzenleri, OPEN parametrelerinde veya veri tabanında bir değişiklik olmasa bile , bir imlecin açılmasından bir diğerine fark göstermektedir. Burada NULL değerler, tüm gerçek değerlerden önce yada sonra oluşur. Önce veya sonra oluşması, seçimi yürütme işlemine bağlıdır.

ORDER BY cümlecığının olmaması, sıralanacak satırların bulunmaması anlamına gelmez. Sadece istenilen düzen seçiminin yapılabilmesi anlamını taşımaktadır. Bununla beraber imleç için, bir sıralama seçimi imlecin CLOSE komutu ile kapanışına bağlıdır.

INSENSITIVE seçeneği, bir imleç içinde o imleçle yapılmayan güncellemeyi göstermekle ilgilidir. Eğer INSENSITIVE belirtilmişse, imleç içindeki tüm satırlar OPEN anında belirtilir ve işlemdeki diğer komutların yaptığı değişikliklerden etkilenmeden imleçde görünür kalır.

SCROLL seçeneği, NEXT haricindeki pazisyon seçeneklerini kullandırtan imleç için FETCH komutunu gösterir. Bunun anlamı, SCROLL seçeneği olmayan tüm FETCH komutları, NEXT'i belirlemeli ve SCROLL seçeneği ile beraber kullanılan FETCH komutu FIRST, LAST, NEXT, PRIOR, RELATIVE yada ABSOLUTE 'i belirleyebilir.

<güncellenebilirlik komutu>, imleci kullanacak hangi kolonun, güncellenebilir veya güncellenemeyeceğine ilave kontrollerle izin verir. Uygulamalarda bu ifade gereksizdir ve gözardı edilir. Bu komutun gösterimi şu şekildedir; FOR { READ ONLY

UPDATE [OF <kolon adı> [{ , <kolon adı> } ...]] }.

<kolon adı>, imleci belirleyen temel tablonun bir kolonunun adıdır. Kolonların imleçde görülmelerine hiç gerek yoktur. İmlecin, sadece gerekli kolonları belirtmesi ve istenilen satırları seçmesi bir avantaj sağlayabilir, bu da veri tabanı ile program arasındaki veri transferini minimuma indirger ve hızlandırır.

FOR UPDATE' in INSENSITIVE veya SCROLL ile kombinasyonu geçersizdir. Eğer <güncellenebilirlik komutu> belirtilmemişse, eğer INSENSITIVE; ORDER BY veya SCROLL belirtilmişse veya eğer <sorgu ifadesi> ile çıkan türemiş tablo güncellenebilirse, FOR READ ONLY kesinlik kazanır, aksi takdirde FOR UPDATE geçerlidir. Eğer FOR UPDATE, OF ile belirtilmemişse, bu belirtilmiş temel tablonun tüm kolonları listesiyle beraber, FOR UPDATE FOR ile aynıdır. READ ONLY demek güncelleme ve silme işlemlerinin yapılamaması demektir. UPDATE ise, UPDATE komutu ile güncellenecek komutları imleç yardımıyla günceller. İmleç belirtmeye örnek verecek olursak,

```
DECLARE Bim_max_kazanan SCROLL CURSOR FOR
SELECT ad, maas
FROM Personel
WHERE blmkod = 'BIM'
AND maas BETWEEN 4000000 AND 10000000
ORDER BY ad
```

Bu imleç örneği, satırları Personel tablosundan seçilmiş iki kolondan oluşur. Default olarak ad kolonu artan oranda sıralanmıştır. İmleç, SCROLL seçeneğine sahip olduğundan, güncellenemez. Pozisyonlandırma opsiyonu, FETCH komutunda kullanılabilir. İmleç, INSENSITIVE seçeneği kullanılmadığından, diğer imleç veya imleçsiz komutlarla oluşturulan Personel tablosunda, yürütmeye bağlı olarak, değişiklik yapabilir veya yapamaz. INSENSITIVE seçeneğinin olmaması kesinlik sağlamaz.. Bunu örnekle açıklarsak;

```
DECLARE Bim_max_kazanan SCROLL CURSOR FOR
SELECT ad, maas FROM Personel
WHERE blmkod = 'BIM'
AND maas BETWEEN 4000000 AND 10000000
ORDER BY ad FOR UPDATE maas, prim
```

ifadesi Personel tablosunun satırlarından, aynı iki kolonu içeren imleç tanımlar. sıralama yürütmenin karar verdiği şekilde gerçekleştirilir. imleç şu anda sadece maas ve prim kolonları (prim kolonu Personal tablosunun bir kolonudur, imlecin kolonu değildir) güncellenebilir, fakat şimdi FETCH komutunda izin verilen tek pozisyonlandırma seçeneği NEXT'dir.

İmleçlerdeki en büyük problem, imlecin kendisinin değilde diğer komutlar tarafından oluşturulan, belirli imleç tablosundaki değişikliklere karşı olan duyarlılığıdır. Aşağıdaki örnekle bu açıklanmıştır;

```
DECLARE List_per SCROLL CURSOR FOR
SELECT perno, ad, maas
FROM Personel
WHERE maas BETWEEN 4000000 AND 10000000
OPEN List_per
FETCH FIRST List_per FROM INTO :progperno, :progad, :progmaas
FETCH NEXT List_per FROM INTO :progperno2, :progad2,
:progmaas2
DELETE Personel WHERE perno = :progperno2
FETCH FIRST List_per FROM INTO :progperno2, :progad2,
:progmaas2
```

buradaki ilk FETCH iele son FETCH'in getirdiği satırlar için aynıdır veya farklıdır garantisi verilemez. Eğer imleç duyarlı ise farklı olacaktır, duyarsız ise aynı olacaktır. Duyarlılık OPEN yapılan bir imleçten diğerine farklılık gösterir. Bu yüzden, büyük işlemlerde bu duruma çok dikkat edilmelidir ve komutlar ona göre verilmelidir.

7.2 OPEN

Bu komut, bir imleci açmak için kullanılır. bu komutun yazım biçimi şu şekildedir; OPEN <imleç adı>

Aynı modül içinde <imleç adı>yla bir imleç tanımlayan DECLARE CURSOR komutu bulunmalıdır. Modül içinde, belirli bir imleç referansı veren sadece bir tane OPEN ifadesi olabilir. Daha sonra imleç açılır, daha önce açılmış

olmaması gerekmektedir. İmleç, DECLARE CURSOR ifadesi içinde bulunan, türemiş tablo tanımlamasının değerlendirilmesi sonucunda açılır. DECLARE CURSOR'da bulunan parametreler, OPEN anındaki değerlerin bir sonucudur. USER, CURRENT_USER, SESSION_USER, SYSTEM_USER, ve tüm tarih-zaman değer fonksiyonları (CURRENT_TIME vb.) OPEN anında bir kez değer kazanırlar, ve bu değerler imleç tanımı içinde daima kullanılır. Eğer DECLARE CURSOR ifadesi bir ORDER BY ifadesi içeriyorsa, imleç sıralanacaktır. Şayet imleç INSENSITIVE olarak tanımlanmış ise, türemiş tablonun bir kopyası yapılır ve bu kopya imleç olarak kullanılır. Varsayalım imleç ilk satırdan önceye konumlandırılmış olsun. Bu imleci tanımlayacak olursak:

```
DECLARE Tum_blm CURSOR FOR SELECT * FROM Bolum
ve bu şu şekilde açılır:
OPEN Tum_blm
```

7.3 FETCH

Bu komut, açık bir imleç vasıtasıyla yapılan türemiş tablonun bir satırını yeniden elde etmek için kullanılır. Bu komutun yazım biçimi şu şekildedir:

```
FETCH [ [ <konum> ] FROM ] <imleç adı> INTO <fetch listesi>
```

<fetch listesi> virgülle ayrılmış parametre listesi veya yüksek düzeyli dil değişkenleridir. <konum> seçeneği aşağıdakilerden biri olmalıdır:

```
NEXT      PRIOR      FIRST      LAST
ABSOLUTE <basit değer ifadesi>
RELATIVE <basit değer ifadesi>
```

<basit değer ifadesi> sözcük , parametre veya yüksek düzeyli dil değişkeni olmalıdır. Özellikleri belirtilmediği takdirde, <konum> seçeneği NEXT olarak kabul edilecektir. Bu işlemden önce imleç açılmış olmalıdır. İmleç önce

konum seçeneğine göre konumlanmalıdır, eğer konum seçeneği NEXT'den farklı ise imleç, SCROLL seçeneği ile tanımlanmış olmalıdır. NEXT imleci, mevcut konumu izleyen satıra konumlanır, eğer imleç x satırından önce konumlanmışsa, bir FETCH NEXT'ten sonra imleç x satırında olacaktır.

PRIOR, imleci mevcut imleç konumundan önceki satıra konumlandırır.

FIRST, imleci imlecin ilk satırına konumlandırır.

LAST, imleci imlecin son satırına konumlandırır.

ABSOLUTE n, eğer n pozitifse başlangıçtan sayarak , eğer n negatifse sondan itibaren sayarak n satırına imleci konumlandırır.

RELATIVE n, eğer n pozitifse n NEXT , n negatif ise n PRIOR ifadesine eşdeğerdir.

RELATIVE 0, eğer imleç bir satırda konumlanmışsa imleç o satırda kalır , aksi takdirde 'veri yok' uyarısıyla bir hata oluşur.

Eğer tabloda hareketi oluşturabilecek yeterince satır mevcut değilse, imleç ilk satırdan önce veya son satırdan sonra konumlanır ve 'veri yok' uyarısı oluşur. Bazı konumlama istekleri, her zaman hata oluşturur. Örneğin ABSOLUTE 0 geçerli olmakla beraber asla bir satır tanımlayamaz. Konumlama başarılı ise, imlecin yeni konumlandığı satırın kolon değerleri hedef parametrelerine, kopya edilir. Eğer tüm hedeflerin değerlerinin transferi sırasında hata olursa, bu hedef belirsizdir ve imlecin konumu mevcut satırda kalır.

Örnek:

FETCH Tum_blm INTO :progad, :progmaas

FETCH LAST Tum_blm FROM INTO :progad, :progmaas

FETCH RELATIVE -4 FROM Tum_blm INTO :progad, :progmaas

7.4 Konumlanmış UPDATE

Bu komut, türetilmiş açık bir imlecin söz konusu olan satırı olarak tanımlanmış temel tablodaki bir satırı güncellemek için kullanılır. Bu komutun yazım şekli şöyledir:

```
UPDATE <tablo adı>
SET <nesne kolon adı> = { <diğer ifadesi> | NULL | DEFAULT }
[ <nesne kolon adı> = { <diğer ifadesi> | NULL | DEFAULT } }...]
WHERE CURRENT OF <imleç adı>
```

Buradaki <imleç adı> ile tanımlanmış imleç, açık, güncellenebilir, bir satıra konumlanmış olmalıdır ve <tablo adı> tarafından belirlenen tablo öyle bir tablo olmalı ki, sadece imleç belirtecinin FROM cümleciğinde tanımlanabilsin.. Standart SQL'in güncelleme işlemi üzerinde çok ciddi kısıtlamaları vardır. Mevcut yetki tanımlayıcısı, <nesne kolon adı> ile belirtilen her kolon üzerinde UPDATE hakkına sahip olmalıdır. Her nesne kolonu imleç içine güncellenebilir olarak dahil edilmelidir.

Hiçbir nesne kolonu listede birden fazla bulunamaz ve hepsi <tablo adı> ile tanımlanan tablonun kolonları olmalıdır. Buna ilave olarak, ORDER BY ifadesinde kullanılan hiçbir kolon güncellenemez.

<değer ifadeleri>, MAX, COUNT vb. küme fonksiyonları olmamalıdır. DEFAULT, nesne kolonu için tanımlanan mevcut varsayılan değerinin varlığını ifade eder. Değer ifadeleri, bir satır güncelleştirilmeden önce değerlendirilir. Tablonun kolonlarına referanslar, herhangi bir kolon güncelleştirilmeden önce olur.

Örnek:

```
DECLARE Tum_blm CURSOR FOR SELECT * FROM Bolum
OPEN Tum_blm
```

şeklinde imleci tanımlarsak Şekil 7.1 deki gibi imleç elde etmiş oluruz.

Bu imlecin konumuna dayanarak:

UPDATE bolum

SET blmad = blmkod || '_' || blmad

blmkod = SUBSTRING (blmad FROM 1 FOR 3)

WHERE CURRENT OF Tum_blm

bunun sonucu da şekil 7.2 de görülmektedir.

Tum_blm	
blmkod	blmad
(a) →	
(b) →	BIM Bilgi işlem
→	PER Personel
(c) →	
→	...
(d) →	
→	SAT Satış
(e) →	

Şekil 7.1 Örnek İmleç

İmleç Konumu	Veritabanına etkisi / Durum	İmlecin Son Konumu
(a) ilk satırdan önce	Yok / Geçersiz İmleç Durumu	Uygulayıcı tanımlanmış
(b) satırda (BIM)	blmkod = BIL blmad = 'BIM_Bilgi İşlem olan satır / başarılı tamamlanmış	Aynı satırda
(c) Bir satır sonra (PER)	Yok / Geçersiz İmleç Durumu	Uygulayıcı tanımlanmış
(d) Bir satır önce (SAT)	Yok / Geçersiz İmleç Durumu	Uygulayıcı tanımlanmış
(e)son satırdan sonra	Yok / Geçersiz İmleç Durumu	Uygulayıcı tanımlanmış

Şekil 7.2 Örnek İmleçde Konumlanmış Update

Güncelleme yaparken ortaya çıkan değerler veya değerlerin işlenmesi sırasında hata meydana gelirse, imleç satırda konumlanmış olarak kalır. Dolayısıyla imleci doğru hale getirmek için ikinci bir girişimde bulunulmalıdır.

7.5 Konumlanmış DELETE

Bu komut, türetilmiş açık bir imlecin söz konusu olan satırı olarak tanımlanmış temel tablodaki bir satırı silmek için kullanılır. Bu komutun yazım şekli şöyledir:

DELETE FROM <tablo adı> WHERE CURRENT OF <imleç adı>

Buradaki <imleç adı> ile tanımlanmış imleç, açık, güncellenebilir, bir satıra konumlanmış olmalıdır ve <tablo adı> tarafından belirlenen tablo öyle bir tablo olmalı ki, ilk ve sadece imleç belirtecinin FROM cümlecğinde tanımlanabilsin. Mevcut yetki tanımlayıcısı, <tablo adı> ile belirtilen tablo DELETE hakkına sahip olmalıdır.

Örnek:

```
DECLARE Tum_blm CURSOR FOR SELECT * FROM Bolum
OPEN Tum_blm
```

şeklinde imleci tanımlarsak Şekil 7.3 deki gibi imleç elde etmiş oluruz.

Bu imlecin konumuna dayanarak:

```
DELETE FROM Bolum WHERE CURRENT OF Tum_blm
```

bunun sonucu da şekil 7.4 de görülmektedir.

	blmkod	blmad
(a) →		
	BIM	Bilgi işlem
(b) →	PER	Personel
(c) →		
	...	...
(d) →		
(e) →	SAT	Satış
(f) →		

Şekil 7.3 Örnek İmleç

İmleç Konumu	Veritabanına etkisi / Durum	İmlecin Son Konumu
(a) ilk satırdan önce	Yok / Geçersiz İmleç Durumu	Uygulayıcı tanımlanmış
(b) Satırda (PER)	Satır silinir/Başarılı Tamamlanış	Sonraki satırdan önce
(c) Bir satır sonra (PER)	Yok / Geçersiz İmleç Durumu	Uygulayıcı tanımlanmış
(d) Bir satır önce (SAT)	Yok / Geçersiz İmleç Durumu	Uygulayıcı tanımlanmış
(e) Son satırda (SAT)	Satır silinir/Başarılı Tamamlanış	Son satırdan sonra
(f)son satırdan sonra	Yok / Geçersiz İmleç Durumu	Uygulayıcı tanımlanmış

Şekil 7.4 Örnek İmleçde Konumlanmış Delete

7.6 CLOSE

Bu komut, açık bir imleci kapatmak için kullanılır. Komutun yazım biçimi şu şekildedir: `CLOSE <imleç adı>`

<imleç adı> ile belirtilen imleç daha evvel açılmış olmalıdır. İmleç kapatılır ve türemiş tablonun tanımı bozulur. Eğer aşağıdaki gibi bir imleç belirtirsek:

```
DECLARE Tum_blm CURSOR FOR SELECT * FROM Bolum
CLOSE Tum_blm
```

komutu ile kapatırız.

BÖLÜM 8. ERİŞİM KONTROL

SQL'de erişim kontrol, yetki tanımlayıcısı kavramına dayanmaktadır. Bu tanımlayıcı, beraberinde belki, zaman içinde verilen herhangi bir noktada, SQL için, bir kullanıcının veya başka bir şeyin kimliğini oluşturmak için kullanılan kısıtlamaları tanımlayan, bazı yürütmeleri olan bir tanımlayıcıdır. Sınırlamalar, tanımlayıcı SQL içinde normal olarak seçilmediğinden ve sistemin içinde geniş güvenlik sistemi bulunduğundan dolayı vardır. Uygulamada, tanımlayıcının kullanıcıyı tek olarak tanımlamada basit bir yol olduğunu kabul ederiz. Mevcut yetki tanımlayıcısı bazında, SQL Veri Tabanı Yönetim Sistemi, kullanıcının referans edeceği konuları, bu konular üzerinde yapılacak hareketleri ve bu konular içindeki verileri belirleyecektir.

