

BÖLÜM 1

BİR VERİ TABANININ OLUŞTURULMASI

YAPISAL SORGULAMA DİLİ SQL

SQL Yapısal sorgulama dili (SQL - Structured Query Language) ilişkisel veri tabanı dilidir. SQL veri tabanında yeni tablo oluşturma, veri ekleme, silme, düzeltme, güncelleme, sorgulama ve koruma ve daha çok sayıda işlemin bir anda yapılmasını sağlar.

SQL dili IBM tarafından geliştirilen System R projesinde geliştirilen matematiksel bir söz dizilimine sahip bir ilk örnek Veri Tabanı yönetim sistemi SQUARE kadar uzanır. Daha sonra oluşturulan bu ilk veri tabanı yönetim sistemi daha kullanışlı olması için İngilizce dilinin söz dizilimine sahip bir hale getirilerek IBM şirketi 1979 yılında SEQUEL (Structured English Query Language) olarak adlandırılan yapısal sorgulama dilini oluşturdular. Daha sonra sonra bu dil geliştirilerek SQL adını almıştır.

SQL dili IBM tarafından geliştirildikten sonra büyük bir ilgi görmüştür. Bu ilgi SQL dilini, ilişkisel veri tabanı yönetim sistemlerinin hemen hemen tamamında kullanılır hale getirmiştir. Günümüzdeki kullanılan veri tabanı yönetim sistemlerinin neredeyse tamamında kullanılan bir sorgulama dili olması nedeniyle veri tabanları konusunda çalışmakta olan tüm teknik personelin mutlaka SQL dilini bilmeleri gerekmektedir.

SQL dili oluşturulduktan sonra SQL dilinin kullanımındaki farklılıkları ortadan kaldırmak ve bu konuda bir standart yakalamak için 1983 yılında ANSI ve ISO bir araya gelerek bir çalışma başlatmış ve 1987 yılında resmi olmayan ilk SQL standardı olan SQL1'i ortaya çıkarmıştır. Bu standardın yetersiz kalmasının ardından 1989 yılında SQL89 adında yine aynı kurumlar tarafından yeni bir standart geliştirilmiştir. Daha sonra 1992 yılında SQL2 diye bir standart çıkarılmıştır.

Yeni gelişmeler karşısında SQL diline bir çok yeni özellikler (özellikle nesneye yönelik olmak üzere) eklenmiş ve yeni uygulamaların ihtiyaçlarını karşılamak için yeni eklentiler yapılmıştır. Böylece resmi olmamakla beraber SQL3 ortaya çıkmış oldu.

SQL, isminin belirttiği gibi sadece bir veri tabanı sorgulamak ve onun verisini idare etmek için değil onu tanımlamak için de kullanılır. SQL aslında iki alandan meydana gelmiştir.

- Veri tabanı ve tabloların oluşturulması için komutlar içeren kısmı
- Sorgu komutları içeren kısmı

Yeni bir veri tabanı dosyası oluşturmak için

Create database <veritabanı adı>;

komutu kullanılır.

Mesela "buroseren" adlı bir veri tabanı dosyası oluşturmak için Create database "buroseren.mdb"; Şeklinde bir komut yeni veritabanı oluşturur

1.1 Giriş

SQL ile gerçekleştirilecek işlemlerde, üzerinde işlem yapılacak olan veri tabanı kütükleri ya da tablolar, bir veri tabanı içinde oluşturulur. Bu veri tabanını oluşturmak için,

CREATE DATABASE isim;

şeklindeki SQL komutunu kullanmak gerekir. Bu komut belirtilen isim'deki veri tabanını oluşturur. (Burada veri tabanı, içinde çok sayıda veri tabanı kütüğü ya da SQL terimleri ile tablo bulunan bir çevre bellek bölgesidir ya da bir alt dizin (subdirectory) adıdır.)

Veri tabanı içerisinde, bir işletmenin çeşitli faaliyetleri ile ilişkili aşağıdaki tür de bilgiler yüklemek istediğimizi varsayalım:

Personel Tablosu:

Sicil No	Sos. Güv. No	Ad	Soyad	Doğum Tarihi	Adres	Cinsiyet	Maaş	Böl-No	Yön. Sos. Güv. No
----------	--------------	----	-------	--------------	-------	----------	------	--------	-------------------

Bölüm Tablosu:

Bölüm Adı	Bölüm No	Yönetici Sosy. Güvenlik No
-----------	----------	----------------------------

Yer Tablosu:

Bölüm No	Bulunduğu Yer
----------	---------------

Proje Tablosu:

Proje Adı	Proje No	Yer	Bölüm No
-----------	----------	-----	----------

Çalışma Tablosu:

Personel Sos. Güv. No	Proje No	Saat
-----------------------	----------	------

Bağımlılık Tablosu:

Personel Sos. Güv. No	Bağl. Old. İsim	Cinsiyet	Doğum Tarihi	İlişki
-----------------------	-----------------	----------	--------------	--------

Parça Tablosu:

Parça No	Parça Adı	Proje No	Fiyat	Ağırlık
----------	-----------	----------	-------	---------

Satıcı Tablosu:

Satıcı No	Adı	Adres
-----------	-----	-------

Parça-Satıcı Tablosu:

Satıcı No	Parça No	Miktar
-----------	----------	--------

Bu tabloların her sütunu, tabloda saklanan verilerle ilişkili bir özelliği (attribute) belirtmektedir. Her tablo satırı, birbiri ile ilişkili verileri saklamaktır. Tablolar arası ilişkiler ise, müşterek özellikler (sütunlar) sayesinde gerçekleştirilecektir. Örneğin, “İstanbul’da yürütülen projelerde çalışan kişileri listeleyniz.” şeklindeki bir talep, Proje, Çalışma ve Personel tablolar arasında ilişki kurulmasını gerektirecek ve örneğin, Proje tablosunda, Yer alanında İstanbul bulunan projelerin Proje No’ları ile Çalışma tablosundaki Proje No’ları mukayese edilecek, aynı olanları için; Çalışma tablosundan alınacak Personel Sos.Güv. No’ları ile Personel tablosunda arama yapılacak, bulunan kişilere ait bilgiler Personel tablosundan listelenecektir. Bu tür işlemlerle ilişkili SQL komutları ilerdeki bölümlerde verilecektir.

1.2 Tabloların Yaratılması

SQL ile giriş bölümünde verilen tabloların yaratılması için, CREATE TABLE komutunu uygun şekli ile kullanmak gerekir. Aşağıdaki, bu tabloları yaratacak SQL komutları verilmiştir.

```
CREATE TABLE tabloadı;
```

komutu ile tablolar oluşturulur.Bu komutu kullanırken SQL’in uygun veri tipleri kullanılır.

ÖRNEK:

```
CREATE TABLE Personel
(sicil INTEGER NOT NULL,
sosy_g_no CHAR(8) NOT NULL
ad CHAR(10) NOT NULL,
soyad CHAR(10) NOT NULL,
dog_tar DATE, adres CHAR(50),
cinsiyet LOGICAL, maas NUMERIC(13,2),
bol_no SMALLINT, yon_s_g_n CHAR(8));
```

Yukarıdaki örnekte Giriş kısmında verilen Personel Tablosunun nasıl oluşturulduğu verilmiştir. Diğer tablolarda aynı şekilde gerekli SQL veri tipleri kullanılarak yaratılır.