Buna ek olarak, SQL nesnenin sahipliği ve nesne üzerindeki hak görüşüne de sahiptir. SQL'de oluşturulan her bir nesnenin bir sahibi vardır. Sahip, nesnenin bağlı olduğu şemanın şema tanımında AUTHORIZATION cümlecisi ile açık veya kapalı olarak yetki tanımlayıcısı tarafından tanımlanır. Sahip, nesne üzerinde belirli hareketleri yapabilecek tek kişidir, gerçekte nesnenin varlığından haberdar olan kişidir. Örnek şemayı gözönüne alırsak, Personel tablosunu tanımlayan CREATE TABLE ifadesi başarılı bir şekilde yerine getirilir ve tablo oluşturulur ancak şemanın sahibi olarak tanımlanan sadece tek kişi Personel tablosuna erişebilir veya bu tabloda bulunan verilerle çalışabilir. Bu tabloya veya onun kolonlarını sınırlı olarak referans verebilir. Kısacası sahip bütün bu bahsedilen şeyleri yapabilecek tek kişidir. Bu durum özel veri tabanları için uygun olmasına rağmen normalde çok faydalı değildir. Tabloları, başkalarının da kullanabilmesi için kesin olarak bunlara kullanabilme hakkının verilmesi gerekmektedir. SQL bu hakları tanımlar ve yönetir. Bunu 'hak tanımlayıcıları' yardımıyla yapar [2]. Hak tanımlayıcıları aşağıdaki gibi çeşitlendirilir:

Kullanım Hak Tanımlayıcıları

Tablo Hak Tanımlayıcıları

Kolon Hak Tanımlayıcıları

Her tanımlayıcı şunları içermektedir;

- Kullanım uygulamaları için nesne (tanım kümesi, tablo, kolon, karakter seti, karşılaştırma veya çevirme) tanımlaması

- Hakkı veren (SQL'de grantor adı verilir) kullanıcının yetki tanımlayıcısını

- Hakkı alan (SQL'de buna grantee adı verilir) kullanıcının yetki tanımlayıcısını

- İzin verilen nesne üzerindeki hareket tanımı

- Hakkın alıcısı tarafından diğer kullanıcılara hakkın verilip verilemeyeceğinin belirlenmesi

Muhtemel hareketler şunlardır

INSERT	UPDATE	DELETE
SELECT	USAGE	REFERENCES

DELETE ve SELECT hareketi sadece, nesne tablo iken kullanılabilir. INSERT, UPDATE ve REFERENCES tablolarla veya kolonlarla kullanılabilir. USAGE hareketi sadece, nesne eğer tanım kümesi, karakter seti, karşılaştırma ve çeviri olduğunda kullanılabilir. Bununla beraber SELECT hakkı bir tablo için verilmişse, kolon hakları tablo içindeki herbir kolon için oluşturulur.

Standart, bir nesne üzerinde şema idare ifadelerinin kullanımına izin verme hareketini tanımlamaz. Bir nesneyi DROP veya ALTER yapma hakkı sadece nesnenin sahibine verilmiştir.

8.1 GRANT

Bir nesneye sahip kullanıcı, yetki tanıtıcısı ile diğer bir kullanıcıya tablo üzerinde erişim hakkı vermek isterse;

```
GRANT SELECT ON Table1 TO DİĞER_KUL
```

Bu komut sayesinde, DİĞER_KUL1 adındaki diğer kullanıcı bu tablo üzerinde referans yapma ve bu tablonun kolonlarını içeren herhangi bir görüntü yapabilir. Ancak, diğer kullanıcıya verilen bu haklarla, tabloya herhangi bir satır ekleyip, güncelleme yapamaz veya satır silemez. Bu hakların verilmesi için de;

```
GRANT INSERT; DELETE; UPDATE
```

```
ON TABLE Table1 TO DİĞER_KUL1
```

Diğer kullanıcıya ihtiyaçlarından dolayı, bir tablonun üzerindeki kolon bazında hak verilebilir;

```
GRANT UPDATE kolon2 ON Tablo2 TO DİĞER_KUL1
```

```
GRANT SELECT ON Tablo2 to DİĞER_KUL1
```

Tablo2 deki herbir satırın kolon2 kolonunu güncelleyebilmek için diğer kullanıcıya, Tablo2 üzerinde SELECT hakkını da vermek gerekmektedir.

Bu örnekte görüldüğü gibi, DİĞER_KUL1 nesne sahibi tarafından yetki tanıtıcısı ile bazı haklara sahip olmuştur. Ancak bu hakları başka birine aktaramaz. Bunu WITH GRANT OPTION komutu ile yapabilir;

```
GRANT UPDATE kolon2 ON Tablo2 TO DİĞER_KUL1
```

```
WITH GRANT OPTION
```

```
GRANT SELECT ON Tablo2 to DİĞER_KUL1
```

```
WITH GRANT OPTION
```

```
GRANT SELECT, INSERT, DELETE, UPDATE ON Tablo1
```

```
TO DİĞER_KUL1 WITH GRANT OPTION
```

DIGER_KUL1, burada Tablo1 üzerinde muhtemel tüm haklara sahiptir, ancak REFERENCES hakkı hariç. REFERENCES, SELECT'e benzeyen bir harekettir. Ancak, tablonun veya kolonun farklı şartlarda kullanımını uygulaması bunun dışındadır. SELECT, veri idaresi ifadeleri için, REFERENCES veri tanımı ifadeleri için hakvermektedir. Bu ayrımın sebebi, SELECT hakkı herhangi bir kısıtlama olmadan başka bir kullanıcıya verilebilir, tablodaki verileri kullanıcı sadece kullanabilir, ama yine de bu erişim garanti edilemez. Eğer tablonu sahibi tablodan satırları siliyorsa ve diğer kullanıcı ON DELETE CASCADE seçimini kullanmıyorsa veriye erişmekte sorun çıkabilir.

```
GRANT SELECT; INSERT; DELETE; UPDATE ON Tablo1
TO DIGER_KUL1 WITH GRANT OPTION
```

Örneğinde verilen ifadeyi şu şekilde de ifade etmek mümkündür;

```
GRANT ALL PRIVILEGES ON Tablo1
```

```
TO DIGER_KUL1 WITH GRANT OPTION
```

DIGER_KUL1 de bu hakları başka bir kullanıcıya aktarmak isterse;

```
GRANT ALL PRIVILEGES ON Tablo1 TO DIGER_KUL2
```

Şayet, bir hakkı herkese vermek gerekiyorsa, bunu tüm kullanıcılar için tek tek hak verilmesine gerek yoktur. Bunun için;

```
GRANT SELECT ON Tablo to PUBLIC .
```

Birçok kullanıcıya birden hak verilebilir;

```
GRANT SELECT ON Tablo1 TO KUL3, KUL4
```

8.2 REVOKE

REVOKE ifadesi, GRANT ifadesinin tersini yapmaktadır. Yani kullanıcıya verilmiş haklar bu komut sayesinde geri alınmaktadır. İki seçeneği vardır:

RESTRICT ve CASCADE. Varsayalım ki KUL4 kulanıcısına verdiğimiz tüm hakları geri almak isteyelim:

```
REVOKE ALL PRIVILEGES ON Tablo1 FROM KUL4 RESTRICT
yada
```

```
REVOKE SELECT ON Tablo1 FROM KUL4 RESTRICT
```

Burada SELECT, sadece verilen haktır. Burada hakları, yalnızca hakları veren geri alabilir. GRANT ile REVOKE , birbirinin yansıması gibi görünse de, GRANT hak tanımlayıcısı oluşturur, REVOKE ise hak tanımlayıcılarını siler veya günceller. REVOKE komutu da WITH GRANT OPTION ile verilebilir. Eğer nesnenin sahibi DİGER_KUL1' verdiği UPDATE hakkını kaldırmak isterse;

```
REVOKE GRANT OPTION FOR UPDATE (kolon1) FROM
DİGER_KUL1 RESTRICT
```

Bu durum eğer DİGER_KUL1, bu kolon üzerindeki hakkını bir başka kullanıcıya vermediyse mümkün olacaktır. Aksi takdirde diğer kullanıcının hakkını silmesi gerekecekti ki bu da RESTRICT yüzünden gerçekleşemez buna da CASCADE ikinci bir kullanıcının haklarına güçlü müdahale sağlanır.

Şekil 8.1 de şema nesnelerini oluşturmak için hangi haklara ihtiyaç vardır onu göstermektedir.

Nesne	Hangi elemanlar üzerinde hak bulunmalı ?
Görüntü	Tablo (En az SELECT, ilave haklar sahibin kendi haklarını genişletmesini saptar) Tanım kümesi Karşılaştırma Karakter kümesi Çeviri
Tanım Kümesi	Karşılaştırma
Tanım Kümesi Kısıtları	Tablo/Kolon (referans hakkı) Tanım kümesi Karşılaştırma Karakter kümesi Çeviri

Şekil 8.1 Şema Nesnelerini Oluşturmak İçin Gerekli Haklar

Tablo kısıtları	Tablo/Kolon (referans hakkı) Tanım kümesi Karşılaştırma Karakter kümesi Çeviri
Assertion	Tablo/Kolon (referans hakkı) Tanım kümesi Karşılaştırma Karakter kümesi Çeviri
Karşılaştırma	Karakter Kümesi
Çeviri	Karakter Kümesi
Hak Tanımlayıcı	Hak

Şekil 8.1 Şema Nesnelerini Oluşturmak İçin Gerekli Haklar (Devamı)

8.3 Hakların Kontrolü

Hakların kontrolü, mevcut yetki tanımlayıcı bazında olmaktadır. Tüm oturumlar için, tanımlanan yürütme veya oturuma başlangıcı sağlayan bağlantının belirteci olan bir yetki tanıtıcısı vardır. Bu yetki tanıtıcısı, SET SESSION AUTHORIZATION komutu ile değiştirilebilmektedir.

Verilen bir SQL komutu için, mevcut yetki tanıtıcısıyla kurulan uygun haklar, komutun başarı ile tamamlanabilmesi için, bu yetki tanıtıcısı tarafından tutulmalıdır. Bu haklar için gerekenler aşağıda sıralanmıştır:

SELECT hakkı, şunlar içinde kullanılan tüm tablolar için gereklidir;
 <araya sokma komutu> için <sorgu ifadesi>,
 <imleç belirtme>,
 <görüntü tanımlama>,
 <doğrudan seçim:çoklu satırlı> veya
 <tablo ifadesi>,
 <seçim komutu:tek satırlı> nu kapsayan bir <seçim listesi> veya

<dinamik seçim komutu:tek satırlı> nu kapsayan <sorgu belirtme> veya <silme komutu> veya >güncelleme komut> içeren <arama şartı> tümüyle <seçim komutu:tek satırlı> kapsayan bir <seçim listesi> veya tümüyle <set ifadesi:aramalı> veya <set ifadesi:komumlu> içeren <değer ifadesi>.

DELETE hakkı, <silme komutu:komumlu> veya <silme komutu:arama> için gereklidir.

INSERT hakkı, açık veya kapalı olarak <araya ekleme kolon listesi> veya <araya ekleme komutu> nda tanımlanmış her bir kolon için gereklidir.

UPDATE hakkı, <nesne kolonu:komumlu> veya <nesne kolonu:arama> da tanımlanmış herbir kolon için gereklidir.

REFERENCES hakkı, <referanssal kısıt tanımı>nda referans edilmiş bütün kolonların veya <kontrol kısıt tanımı> veya <assertion tanımı> içeren <arama şartı> nda referans edilmiş bütün kolonlarda veya bütün tabloların en az bir kolonun üzerinde gereklidir.

USAGE hakkı, kendini yaratan komutlar dışında herhangi bir komuta referans edilmiş herbir tanım kümesi, karşılaştırma, karakter kümesi veya çeviriler için gereklidir.

BÖLÜM 9. TÜREMİŞ TABLOLAR

Bu bölüm, SQL dilinin türemiş tabloların oluşturulmasında kullanılan elemanlarını içermektedir.

9.1. Satır Değeri Yapıcıları

Bir satır değeri yapıcısı, tablodaki bir satır gibi düzenlenmiş değerler kümesidir. Şu şekli alır :

```
<Satır Değeri Yapıcısı> ::=  
    {<Değer İfadesi> | NULL | DEFAULT}  
    | {<Değer İfadesi> | NULL | DEFAULT}  
    [{,<Değer İfadesi> | NULL | DEFAULT}...]  
    | <Alt Sorgu>
```

Eğer <Alt Sorgu> belirtilmişse, o sadece sıfır veya bir satır olarak geri dönmelidir. Eğer <Alt Sorgu> satır aralarına geri dönerse <Satır Değeri Yapıcısı>'nın sonucu geçersiz bir satırdır. Aksi halde o geri dönmüş bir satırdır. Satır değeri yapıcısının sonucu şu kolon satırlarından oluşur; <Değer İfadesi>, NULL, ya da eğer DEFAULT belirtilmemişse default değer hedef kolon belirtilir. Eğer satır değeri yapıcısı bir INSERT ifadesi için, girdi verisinin bölümünü oluşturuyorsa bir sınırlandırma vardır. Ancak NULL ve DEFAULT belirtilebilir (NULL'un kapalı veri çeşitinin olmaması ve bir satır değeri yapıcısı içindeki kolonun default değeri almaması nedeniyle kolonun veri çeşitini veya default değerini belirtemeyiz). Eğer sadece bir kolon varsa satır değeri yapıcısının paranteze alınması gerekmez [2].

Örnek:

```
perno  
( 'A text string' ,NULL, 2.3)  
(SELECT * FROM Bolum WHERE blmkd = 'SAT')
```

9.2 Tablo Değeri Yapıcısı

Tablo değeri yapıcısı satır değeri yapıcısından oluşan bir kümedir. İşlemleri aynen yerine getiren, güncellenemeyen bir tablo çeşitidir ve şu forma sahiptir:

<Tablo Değeri Yapıcısı>::=

VALUES <Satır Değeri Yapıcısı>[{,<Satır Değeri Yapıcısı>}...]

Tüm tablo değeri yapıcıları aynı sayıda değerlere sahip olmalıdır. Her bir satır değeri yapıcısı uygun değerler içeren veri tiplerine sahip kolonlarla beraber bir satır tablosu kabul edilebilir. Tablo değeri yapıcısının sonucu, satır değeri yapıcısının temsil ettiği tüm tabloların UNION'undan (birleştirilmesinden) oluşan bir tablodur. Örneğin Bolum tablosu olarak aynı yapıda iki satır oluşturarak:

VALUES ('SAT', 'Satis') , ('BIM', 'Bilgi Islem')

İki ayrı satır artı bir satırdan oluşan bir tablo oluşturmak için ki bu artı satır birinci anahtarı 'PRD' olan Bolum tablosundan alınacak,

VALUES ('SAT', 'Satis'), ('BIM', 'Bilgi Islem'),
(SELECT * From Bolum WHERE blmkd='PRD')

9.3 Sorgu İfadesi

Sorgu ifadesi, türemiş bir tablo tanımlamada en temel yoktur. Görüntülerin açıklanması için başlama noktası, imleçler ve alt-sorgulardır. Bir türemiş tablonun sınırlanması için format içinde birçok seviyeler ve tercihler vardır. Bu ara seviyelerden iki tanesi komutlar içinde bir türemiş tablo belirtilmesi için başlangıç noktaları olarak kullanılırlar.

Bunlardan birincisi <Sorgu Özelliği>dir ki, bu <Dinamik Tek Satır Seçme İfadesi>'ne denktir ve INTO olmaksızın <Seçim İfadesi : Tek Satır> türemiş tablosunun ikinci formunu kullanır- Bir <tablo ifadesi>. Bunun nedeni daha çok tarihseldir ve seçilmiş ifadelerin harf yapısındaki uygunluğunu artırarak sürdürmek için tutulmuştur. Göreceğimiz gibi, sorgu ifadesi için tablo ifadesi tanımları ve sorgu ifadesi tayini bileşke olarak diğer formları da içerir (Örneğin tanımlar tekrarlanır), fark öyle önemli bir problem değildir. Sorgu ifadesinin format yapısı :

<Sorgu İfadesi>::=

<Non-join Sorgu İfadesi>

| <Joined Tablo>

<Non-Join Sorgu İfadesi>::=

<Non-Join Sorgu Terimi>

| <Sorgu İfadesi> UNION [ALL] [<Karşılık Gelen Özelliği>]

<Sorgu Terimi>

| <Sorgu İfadesi> EXCEPT [ALL] [<Karşılık Gelen Özelliği>]

<Sorgu Terimi>

<Sorgu İfadesi>::=

<Non-join Sorgu Terimi>

| <Join Edilmiş Tablo>

<Non-join Sorgu Terimi> ::= <Non-join Sorgu Önceliği>

| <Sorgu Terimi> INTERSECT [ALL] [<Karşılık Gelen Özelliği>]

<Sorgu Önceliği>

<Sorgu Önceliği> ::= <Non-join Sorgu Önceliği>| <Join Edilmiş

Tablo>

<Non-join Sorgu Önceliği>::=<Basit Tablo>(<Non-join Sorgu İfadesi>)

<Basit Tablo>::=<Sorgu Özelliği>| <Tablo Değeri Yapıcısı>

| TABLE <Tablo İsmi>
 <Karşılık Gelen Özelliği>::=
 CORRESPONDING [BY (<Karşılık Gelen Kolon Listesi>)

 <Karşılık Gelen Kolon Listesi>::=
 <Kolon İsmi> [{Kolon İsmi}>...]

Şeklinde yorumlanır.

Bu demektir ki, bir sorgu ifadesi iki çeşitten biri olabilir. Bir tabloya join operatorlerinden ya da non-join operatörlerden oluşur. Birinci durumda operatörler basit tablo isimleri veya sorgu gruplarıdır. İkinci durumda operatörler alt sorgular, tablo değeri yapıcılar veya sorgu özellikleri ya da TABLE kelimesi ile oluşturulan tablo isimleridir. Bu son form "SELECT * FROM <Tablo İsmi>" 'e eşittir ve şüpheli, hatalı durumları önlemek için TABLE ifadesine ihtiyacı vardır. Karşılık sorgu ifadelerinde önce parantez içindeki değerler değerlendirilir. Sonra birleştirilmiş tablolar ve sonradan UNION ve EXCEPT işlemleri değerlendirmeye sokulur. UNION ve EXCEPT soldan sağa doğru değerlendirilir.

UNION, INTERSECT ve EXCEPT non-join küme operatörleridir. Tüm tabloların bütün kolonlarının bir ismi vardır ve bunlar tablo içindedirler. Tablolar sorgu ifadesi yardımıyla türetildiğinde son tablodaki kolonun adı ya kendisinden türetilen kolonunki ya da UNION, EXCEPT ya da INTERSECT durumlarında kendisinden türetilen iki kolonun ortak adıdır. Eğer türetme basit değilse veya küme operatörlerin kolon isimleri farklılaştığında sistem uygun bir isim sağlar. Bir <Sorgu İfadesi>nden gelen türemiş tablo sadece güncellenebilir eğer o sadece bir <Non-join Sorgu Önceliği>nden ibaretse (hiçbir küme operatörü belirtilmemiştir vb.) ve belirlenmiş <basit tablo> veya <Non-join Sorgu İfadesi> kendisi için güncellenebilir. Bir non-join küme operatörü için iki işlem tablosunun aynı sayıda kolonlara sahip olması gerekir.

Eğer CORRESPONDING seçeneği belirtilmişse o zaman bu, aşağıdaki sorgu ifadesine eşittir :

```
(SELECT Select-List FROM Operand1)
op
(SELECT Select-List FROM Operand2)
```

Op-küme operandı, operand1 birinci operand, operand2 ikinci operanddır. Select-list ise şöyle açıklanır:

- Eğer karşılık gelen hiçbir kolon listesi belirtilmemişse o zaman bu her iki operant tabloda yer alan bütün kolon isimlerinin bir listesidir. En azından bir ortak kolon olması gerekir.

- Aksi halde, karşılık gelen kolon listesinde verilne düzendir. Bütün belirlenen kolonlar her iki operant tablosunda bulunmalıdır. Herhangi bir kolon ikilemesinin ortaya çıkmaması için her iki ayrı tabloda çift kolon isimlerinin olmaması gerekir. Örneğin türemiş tablo 'A JOIN A'da, bir durum ortaya çıkaracaktır.Şunlar geçerli sorgu ifadeleridir:

```
VALUES ('SAT', 'Satis'), ('BIM'), Bilgi Islem)
bir tablo değeri yapıcısı,
```

```
(SELECT * FROM personel)
EXCEPT CORRESPONDING
(SELECT * FROM Seyahat-İst)
```

EXCEPT edilmiş bir tablodur. Bu tablo seyahat talebinde bulunmayan peronelin listesinden oluşuyor.

```
SELECT DISTINCT Tedarikci FROM Sipariş
bir sorgu özelliğidir.
```

Sipariş NATURAL JOIN Sipariş-Sırası

Join edilmiş (Katılmış) bir tablo,

```
(SELECT Perno
FROM Per-ext
WHERE Dogum_trh < CURRENT-DATE-INTERVAL '60'
YEAR)
```

```
UNION
(SELECT perno
FROM Per-ext
WHERE Dogum_trh > CURRENT-DATE-INTERVAL '20'
YEAR)
```

Bu UNION edilmiş bir tablodur. 60'ın üzerinde 20'nin altında doğum tarihi olan işçileri içermektedir.

9.3.1. Tablo Referansları

Sorgular içinde birçok yerde değişik türden çeşitli tabloları referans göstermek gerekir. Temel tablolar, görüntüler veya alt sorgulardan gelen türemiş tablolar vb. Bu tabloların yer isimlerini de birleştirmek de mümkündür. Bu ya sorgunun geride kalan yazımını basitleştirmek ya da aynı tablonun farklı kullanımlarını birbirinden ayırmakla yapılabilir [1]. Bir tablo referansının format yapısı şöyledir :

```
<Tablo Referansı>::=
<Tablo İsmi>[[AS]<Bağlantı İsmi>[(<Türemiş Kolon Listesi>)]]
<Tablo Alt Sorgusu>[[AS]<Bağlantı İsmi>[(<Türemiş Kolon
Listesi>)]]
<Join Edilmiş Tablo>
<Türemiş Kolon Listesi>::=<Kolon İsmi>[{,<Kolon İsmi>}..]
```

- Eğer veri tipleri karakter katarsa tüm veri tipleri de karakter katarı ve aynı karakter grubundan olmalıdır. Sonuç da bir karakter katarı olmalıdır. Sıraların düzenlenmesi ve sonucun zorlanabilirliği Tablo 4.3'de işlenmiştir. (Bkz. Bölüm CREATE COLLATION).

- Eğer hepsi sabit uzunlukta ise sonuç da veri tiplerinin uzunluklarının maksimum değerinde sabit bir uzunluğa sahiptir. Eğer değişken uzunluktaysalar, sonuç da küme içinde sabit uzunlukta ve değişken uzunlukta veri tiplerinin maksimum uzunluğuna eşit değerde değişken olur.

9.4 Tablo İfadesi

Bir tablo ifadesinin amacı türemiş bir tablo tanımlamaktır. Bu, türemiş tablonun gruplanabileceği bir sorgu ifadesinden farklıdır. Yani, bazı ortak kriterli satır grupları birlikte işletilebilir. Gruplanmış tabloları normal tğremiş tabloya çevirme işlemini bir sorgu özelliği ile gerçekleştirebiliriz.

Tablo ifafdesinin format yapısı şöyledir:

```
<Tablo İfadesi>::=
  <From İfadesi>
  [<Where İfadesi>]
  [<Group By İfadesi>]
  [<Having İfadesi>]
```

Türemiş bir tablo tanımlar ki bu tablo seçeneğe bağlı ifadeleri başarıyla tamamlayarak geliştirir, uyarlayabilir. Tablo ifadesinin sonucu belirtilen tüm ifadelerin uygulamasından çıkan son türemiş tablodur. Çünkü <Tablo İfadesi> öyle çok bileşene ve ilişkili parçalara, sorgu özelliğinin form parçalarına sahip değildir.

9.4.1 FROM

Bir sorgu özelliğinin başlangıç noktasıdır ve türemiş bir tablo tanımlar. Format yapısı şöyledir :

<From İfadesi> ::= FROM <Tablo Referansı> [{,Tablo Referansı}..]

Türemiş tablo (T) tek bir tablo referansı tarafından veya bir tablo referansları listesi ile tanımlanmış tüm tabloların bir CROSS JOIN'i tarafından tanımlanan tek bir tablodur. <From İfadesi> içinde yer alan tablolar basit tablolar veya türemiş tablolar olabilir. (Tablo alt sorguları veya join edilmiş tablolar).

9.4.2 WHERE

Bu ifade türemiş tablodan (T) yeni bir türemiş tablo meydana getirir. Ki, bu tablo (T) sadece seçilmiş satırları tutan from ifadesi ile tanımlanmıştır. Format yapısı şöyledir :

<Where İfadesi> ::= WHERE <Arama Durumu>

Arama durumundaki kolon referansları T'deki kolonlara referans vermeli veya bir ilişik türemiş tablo tanımlamasındaki kolonlara referans olmalıdır. (Daha dıştaki (outer) referans). Bu, iç içe girmiş türemiş tablonun parçası olan WHERE ifadesinin olduğu yerde ortaya çıkar. Arama durumu bu <Where İfadesi> kendi parçası veya daha yüksek seviyedeki Having ifadesi veya bir seçim listesi ki, onda durum T'ye referans vermeyebilir sadece outer referanslar, olmadıkça küme fonksiyonlarını içermeyebilir.

Türemiş tablo sonucu arama durumu doğru değerini taşıyan sadece T'nin satırlarını içerir. Arama durumu teorik olarak T'nin her satırı için tekrar değerlendirme yapar (Ki bu pratikte her zaman gerekmebilir). Kitapta WHERE ifadesi için bir çok örnek verilmiştir. Ama bir outer referansı gösteren özel bir örnek verebiliriz (İkinci WHERE ifadesindeki Sipariş.Siparişno) :

```
SELECT * FROM Sipariş
WHERE 10 = (SELECT COUNT(*)
FROM Sipariş-Sırası
WHERE Sipariş.Siparişno=Sipariş-Sırası
```

9.4.3 GROUP BY

Group By ifadesi <From İfadesi>'den türemiş tabloyu alır ve eğer varsa bir gruplanmış tablo üretir. Gruplanmış tablo, satırların satır setlerine ayrıldığı yerdedir ve her bir satır kümesi için ayrı olmayan kesin belirlenmiş kolonların değerleri vardır. Bu gruplama setlerin veya grupların sayısı minimum alınarak yapılmıştır. Format yapısı şöyledir :

```
<Group By İfadesi>::= GROUP BY <Kolon Referansı> [<Collate
ifadesi>]
[ {,<Kolon Referansı> [<Collate İfadesi>]} ... ]
```

Kolon referansları doğal olarak türemiş tablodakilerden biri olmalıdır. <Collate İfadesi> eğer referans edilmiş kolon bir karakter veri tipine sahipse o zaman kullanılabilir. Bu ifade, eğer varsa, EXPLICIT ile kolonların eşleme sıralarını belirli sıralara dönüştürür. Bu eşleme sıraları gruplanmaları için satırlar karşılıklı eşlenirken kullanılırlar.

Her zaman bir gruplama kolonuna grup içindeki bir satır için referans verilirse bu kolonun değeri geri gelir. Kesin olarak benzemeyen değerler eşit şekilde eşlenebileceği için (Özellikle eşleme sırasının kullanımı ile) geri dönen değer muhtemel adaylardan (sonuçlardan, değerlerden) biri olabilir. Bununla beraber, gruplanmış tablodaki kolonlara alt sıra referansları ya gruplama kolonuna olmalı ya da bir küme fonksiyonu içinde tutulmalıdır. Bu şekilde Group By ifadesi grup başına en fazla bir satır ile bir tablo oluşturulmasını güçlendirir. Bu tablo eleman bazında tek tek olmayan genel bilgilerini sağlar bu grubun. Gruplanmayan kolonlar için bir küme fonksiyonun kullanımı bunları küme fonksiyon tek değerine indirger.

Örneğin, eğer personel numaraları ve bolum listesini almak için aşağıdaki program yazılır;

```
SELECT BlmKod,BlmAdı, COUNT (*)
FROM Bölüm LEFT JOIN İşçi
GROUP BY BlmKod,BlmAdı
```

Dikkat edin, burada BlmAdı'nı bir küme fonksiyonu içinde bir yere sahip olmasından kaçınmak için basitçe GROUP BY ifadesi içine dahil edildi. Bu durumda sınırlama mantıklı değil, çünkü BlmAdı fonksiyonel olarak zaten yine GROUP BY ifadesi içinde olan BlmKod'a bağlı. SQL böyle özel durumları ayıramaz. BlmAdı'nı GROUP BY ifadesi içine yerleştirmeye bir alternatif olarak seçim listesindeki bir MIN(BlmAdı) veya MAX(BlmAdı) içine yazabilinecekti. Öyleki yine aynı sonuç alınır. Eğer cinsiyete göre personel sayısını alınmak istenirse, o zaman şöyle yapılır;

```
SELECT Cins, COUNT(*)
FROM İşçi
GROUP BY Cins
```

9.4.4 HAVING

Having ifadesi, önceki <From İfadesi>'nin, <Where İfadesi>'nin ve <Group By İfadesi>'nin birleştirilmesinin sonucunda türemiş tablodan (T) tüm grupların elimine edilmesine izin verir. Öncesinde <Group By İfadesi> yoksa, o

zaman tüm türemiş tablo tek bir grup gibi olur. (Herhangi bir gruplama kolonu olmaksızın). Format yapısı :

<Having İfadesi>::= HAVING <Arama Durumu>

Arama durumundaki kolon referansları T'deki kolonlara referans vermeli veya ilişik tablo tanımlamasındaki kolonlar için referanslar olmalıdır (Outer referans) ve , daha önce belirtildiği gibi, T'deki kolonlara giden referanslar ya bir gruplama kolonu için olmalı ya da bir küme fonksiyonunun içinde bulunmalıdır. Türemiş tablonun sonucu arama durumunun doğru değeri için sadece T'nin gruplarını içerir. <Arama Durumu> teorik olarak T'nin herbir grubu için yeniden değerlendirme yapar. Örneğin, içindeki personelin sayısı ile birlikte 20'den fazla personel çalıştıran departmanların listesi istenirse ifadeler şöyle olur :

```
SELECT BlmKod,MIN(BlmAdı),COUNT(*)
FROM Bölüm LEFT JOIN İşçi
GROUP BY BlmKod
HAVING COUNT(*) > 20
```

ya da 1000 kaleminden fazla ürün sipariş miktarlarının ortalaması;

```
SELECT Ürün-Kodu, AVG(Miktar)
FROM Sipariş-Satırı
GROUP BY Ürün-Kodu
HAVING SUM (Miktar) > 1000
```

9.5 Sorgu Özelliği

<Sorgu Özelliği>'nin birincil fonksiyonu, bir tablo ifadesi tarafından tanımlanmış bir tablo veya gruplanmış tablodan bazı veya tüm kolonları seçmek için türemiş bir tablonun (T) izdüşümünü almaktır. Kolonları seçmesine ek olarak, <Sorgu

<Özellik> değer ifadeleri kullanarak kolonlar da oluşturulabilir. Bu değer ifadeleri tabloya kolon veya kolonlar sokarlar. Format yapısı şöyledir :

<Sorgu Özelliği> ::= SELECT [ALL | DISTINCT] <Seçim Listesi>
<Tablo İfadesi>

<Seçim Listesi> ::= * | <Seçim Alt Listesi> [{ , <Seçim Alt Listesi> }]

<SeçimAltListesi> ::= <niteleyen>* | <Değerİfadesi> [[AS] <Kolon adı>]

sonuç tablosunun kolonlarını tanımlar. İki uygun kısa yol vardır, * ve <Qualifier> *. Birincisi, *, tablo ifadesinin sonucundaki tüm kolonları gösterir. İkincisi, <Qualifier> *, <Qualifier> tarafından belirlenen tablonun, ortak kolonlar hariç, tüm kolonlarına işaret eder. Dikkat edilmelidir ki, *, yürütme anındaki tüm uygulanabilirleri ifade eder., ve temel tabloya giden kolonların ilave sıralarından etkilenmez. Ortak kolon, join edilmiş bir tablodaki aşıkâr veya kapalı <Join Kolon Listesi>'nde meydana gelen bir kolondur. (NATURAL'ın kullanımı ile). Bunun sebebi şudur; bu kolonlar, join edilmiş iki tablodaki kolonlar üzerinde bulunan bir COALESCE fonksiyonundan türerler ve orijinal kolonlar sonucun daha uzun parçası değildirler. Kolonlar AS opsiyonu kullanılarak denk geldikleri kolonların isimlerini alırlar.

Son olarak, eğer DISTINCT belirtilmişse (Default ALL'dır) tekrarlanmış satırlar sonuçtan çıkarılırlar. O zaman,

SELECT ALL İşçino FROM Seyahat-Talebi

herbir Seyahat-Talebi için geçerli bir satırlı sonuç tablosu oluşturur, halbuki,

SELECT DISTINCT İşçino FROM Seyahat-Talebi

Seyahat Talebinde bulunan işçilerin listesinden oluşan bir sonuç tablosu oluşturur, ama burada hiçbir işçi sadece bir kere gözükür, kaç kere talepte bulunduğu bakılmaz.