NOT NULL ifadesi söz konusu alan ile ilişkili olarak mutlaka veri yüklenmesi gerektiğini, ilgili alanın boş bırakılmayacağını anlatmaktadır.NOT NULL ifadesi yoksa, o alan NULL anlamındadır.Yani o alan ilişkili olarak veri yüklenmemesi durumuna da müsaade edilmektedir.

1.3 SQL’de Veri Tipleri

Tabloların oluşturulması için kullanılan CREATE TABLE komutu içerisinde, tablonun her sütunu için kullanılan isimlerin her birinin farklı tiplerde tanımlandığını gözlediniz. Bu tanımları içinde CHAR, VARCHAR, INTEGER, SMALLINT, DATE, LOGICAL, NUMERIC(x,y) gibi ifadeler, SQL içinde tablodaki alanlara (sütun adı, attribute) yüklenebilecek veri tiplerini belirten sözcüklerdir. Tablo 1.1’de SQL’de mümkün veri tipleri ve özellikleri görülmektedir.

Tablo 1.1 SQL’de Veri Tipleri.

VERİ TİPİ	SQL KOMUTU	ÖZELLİĞİ
1)Sabit uzunluklu karakter	CHAR(Uzunluk) ÖRNEK: CHAR(15) gibi.	Sayısal işleme sokulmayacak veriler için kullanılır.Adres,isim,açıklama v.b. Uzunluk sabit olarak belirlenir. Maksimum uzunluk 254 karakterdir.
2)Değişken uzunluklu karakter	VARCHAR(Uzunluk) Buradaki uzunluk, maksimum uzunluktur.Veri daha kısa ise, uzunluğun gerektiği kadarı kullanılır. ÖRNEK: VARCHAR(23) gibi.	Karakter türündeki veriler gibidir. Tek fark, uzunluk değişkendir; yani verinin gerektirdiği uzunluk kullanılır.Bu veri tipi standart değildir.Her derleyicide bulunmaz.
3)Nümerik tam sayı	INTEGER	-2147483648 ile +2147483647 arasındaki tam sayılar için kullanılır.Ondalıklı nokta kullanılamaz.Bellekte 4 byte’lik yer kaplar.
4)Nümerik kısa tam sayı	SMALLINT	-32768 ile +32767 arasındaki tamsayılardır.Bu veri tipi standart değildir.Bazı derleyicilerde tanımlanmamıştır.Bu durumda veri tipi INTEGER’a çevrilir.
5)Ondalık sayı	DECIMAL(x,y) REAL(x,y) ya da NUMERIC(x,y) şeklinde tanımlanabilir.	x sayısı maksimum hane (digit) sayısı, y ondalık noktadan sonraki hane sayısıdır. x en fazla 20 olabilir. y 0-18 arasındadır. Örnek: -10.22 veya 1056.82 gibi sayılar.
6)Üstel sayı	FLOAT(x,y)	x sayısı toplam hane (digit) miktarıdır.Maksimum değer 20’dir. y ondalık noktadan sonraki hane sayısıdır. 0-18 arasındadır. 0.1 E-307 ile 0.9 E+308 arasındaki değerleri alabilir.Çok küçük ve çok büyük sayılar için uygun olan veri tipidir.
7)Tarih türü veriler	DATE	Tarih türü bilgiler üzerinde işlem yapma imkanı sağlar.SQL için standart bir veri tipi değildir.Bir gerçekleştirimden diğerine farklı şekilde kullanımı mümkündür.
8)Mantıksal veri	LOGICAL	Doğru (True, .T.) ya da yanlış (False, .F.) şeklinde değer alabilen veriler için kullanılır.
9)Zaman türü veriler	TIME	ss:dd:snsn (saat, dakika, saniye) şeklinde veriler için kullanılır. Standart değildir.
10)Tarih ve zaman türü veriler	TIMESTAMP	Tarih ve zaman türü verilerin bir karışık şeklinde kullanılan veriler içindir.Standart değildir.
11)Grafik türü veriler	GRAPHIC(n)	n adet 16 bitlik karakterden oluşan bir sabit uzunluklu bilgi tanımlamak için kullanılır. (Karakter türü veri, 8 bitlik karakterden oluşmaktadır.) Standart değildir.
12)Değişken uzunluklu grafik türü veri	VARGRAPHIC(n)	n adet 16 bitlik karakterden oluşan değişken uzunluklu (n maksimum uzunluktur.) bir bilgi tanımlamak için kullanılır. Standart değildir.

SQL’de kabul edilen, zaman ve tarih formatları Tablo 1.2 ve Tablo 1.3’te verilmiştir.

Tablo 1.2 SQL’de Tarih Formatları.

STANDART ADI	TARİH FORMATI	ÖRNEK
Uluslararası Standartlar Organizasyonu (ISO)	yyyy-aa-gg y –yıl a –ay g –gün	1992-11-29
IBM ABD Standardı USA	aa/gg/yyyy	11/29/1992
IBM Avrupa Standardı EUR	gg.aa.yyyy	29.11.1992
Japon Endüstri Standardı JIS	yyyy-aa-gg	1992-11-29

Tablo 1.3 SQL’de Zaman Formatları.

STANDART ADI	ZAMAN FORMATI	ÖRNEK
Uluslararası Standartlar Organizasyonu (ISO)	SS.dd.ss S –saat d -dakika s –saniye	16.25.07
IBM ABD Standardı USA	SS:dd AM veya PM	16:25 PM
IBM Avrupa Standardı EUR	SS.dd.ss	16.25.07
Japon Endüstri Standardı JIS	SS:dd:ss	16:25:07

1.4 Tablolara Veri Yüklenmesi

Bir tabloya veri girişi (veri yüklemesi) işlemi için;
INSERT INTO Tabloadı VALUES

komutu kullanılır.

ÖRNEK: INSERT INTO Kimlik
VALUES ('Ahmet','Yılmaz','25.12.1975','Trabzon');

1.5 Tablodaki Sütün İsimleri ve Tablo İsimleri İle İlişkili Kurallar

- İsim uzunlukları 18 karaktere kadar olabilir. (Bazı SQL uygulamalarında 8 karaktere kadar)
- İlk karakter bir harf olmalıdır. Onu izleyen karakterler, harf, rakam ya da alt çizgi sembolü (_) olabilir.

BÖLÜM 2 TEK TABLO İÇİNDE SORGULAMALAR

2.1 Giriş

SQL içinde, tek bir tablo içinde çeşitli kriterlere göre bilgi sorgulama, bilgiyi sıralı olarak elde etme, bilgi özetleme, ortalama vb. gibi matematiksel işlemleri gerçekleştirmeyi sağlayan komut ve fonksiyonlar vardır. Ayrıca, doğal olarak, aynı tipte işlemleri birden çok tabloyu birlikte ele alarak gerçekleştirmekte mümkündür.

Bu bölümde öncelikle tek tablo ile ilişkili sorgulamalar ve gerekli SQL komutları incelenecektir.

2.2 Temel SQL Sorgulamaları Select Komutu

Tek tablodan gerekli bilgileri elde etmek için sorgulama yapabilecek SQL komutu olan SELECT’in en basit şekli aşağıdaki gibidir.

```
SELECT *  
FROM Tabloadı;
```

Bu komut Tabloadı kısmında adı yazılı tablo içindeki bütün bilgileri koşulsuz olarak listeleyecektir. SELECT sözcüğünü izleyen kısımda * sembolünün bulunması, ilgili tablodaki bütün sütun (kolon) isimlerinin ve ilgili bilgilerin listelenmesini sağlayacaktır. Burda istersek * sembolü yerine bütün alanların adını veya işlem yapacağımız alan ya da alanların adını yazarız.

ÖRNEK: SELECT *
FROM Kimlik;

ÖRNEK: SELECT ad,soyad,dogumtar,dogumyer
FROM Kimlik;

Yukarıdaki iki örnekte Kimlik isimli tablodaki bütün alanları listeler. Eğer sadece ad alanının listelenmesini istersek;

```
SELECT ad  
FROM Kimlik;
```

yazarız.

2.3 Tekrarlı Satırların Ortadan Kaldırılması

SQL dilindeki tablo yapısı, formal olarak tanımlanan ilişkisel veri tabanı tablo yapısından farklıdır. SQL de tablo içinde, birbirinin aynı data içeren satırlara müsaade edilir.(İlişkisel veri modelinde ise müsaade edilmez.)

Birbirinin aynı olan satırların, listeleme esnasında, bir kez yazılması için, SELECT komutuna DISTINCT sözcüğü eklenir.

```
SELECT DISTINCT Sat_no
FROM Par_sat;
```

Bu komut ile Par_sat adlı tablodan Sat_no'lar tekrarsız olarak listelenecektir. Örneğin;

Par_sat

Sat_no	Par_no	Miktar
S1	P1	200
S1	P3	300
S2	P1	50
S2	P4	150
...
...

şeklinde bilgi içeriyorsa, komutun icrası sonucu

Sat_no

S1
S2

listesi elde edilecektir.

2.4 Tablo Bilgilerinin Sıralanmış Olarak Listelenmesi

Tablodan listelenecek bilgilerin, belirli bir sütun adına göre (ad'a göre v.b.) sıralanmış olarak görüntülenmesi için, SELECT komutuna ORDER BY sözcüğü ilave edilir.

ÖRNEK: Personel isimli bir tabloda sicil, ad, soyad, maas sütunları olsun. Maaşa göre artan sırada (küçükten büyüğe doğru) sıralı olarak listeleyiniz.

sicil	ad	soyad	maas
2746	Ali	Can	7800000
1728	Ayşe	Şener	5600000
1116	Mert	Okan	8950000
1022	Melih	Berkan	7500000
...
...

```
SELECT sicil,ad,soyad,maas
FROM Personel
ORDER BY maas ASC;
```

SONUÇ:

sicil	ad	soyad	maas
1728	Ayşe	Şener	5600000
1022	Melih	Berkan	7500000
2746	Ali	Can	7800000
1116	Mert	Okan	8950000
...
...

ASC sözcüğü ascending (artan) anlamındadır. Veriler azalan sırada (büyükten küçüğe ya da alfabetik olarak Z'den A'ya doğru) sıralamak için ASC yerine DESC (descending) sözcüğü kullanılmalıdır.

2.5 Birden Çok Alana Göre Sıralama

Bir tablo içinde verilerin aynı anda birden çok sütun (alana) göre sıralamakta mümkündür. Örneğin Personel tablosunu ad ve maas alanlarına göre sıralamak isteyelim.

```
SELECT sicil,ad,soyad,maas
FROM Personel
ORDER BY ad,;
```

Burada tablo öncelikle ad'a göre artan sırada (A'dan Z'ye doğru) sıralanacak, sadece aynı ad'a sahip olanlar kendi aralarında maas'a göre küçükten büyüğe (artan) sıralanacaktır.

sicil	ad	soyad	maas
1215	Ali	Can	2000000
3712	Ali	Okan	6000000
1152	Ali	Mert	8000000
3712	Birol	Akın	4000000
8145	Birol	Çelen	8500000
1248	Birol	Okur	11000000
...

Şekil 2.1 Ad ve Maaş'a Göre Sıralama.

Burada, çok sayıda alana göre sıralama, farklı sıralama kriterlerine göre gerçekleştirilebilir. Örneğin aşağıdaki SELECT komutu ile ad alanına göre artan, soyad alanına göre azalan, maas alanına göre artan sıralanmış tablo elde edilmektedir.

```
SELECT sicil,ad,soyad,maas
FROM Personel
ORDER BY ad ASC,soyad DESC,maas ASC;
```

veya aynı komut için alternatif yazılış:

```
SELECT sicil,ad,soyad,maas
FROM Personel
ORDER BY ad,soyad DESC,maas;
```

şeklinde olacaktır. Örnek çıktı aşağıdaki gibidir.

sicil	ad	soyad	maas
2742	Ahmet	Kaner	8000000
1712	Ahmet	Kaner	16000000
3112	Ahmet	Berk	17000000
2712	Melih	Caner	12000000
1317	Melih	Berat	18000000
2718	Zerhan	Şen	7000000

Şekil 2.2 Ad'a Göre Artan, Soyad'a Göre Azalan, Maaş'a Göre Artan Sıralama.

2.6 Koşula Bağlı Olarak Listeleme

SELECT komutu ile bir tablonun satırları içinde sadece verilen bir koşulu sağlayanlar da listelenebilir. Örneğin, maaşı 5000000 'dan fazla olan personel listelenmek istenirse, SELECT komutu aşağıdaki gibi yazılmalıdır:

```
SELECT *
FROM Personel
WHERE maas>5000000;
```

Burada WHERE sözcüğünü izleyen kısımda koşul belirtilmektedir. Koşul belirtilirken iki veri birbiri ile karşılaştırılmaktadır. SQL içinde, verileri çeşitli açılardan karşılaştırılmak için kullanılacak, karşılaştırma operatörleri Tablo 2.4'te verilmiştir.

Karşılaştırma ifadesinde karşılaştırılan verilerin türü aynı olmalıdır. Yani, bir karakter türü veri ile ancak karakter türünde başka bir veri, bir nümerik veri ile ancak nümerik olan başka bir veri karşılaştırılabilir.

Tablo 2.4 SQL'de Karşılaştırma Operatörleri.

OPERATÖR	ANLAMI
<	Küçük
>	Büyük
=	Eşit
<=	Küçük veya eşit
>=	Büyük veya eşit
<>	Eşit değil
!=	Eşit değil

Bazı SQL gerçekleştirmelerinde != (den küçük değil) ve != (den büyük değil) operatörleri de mevcuttur.

2.7 Çeşitli Veri Tipleri İçin Basit Sorgulamalar

Nümerik veri tipi

Nümerik (sayıda) veri tipi, SMALLINT, INTEGER, DECIMAL, NUMERIC ya da FLOAT tipi bildiri sözcüklerinden biri ile tanımlanan, matematiksel işlemlere sokulabilen, özel semboller içine alınmayan (" " sembolleri gibi) verileri kapsar. Maaş, ürün miktarı vb. tipteki bilgiler bu türde olmalıdır.