9.6 Alt Sorgular

Alt sorgular üç dala ayrılır: Ölçek, satır ve tablo. Herbiri parantez içinde bir <Sorgu İfadesi>dir.Örneğin :

```
( SELECT BlmAdı FROM Bölüm WHERE BlmKod = 'BIM' )
```

tek kolon ve satır oluşturan ölçek bir alt sorgudur. Ayrıca bir satır alt sorgusu veya bir tablo alt sorgusu gibi de hareket ettirilebilir.

```
( SELECT * FROM Bölüm WHERE BlmKod = 'BIM'
```

birçok kolonu ama sadece tek bir satırı oluşturan bir satır alt sorgusudur. Tabii olarak bir tablo alt sorgusu olarak da uygulanabilir.

```
( SELECT BlmAdı FROM Bölüm )
```

Bir satırdan fazla satıra sahip olan Bölüm tablosundan oluşturulan bir satır alt sorgusudur. Bir ölçek alt sorgusu prensipte tek bir değere ihtiyaç duyan her yerde kullanılabilir. Bir ölçek alt sorgu tüm veri ifadeleri sentaksında ortak önceliğe sahip bir opsiyondur. Bir satır alt sorgusu veri yapıcıya ihtiyaç duyulan her yerde kullanılabilir, çoklukla önermelerde. Bir tablo alt sorgusu bir tabloya ihtiyaç duyulan her yerde örneğinde veya IN önermesinde kullanılabilir.

Bir değer ifadesindeki bir ölçek alt sorgu tüm sorgu ifade yollarını açar. İhtimalleri göstermek için bu özelliği kullanarak bir çift sorgu hazırlanmıştır aşağıda. Tüm programcıların isimlerini seçmek için şu anki maaşları ile önceki yüksek maaşları arasındaki fark baz alınarak şu sorgu kullanılır :

```
SELECT İsim, Maaş -  
( SELECT MAX (Maaş)  
FROM Meslek-geçmişi  
WHERE Meslek-geçmişi.İşçiNo = İşçi.İşçiNo )
```

```
FROM İşçi
WHERE Meslek = 'PROGRAMCI'
```

işleri için belirtilmiş olan bir yüzdeyle tüm işçilerin maaşlarını yükseltmek için (meslek ismini ve yükseltme oranını yani 10percent =.10 içeren bir geçici tablo varsayın. Aşağıdaki tablo kullanılır :

```
UPDATE İşçi
    KÜME Maaş = Maaş *
        ( 1 + ( SELECT yükseltme-oranı
                FROM yükseltme-tablosu R*
                WHERE R.Meslek = İşçi.Meslek)
```



BÖLÜM 10. ÖNERMELER VE ARAMA ŞARTLARI

Önermeler; tablolardan satırları seçmek için arama şartlarını belirtmede veya kayıtların olup olmadığını belirlemede kullanılmaktadır.

10.1 Arama Şartları

Arama şartı, doğru yada yanlış değeri olan mantıksal operatörlerle bağlanmış cümle yada cümle gruplarından oluşur. Burada mantıksal operatörler doğru, yanlış veya belirsiz değer alabilirler. Arama şartları belli bir kritere göre satırlarda veya satır gruplarında arama yapar [1]. Arama şartlarının notasyonu aşağıdaki gibidir;

$\langle \text{arama şartı} \rangle ::= \langle \text{mantıksal terim} \rangle | \langle \text{arama şartı} \rangle \text{OR} \langle \text{mantıksal terim} \rangle$

$\langle \text{mantıksal terim} \rangle ::= \langle \text{mantıksal faktör} \rangle | \langle \text{mantıksal terim} \rangle$

$\text{AND} \langle \text{mantıksal faktör} \rangle$

$\langle \text{mantıksal faktör} \rangle ::= [\text{NOT}] \langle \text{mantıksal test} \rangle$

$\langle \text{mantıksal test} \rangle ::= \text{mantıksal birincil}$

$[\text{IS} [\text{NOT}] \{ \text{TRUE} | \text{FALSE} | \text{UNKNOWN} \}]$

$\langle \text{mantıksal birincil} \rangle ::= \langle \text{önerme} \rangle | \langle \text{arama şartı} \rangle$

Bir mantıksal test sırasında 'x IS NOT y' ifadesi 'NOT x IS y' ifadesi ile eşdeğerdir. Birden fazla önerme içeren arama şartlarında işlem sırası şu şekildedir;

- 1-) Parantezler
- 2-) NOT'dan önce gelen IS
- 3-) AND
- 4-) OR

Eğer aynı önceliğe sahip iki operatör varsa öncelik soldan sağa doğrudur. Aşağıdaki tablolarda her operatörün doğruluk tabloları verilmiştir

OR	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

Şekil 10.1 OR Doğruluk Tablosu

OR	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

Şekil 10.2 AND Doğruluk Tablosu

NOT	
true	false
false	true
unknown	unknown

Şekil 10.3 NOT Doğruluk Tablosu

IS	true	false	unknown
true	true	false	false
false	false	true	false
unknown	false	false	true

Şekil 10.4 IS Doğruluk Tablosu

10.2 Önermeler

Önerme, doğru, yanlış veya bilinmeyen değer alabilen bir cümledir. SQL'de 10 çeşit önerme vardır:

- Karşılaştırma Önermesi : true, false, unknown değerlerini alır.
- Arasında Önermesi : true, false, unknown değerlerini alır.
- Benzerlik Önermesi : true, false, unknown değerlerini alır.
- Boş olma Önermesi : true, false değerlerini alır.
- İçinde olma Önermesi : true, false, unknown değerlerini alır.
- Nicelenmiş Karşılaştırma Önermesi : true, false, unknown değerini alır.
- Eşleme Önermesi : true, false değerlerini alır.
- Çakışma Önermesi : true, false, unknown değerlerini alır.
- Varolma Önermesi : true, false değerlerini alır.
- Tek olma Önermesi : true, false değerlerini alır.

10.2.1 Karşılaştırma Önermesi

Bu önerme, iki satır yapısını karşılaştırmak için kullanılır [2]. Bu önermenin yazım formatı şu şekildedir;

$\langle x \rangle \langle op \rangle \langle y \rangle$

$\langle x \rangle$, $\langle y \rangle$: Aynı derecede iki satır değeridir. Bunların aldığı değerler karşılaştırılabilir olmalıdır. Yani karakter dizisi, sayısal, tarih-zaman veya aralık sözcüğü olabilir. $a \langle op \rangle b$ en basit karşılaştırma a veya b den biri null değere sahipse o zaman karşılaştırma sonucu bilinmeyen değerini alır. Diğer karşılaştırma durumları;

$a = b$	$a \langle \rangle b$	$a < b$
$a > b$	$a \leq b$	$a \geq b$

Karşılaştırma kuralı karşılaşılan değerlerin tipine bağlıdır.

- Nümerik değerler, cebirdeki kurallara göre yapılır.

- İkili diziler, soldan sağa karşı gelen ikililer karşılaştırılır.

- Karakter dizileri, karşılaştırılan dizilerin uzunlukları eşit değilse (PAD SPACE kullanılmışsa), kısa olan dizinin sağ tarafı uzun olanın boyuna kadar boşluklarla doldurulur. Eğer NO PAD tanımlanmışsa, uzunluğu kısa olan büyük olandan küçük kabul edilir.

- Tarih-zaman değerlerinin karşılaştırılması, iki değer farkının alıp sonucu sıfır aralığı ile karşılaştırmakla yapılır. Çıkarmanın hassasiyeti işleme giren tarih-zaman alanlarının son hanelerine bağlıdır. Bu nedenle tarih-zaman alanları aynı hassasiyete sahip olmalıdır.

Karşılaştırma önermelerine örnek verecek olursak;

- Personel tablosunda maaşı 15000 üzerinde olan personel numaraları;

SELECT perno FROM Personel WHERE maas = 15000

- Bilgi işlemde bölümünde çalışan kadınların personle numaraları;

SELECT perno FROM Personel

WHERE (cins, blmkod) = ('K', 'BIM')

- Aynı cinsiyete sahip ve aynı bölümde çalışan, delege olarak 12 nolu seçmenlerde çalışan personel numaraları;

SELECT perno

FROM Personel

WHERE (cins, blmkod) = (SELECT cins, blmkod

FROM Personel P

WHERE (SELECT delege

FROM Secmen

WHERE secmenler = 12) = P.perno)

10.2.2 Arasında Önermesi

Bu önerme, bir satır değerli yapının, başka iki ifadenin arasında olup olmadığını kontrol eder. Bu önermenin iki çeşit yazım biçimi vardır;

1. Yazım: `x BETWEEN y AND z`

2. Yazım: `x NOT BETWEEN y AND z`

`SELECT perno FROM Personel`

`WHERE maas BETWEEN 15000 AND 45000`

10.2.3 Benzerlik Önermesi

Bu önerme sadece karakter dizilerinin, belirlenen bir şekle uyup uymadığını test etmek için kullanılır.

`<x> [NOT] LIKE <y> [ ESCAPE <z> ]`

`<x>`, `<y>`, `<z>`: karakter değerli ifadelerdir.

`NOT ( <x> LIKE <y> [ ESCAPE <z> ] )` ifadeside yukarıdaki ile eşdeğerdir. Test edilecek şekil şu şekilde belirlenir;

- `_` (alt çizgi): Tek bir karakteri temsil eder

- `%` : Sıfır veya daha fazla karaktere sahip diziye temsil eder

bunun dışındaki tüm karakterler ise kendilerini temsil ederler. Eğer `<z>` varsa, bu tek bir karakter olmalıdır, ve kendisinden, `'_'` dan, `'%'` dan sonra gelmelidirler.

Örnekler:

- `ad LIKE 'Can%'` ifadesi eğer `ad` 'Can' ile başlıyorsa doğru olacaktır

- `deger LIKE '5_#%' ESCAPE '#'` ifadesi eğer, `deger` kolonu '5.' ile başlıyor ve arkasından gelen değer `'_'` ile yani herhangi bir değer, ve ondan sonra sıfır veya daha fazla karaktere sahip dizi geliyorsa doğru değerini alacaktır.

- `'ahmet' LIKE 'A%HM%ET'` ifadesi doğrudur.

10.2.4 Boş Olma Önermesi

Bu önerme, satır değerinin boş olup olmadığını kontrol eder.

x IS NULL veya

X IS NOT NULL şeklinde yazılır. X 'in belirttiği satırdaki değerler eğer NULL ise o zaman bu önerme doğru değerini alır. NULL önermesinin tüm durumlarını gösteren doğruluk tablosu aşağıdadır:

x 'in elemanlarının değeri	x IS NULL	x IS NOT NULL	NOT x IS NULL	NOT x IS NOT NULL
bütün elemanlar NULL	true	false	false	true
bazıları NULL, bazıları değil	false	false	true	true
hiçbir eleman NULL değil	false	true	true	false

Şekil 10.5 NULL Doğruluk Tablosu

10.2.5 İçinde Önermesi

Bu önerme, bir satır değeri yapısının, alt sorgulama tarafından tanımlanan tablo içinde olup olmadığını test eder. Bu önermenin yazım şekli şöyle;

x NOT IN y veya NOT (x IN y) aynı anlamı ifade eder.

Örnek: $blmkod$ IN ('BIM' , 'SAT')

10.2.6 Nicelenmiş Karşılaştırma Önermesi

Bu önerme, bir satır değeri yapısının, bir tablo altsorgulamasında elde edilen değerlerin karşılaştırılmasında kullanılır. Tablo altsorgulamasından elde edilen

değerlerin, satırlardakine uygun olup olmadığını kontrol eder, eğer uygunsa önerme doğru, uygun değilse önerme yanlış değerini almaktadır. Yazım biçimi;

<satır değer yapısı> <op> { ALL | SOME | ANY } <tablo altsorgu>

satırdaki değerlerin yapısı, tablo altsorgulamasındaki kolonların sayısı kadar olmalıdır. <op> herhangi bir karşılaştırma operatörüdür. SOME ve ANY eşanlamlıdır, o halde burada etkili iki seçenek var: ALL ve ANY.

Eğer 'rve' bir satır değer yapısını, 'tsr' bir tablo altsorgulamasının sonucunu gösteriyorsa bu durumda, karşılaştırma önermesi 'rve <op> tsr' alt sorgulamanın bütün satırlarına uygulanır.

Aşağıdaki durumlarda bu önerme doğrudur:

ALL opsiyonu tanımlanmışsa

- eğer altsorgulamanın sonucunda oluşan tablo boş ise
- eğer karşılaştırma önermesi, altsorgulamadaki her satır için doğru ise

Eğer ANY tanımlanmış ve önerme sorgulamadaki en az bir satır için doğru ise

Aşağıdaki durumlarda bu önerme yanlıştır:

ANY opsiyonu tanımlanmışsa

- eğer altsorgulamanın sonucunda oluşan tablo boş ise
- eğer karşılaştırma önermesi, altsorgulamadaki her satır için yanlış ise

Eğer ALL tanımlanmış ve önerme sorgulamadaki en az bir satır için yanlış ise

Örnek:

'BIM' = ANY (SELECT blmkod FROM Bolum) İfadesi eğer en az bir satır 'BIM' içeren bir satır bulunursa doğru olacaktır

SELECT perno FROM Per_Ext

WHERE dogum_trh <= ALL

(SELECT dogum_trh FROM Per_Ext)

Bu ifade de, Personel tablosundan en yaşlı elemanlar isteniyor.

10.2.7 Çakışma önermesi

Bu önerme, çakışan zaman aralıklarını kontrol eder. Karşılaştırılabilir önerme gruplarının birlikte kullanılmasıyla yazılır:

(a, b) OVERLAPS (c, d)

a, c: Karşılaştırılabilir tarih-zaman tipi

b, d: Karşılaştırılabilir tarih-zaman tipi veya a ve c'ye eklenecek zaman aralıkları.

Örnek:

(DATE '1989-02-01', DATE'1989-05-01')

OVERLAPS

(DATE '1989-05-01', DATE'1989-06-01')

bu önerme yanlıştır,

(DATE '1989-06-03', INTERVAL '-3' DAY)

OVERLAPS

(DATE '1989-05-01', DATE'1989-06-01')

bu önerme doğrudur.

10.2.8 Eşleme Önermesi

Bu önerme, birbirlerine uyan satırları test eder. Bu önermenin gösterimi şu şekildedir;

<satır değer yapısı> MATCH [UNIQUE]

[PARTIAL | FULL] <tablo altsorgulama>

Burada <satır değer yapısı>ndaki değerlerin adedi, tablo altsorgusundaki kolonların sayısı ile aynı olmalıdır. Bu önermenin kullanıldığı bir altsorgulama yazacak olursak:

```
CHECK ( ( perno, plaka_no )
```

```
    MATCH ( SELECT perno, plaka_no FROM Personel ) )
```

Burada Taşıt tablosundaki her satırdan perno ve plaka_no kolonları alınır; Personel tablosundan seçilen perno ve plaka_no ile karşılaştırılır; eşit olup olmadığı kontrol edilir. Her satır bu şarta uyuyorsa önerme doğru, diğer durumlarda yanlış olur.

UNIQUE belirtilmişse; bu önermede verilen değerlerin sadece uyup uymadığına bakmaz, aynı zamanda uyan bu değerlerin tek olup olmadığına da bakar. Eğer uyan birden fazla satır varsa o zaman önerme yanlış olur.

PARTIAL belirtilmişse; dolu olan kolonlarda uyan kayıtlar varsa önerme doğru değerini alır, diğer durumlarda yanlış olur.

FULL belirtilmişse;eğer tüm satırlar boş ise önerme doğru, en az bir boş değer varsa önerme yanlış değerini almaktadır. Eğer tüm değerler boş değilse; ve şarta uyan bir kayıt bulunursa önerme doğru olur.

10.2.9 Var Olma Önermesi

Bu önerme diğerlerine göre farklıdır; altsorgulamanın sonucuna benzer özellikte değerler alır. EXISTS bir tablonun boş olup olmadığını kontrol eder. Bu nedenle, EXISTS sadece doğru veya yanlış değeri alır. Yazım biçimi:

```
EXISTS <altsorgulama>
```

Örnek: EXISTS (SELECT * FROM Bolum) bu sorgulamanın sonucunda eğer Bolum tablosu en az bir satır içeriyorsa önerme doğru, diğer durumlarda yanlış olacaktır.

10.2.10 Tek Olma Önermesi

Bu önerme, boş olmayan kayıtlardan birden fazla bulunup bulunmadığını kontrol eder. Bu önermenin yazım biçimi şu şekildedir;

UNIQUE <altsorgulama> . Bu önermenin sonucu eğer altsorgulama sonucu elde tekrar etmeyen satırlar var ise doğrudur, diğer durumlarda ise yanlıştır.

Örnek: UNIQUE (SELECT blmkod FROM Bolum) önermesi; eğer Bolum tablosunda aynı bölüm koduna sahip kolon varsa önerme sonucu yanlış, değilse doğru olarak dönecektir.

BÖLÜM 11. FONKSİYONLAR VE İFADELER

Bu bölümde, çeşitli nümerik değerleri ve küme fonksiyonlarını hesaplamak için kullanılan ifadelerden bahsedilecektir [1].

11.1 Değer İfadeleri

Gerçek değerlerin ve operatörlerin kombinasyonundan türeyen değerler; bu ifadenin değerini oluşturur. Gerçek değerler, sözcükler, parametreler, bağlanan dilin değişkenleri, kolonlar, fonksiyonlar ve bunlardan türeyenlerdir. Sonucun veri tipine bağlı olarak birçok çeşit değer ifadeleri vardır. Bunlar;

<değer ifadesi> ::= <sayısal değer ifadesi>
<karakter dizi değer ifadesi>
<tarikh-zaman değer ifadesi>
<aralık değer ifadesi>

Geçerli olan gerçek değerler, 'genel gerçek' olarak adlandırılırsa;

<genel gerçek> ::= <işaretsiz değer tanımı>
<kolon referansı>
<küme fonksiyon tanımı>
<skaler altsorgulama>
<durum ifadesi>
<döküm ifadesi>
<değer ifadesi>

11.1.1 Sayısal Değerli İfadeler

Bu ifadeler, tam yada yaklaşık sayısal veri tipine uygun olan değerler sonuçlandırır. Bunun yazım biçimi:

```
<sayısal değerli ifade> ::= <terim>  
    | <sayısal değerli ifade> + <terim>  
    | <sayısal değerli ifade> - <terim>  
  
<terim> ::= <faktör> | <terim>*<faktör> | <terim>/<faktör>  
  
<faktör> ::= [ + | - ] <sayısal gerçel>  
  
<sayısal gerçel> ::= <genel gerçel>  
    | <konum ifadesi>  
    | <çıkarma ifadesi>  
    | <karakter uzunluk ifadesi>  
    | <sekizli uzunluk ifadesi>  
    | <ikili uzunluk ifadesi>
```

Örnek:

- 3.14
- miktar + 1000
- (SELECT MAX(maas) FROM Personel)
/ (SELECT AVG(maas) FROM Personel)
- CHARACTER_LENGTH(adres) + CHARACTER_LENGTH(ad)

Herhangi bir <genel gerçel> veri tipi sayısal olmalıdır. İfadenin sonucunun veri tipi; işleme ve işleme giren terimlere bağlıdır. (+,-) veri tipini değiştirmez. '+', '-', '*', '/' işlemlerinde sonuç tam yada yaklaşık sayısal olabilir. Eğer <sayısal gerçel> boş veya <skaler altsorgulama> da sonuç boş bir tablo ise, ifade de boş değerini alır. İşlem sırası, önce parantez içi olmak üzere; '*', '/' ve daha sonra '+', '-', gelmektedir. Aynı önceliğe sahip iki operatörde, işlem sırası soldan sağa doğrudur.

11.1.2 Karakter Dizi Değerli İfadeler

Bu ifadelerin sonucu, ikili veya karakter dizisi olan ifadelerdir.

```
<karakter dizi değeri ifade> ::=
    <karakter değeri ifade> | <ikili değeri ifade>
```

<karakter değerli ifade> ::=

<karakter değerli ifade> || <karakter gerçel>[<karşılaştırma cümlesi>]

|<karakter gerçel>[<karşılaştırma cümlesi>]

<karakter gerçel> ::= <genel gerçel>

| <alt dizi>

| <upper>

| <lower>

| <karakter çevirme>

| <kullanım şekli çevirme>

Örnek: USER

UPPER ((SELECT MAX(ad) FROM Personel))

Eğer <karakter gerçel> boş veya <skaler altsorgulama> da sonuç boş bir tablo ise, ifade de boş değerini alır. || operatörü, karakter dizi toplama operatörüdür, ikinci diziye birincinin sonuna ekler.

<ikili değerli ifade> ::= <ikili değerli ifade> || <ikili gerçel>|<ikili gerçel>

<ikili gerçel> ::= genel gerçel> | <altdizi>

Burada <genel gerçel> ve <alt dizi>nin veri tipi ikili olmalıdır. Eğer <ikili gerçel> boş veya <skaler altsorgulama> da sonuç boş bir tablo ise, ifade de boş değerini alır.

Örnek: x'2F' || B'010111'

11.1.3 Tarih-Zaman değerli ifadeler

Bu ifadenin sonucu tarih-zaman olur. Bu ifadelerin yazım biçimi;

<tarıh-zaman değeri ifade> ::=
 <tarıh-zaman terimi>
 <aralık değeri ifadesi> + <tarıh-zaman terimi>
 | <tarıh-zaman değeri ifadesi> {+|-} <aralık terimi>
 <tarıh-zaman terimi> ::= <tarıh-zaman gerçel> [<zaman dilimi>]
 <tarıh-zaman gerçel> ::= <genel gerçel>
 | <güncel tarıh fonksiyonu>
 | <güncel saat fonksiyonu>
 | <güncel timestamp fonksiyonu>
 <zaman dilimi> ::= AT {LOCAL | TIME ZONE <basit değeri tnm>}

Örnekler:

başlama_trh
 INTERVAL '3' MONTH + giriş_trh
 doğum_trh - INTERVAL '9' MONTH

Herhangi bir <genel gerçel> veri tipi tarıh-zaman olmalıdır. Eğer <tarıh-zaman gerçel> boş veya <skalere altsorgulama> da sonuç boş bir tablo ise, ifade de boş değerini alır. İşlem sırası, önce parantez içi olmak üzere soldan sağa doğrudur.

11.1.4 Aralık Değeri İfadeler

Bu ifade, tipi aralık tipi olan değerlere sonuçlandırır. Bu ifadenin gösterimi şöyledir;

<aralık değeri tipi> ::= <aralık terimi>
 | (<tarıh-zaman değeri ifade> - <tarıh-zaman terimi>) <aralık niteleyici>
 | <aralık değeri ifade> {+ | -} <aralık terimi>

$\langle \text{aralık terimi} \rangle ::= \langle \text{aralık faktörü} \rangle$
 $| \langle \text{aralık terimi} \rangle * \langle \text{faktör} \rangle$
 $| \langle \text{aralık terimi} \rangle / \langle \text{faktör} \rangle$
 $| \langle \text{terim} \rangle * \langle \text{aralık faktörü} \rangle$

$\langle \text{aralık faktörü} \rangle ::= [+, -] \langle \text{genel gerçel} \rangle [\langle \text{aralık niteleyicisi} \rangle]$

Örnekler:

$[\text{giriş_trh} - \text{doğum_trh}] \text{ YEAR (işe kaç yaşında başladığı)}$

$\text{MAX} ((\text{gelis_trh} - \text{gidis_trh}) \text{ DAY} + 1 \text{ DAY})$ (en uzun kalış süresi)

Herhangi bir $\langle \text{genel gerçel} \rangle$ veri tipi aralık olmalıdır. Eğer $\langle \text{aralık gerçel} \rangle$ boş veya $\langle \text{skaler altsorgulama} \rangle$ da sonuç boş bir tablo ise, ifade de boş değerini alır. Diğer durumlarda, normal tarih ve zaman işlemleri kuralları kullanılarak aralık değerleri için toplama, çıkarma, çarpma ve bölme işlemleri yapılır. İşlem sırası, önce parantez içi olmak üzere soldan sağa doğrudur.

11.2 Skaler Fonksiyonlar

Bu fonksiyonlar, kolonlara atama yapmakta, değer ifadelerinin sıralanmasında veya karşılaştırılmalarında kullanılır.

11.2.1 Tarih-Zaman Fonksiyonları

Bu fonksiyonlar, ya tarih-zaman veri tipini döndürür yada tarih-zaman değeri üzerinde işlem yapar.

CURRENT_DATE fonksiyonu: Zaman dilimine göre güncel olan tarihi dönüştürür. Fonksiyonun sonucu DATE tipindedir.

<güncel tarih fonksiyonu> ::= CURRENT_DATE.

CURRENT_TIME fonksiyonu: Zaman tipine göre güncel olan zamanı verir. Sonucun tipi TIME tipindedir. <zaman hassasiyeti> doğruluğu belirler ve varsayılanı sıfırdır.

<güncel zaman fonksiyonu> ::= CURRENT_TIME [(<hassasiyet>)]

Örnek: CURRENT_TIME saniye hassaslığında

CURRENT_TIME (6) mikrosaniye hassaslığında

CURRENT_TIMESTAMP fonksiyonu: Zaman dilimine göre güncel olan tarih ve zamanı verir. Sonucun tipi TIMESTAMP tipindedir. <zaman hassasiyeti> doğruluğu belirler ve varsayılanı 6, başka deyişle mikro saniyedir.

<güncel timestamp fonksiyonu> ::= CURRENT_TIMESTAMP
[(<hassasiyet>)]

Örnek: CURRENT_TIMESTAMP mikrosaniye hassaslığında

CURRENT_TIME (0) saniye hassaslığında

EXTRACT fonksiyonu: Bu fonksiyon, bir tarih zaman alanından veya aralıktan, istenen kısmını alır. Eğer tarih-zaman veya aralık boş değere sahipse, sonuç değer de boş olur. Fonksiyonun sonucu sayısal bir değerdir.

<çıkarılan ifade> ::= EXTRACT

({ YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
| TIMEZONE_HOUR | TIMEZONE_MINUTE })

FROM { <tarih-zaman değerli ifade> | <aralık değerli ifade> })

Örnek:

EXTRACT (YEAR FROM Personel.doğum_trh)

Personelin doğduğu yılı verir.

11.2.2 Karakter Dizi Fonksiyonları

Bu gruptaki fonksiyonlar, karakter veya ikili diziler üzerinde işlem yaparlar. Sonuç değer, işleme girmeden evvelki veri tipinde veya karakter, ikili veya sayısal değer olur. Eğer dizilerden biri boş ise sonuç boş olacaktır.

SUBSTRING fonksiyonu: Bir dizinin verilen kısmını alt diziye çevirir.

<altdizi> ::= SUBSTRING (<dizi değerli ifade>

FROM <başlangıç konumu> [FOR <dizi uzunluğu>])

<dizi değerli ifade> ::= karakter veya ikili dizidir sonuç buna bağlıdır.

Eğer <dizi uzunluğu> verilmemişse,

alınacak karakter sayısı= dizinin uzunluğu - başlangıç konum+1 veya 0.

Örnek:

SUBSTRING ('mustafa' FROM 3 FOR 2) sonuç: 'st'

SUBSTRING ('mustafa' FROM 8) sonuç: hatalı -negatif

varsayılan kayıt uzunluğu $7-8+1 = 0$ dır.

SUBSTRING (B'1010101' FROM 5 FOR 3) sonuç: '101'

CHARACTER_LENGTH fonksiyonu: Bir dizideki karakterlerin sayısını verir. Yazım biçimi şu şekildedir:

<karakter uzunluk fonksiyonu> ::=

{CHAR_LENGTH | CHARACTER_LENGTH }<dizi değerli ifade>}

Örnek:

CHAR_LENGTH ('XF0F0') sonuç:2

CHARACTER_LENGTH ('mustafa') sonuç:7

OCTET_LENGTH fonksiyonu: Dizinin uzunluğunu sekizli olarak verir.

BIT_LENGTH fonksiyonu: Dizinin uzunluğunu ikili olarak verir.

11.2.3 Karakter fonksiyonları

Bu gruptaki fonksiyonlar, karakter üzerinde işlem yaparlar sonuç ise, karakter veya sayısal bir değerdir. Eğer herhangi bir dizi boş ise, sonuç boştur.

POSITION fonksiyonu: Bir dizinin başka dizi içindeki pozisyonunu verir.

<konum ifadesi> ::= POSITION
(<karakter değerli ifade> IN <karakter değerli ifade>)

Eğer ilk dizinin uzunluğu sıfır ise, sonuç 1 olur. Diğer durumlarda birinci dizinin ilk karakterinin, ikinci dizide başladığı yere kadar olan karakter sayısı alınır. Böyle bir dizi yoksa sonuç sıfırdır.

POSITION ('sta' IN 'mustafa') sonuç:3

POSITION ('a' IN 'mustafa') sonuç:5

POSITION ('el' IN 'mustafa') sonuç:0

TRIM fonksiyonu: Bu fonksiyonla, bir dizinin başından veya sonundan istenildiği kadar karakter silinir.

<trim fonksiyonu> ::= TRIM ([[{ LEADING|TRAILLING|BOTH }]
[<trim karakter>] FROM] <trim kaynak>)

LEADING veya TRAILLING tanımlanmamışsa, BOTH varsayılandır.

Örnek:

TRIM (LEADING '*' FROM '***ahmet') sonuç 'ahmet'

TRIM (' a h m e t ') sonuç:'ahmet'

UPPER fonksiyonu: Küçük harfleri büyük harflere çevirir.

LOWER fonksiyonu: Büyük harfleri küçük harflere çevirir.

UPPER('ahmet') sonuç:'AHMET'

LOWER('AHMET') sonuç:'ahmet'

TRANSLATE fonksiyonu: Çeviri tanımı kullanarak, bir karakter kümesini başka bir kümeye çevirir.

<karakter çevirisi> ::= TRANSLATE

(<karakter değerli ifade> USING <çeviri adı>)

TRANSLATE ('ÇAĞATAY' USING TRK_TO_ING)

sonuç: 'CAGATAY'

CONVERT fonksiyonu: Kullanılan fiziksel temsil şeklini başka birine çevirir. <kullanım şekli> ::= CONVERT

(<karakter değer ifade> USING <dönüşüm ad>)

CAST fonksiyonu: bir veri tipini başka bir veri tipine çevirir.

CAST({<değer ifadesi>|NULL} AS {<tanım kümesi adı>|<veri tipi>})

Örnek: CAST ('100' AS FLOAT)

Aşağıdaki Tablo 11.1 de izin verilen veri tipi dönüşümü verilmiştir.

Bu tabloda E: kaynak veri tipi hedef veri tipine CAST ile çevrilebilir.

H: kaynak veri tipi hedef veri tipine CAST ile çevrilemez.

kaynak veri tipi	hedef veri tipi								
	tam sayısal	yaklaşık sayısal	karakter	ikili	tarih	zaman	time-stamp	yıl-ay aralığı	gün-zaman aralığı
tam sayısal	E	E	E	N	N	N	N	tek tarih-zaman alanı varsa	tek tarih - zaman alanı varsa
yaklaşık sayısal	E	E	E	N	N	N	N	N	N
karakter	E	E	repertuarı aynı ise	E	E	E	E	E	E
ikili	N	N	E	E	N	N	N	N	N
tarih	N	N	E	N	E	N	E	N	N
zaman	N	N	E	N	N	E	E	N	N
time-stamp	N	N	E	N	E	E	E	N	N
yıl-ay aralığı	tek tarih-zaman alanı varsa	N	E	N	N	N	N	E	N
gn-zmn aralığı	tek tarih-zaman alanı varsa	N	E	N	N	N	N	N	E
bilinmeyen	E	E	E	E	E	E	E	E	E

Şekil 11.1 Geçerli Veri Tipi Dönüşümü

11.3 Şartlı Değer İfadeleri

Bu ifadeler, bir şarta bağlı olarak değer seçerler. Diğer şart ifadeleri ile karıştırılmamalıdır (IF THEN ELSE vb. Bu ifadeler zaten SQL de bulunmazlar). Burada sadece fonksiyon ile dönen değer şarta bağlıdır.

CASE fonksiyonu: Bir şarta bağlı olarak değer döndürür.

```
<durum ifadesi> ::= CASE [<durum op.>]<when cümlesi> ...
                        [else cümlesi> ] END
<durum op.> ::= <değer ifadesi>
<when cümlesi> ::= WHEN {<arama şartı>|<when op.>}
                        THEN <sonuç>
<else cümlesi> ::= ELSE <sonuç>
<sonuç> ::= {<değer ifadesi> |NULL }
<when op.> ::= <değer ifadesi>
```

Örnek:

```
CASE cins WHEN 'K' THEN 'KADIN'
           WHEN 'K' THEN 'KADIN'
           ELSE 'Bilinmiyor' END
CASE ağırlık WHEN <45 THEN 'zayıf'
              WHEN BETWEEN 46 AND 60 THEN 'ideal'
              ELSE 'şişman' END.
```

NULLIF fonksiyonu: Eğer verilen bir değer bulunursa, onun yerine NULL değerini koyar. NULLIF (<değer ifadesi>, <değer ifadesi>) Bu ifade

NULLIF (D1, D2) olarak gösterilirse

CASE WHEN D1=D2 THEN NULL ELSE D1 END ifadesi ile aynı.

Bu fonksiyon sıfıra bölme gibi hataları önlemede kullanılabilir.

Örnek:

```
SELECT maas / (NULLIF(prim,0) FROM Personel)
```

COALESCE fonksiyonu: Belirtilen değerler listesinden ilk NULL olmayan değeri getirir. COALESCE(<değer ifadesi>{, <değer ifadesi>}...)

COALESCE (D1, D2) olarak gösterilirse aşağıdaki ifade ile aynı:

```
CASE WHEN D1 IS NOT NULL THEN D1 ELSE D2 END .
```

11.4 Küme Fonksiyonları

Skaler fonksiyonlar, bir değer üzerinde işlem yaparlar, küme fonksiyonları ise değerler kümesi üzerinde işlem yaparlar.

<küme fonksiyon tanımı> ::= COUNT(*)
 | {AVG | MAX | MIN | SUM | COUNT }
 ([DISTINCT | ALL] <değer ifadesi>)

AVG ve SUM; sadece sayısal ve aralık değerleri için kullanılır. <değer ifadesi> başka bir küme fonksiyonu veya altsorgulama içeremez.

Örnek:

```
SELECT c1
FROM T1 A
GROUP BY c1
HAVING EXISTS (SELECT *
                FROM T2 B
                WHERE B.c2 = ( SELECT SUM(D.c3)
                              FROM T3 d
                              WHERE D.c4=SUM( B.c5 ) ) )
```

COUNT fonksiyonu: Bu fonksiyon satırları sayar. 3 şekilde kullanılabilir;

COUNT(*), COUNT DISTINCT, COUNT ALL.