ÖRNEK: Maaşı 8000000'dan fazla olmayan personeli listelemek.

```
SELECT *
FROM Personel
WHERE maas<=8000000;
```

Karakter (char) veri tipi

Karakter türündeki veriler, çift tırnak (" ") veya tek tırnak (' ') sembolleri içine yazılır. Bu tip veriler, rakamdan oluşsa bile, matematiksel işlemler içinde kullanılamazlar. Örneğin "235"+2 işlemi geçersizdir.

ÖRNEK: Adı Ali olmayan personele ait kayıtları listelemek.

```
SELECT *
FROM Personel
WHERE ad<>"Ali";
```

Veya

```
SELECT *
FROM Personel
WHERE ad != "Ali";
```

Bu komutlar sonucu aşağıdaki kayıtlar listelenebilir;

sicil	ad	soyad	sos_g_no
275	Ahmet	Caner	272534
117	Melih	Akın	387265
287	ali	çokşen	277421
444	ali	cansu	1722433

Adı'ı Ali olanları listelemek istememize rağmen, ali cansu ve ali çokşen listelenmiştir. Bunun sebebi, bilindiği gibi, bilgisayar açısından Ali karakter verisi ile ali karakter verisinin farklı veriler olmasıdır. Eğer ali cansu, Ali cansu şeklinde yazılmış olsa, listelemeyecekti. Bu özellik karakter türü verilerle işlem yaparken göz önünde bulundurulması gereken önemli bir noktadır.

Tarih veri tipi

Tarih tipi veriler, {} sembolleri içinde yazılmalıdır.

ÖRNEK: Hangi personelin, doğum tarihi 1960 yılından daha öncedir?

```
SELECT *
FROM Personel
WHERE dog_tar <={1959};
```

Burada kullanılan SQL versiyonunda, tarih tipi verinin aa/gg/yy (ay/gün/yıl) formatında temsil edildiği varsayılmıştır.

SONUÇ:

sicil	ad	soyad	dog_tar
2712	Ali	Okan	05/02/58
3718	Hasan	Akın	04/03/59

Mantıksal (lojik) veri tipi

Mantıksal veriler için mümkün olabilen sadece iki değer sözkonusudur.dođru (true, .T.) ile simgelenir, yanlış (false, .F.) ile simgelenir.

Personel tablosunda, personelin cinsiyetini belirleyen bir alanımız olsun. Cinsiyet adlı bu alanda cinsiyeti erkek olanlar true (.T.), kadın olanlar false(.F.) ile kodladığımızı kabul edersek, işletmede çalışan personel içinden erkek olanları listelemek için aşağıdaki gibi bir SQL kodu yazmak gerekecektir.

```
SELECT *
FROM Personel
WHERE cinsiyet=.T.;
```

Bu komut aşağıdaki şekilde de kullanılabilir;

```
SELECT *
FROM Personel
WHERE cinsiyet;
```

Bu durumda cinsiyet alanı .T. olanlar (erkek olanlar) listelenir.

2.8 Birden Çok Koşula Dayalı Sorgulamalar (Not, And, Or)

NOT, OR ve AND mantıksal operatörleri yardımı ile birden çok koşulun gerçekleşmesine bağlı olarak ifade edilebilecek karmaşık ya da birleşik koşullu listelemeleri gerçekleştirmek mümkün olmaktadır.

ÖRNEK: Maaşı 5000000 TL'den fazla olan ve cinsiyeti erkek olan personelin listelenmesi gibi bir işlemde, söz konusu personel için iki koşul verilmekte ve ikisinin de gerçekleşmesi istenmektedir:

- 1.Koşul → Maaşın 5000000'dan fazla oluşu
- 2.Koşul → Cinsiyetin Erkek olması

Her iki koşulunda aynı anda gerçekleşmesi istendiđi için, VE (AND) sözcüğü ile birbirlerine bağlanmıştır. Bu işlemi gerçekleştiren SQL komutu aşağıdaki gibidir.

```
SELECT *
FROM Personel
WHERE maas>5000000 AND cinsiyet=.T.;
```

NOT, AND ve OR operatörlerinin etkileri aşağıdaki tablolarda (Tablo 2.5, 2.6, 2.7) verilmiştir.

Tablo 2.5 NOT Operatörü.

Koşul	NOT koşul
.T.	.F.
.F.	.T.

Tablo 2.6 AND Operatörü.

1. Koşul	2. Koşul	1. Koşul AND 2. Koşul
.T.	.T.	.T.
.T.	.F.	.F.
.F.	.T.	.F.
.F.	.F.	.F.

Tablo 2.7 OR Operatörü.

1. Koşul	2. Koşul	1. Koşul OR 2. Koşul
.T.	.T.	.T.
.T.	.F.	.T.
.F.	.T.	.T.
.F.	.F.	.F.

Aşağıdaki örnek sorularla, mantıksal işlem operatörlerinin kullanımı konusunda fikir verilmektedir.

SORU: Doğum tarihi 1960'tan önce olan, maaşı 6000000 – 10000000 arasında olan bayan personel kimlerdir.

```
SELECT *
FROM Personel
WHERE dog_tar<{01/01/60} AND
Maas>=6000000 AND maas<=10000000
AND cinsiyet=.F.;
```

SORU: Satış bölümü ile muhasebe bölümündekiler kimlerdir?

Satış bölümünün bölüm no'sunun (bol_no) 1 ve muhasebe bölümünün bol no 'sunun 2 olduğunu varsayalım.

```
SELECT *
FROM Personel
WHERE bol_no=1 OR bol_no=2;
```

şeklinde listeleme gerçekleştirilebilir.

SORU: Doğum tarihi 1960'tan önce olmayan, cinsiyeti erkek olan veya doğum tarihi 1965'ten önce olmayan ve cinsiyeti kadın olan personeli listeleyiniz.

```
SELECT *
FROM Personel
WHERE dog_tar>={01/01/60} AND
cinsiyet=.T. OR dog_tar>={01/01/65}
AND cinsiyet=.F.;
```

SORU: Bölümü satış veya muhasebe olan kadın personeli listeleyiniz.

```
SELECT *
FROM Personel
WHERE bol_no=1 OR bol_no=2
AND cinsiyet=.F.;
```

şeklindeki SELECT komutu doğru görünüyorsa da, dikkatle düşünüldüğünde istenileni vermeyeceği anlaşılabilir. AND ; OR 'a göre öncelikli olduğu için, değerlendirme:

Satış bölümünde çalışan (cinsiyeti ne olursa olsun)
veya
muhasebe bölümündeki kadın personel; şeklinde listeleme yapılacaktır.

Doğru cevap ()'ler ile öncelikleri değiştirerek aşağıdaki ifade ile elde edilebilir.