Örnek:

```
SELECT COUNT (*) FROM Personel
```

Personel tablosundaki tüm satırlarını gösteren tek satırlı bir tablo.

```
SELECT COUNT (ALL ftr_str) FROM Fatura_detay
```

ftr_str kolonundaki tüm satırların adedini gösteren tek satırlı bir tablo.

```
SELECT COUNT (DISTINCT ftr_no) FROM Fatura_detay
```

ftr_no kolonundaki farklı kayıtların adedini gösteren tek satırlı bir tablo.

AVG fonksiyonu: Boş olmayan değerlerin ortalamasını verir.

SELECT AVG (ALL maas) FROM Personel.

SELECT AVG (DISTINCT maas) FROM Personel.

MAX fonksiyonu: Boş olmayan değerlerin en büyüğünü verir.

SELECT MAX (ALL maas) FROM Personel.

SELECT MAX (DISTINCT maas) FROM Personel.

Burada ALL ve DISTINCT sonucu etkileyemez

MIN fonksiyonu: Boş olmayan değerlerin en küçüğünü verir.

SELECT MIN (ALL maas) FROM Personel.

SELECT MIN (DISTINCT maas) FROM Personel.

Burad ALL ve DISTINCT sonucu etkileyemez

SUM fonksiyonu: Boş olmayan değerlerin toplamının verir.

SELECT SUM (ALL maas) FROM Personel.

SELECT SUM (DISTINCT maas) FROM Personel.

BÖLÜM 12. OTURUM YÖNETİMİ

SQL_Client üzerinde çalışan bir uygulama programı ile SQL_Server arasındaki bağlantıya 'Oturum' denir. SQL_Server'da bir oturumu açma kuralları, özellikleri, bunların yerlerini belirleyecek tüm ifadeler başlangıçta tanımlanmalı [1].

Oturumu yönetme komutları, uygulama programına çeşitli SQL_Server'larla bağlantı kurmaya ve bunlar arasında bağlantıyı değiştirmeye imkan verirler. SQL'deki bir uygulama programı tek bir bulgu alanına sahiptir ve bir yada daha fazla modül kullanabilir. GET DIAGNOSTICS komutu ve oturum yönetim komutları hariç diğer tüm komutlar bir SQL_Server içinde kullanılabilirler. Bu istisna komutlar SQL_Client 'da kullanılırlar. Bir SQL_Server 'a bağlanıp oturum açıldığında, uygulama programı ile ilgili tüm modüller bu server'a taşınır. Server ile ilgili komutlar, eğer varsa mevcut oturum içinde icra edilir. Eğer aktif bir oturum yoksa varsayılan aktifleştirilir. Eğer bu yapılamazsa, orjinal SQL programından önce CONNECT TO DEFAULT komutu icra edilir. Her SQL_Client için daima varsayılan server mevcuttur ki bunun yerel yada uzaktan olması önemli değildir .

Her oturum için bir bilgi kümesi vardır. Bu küme oturum etkisiz hale getirildiğinde saklanır, tekrar aktif yapıldığında ise yeniden yüklenir. Bu bilgi kümesi şunlardan oluşur.

- oturum tanıtıcısı
- mevcut yetki tanıtıcısı
- tüm geçeci tabloların özellikleri
- açık olan tüm imleç konumları
- dinamik SQL tanıtıcı alanlarının özellikleri ve içerikleri
- SQL_oturum modülü

- mevcut varsayılan katalog ismi
- mevcut varsayılan nitelenmemiş şema ismi
- mevcut karakter küme ismi
- mevcut zaman dilimi
- tüm bütünlük kısıtlarının modları
- mevcut işlem erişim modu
- mevcut işlem yalıtım seviyesi
- mevcut işlem bulgu alan limiti

12.1 CONNECT

SQL_Client ile SQL_Server arasındaki bağlantı bu komut sayesinde olmaktadır. Bu komutun yazım biçimi şu şekilde olmaktadır [2]:

CONNECT TO

{ <SQL_Server>[AS <bağlantı adı>] [<kullanıcı adı>] } | DEFAULT

<bağlantı adı> tanımlanmazsa, SQL_Server'dan varsayılan değer alınır.

DEFAULT 'Server AS bağlantı USER kullanıcı' tanımına eşittir. Aynı bağlantı adı ile aynı anda birden fazla oturum açamaz:

SET CONNECTION <bağlantı adı>

SET SESSION AUTHORIZATION <kullanıcı adı>

12.2 SET CONNECTION

CONNECT komutu ile açılan oturumlardan birini seçmek için bu komut kullanılmaktadır. Bu komutun yazım biçimi şu şekildedir;

SET CONNECTION <bağlantı adı> | DEFAULT<bağlantı adı> o anda kullanılan veya etkisiz olan bir bağlantının adı olmalıdır. Bu komut, o anki oturumu

etkisiz yapar, 'bağlantı bilgi kümesini' saklar, >bağlantı adındaki oturumu aktif hale getirir.

12.3 DISCONNECT

Bu komut, SQL_Server ile olan bağlantıyı keser ve oturumu bitirir. Bu komutun yazım biçimi şu şekildedir:

DISCONNECT { <bağlantı adı> | CURRENT | ALL | DEFAULT }

<bağlantı adı>, aktif yada etkisiz olan bir bağlantı adı olmalıdır. CURRENT o anda aktif , ALL seçeneği ile de bütün oturumları kapatır.

Eğer kapatılacak oturumlardan biri aktifse, bir başka deyişle işlem hala çalışmakta ise ve bu işlem başladığından itibaren oturumda herhangi bir SQL komutu icra edilmiş ise hata oluşacaktır. Çünkü uygulama bitmeli (COMMIT) veya geri alınmalıdır (ROLLBACK) . Komut tarafından belirtilen tüm oturumlar kapatılır, eğer herhangi bir hata varsa bunlar bulgu alanına kaydedilir.

Bir oturumu kapatmak, ilgili tüm geçici tabloları silmek, sistem tanıtım alanlarını ve bilgi kümesinin geri kalanını boşaltmak demektir.

BÖLÜM 13. HATA BULMA VE DÜZELTME

Herhangi bir SQL komutu icra edildiğinde, bir yada daha fazla durum kodu oluşmaktadır. Bu kodlar 'bulgu alanı' içinde saklanır. Her program için sadece bir tane bulgu alanı vardır. Uygulama programı bu alanın büyüklüğünü seçebilir.

GET DIAGNOSTICS komutunun haricindeki tüm SQL komutlarının icrasından sonra, bulgu alanı silinir ve yeni değerler yerleştirilir. GET DIAGNOSTICS komutunda ise, sadece SQLSTATE parametresi değiştirilir. Tek bir komutun icrası, birden fazla hataya yol açabilir. Buna örnek verecek olursak, bir kolondaki değerin değiştirmek iki yada daha fazla kısıtın ihlal edilmesine yol açabilir.

13.1 GET DIAGNOSTICS

Bu komut, bulgu alanındaki tüm bilgiye ulaşılmasını sağlar. Bu bilgiler arasında; tanıtıcılar, ilgili tabloların, kolonların, imleçlerin, adları vardır. Varlığı bir hataya bağlı olan bir bilgi üzerinde kullanıcının hakkı yoksa, geri dönen bu bilgi sıfır uzunluğunda bir dizi olarak görünür [1]. Buna örnek verecek olursak:

```
GET DIAGNOSTICS :değiş = NUMBER
```

```
GET DIAGNOSTICS :değiş, fazla= MORE,  
sonkomut= COMMAND FUNCTION
```

```
GET DIAGNOSTICS EXCEPTION 3
```

```
: kısıtkat = CONSTRAINT_CATALOG
```

```
: kısıtsem=CONSTRAINT_SCHEMA
```

```
: kısıtadı = CONSTRAINT_NAME
```

SONUÇLAR

Bu çalışmada, veri tabanı sistemlerinde sorgulama optimizasyonu ve SQL standartları incelenmiştir.

Veri Tabanı Yönetim Sistemi (VYTS), veri tanımlama ve veri işleme olmak üzere iki parçadan oluşmaktadır. Veri işleminin bileşenleri; sorgulama, raporlama ve sistem yardımlarıdır. Sorgulama, veri tabanındaki veriyi incelemek için kullanılmaktadır. Bir sorgulamayı çözen en iyi çözümü seçme işlemine sorgulama optimizasyonu denir. Sistem, sorgulamayı çözen bütün ihtimalleri tanımayabilir. Zaten mevcut bütün çözümleri sistemin değerlendirmesi oldukça uzun zaman alacaktır. Bu sebeple, sistem, sadece mevcut olan sorguyu iyileştirmeye çalışmaktadır. O halde mevcut olan veri tabanının da iyi tasarlanmış olması gerekmektedir.

Sorguyu iyileştiren iki unsur vardır; zaman ve yer. Burada zaman, sorgulamayı çözme işleminin gerçekleşmesi için geçen süreyi, yer ise, sorgulamayı çözen tabloların sistemde ne kadar yer tutacağını ifade etmektedirler. Sorgulamayı optimize etmek, veri tabanı yönetim sisteminin görevi olmalıdır ancak mevcut veri tabanı tasarımının da iyi tasarlanmış olması gerekmektedir.

Bu tezde öncelikle bir sistem ele alınmıştır. Bu sistemin bilgi ve işlem gereksinimleri gözönüne alınarak bir veri tabanı tasarımı yapılmıştır. Bu veri tabanında amaç optimizasyon olduğundan bir tasarım yöntemi temel alınarak bu sistemin iyileştirilmesi yapılmıştır. Burada tabiki hangi veri tabanı tasarım yöntemlerinin kullanılacağı düşünülecek olunursa günlük hayatta uygulanan bir

yöntem daha iyi olacaktır. Çünkü, matematiksel tasarımlarda herhangi bir adımda yapılacak hata uygunsuz sonuçlar doğacaktır.

Tasarımın performansı için mantıksal kayıt erişim sayıları yaklaşımı kullanılmak daha iyi olacaktır. Burada dikkat edilmesi gereken nokta, yapılan tasarımın hazırlanmış olan uygulamalarda düşük maliyetli değişikliklerle, olabildiğince değişikliklere açık olmasıdır. Bu tasarım yöntemi ile mantıksal kayıt erişim sayıları ve taşıma hacimleri en aza indirgenir. Eğer mantıksal kayıt erişim sayıları indirgenirse sorgulama optimizasyonundaki zaman faktörü, taşıma hacimlerinin indirgenmesi ile yer faktörü istenene ulaşacaktır.

Tasarımdan sonra kullanılacak sorgulama dilinin de iyi belirlenmesi ile sorgulama optimize edilir. Bu çalışmada sorgulama dili olarak 1987 yılından itibaren International Standard Organization (ISO) tarafından standart bir dil olarak kabul edilen SQL kullanılmıştır. SQL ilişkisel veri tabanları için olan bir sorgulama dilidir ve ilişkisel veri tabanı modelleri içinde en yaygın olanıdır.

Sonuç olarak veri tabanı sistemlerinde sorgulama optimizasyonu yapmak için öncelikle veri tabanı tasarımının, az maliyetli ve değişikliğe açık bir tasarım olması gerekmektedir. Buna ek olarak kullanılan sorgulama dilinde optimizasyonu etkileyeceği kesindir. Burada SQL sorgulama dili optimizasyonda iyi bir sonuç verecektir.

KAYNAKLAR

- [1] CANNAN, S., OTTEN, G., SQL The Standard Handbook,
McGRAW-HILL Book Company Europe, (1993).
- [2] GROFF, J.R., WEINBERG, P.N., Using SQL,
Osborne McGRAW-HILL, (1990).
- [3] BISLAND, R.B., DATABASE MANAGEMENT Devolving
Application Systems Using ORACLE,
PRENTICE HALL, Inc, (1989).
- [4] IBM Corp, IBM AS/400 Sistem Uygulama Mimarisi SQL/400
Başvuru Kılavuzu, SC16-0328-02, Copyright IBM
Corporation, EYLÜL (1992).

EKLER

İLK VERİ TABANI TASARIMI

```
*****  
** TT1 ANA FİRMA YEDEK PARÇA TANIMLAMA HAREKETİ **  
*****
```

```
SELECT * FROM YPK_STOK  
WHERE YPK_PARNO = '6000046003';
```

```
INSERT INTO YP_STOK VALUES  
(6000046003, 'HAVA FİLTRE ELEMANI R9', 'FİLTRE', '0009',  
51000, '01', '001', '001', NULL, NULL, NULL, 15, NULL, NULL,  
42500, NULL, NULL);
```

```
*****  
** TT2 YAN SANAYİ YEDEK PARÇA TANIMLAMA HAREKETİ **  
*****
```

```
INSERT INTO YP_STOK VALUES  
(7800100005, 'CASTROLL MOTOR YAĞI', 'YAGLAR', '9000',  
7800, '01', '001', '002', NULL, NULL, NULL, 100, NULL, NULL,  
6800, NULL, NULL);
```

```
*****  
** TT3 OTOTİP TANIMLAMA HAREKETİ **  
*****
```

```
INSERT INTO OTOTİP VALUES  
(L423', 'RENAULT 9 BROADWAY', 'R9', 196000,  
142000, 164000, 155000);
```

```
*****  
** TS1 MÜŞTERİ TANIMLAMA HAREKETİ **  
*****
```

```
INSERT INTO CARMAS VALUES  
(10154, 'FMV ÖZEL IŞIK LİSESİ', 'TEŞVİKİYE-İSTANBUL',  
'246 60 47', 'BEYOĞLU VD.', '0986020', 0, 0, 0, 0, 10.00, 5.00);
```

```
*****  
** TS2 ARAÇ BİLGİLERİ TANIMLAMA HAREKETİ **  
*****
```

```
INSERT INTO PLAKA VALUES  
(34 AGK 47', 'L423', 8292929, 2848446, '010194', 1993,  
10154, 1500);
```

** TS3 ONARIM EMRİ AÇMA HAREKETİ **

SELECT * FROM CARMAS WHERE CARNO = 10154;

SELECT * FROM PLAKA VHERE PLKCAR = 10154;

INSERT INTO ONARIM VALUES

(35196, 105, CURRENT DATE, '34 AGK 07', 10154, NULL,
NULL, NULL, NULL, NULL, NULL);

INSERT INTO ONRHRK VALUES

(35196, '34 AGK 07', 'S', 0, 0, CURRENT DATE, NULL, NULL,
NULL, NULL, NULL, 'ILK BAKIM YAPILACAK');

** TS4 TRANSFER FİŞİ KESME HAREKETİ **

SELECT * FROM ONARIM WHERE ONNO = 35196;

SELECT * FROM YP_STOK WHERE YP_PARN0 '6000046003';

INSERT INTO ONRHRK VALUES

(35196, '34 AGK 07', 'P', 25816, 1, CURRENT DATE,
51000, 42500, '6000046003', 1, 0, 'HAVA FİLTRE ELEMANI R9');

INSERT INTO STKHRK VALUES

('6000046003', CURRENT DATE, 23, 10154, 35196, 25816, 1,
1, 42500, 51000);

UPDATE YP_STOK

SET YP_CIKMIK = YP_CIKMIK + 1,
YP_CIKMLY = YP_CIKMLY + 1
WHERE YP_PARN0 = '6000046003';

SELECT * FROM YP_STOK WHERE YP_PARN0 '7800100005';

INSERT INTO ONRHRK VALUES

(35196, '34 AGK 07', 'P', 25816, 2, CURRENT DATE,
7800, 6500, '7800100005', 1, 0, 'CASTROLL MOTOR YAGI');

INSERT INTO STKHRK VALUES

('7800100005', CURRENT DATE, 23, 10154, 35196, 25816, 2,
1, 6500, 7800);

UPDATE YP_STOK

SET YP_CIKMIK = YP_CIKMIK + 1,
YP_CIKMLY = YP_CIKMLY + 1
WHERE YP_PARN0 = '7800100005';

```
*****
** TS5 İŞÇİLİK GİRME HAREKETİ **
*****
```

```
SELECT * FROM ONARIM WHERE ONNO = 35196;
```

```
SELECT * FROM PLKA WHERE PLKNO = '34 AGK 07';
```

```
SELECT * FROM OTO TIP WHERE OTOTIP = 'L423';
```

```
SELECT FROM TAMIRK
      WHERE IOTOKATTIP = 'R9' AND ISCKOD = '0061'
```

```
INSERT INTO ONRHRK VALUES
      (35196, '34 AGK 07', 'T', 0, 0, CURRENT DATE, 500000,
       500000, '0061', 0, 1.8, 'DIAGNOSTIC BAKIM');
```

```
SELECT FROM TAMIRK
      WHERE IOTOKATTIP = 'R9' AND ISCKOD = '0050'
```

```
INSERT INTO ONRHRK VALUES
      (35196, '34 AGK 07', 'T', 0, 0, CURRENT DATE, 50000,
       50000, '0050', 0, 1.8, 'HAVA FLT. DEĞİŞ.');
```

```
SELECT FROM TAMIRK
      WHERE IOTOKATTIP = 'R9' AND ISCKOD = '0071'
```

```
INSERT INTO ONRHRK VALUES
      (35196, '34 AGK 07', 'T', 0, 0, CURRENT DATE, 20000,
       20000, '0071', 0, 1.8, 'MOT.YAGI DEĞİŞ.');
```

```
*****
** TS6 ONARIM FATURASI KESME HAREKETİ **
*****
```

```
SELECT * FROM STKHRK WHERE HRKFAT = 35196;
```

```
SELECT * FROM ONRHRK WHERE ONRNO = 35196 AND ONRTIP = 'T';
```

```
SELECT FROM CARMAS WHERE CARNO = 10154;
```

```
INSERT INTO FATURA VALUES
      (10154, 23, 35196, CURRENT DATE, 'K', 798208, 96288, 5880,
       6200000, 'FMV ÖZEL IŞIK LİSESİ');
```

```
UPDATE CARMAS
      SET CARBORTUT = CARBORTUT + 738208;
      WHERE CARNO = 10154;
```

```
INSERT INTO CARHRK VALUES
(10154, 'B', 35196, CURRENT DATE, 738208, 'OTO ONARIM');
```

```
UPDATE ONARIM SET
ONFATTRH = CURRENT DATE,
PARSAT = 4,
TOPSAT = 7,
ONFATTUT = 738208,
ONFATIND = 102160,
ONPARTUT = 58800,
ONISCTUT = 620000
WHERE ONNO = 35196;
```

```
*****
** TS7 YEDEK PARÇA FATURASI KESME HAREKETİ **
*****
SELECT * FROM CARMAS WHERE CARNO = 00098;
```

```
INSERT INTO STKHRK VALUES
('6000046004', CURRENT DATE, 24, 00098, 0, 15960, 0,
1, 10000, 12000);
```

```
UPDATE YP_STOK
SET YP_CIKMIK = YP_CIKMIK + 1,
    YP_CIKMLY = YP_CIKMLY + 10000
WHERE YP_PARN0 = '6000046004';
```

```
INSERT INTO STKHRK VALUES
('6000046005', CURRENT DATE, 24, 00098, 0, 15960, 0,
1, 20000, 24000);
```

```
UPDATE YP_STOK
SET YP_CIKMIK = YP_CIKMIK + 1,
    YP_CIKMLY = YP_CIKMLY + 20000
WHERE YP_PARN0 = '6000046005';
```

```
INSERT INTO STKHRK VALUES
('6000046006', CURRENT DATE, 24, 00098, 0, 15960, 0,
1, 30000, 36000);
```

```
UPDATE YP_STOK
SET YP_CIKMIK = YP_CIKMIK + 1,
    YP_CIKMLY = YP_CIKMLY + 30000
WHERE YP_PARN0 = '6000046006';
```

INSERT INTO FATURA VALUES

(00098, 24, 15960, CURRENT DATE, 'K', 120000, 0, 20000,
100000, 0, 'AGK REKLAMCILIK AŞ.');

UPDATE CARMAS

SET CARBORTUT = CARBORTUT + 120000;
WHERE CARNO = 00098;

INSERT INTO CARHRK VALUES

(00098, 'B', 15960, CURRENT DATE, 120000, 'YEDEK PARÇA');

** TF1 YEDEK PARCA FİYAT DEĞİŞİKLİĞİ HAREKETİ **

UPDATE YP_STOK

SET YP_SATFYT = YPK_SATFYT
WHERE YP_PARN0 IN
(SELECT YPK_PARN0 FROM YPK_STOK);

** TF2 TAMİR KATALOĞU FİYAT DEĞİŞİKLİĞİ **

UPDATE TAMIRK

SET ISCFYT = ISSUCB
WHERE IOTKTİP IN
(SELECT OTOKATTİP FROM OTOTİP);

** TR1 GÜNLÜK İCMAL HAREKETİ **

SELECT * FROM FATURA WHERE FATTRH = CURRENT DATE;

** TS5 AYLIK STOK HAREKET RAPORU **

SELECT * FROM STKHRK;

Y ENİ VERİ TABANI TASARIMI

** YTS2 ARAÇ BİLGİLERİ TANIMLAMA HAREKETİ **

SELECT * FROM OTOTIP WHERE OTOTIP = 'L423';

INSERT INTO PLAKA VALUES

('34 AGK 47', 'L423', 8292929, 2848446, '010194', 1993,
10154, 1500);

** YTS3 ONARIM EMRİ AÇMA HAREKETİ **

SELECT * FROM CARMAS WHERE CARNO = 10154;

SELECT * FROM YPLAKA VHERE PLKCAR = 10154;

INSERT INTO ONARIM VALUES

(35196, 105, CURRENT DATE, '34 AGK 07', 10154, NULL, NULL);

INSERT INTO FATHRK VALUES

(35196, 23, 'S', 10154, '34 AGK 07', 0, 0, CURRENT DATE, 0,
NULL, NULL, NULL, 'ILK BAKIM YAPILACAK');

** YTS4 TRANSFER FİŞİ KESME HAREKETİ **

SELECT * FROM ONARIM WHERE ONNO = 35196;

SELECT * FROM YP_STOK WHERE YP_PARN0 '6000046003';

INSERT INTO FATHRK VALUES

(35196, 23, 'P', 10154, '34 AGK 07', 25816, 1, CURRENT DATE,
'6000046003', 1, 42500, 51000, 'HAVA FİLTRE ELEMANI R9');

SELECT * FROM YP_STOK WHERE YP_PARN0 '7800100005';

INSERT INTO FATHRK VALUES

(35196, 23, 'P', 10154, '34 AGK 07', 25816, 2, CURRENT DATE,
'7800100005', 1, 42500, 51000, 'CASTROLL MOTOR YAĞI');

** YTS5 İŞÇİLİK GİRME HAREKETİ **

SELECT * FROM ONARIM WHERE ONNO = 35196;

```
SELECT * FROM PLAKA WHERE PLKNO = '34 AGK 07';
```

```
SELECT FROM TAMIRK
      WHERE IOTOKATTIP = 'R9' AND ISCKOD = '0061'
```

```
INSERT INTO FATHRK VALUES
      (35196, 23, 'I', 10154, '34 AGK 07', 0, 0, CURRENT DATE,
       '0061', 1.8, 500000, 500000, 'DIAGNOSTIC BAKIM');
```

```
SELECT FROM TAMIRK
      WHERE IOTOKATTIP = 'R9' AND ISCKOD = '0050';
```

```
INSERT INTO FATHRK VALUES
      (35196, 23, 'I', 10154, '34 AGK 07', 0, 0, CURRENT DATE,
       '0050', 1.8, 50000, 50000, 'HAVA FLT.DEGİŞİM');
```

```
SELECT FROM TAMIRK
      WHERE IOTOKATTIP = 'R9' AND ISCKOD = '0071';
```

```
INSERT INTO FATHRK VALUES
      (35196, 23, 'I', 10154, '34 AGK 07', 0, 0, CURRENT DATE,
       '0071', 1.8, 20000, 20000, 'MOTOR YAG DEĞİŞ.');
```

```
*****
** YTS6 ONARIM FATURASI KESME HAREKETİ **
*****
```

```
SELECT * FROM FATHRK
      WHERE HRKNUM = 35196 AND HRKTIP IN (I, P) ;
```

```
SELECT FROM CARMAS WHERE CARNO = 10154;
```

```
INSERT INTO FATURA VALUES
      (10154, 23, 35196, CURRENT DATE, 'K', 798208, 96288, 5880,
       6200000, 'FMV ÖZEL IŞIK LİSESİ');
```

```
INSERT INTO CARHRK VALUES
      (10154, 'B', 35196, CURRENT DATE, 738208, 'OTO ONARIM');
```

```
*****
** YTS7 YEDEK PARÇA FATURASI KESME HAREKETİ **
*****
```

```
SELECT * FROM CARMAS WHERE CARNO = 00098;
```

```
INSERT INTO FATHRK VALUES
      (35196, 23, 'P', 10154, '34 AGK 07', 0, 0, CURRENT DATE,
       '6000046003', 1, 10000, 12000, 'HAVA FİLTRE ELEMANI R11');
```

```
INSERT INTO FATHRK VALUES
      (35196, 23, 'P', 10154, '34 AGK 07', 0, 0, CURRENT DATE,
       '6000046003', 1, 10000, 12000, 'HAVA FİLTRE ELEMANI R21');
```

INSERT INTO FATHRK VALUES

(35196, 23, 'P', 10154, '34 AGK 07', 0, 0, CURRENT DATE,
'6000046003', 1, 10000, 12000, 'HAVA FİLTRE ELEMANI R25');

INSERT INTO FATURA VALUES

(00098, 24, 15960, CURRENT DATE, 'K', 120000, 0, 20000,
100000, 0, 'AGK REKLAMCILIK AŞ.');

INSERT INTO CARHRK VALUES

(00098, 'B', 15960, CURRENT DATE, 120000, 'YEDEK PARÇA');

** YTR2 AYLIK STOK HAREKET RAPORU **

SELECT * FROM FATHRK WHERE HRKTIP = 'P';

** YTK1 AYLIK STOK KAPANIŞI **

UPDATE YP_STOK

SET YP_CIKMIK = YP_CIKMIK + HRKMIK

WHERE YP_PARNNO IN

(SELECT HRKPARISC FROM FATHRK

WHERE HRKTIP = 'P');

** YTK2 AYLIK CARİ KAPANIŞI **

UPDATE CARMAS

SET CARBORTUT = CARBORTUT + CRXTUT

WHERE CARNO IN

(SELECT CRXNO FROM CARHRK

WHERE CRXTIP = 'B');

Member : GP40221346 Mode L/Serial : 84/44-A6035 Main storage : 24.0 M Started : 01/22/94 15:47:57
 Library : GPFRRDATA System name : M402 Version/Release : 2/2.0 Stopped : 01/22/94 15:22:37

Job Name	User Name	Job Number	T P			Start	Stop	Elapsed Seconds	CPU		Disk		Sync		Async		Synchronous		Backup	
			Y	P	Y				P	Seconds	Util	I/O	Disk	I/O	Disk	BCPU	DIO	ELP	Act	Ded
QDIRSHDCTL	QDDC	111559	02	8	50	15.47.56	16.22.37	2080.194	594	1.4	3							1947	42054.554	
ECS	QSYS	111570	02	8	20	15.47.56	16.22.37	2080.196	937	23.6	3							3290	32064.365	
QSYSECO	QPMR	111572	02	8	10	15.47.56	16.22.37	2080.196		72.7									2080.176	
M400	QUSER	114641	02	8E	50	15.47.56	16.22.37	2080.199		68.7									2080.179	
M400	QUSER	114727	02	8E	50	15.47.56	16.22.37	2080.202		47.2									2080.202	
M400	QUSER	114730	02	8E	50	15.47.56	16.22.37	2080.202		64.7									2080.202	
QROUTER	QSNADS	114745	02	8	40	15.47.56	16.22.37	2080.203	185		58	4						0032	281290.721	
M400	QSNADS	114746	02	8	40	15.47.56	16.22.37	2080.208	562		87	6						0065	251293.521	
DIA	QSNADS	114747	02	8	40	15.47.56	16.22.37	2080.208											2080.208	
QDIALICAL	QSNADS	114748	02	8	20	15.47.56	16.22.37	2080.209											2080.209	
QDIAINDIUSR	QSNADS	114749	02	8	50	15.47.56	16.22.37	2080.208											2080.208	
QDIANSIPRT	QSNADS	114750	02	8	50	15.47.56	16.22.37	2080.209											2080.209	
QNETF	QSNADS	114751	02	8	40	15.47.56	16.22.37	2080.211											2080.211	
QPFMRN	QPMR	114756	02	8	00	15.47.56	16.22.37	2080.212	29.381	1.4	2039	940						0144	211983.457	
IF1	MEIPARE	114757	02	8	50	15.49.08	15.50.31	13.014	19.523	23.6	815	69						0241	10 18	141
IF2	MEIPARE	114758	02	8	50	15.50.32	15.50.54	22.632	16.498	72.7	106	67						1556	5 5	009
IK1	MEIPARE	114759	02	8	50	15.50.55	15.51.14	18.411	12.680	68.7	118	86						1075	6 6	7
IK2	MEIPARE	114760	02	8	50	15.51.14	15.51.27	15.214	9.457	68.2	91	58						1039	6 6	7
IR1	MEIPARE	114761	02	8	50	15.51.30	15.51.59	29.149	13.958	47.7	383	68						0364	13 13	14
K2	MEIPARE	114762	02	8	50	15.52.00	15.53.11	70.983	51.293	72.3	230	119						2052	4 4	4
S1	MEIPARE	114763	02	8	50	15.53.12	15.53.25	13.213	8.576	64.7	37	62						2318	3 3	4
S2	MEIPARE	114764	02	8	50	15.53.26	15.53.39	13.101	8.662	66.1	70	63						1237	5 5	6
S31	MEIPARE	114765	02	8	50	15.53.39	15.53.57	17.961	12.751	72.1	96	68						1349	5 5	6
S32	MEIPARE	114766	02	8	50	15.53.58	15.54.14	16.536	12.234	74.0	73	65						1676	4 4	5
S33	MEIPARE	114767	02	8	50	15.54.15	15.54.28	13.130	8.396	63.8	75	65						1119	6 6	7
S34	MEIPARE	114768	02	8	50	15.54.28	15.54.42	13.583	8.586	63.0	64	65						1337	5 5	6
TS41	MEIPARE	114769	02	8	50	15.54.42	15.54.00	17.336	12.557	72.3	90	67						1395	5 5	6
TS42	MEIPARE	114770	02	8	50	15.55.00	15.55.20	20.132	13.643	67.8	194	71						0703	10 10	10
TS43	MEIPARE	114771	02	8	50	15.55.21	15.55.32	10.931	9.018	82.5	29	66						3110	3 3	3
TS44	MEIPARE	114772	02	8	50	15.55.32	15.55.43	10.967	8.551	78.7	37	65						2311	3 3	4
TS45	MEIPARE	114773	02	8	50	15.55.44	15.55.54	10.812	8.633	79.0	49	58						1762	5 5	5
TS46	MEIPARE	114774	02	8	50	15.55.55	15.56.13	18.170	13.405	73.8	118	69						1136	6 6	7
TS47	MEIPARE	114775	02	8	50	15.56.14	15.56.24	10.591	8.908	84.1	24	65						3712	2 2	2
TS48	MEIPARE	114776	02	8	50	15.56.25	15.56.35	10.125	8.392	82.3	27	66						3108	3 3	3
TS49	MEIPARE	114777	02	8	50	15.56.35	15.56.45	9.867	8.334	84.5	24	57						3473	2 2	3
TS51	MEIPARE	114778	02	8	50	15.56.45	15.57.00	15.036	12.246	81.4	68	66						1801	5 5	5
TS52	MEIPARE	114779	02	8	50	15.57.01	15.57.15	14.301	11.726	82.0	69	66						1692	5 5	5
TS53	MEIPARE	114780	02	8	50	15.57.15	15.57.32	16.434	11.759	72.8	81	65						1476	5 5	6
TS54	MEIPARE	114781	02	8	50	15.57.32	15.57.52	19.221	11.034	55.4	73	64						1512	4 4	5
TS55	MEIPARE	114782	02	8	50	15.57.53	15.58.03	10.840	8.416	77.6	50	65						2805	3 3	3
TS56	MEIPARE	114783	02	8	50	15.58.04	15.58.17	12.777	10.886	85.1	38	63						2865	3 3	3
TS57	MEIPARE	114784	02	8	50	15.58.17	15.58.28	10.583	8.449	79.8	26	65						3250	2 2	3
TS58	MEIPARE	114785	02	8	50	15.58.28	15.58.43	14.778	10.882	73.6	47	64						2315	3 3	4
TS59	MEIPARE	114786	02	8	50	15.58.43	15.59.01	17.466	9.063	51.9	26	64						3486	1 1	2