NOT: AND operatörü OR operatörüne göre daha önceliklidir.

```
SELECT *
FROM Personel
WHERE (bol_no=1 OR bol_no=2) AND cinsiyet=.F.;
```

SORU: Bölümü satış ya da muhasebe olmayan, 1960'tan sonra doğmuş, bayan personeli listelemek.

```
SELECT *
FROM Personel
WHERE NOT (bol_no=1 OR bol_no=2)
AND dog_tar>={01/01/60}
AND cinsiyet=.F.;
```

Eşdeğer cevap:

```
SELECT *
FROM Personel
WHERE bol_no<>1 AND bol_no<>2
AND dog_tar>={01/01/60} AND cinsiyet=.F.;
```

2.9 Bir Veri Kümesi İçinde Arama (In Operatörü)

Aşağıdaki örnek sorunun cevabını, şu ana kadar öğrendiğimiz SQL komutları ile gerçekleştirebiliriz:

SORU: bol_no'su 1,2 ya da 3 olan personeli listeleyiniz.

```
1-) SELECT *
FROM Personel
WHERE bol_no=1 OR bol_no=2 OR bol_no=3;
```

Fakat SQL'de bu işlemi gerçekleştirmenin daha kısa ve daha şık bir yolu vardır; IN sözcüğünü kullanarak bu işlemi yaparız.

```
2-) SELECT *
FROM Personel
WHERE bol_no IN(1,2,3);
```

Bu komut, OR ile düzenlenen 1. SELECT komut grubuna denktir. Fakat belirtildiği gibi daha kısa ve anlaşılır bir ifade oluşmaktadır.

IN operatörü NOT ile birlikte de kullanılabilir.Örneğin aşağıdaki soru ile ilişkili üç ayrı eşdeğer SELECT komutu verilmiştir.

SORU: Bölüm 1,2 ve 3 olmayan personel kimlerden oluşmaktadır.

CEVAP 1:

```
SELECT *
FROM Personel
WHERE NOT(bol_no=1) AND
NOT(bol_no=2) AND NOT(bol_no=3);
```

CEVAP 2:

```
SELECT *
FORM Personel
WHERE bol_no<>1 AND
bol_no<>2 AND bol_no<>3;
```

CEVAP 3:

```
SELECT *
FORM Personel
WHERE bol_no NOT IN(1,2,3);
```

2.10 Aralık Sorgulaması (Between Sözcüğü)

ÖRNEK: Maaşı 5000000 – 10000000 arasında olan personel kimlerdir.

```
SELECT *
FORM Personel
WHERE maas>=5000000 AND
Maas<=10000000;
```

şeklindeki bir SELECT komutu ile bu işlem gerçekleştirilebilir. Aynı soruya daha kısa ve daha etkin cevap verebilecek bir SQL komutu ise BETWEEN sözcüğü ile aşağıdaki gibi düzenlenebilir.

```
SELECT *
FORM Personel
WHERE maas BETWEEN 5000000
AND 10000000;
```

2.11 Karakter Türü Bilgi İçinde Arama Yapma (Like Sözcüğü)

Personel tablosu içinde adres adlı 50 karakter uzunluğunda bir alanımız olsun. Adres bilgisinin aşağıdaki şekilde verildiğini varsayalım:

Cumhuriyet Cad. 46/9 Taksim/İSTANBUL

Burada, adres içinde, semtinde belirtildiğini ve bunun ayrı bir sütun (alan) olmadığına dikkat çekelim. Şimdi belirli bir semtte ikamet eden personeli listelemek istersek, semt adını, adres alanı içinde aramak gerekecektir. Bu işlemi gerçekleştirmek için SQL'de LIKE sözcüğü kullanılır. Aşağıdaki SELECT komutunu inceleyelim:

```
SELECT *
FROM Personel
WHERE adres LIKE '%Taksim%';
```

Bu komut ile "Taksim" semtinde ikamet eden personel listelenmek istenmektedir. Bu komut gerçektede "Taksim" de oturan kişileri listeleyecektir. Ama bu arada

“Taksim Caddesi 22-7 Kadıköy - İST”

şeklindeki adresleri de listeleyecektir.

adres LIKE ‘%Taksim%’

ifadesi, adres içinde Taksim’i arayacaktır. Adres içinde, herhangi bir yerde, bulunduğu takdirde, bu satırı (bu kayıt, personeli) listeleyecektir.

% sembolü, Taksim sözcüğünün öncesi ve sonrasındaki karakterler ne olursa olsun anlamındadır.

LIKE sözcüğünü, alt çizgi (-) sembolü ile birlikte kullanmak da mümkündür.

ÖRNEK: SELECT *
FROM Personel
WHERE ad LIKE ‘Mehmet----’;

şeklindeki komut ile ad alanı “Mehmet” ile başlayan ve ad alanı uzunluğu 10 karakter olan isimlere sahip personeli listeleyecektir. ”Mehmet Ali”, “Mehmet Can”, “Mehmetcik” gibi isimler listelenecektir. Anlaşılacağı gibi - sembolü, tek karakterlik bir bilgiyi temsil etmektedir.

BÖLÜM 3 SQL’DE ARİTMETİKSEL İFADELER ve FONKSİYONLAR

3.1 Aritmetiksel İfadeler

SELECT komutu ile, veri tabanında mevcut tablolardan listeleme yaparken, tabloda ayrı bir sütun (alan) olarak yer almamış ve ancak bir hesaplama sonucunda üretilebilecek bilgileri de liste içine katmak mümkündür.

Aşağıdaki SELECT komutu ile, personelin şu anda geçerli olan maaşı ile, bu maaşın %32 zamlı şekli listelenmektedir.

```
SELECT ad,soyad,maas,maas*1.32  
FROM Personel;
```

Hesaplanmış alanları elde etmek için oluşturulacak aritmetiksel ifadelerde, Tablo 3.1’de belirtilen semboller kullanılabilir.

Tablo 3.1 SQL’de Aritmetiksel Semboller.

OPERATÖR	İŞLEVİ
** veya ^	Üs alma
*	Çarpma
/	Bölme
+	Toplama
-	Çıkarma

Öncelikli sırası, matematikte ve diğer bilgisayar dillerinde olduğu gibidir. Üs alma, hepsinden öncedir. Sonra çarpma (*) ve bölme (/) gelir. Toplama (+) ve çıkarma (-) en son önceliklidir.

Parantez kullanılarak öncelikler değiştirilebilir.

3.2 Kümeleme Fonksiyonları

SQL tablo içinden, çeşitli matematiksel işlemlerin sonucunu otomatik olarak üretmeyi sağlayan, fonksiyonlara sahiptir. Bu fonksiyonlar, örneklerle birlikte aşağıda verilmiştir:

3.2.1 SUM fonksiyonu

Fonksiyonun argümanı olarak belirtilen sütun ile ilişkili olarak toplama işlemini gerçekleştirir.

SORU: İşletmedeki personelin maaşlar toplamı ne kadardır?

ÇÖZÜM: SELECT SUM(maas)
FROM Personel;

SORU: Bilgi işlem bölümündekilerin maaşları toplamı ne kadardır?

ÇÖZÜM: SELECT SUM(maas)
FROM Personel
WHERE bol_no=5;

ifadesi ile sonuç elde edilebilir. Sonuç, sadece bilgi işlem bölümdekilerin maaşları toplamı şeklinde olacaktır.

SORU: Satış, muhasebe ve bilgi işlem bölümündeki personelin maaşları toplamı nedir?

ÇÖZÜM: Satış bölümü için bol_no 1, muhasebe için 2 ve bilgi işlem için bol_no 5 olarak alınırsa

```
SELECT SUM(maas)
FROM Personel
WHERE bol_no IN(1,2,5);
```

SORU: Maaşları 5000000 TL'nin altında olan bayan personelin maaşları toplamı nedir?

ÇÖZÜM: Bayan personeli, daha önceden cinsiyet alanına .F. yerleştirerek kodlamış isek

```
SELECT SUM(maas)
FROM Personel
WHERE cinsiyet=.F. AND
maas<5000000;
```

ifadesi istenilen çözümü verecektir.

3.2.2 AVG fonksiyonu

Aritmetiksel ortalama (average) hesaplamak için kullanılır.

```
SELECT AVG(maas)
FROM Personel;
```

Komutu, işletmedeki ortalama maaşı hesaplayarak görüntüleyecektir. Bu fonksiyon ile de, koşula bağlı olarak hesaplatma yaptırılabilir.

SORU: Bilgi işlem bölümündekilerin maaş ortalaması ne kadardır?

ÇÖZÜM: Bilgi işlem bölümünün bol_no'su 5 ise

```
SELECT AVG(maas)
FROM Personel
WHERE bol_no=5;
```

ifadesi istenilen çözümü verecektir.

3.2.3 MAX fonksiyonu

Tablo içinde, belirlenen sütun (alan) içindeki en büyük değeri bulur.

SORU: İşletme içindeki en yüksek maaş ne kadardır?

ÇÖZÜM: SELECT MAX(maas)
FROM Personel;

SORU: Bilgi işlem bölümündeki en yüksek maaş ne kadardır?

ÇÖZÜM: Bilgi işlem bölümünün 5 ile kodlandığı varsayımı ile

```
SELECT MAX(maas)
FROM Personel
WHERE bol_no=5;
```

SORU: Bayan personel içinde en yüksek maaş ne kadardır?

ÇÖZÜM: SELECT MAX(maas)
FROM Personel
WHERE cinsiyet=.F.;

3.2.4 MIN fonksiyonu

Tablo içinde, belirlenen sütun (alan) içindeki en küçük değeri bulur.

SORU: İşletme içinde 4 Mayıs 1970'den önce doğanlar için, asgari ücret nedir?

ÇÖZÜM: SELECT MIN(maas)
FROM Personel
WHERE dog_tar>{05/04/70};

3.2.5 COUNT fonksiyonu

Tablo içerisinde herhangi bir sayma işlemi gerçekleştirmek için kullanılır.

SORU: Personel tablosunda kaç satır vardır? (Bu, her satırda farklı bir personel bulunduğu düşünülürse, personel sayısı anlamına gelmektedir.)

ÇÖZÜM: SELECT COUNT(*)
FROM Personel;

SORU: Maaşı 6000000'dan fazla olan personel sayısı nedir?

ÇÖZÜM: SELECT COUNT(*)
FROM Personel
WHERE maas>6000000;

COUNT fonksiyonu, DISTINCT sözcüğü ile de kullanılabilir. Örneğin, personel tablosunda mevcut personelin, işletme içinde kaç tane farklı bölümde çalıştığı bulunmak istenirse aşağıdaki SELECT komutu kullanılabilir.

```
SELECT COUNT(DISTINCT bol_no)  
FROM Personel;
```

COUNT komutunda, * argümanının kullanılması, bütün sütunların (alanların) işleme sokulması, alan adının belirtilmesi ise (COUNT(bol_no) gibi), sadece, belirtilen sütunun işleme sokulmasını sağlar.

3.3 Gruplandırarak İşlem Yapma

SUM, AVG, MAX, MIN, COUNT fonksiyonları, tablodaki bilgileri, bazı özelliklere göre gruplandırarak bu gruplandırılmış veri üzerine de uygulamak mümkündür. Bu işlem, GROUP BY sözcükleri yardımı ile gerçekleştirilebilir. Örneğin, aşağıdaki soru bu konuda bir fikir verecektir:

SORU: Her bölümdeki ortalama maaş nedir?

Burada istene, bölümler bazında ortalama maaş olduğuna göre, personel tablosundaki satırlar, bölüm numaralarına göre (bol_no) gruplandırarak, her bir grubun maaş ortalaması ayrı ayrı hesaplanarak listelenebilir. Aşağıdaki SELECT komutu bu işlemi gerçekleştirmektedir.

```
SELECT bol_no,AVG(maas)  
FROM Personel  
GROUP BY bol_no;
```

SONUÇ:

bol_no	AVG(maas)
1	2500000
2	6800000
3	7400000
4	12500000

Her bölümdeki en yüksek maaşı alan kişiler listelenmek istenirse, aşağıdaki komut kullanılabilir:

```
SELECT bol_no,MAX(maas),ad,soyad  
FROM Personel  
GROUP BY bol_no;
```

Personel tablosundaki bilgiler

maas	ad	soyad	bol_no
5000000	Ali	Can	1
3000000	Ayşe	Okan	1
8000000	Akın	Oran	2
10000000	Yeşim	Şensoy	2

şeklinde ise, yukarıdaki SELECT komutunun çıktısı;

bol_no	Max_maas	ad	soyad
1	5000000	Ali	Can
2	10000000	Yeşim	Şensoy

şeklinde olacaktır.

Gruplandırarak, kümeleme fonksiyonlarını uygularken, koşul da verilebilir. Bu durumda, grup üzerindeki hesaplamalarla ilişkili koşul belirtirken, HAVING sözcüğü kullanmak gerekir. Aşağıdaki örnek soru, bu konuda fikir vermektedir.

SORU: En yüksek maaşın, 9000000'dan fazla olduğu bölümlerdeki personele ait ortalama maaşları listeleyiniz.

ÇÖZÜM: SELECT bol_no,AVG(maas)
FROM Personel
GROUP BY bol_no
HAVING AVG(maas)>9000000;

Personel tablosunda aşağıdaki veri mevcut olsun:

.....	bol_no	maas
	1		6000000
	1		17000000
	2		7500000
	2		8000000
	3		12000000
	3		11000000
	1		14000000
	1		18000000

Yukarıdaki SELECT komutu sonucunda

bol_no	AVG_maas
1	13750000
3	11500000

tablosu elde edilecektir.

HAVING sözcüğü, SELECT komutunda GROUP BY sözcükleri bulunmadığı zaman, geçersizdir. HAVING sözcüğünü izleyen ifade içinde, SUM, COUNT (*), AVG, MAX ya da MIN gibi kümeleme fonksiyonlarından en az biri bulunmalıdır.

WHERE sözcüğü bir tablonun tek tek satırları üzerinde işlem yapan koşullar için geçerli iken, HAVING sözcüğü sadece, gruplanmış veriler üzerindeki işlemlerde geçerlidir.

Bazı durumlarda, HAVING ve WHERE sözcükleri birlikte, SELECT komutu içinde kullanılabilir.

SORU: Personel tablosu içinde, her bölümde, erkek personele ait maaşlar için, ortalamanın 9000000'den fazla olduğu bölümleri listeleyiniz.

ÇÖZÜM: SELECT bol_no,AVG(maas)
FROM Personel
WHERE cinsiyet=.T.
GROUP BY bol_no
HAVING AVG(maas)>9000000;

Personel tablosunda aşağıdaki bilgiler olsun:

.....	bol_no	maas	cinsiyet
	1	6000000	.T.	
	1	17000000	.F.	
	2	7500000	.F.	
	2	8000000	.F.	
	3	12000000	.T.	
	3	11000000	.F.	
	1	14000000	.T.	
	1	18000000	.T.	