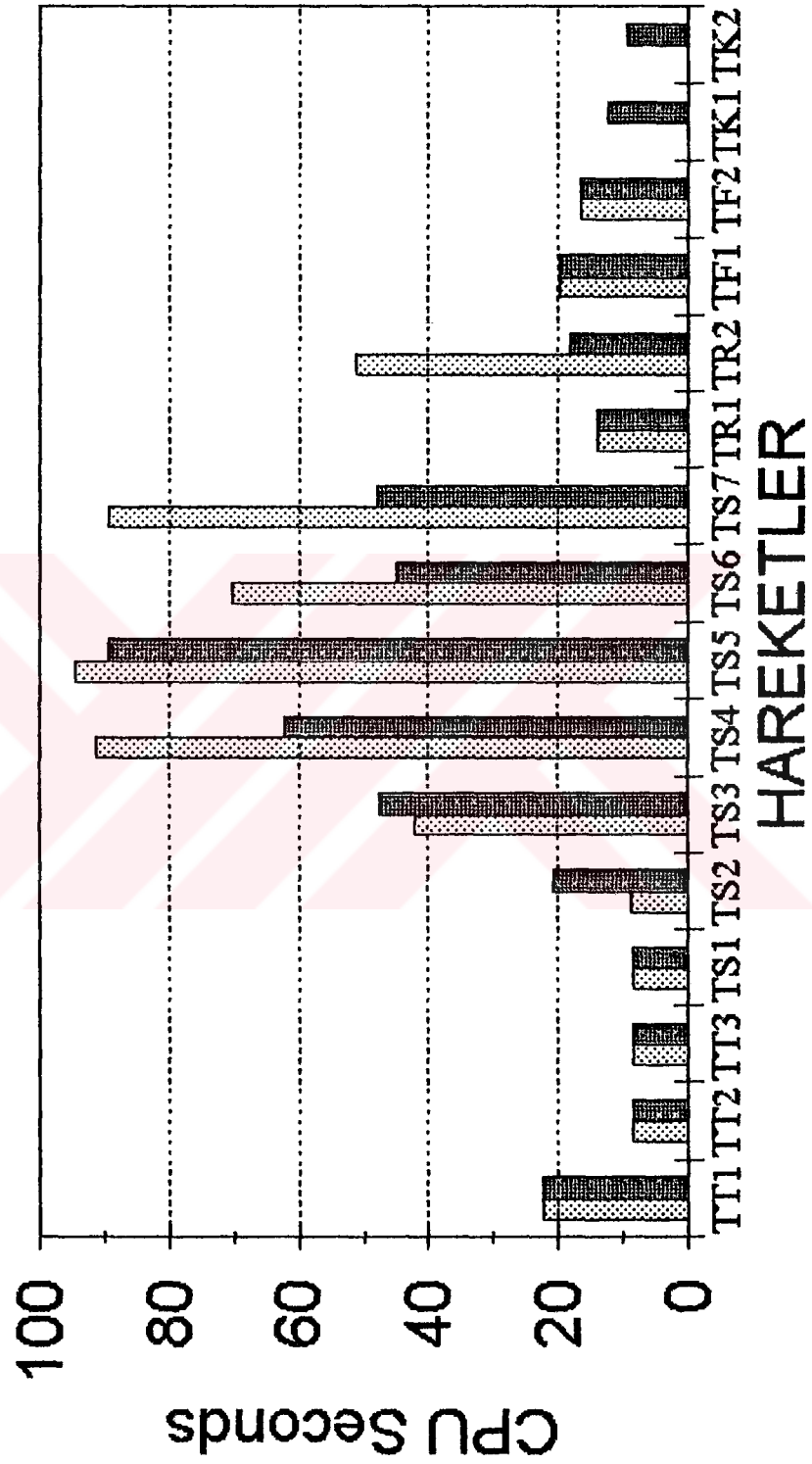
Member : : Q240221546 Model/Serial : 345/44-A4035 Main storage : 24.0 M Started : 01/22/94 15:47:57
 Library : : QPFRDATA System name : : H402 Version/Release : 2/ 2.0 Stopped : 01/22/94 15:22:37

Job Name	User Name	Job Number	P l p y	t y	Start	Stop	Elapsed Seconds	CPU Seconds	CPU Util	Sync Disk I/O	Asynchronous			Excp Wait Sec
											Disk I/O	BCPU /DIO	Elp Act Ded	
TS61	MEMPAR	114737	02	8	50	15.59.01	15.59.21	12.570	42.2	73	67	1722	4 4 5	.008
TS62	MEMPAR	114738	02	8	50	15.59.22	15.59.43	13.158	60.2	75	67	1756	3 3 5	.139
TS63	MEMPAR	114739	02	8	50	15.59.44	16.00.03	12.144	62.7	101	67	1202	5 5 7	.005
TS64	MEMPAR	114740	02	8	50	16.00.04	16.00.15	8.347	76.9	24	63	3478	2 2 3	.007
TS65	MEMPAR	114741	02	8	50	16.00.16	16.00.26	8.314	74.2	25	58	3326	2 2 3	.007
TS66	MEMPAR	114742	02	8	50	16.00.26	16.00.38	8.199	73.8	31	62	2645	3 3 3	.006
TS67	MEMPAR	114743	02	8	50	16.00.38	16.00.49	8.633	80.8	21	58	4111	2 2 2	.008
TS71	MEMPAR	114744	02	8	50	16.00.49	16.01.08	12.045	63.1	98	68	1227	5 5 6	.006
TS710	MEMPAR	114745	02	8	50	16.01.08	16.01.20	8.166	72.2	26	63	3141	2 2 3	.007
TS72	MEMPAR	114746	02	8	50	16.01.20	16.01.33	8.273	66.7	28	65	2953	2 2 3	.320
TS73	MEMPAR	114747	02	8	50	16.01.33	16.01.44	8.363	78.6	24	57	3485	2 2 3	.078
TS74	MEMPAR	114748	02	8	50	16.01.44	16.01.57	8.273	65.9	25	65	3317	2 2 3	.008
TS75	MEMPAR	114749	02	8	50	16.01.57	16.02.11	8.437	62.2	24	58	3515	2 2 3	.006
TS76	MEMPAR	114800	02	8	50	16.02.11	16.02.25	7.117	65.5	22	64	4144	2 2 2	.030
TS78	MEMPAR	114801	02	8	50	16.02.25	16.02.43	8.838	38.2	29	58	3048	1 1 3	.009
TS79	MEMPAR	114802	02	8	50	16.02.49	16.03.03	8.550	52.5	34	63	2518	2 2 4	.044
TS79	MEMPAR	114803	02	8	50	16.03.06	16.03.20	7.100	64.1	31	57	2935	2 2 3	.034
TT11	MEMPAR	114804	02	8	50	16.03.20	16.03.39	14.034	74.4	69	73	2032	4 4 4	.006
TT12	MEMPAR	114805	02	8	50	16.03.39	16.03.53	13.903	60.9	28	63	3023	2 2 3	.018
TT2	MEMPAR	114806	02	8	50	16.03.54	16.04.04	10.632	77.5	25	62	3386	2 2 3	.006
TT3	MEMPAR	114807	02	8	50	16.04.05	16.04.16	11.549	71.2	31	65	2657	3 3 3	.007
NETFLRCV	MAIS	114811	02	8	50	16.04.17	16.04.42	25.916	8.0	270	19	.0077	10 23 25	14.066
H400	QJSEB	114813	02	BE	50	16.04.44	16.04.51	7.060	15.0	95	16	.0124	12 16 22	1.720
H400	QJSEB	114814	02	BE	50	16.04.52	16.04.55	.622	21.2	25	10	.0249	9 9 17	.227

Member . . . : 0940221359 Model/Serial . . . : 845/44-A6035 Main storage . . . : 24.0 M Started . . . : 01/22/94 14:00:57
Library . . . : QPFRDATA System name . . . : M402 Version/Release : 2/ 2.0 Stopped . . . : 01/22/94 14:15:52

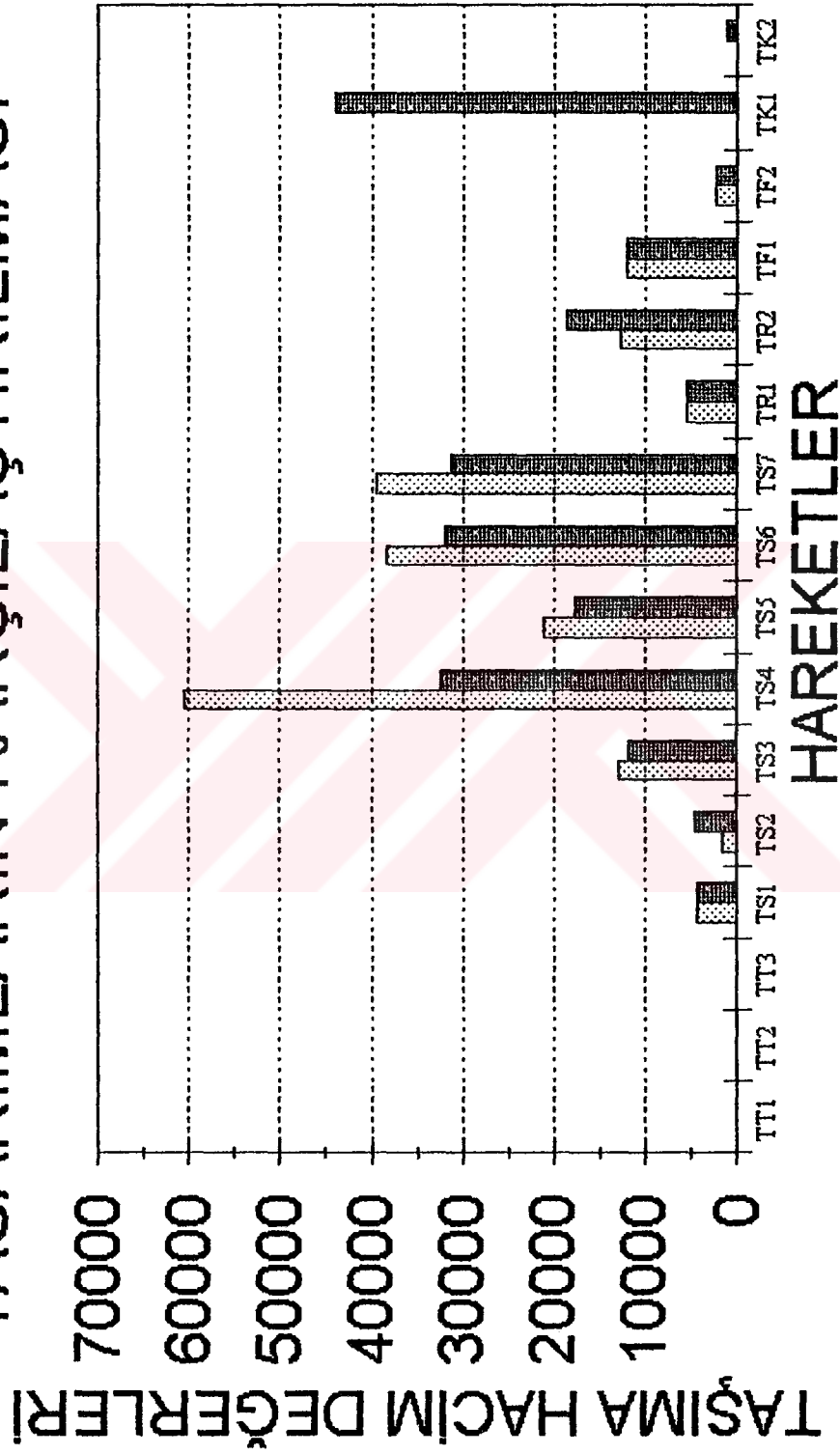
Job	User	Job	P	t	Start	Stop	Elapsed	CPU	CPU	Sync	Async	Synchronous		Excp
Name	Name	Number	P	t			Seconds	Seconds	Util	Disk	Disk	BCPU	Wait	
										I/O	I/O	/DIO	Sec	
QDIRSHDCTL	QDDC	111569	02	B	50	14.00.57	14.15.52	895.369	.240					
QEC8	QSYS	111570	02	B	20	14.00.57	14.15.52	895.369					871.684	
QSYSSCD	QPGMR	111572	02	B	10	14.00.57	14.15.52	895.370	.392				876.058	
M400	QUSER	114224	02	BE	50	14.00.57	14.15.52	895.375					895.375	
M400	QUSER	114225	02	BE	50	14.00.57	14.15.52	895.376					895.376	
M400	QUSER	114524	02	BE	50	14.00.57	14.15.52	895.378					895.378	
QPFRRMON	QPGMR	114558	02	B	00	14.00.57	14.15.52	895.380						
YTS21	QDDEN	114559	02	B	50	14.01.38	14.02.29	50.732	18.300	36.1	228	.0803	.032	
YTS22	QDDEN	114560	02	B	50	14.02.30	14.02.51	20.824	12.164	58.4	228	.0534	.056	
YTS31	QDDEN	114561	02	B	50	14.02.51	14.03.03	11.991	8.404	70.1	44	.1910	.009	
YTS32	QDDEN	114562	02	B	50	14.03.04	14.03.23	19.269	13.397	69.5	133	.1007	.009	
YTS33	QDDEN	114563	02	B	50	14.03.24	14.03.41	17.588	12.361	70.3	96	.1288	.010	
YTS34	QDDEN	114564	02	B	50	14.03.42	14.03.52	10.542	8.265	72.4	34	.2431	.009	
YTS41	QDDEN	114565	02	B	50	14.03.53	14.04.05	11.976	8.691	72.6	48	.1811	.011	
YTS42	QDDEN	114566	02	B	50	14.04.05	14.04.25	20.100	16.477	82.0	68	.2423	.011	
YTS43	QDDEN	114567	02	B	50	14.04.26	14.04.47	21.122	14.433	68.3	174	.0829	.009	
YTS44	QDDEN	114568	02	B	50	14.04.48	14.04.59	11.451	8.461	73.9	38	.2227	.009	
YTS45	QDDEN	114569	02	B	50	14.04.59	14.05.21	21.789	14.417	66.2	161	.0895	.047	
YTS51	QDDEN	114570	02	B	50	14.05.22	14.05.32	10.769	8.396	78.0	34	.2469	.009	
YTS52	QDDEN	114571	02	B	50	14.05.33	14.05.57	24.476	16.879	69.0	60	.2813	.008	
YTS53	QDDEN	114572	02	B	50	14.05.58	14.06.14	16.899	12.520	74.1	90	.1391	.008	
YTS54	QDDEN	114573	02	B	50	14.06.15	14.06.30	15.568	11.795	75.8	74	.1594	.058	
YTS55	QDDEN	114574	02	B	50	14.06.31	14.06.41	10.024	8.330	83.1	23	.3622	.006	
YTS56	QDDEN	114575	02	B	50	14.06.41	14.06.56	14.539	11.263	77.5	56	.2011	.006	
YTS57	QDDEN	114576	02	B	50	14.06.56	14.07.08	11.491	8.510	74.1	29	.2934	.009	
YTS58	QDDEN	114577	02	B	50	14.07.08	14.07.23	15.210	11.173	73.5	74	.1510	.010	
YTS61	QDDEN	114578	02	B	50	14.07.24	14.07.35	11.609	8.777	75.6	35	.2508	.009	
YTS62	QDDEN	114579	02	B	50	14.07.36	14.07.56	20.225	14.495	71.7	141	.1028	.008	
YTS63	QDDEN	114580	02	B	50	14.07.56	14.08.15	18.957	13.577	71.6	139	.0977	.008	
YTS64	QDDEN	114581	02	B	50	14.08.15	14.08.26	10.807	8.666	80.2	30	.2889	.007	
YTS71	QDDEN	114582	02	B	50	14.08.27	14.08.37	10.317	8.269	80.1	30	.2756	.006	
YTS72	QDDEN	114583	02	B	50	14.08.37	14.08.55	18.108	12.799	70.7	135	.0948	.006	
YTS73	QDDEN	114584	02	B	50	14.08.56	14.09.07	11.468	8.703	75.9	24	.3626	.006	
YTS74	QDDEN	114585	02	B	50	14.09.08	14.09.18	10.609	8.420	79.4	25	.3368	.008	
YTS75	QDDEN	114586	02	B	50	14.09.19	14.09.29	10.435	8.416	80.7	23	.3659	.006	
YTS76	QDDEN	114587	02	B	50	14.09.29	14.09.42	12.964	9.612	74.1	39	.2465	.011	
YTS77	QDDEN	114588	02	B	50	14.09.43	14.09.53	10.373	8.625	83.1	23	.3750	.009	
TF1	QDDEN	114589	02	B	50	14.11.04	14.12.03	59.564	17.219	28.9	182	.0946	.017	
TF2	QDDEN	114590	02	B	50	14.12.04	14.12.26	22.507	15.580	69.2	51	.3055	.023	
TK1	QDDEN	114591	02	B	50	14.12.27	14.12.43	16.005	12.262	76.6	86	.1426	.016	
TK2	QDDEN	114592	02	B	50	14.12.43	14.12.55	12.216	9.330	76.4	49	.1904	.008	
TR1	QDDEN	114593	02	B	50	14.13.56	14.13.15	19.096	13.381	70.1	132	.1014	.007	
TR2	QDDEN	114594	02	B	50	14.13.15	14.14.14	59.554	50.544	85.4	273	.1851	.006	
TS1	QDDEN	114595	02	B	50	14.14.15	14.14.27	12.108	9.219	76.1	19	.4852	.006	

TASARIMLARIN KARŞILAŞTIRILMASI



İlk Durum İkinci Durum

TASARIMLARIN KARŞILAŞTIRILMASI



ÖZGEÇMİŞ

Güler Çelik, 1969 yılında Malatya'da doğdu. Öğrenim hayatına 1975 yılında Ahmet Rasim İlkokulu'nda başladı ve daha sonra orta öğrenimini Bostancı Ortaokulu (1983), Fenerbahçe Lisesi'nde (1986) tamamladı. Aynı yıl İTÜ Fen Edebiyat Fakültesi Matematik Mühendisliği Bölümü'nde yüksek öğrenimine başladı. 1990 yılında bu fakülteden bölüm birincisi olarak mezun oldu. Akabinde İTÜ Fen Bilimleri Enstitüsü Mühendislik Bilimleri Anabilim Dalı Sistem Analizi Yüksek Lisans programına başladı. Çalışma hayatına 1991 yılında Elginkan Holding 'de (ECA), sistem analisti olarak atıldı. Halen RENAULT MAİS A.Ş.'de bilgisayar programcısı olarak çalışmaktadır.