Yukarıdaki uygulanan SELECT komutu, her bölümdeki erkek personele ait ortalama maaşı hesaplayacak (erkek personel .T. ile belirtilmiştir) ve erkek personel maaş ortalaması, 9000000'den olan bölümler listelenecektir. Komutun çıktısı, aşağıdaki gibidir.

bol_no	AVG_maas
1	12666666.67
3	12000000

BÖLÜM 4 BİRDEN FAZLA TABLOYU İLİŞKİLENDİREREK SORGULAMAK

4.1 Giriş

Bu bölüme kadar, SQL ile, sadece tek tablo üzerinde sorgulamalar gerçekleştirilmiştir. Fakat işletme uygulamalarında ya da gerçek veri tabanı uygulamalarında, bu işin elemanter bir yönüdür. Daha sık karşılaşılan ve güç olan sorgulamalar, birden çok tablonun bir biri ile ilişkilendirilmesini gerektiren sorgulamalardır.

SQL'in sorgulama gücü daha ziyade bu tip sorgulamalarda ortaya çıkmaktadır.

4.2 Birleştirme (Join) İşlemi

Birleştirme işlemini anlayabilmek için, örnek veri tabanımızdaki, Personel ve Bölüm tablolarını göz önüne alalım.

Personel

Sicil No	Sos. Güv. No	Ad	Soyad	Doğum Tarihi	Adres	Cinsiyet	Maaş	Böl-No	Yön. Sos. Güv. No
----------	--------------	----	-------	--------------	-------	----------	------	--------	-------------------

Bölüm

bölüm_ad	bölüm_no	y_sos_g_n	y_is_bas_tar
----------	----------	-----------	--------------

Bu tablolar ile ilişkili olarak soruyu soralım:

Çalışan her personel ve bu personelin yöneticisi ile ilişkili bilgiler nelerdir?

Belirli bir personel ile ilişkili bilgiler, personel tablosunun o personele ait satırında mevcuttur. Ancak personelin yöneticisi ile ilişkili bilgilerin bir kısmına ise bölüm tablosunda erişilebilir. Bu durum zorunlu olarak personel ile bölüm tabloları arasında ilişki kurulmasını gerektirir. Bu ilişki, ancak müşterek bir alan yardımı ile kurulabilir.

Müşterek alan, burada bölüm numarasıdır ve personel tablosunda bol_no, bölüm tablosunda bölüm_no adı ile yer almaktadır.

Müşterek alana göre, personel ve bölüm tablolarının birleştirilmesi (JOIN) demek, her iki tablodaki tüm sütunları içeren yeni bir tablo oluşturmak demektir. Yalnız bu tabloda sadece, her iki tabloda da mevcut olan bölüm numaraları ile ilişkili satırlar yer alacaktır.

Birleştirme (JOIN) işlemi ile listeleme aşağıdaki SQL komutu ile gerçekleştirilebilir.

```
SELECT *  
FROM Personel,bölüm  
WHERE Personel.bol_no=bölüm.bölüm_no;
```

ilişkilendirme, kolayca görüleceği gibi

```
WHERE Tabloadı.Kolonadı=Tabloadı.Kolonadı;
```

ifadesi ile sağlanmaktadır. Aşağıdaki örnek veri için bu JOIN işlemi sonucu Tablo 4.1'de görülmektedir. Örnek veri:

Personel

sicil	sos_g_no	ad	soyad	dog_tar	adres	cinsiyet	maas	bol_no	yon_s_g_n
112	27641	Ali	Can	01/06/60	Fatih	.T.	8000000	1	037165
175	3777654	Ayşe	Şen	04/07/65	Kadıköy	.F.	7000000	1	037165
217	176241	Akın	Öncel	11/07/64	Üsküdar	.T.	6000000	2	277143
517	27615	Can	Öner	05/08/65	Fatih	.T.	4000000	2	277143
618	57253	Beril	Meral	08/07/62	Pendik	.F.	3750000	2	277143
1540	44721	Ayşe	Cansu	07/08/63	Beşiktaş	.F.	4800000	3	577211

Bölüm

bölüm_ad	bölüm_no	y_sos_g_n	y_is_b_tar
Satış	1	037165	01/07/89
Muhasebe	2	277143	01/08/91
Üretim	3	577211	04/06/92
Eğitim	4	443421	01/05/91
Bilgi işlem	5	288111	05/02/92

```
Select soyad,bölüm_ad FROM Personel,Bölüm WHERE Personel.bol_no=Bölüm.bölüm_no;
```

Tablo 4.1 Personel ve Bölüm Tablolarının, Müşterek Alan Olan Bölüm Numarası Üzerinde JOIN (Birleştirme) İşlemine Tabi Tutulması Sonucu Elde Edilen Bilgi.

sicil	sos_g_no	ad	soyad	dog_tar	adres	cin_siyet	maas	bol_no	yon_s_g_no	bölüm_ad	bölüm_no	y_sos_g_no	y_is_b_tar
112	27641	Ali	Can	01/05/60	Fatih	.T.	8000000	1	037165	Satış	1	037165	01/07/89
175	37654	Ayşe	Şen	04/07/65	Kadıköy	.F.	7000000	1	037165	Satış	1	037165	01/07/89
217	176241	Akın	Öncel	11/07/64	Üsküdar	.T.	6000000	2	277143	Muhasebe	2	277143	02/08/91
517	27615	Can	Öner	05/08/65	Fatih	.T.	4000000	2	277143	Muhasebe	2	277143	02/08/91
618	57253	Beril	Meral	08/07/62	Pendik	.F.	3750000	2	277143	Muhasebe	2	277143	02/08/91
1540	44721	Ayşe	Cansu	07/08/63	Beşiktaş	.F.	4800000	3	577211	Üretim	3	577211	04/06/92

Tablo 4.1’de görüleceği üzere, Personel ve bölüm tablolarında sadece her iki tabloda da aynı olan bölüm numaralarına ait satırlar alınarak birleştirilmiş ve listelenmiştir.

Listelenen birleştirilmiş tabloda, her iki tablodan da alındığı için tekrar eden bölüm numarası ve yönetici sosyal güvenlik numarası gibi sütun başlıklarının bir kez yazılması isteniyorsa, o takdirde SELECT komutunda * sembolü yerine sadece, sonuçta yazılması istenen sütun başlıkları kullanılmalıdır:

Örneğin aşağıdaki SELECT komutu ile aşağıdaki tablodaki sonuçlar üretilecektir:

```
SELECT sicil,ad,soyad,bol_no,yon_s_g_no
FROM Personel,bölüm
WHERE Personel.bol_no=bölüm.bölüm_no;
```

Tablo 4.2 Join İşleminde, Sadece Arzu Edilen Alanların Listelenmesi Sonucu Üretilen Bilgi.

sicil	ad	soyad	bol_no	yon_s_g_no
112	Ali	Can	1	037165
175	Ayşe	Şen	1	037165
217	Akın	Öncel	2	277143
517	Can	Öner	2	277143
618	Beril	Meral	2	277143
1540	Ayşe	Cansu	3	577211

Personel ve bölüm tablolarından yararlanılarak aşağıdaki şekilde yazılacak bir SELECT komutu ise, ilgili sütun elemanları ile ilişkili kartezyen çarpımı işlemi gerçekleştirecektir.

```
SELECT sos_g_no,bölüm_ad
FROM Personel,bölüm;
```

Tablo 4.3 Personel ve Bölüm Tabloları Üzerinde Kartezyen Çarpımı İşlemi.

sosy_g_no	bölüm_ad
27615	Satış
27615	Muhasebe
27615	Üretim
27615	Eğitim
27615	Bilgi işlem
57253	Satış
57253	Muhasebe
57253	Üretim
57253	Eğitim
57253	Bilgi işlem
...	...
...	...
...	...
...	...
44721	Satış
44721	Muhasebe
44721	Üretim
44721	Eğitim
44721	Bilgi işlem

Burada Personel tablosundaki her sosyal güvenlik numarası, sıra ile, bölüm tablosundaki her bölüm adı ile eşlenerek listelenmiştir (6 personel ve 5 bölüm varsa, toplam 30 satır listelenecektir). Bu tür bir listelemin pekçok durum için faydasız ve anlamsız olacağı açıktır.

NOT: $A=(a,b,c,d)$ ve $B=(x,y,z)$ kümelerinin kartezyen çarpımı
 $A \times B = ((a,x),(a,y),(a,z),(b,x),(b,y),(b,z),(c,x),(c,y),(c,z),(d,x),(d,y),(d,z))$
şeklinde tanımlanır.

4.3 Bir Tablonun Farklı İsimlerdeki Eşdeğerleri İle Sorgulama

Daha önceden tanımlanmış bir tablonun, farklı isimli, bir eşdeğerini oluşturarak sorgulamalarda kullanmak mümkün olabilir. Örneğin, aşağıdaki sorgulamayı göz önüne alalım:

Her personel için, personel sicil numarası, ad ve soyadı ile bu personelin yöneticisinin ad, soyad ve doğum tarihini listeleyiniz: Bu isteğe cevap teşkil edebilecek SQL ifadelerinden bir tanesi, aşağıdaki gibi düzenlenebilir.

```
SELECT A.sicil,A.ad,A.soyad,B.ad,
B.soyad,B.dog_tar
FROM Personel A B
WHERE A.yon_s_g_n=B.sosy_g_no ;
```

Personel tablosu aşağıdaki gibi olsun:

sicil	sosy_g_no	ad	soyad	dog_tar	yon_s_g_n
117	276543	Ali	Can	01/05/67	372651
247	372651	Can	Okan	01/06/60	555115
348	572416	Ayşe	Akın	01/05/70	372651
447	443211	Beril	Caner	02/04/65	555115
542	555115	Akın	Öner	02/07/54	771727
...
...

Yukarıdaki SELECT komutunun çıktısı aşağıdaki tabloda verilmiştir.

Tablo 4.4 Eşdeğer Tablo Yardımı İle Sorgulama Sonucu.

sicil	ad_a	soyad_a	ad_b	soyad_b	dog_tar
117	Ali	Can	Can	Okan	01/06/60
348	Ayşe	Akın	Can	Okan	01/06/60
247	Can	Okan	Akın	Öner	02/07/54
447	Beril	Caner	Akın	Öner	02/07/54

Bu SELECT komutu ile, personel tablosunun A ve B isimli birer kopyası oluşturulur. Bu kopyalara personel kütüğünün eşdeğerleri ya da takma adları (aliases) adı verilir. SELECT komutu, personel tablosunun eşdeğeri olan ve yönetilenleri temsil eden B tablosundaki sosyal güvenlik numarasına eşit olan satırları kontrol ederek her personelin sicil no, ad, soyad bilgilerini ve bu personelin yöneticisinin ad, soyad ve doğum tarihini listeler (yukarıdaki tabloda verilmiştir). Buradaki yöntem ile bir tablonun kendisi ile birleştirilmesi işlemi kendi üzerinde birleştirme (self-join) adını almaktadır. Bazı SQL gerçekleştirmelerinde, bu SELECT komutunun,

```
SELECT A.sicil,A.ad,A.soyad,B.ad,B.soyad,B.dog_tar
FROM Personel A,Personel B
WHERE A.yon_s_g_n=B.sosy_g_no;
```

şeklinde yazılması gerekir.

Aşağıdaki örnek soru için, eşdeğer tablo oluşturma özelliğinden yararlanarak, benzer şekilde SELECT komutu düzenleyebiliriz:

SORU: Satış bölümünde çalışan tüm personelin ad, soyad ve adreslerini listeleyiniz.

ÇÖZÜM: SELECT a.ad,a.soyad,a.adres
FROM Personel a, Bölüm b
WHERE b.bölüm_ad="Satış" AND
a.bol_no=b.bölüm_no;

şeklinde olacak ve tablo aşağıdaki gibi olacaktır.

Personel tablosu

sicil	ad	soyad	adres	bol_no
117	Ali	Can	Fatih	1
247	Selin	Akın	Bakırköy	2
348	Can	Seler	Üsküdar	1
547	Beril	Caner	Pendik	1
648	Akın	Çalışır	Kadıköy	4

Bölüm tablosu

bölüm_ad	bölüm_no	y_sos_g_n	y_is_b_tar
Satış	1	276241	01/05/87
Muhasebe	2	372615	02/04/90
Bilgi işlem	3	44272	04/08/91

şeklinde ise Tablo 4.5'teki sonuç elde edilecektir.

Tablo 4.5 Satış Bölümündekilerin Eşdeğer Tablo Oluşturma Yöntemi İle Listelenmesi Sonucu.

ad	soyad	adres
Ali	Can	Fatih
Can	Seler	Üsküdar
Beril	Caner	Pendik

4.4 İç İçe Select Komutları (Nested Selects)

Bazı sorgulamalar, özelliği itibarı ile iç içe SELECT komutlarının kullanılmasını gerektirebilir. İçteki SELECT komutunun bulduğu sonuç, dıştaki SELECT komutunun işlevini yerine getirmesi için kullanılır. Örnek olarak, aşağıdaki soruyu göz önüne alalım:

SORU: Parça numarası 24 olan parçayı kullanan projelerde çalışan personeli listeleyiniz.

Örnek olarak aşağıdaki verileri göz önüne alalım:

Parça

par_no	par_ad	pr_no	fiyat	ağırlık
24	Vida	2	2000	500
24	Vida	4	2000	500
37	Civata	2	6000	800
87	Conta	2	7000	5000
112	Pim	5	6000	70

Proje

proje_ad	proj_no	yer	bl_no
1	1	İstanbul	4
2	2	İstanbul	4
3	3	Ankara	5
4	4	Ankara	5
5	5	İzmir	4

Personel

sicil	sosy_g_no	ad	soyad	dog_tar	bol_no	adres
117	274251	Ali	Can	05/01/60	4	Akar sok. 2 Fatih
247	527241	Hasan	Okan	04/07/62	4	Merk cad. 3 Pendik
348	527672	Ayşe	Pekcan	04/08/65	5
548	443211	Akın	Pekol	07/02/70	4
1148	527625	Mert	Caner	04/08/70	5

Çalışma

per_s_g_no	proje_no	saat
274251	1	250
527241	2	350
527672	3	400
443211	5	300
527625	4	250

Bu tablolardan yararlanarak aşağıdaki SELECT komutları ile arzu edilen işlem gerçekleştirilebilir:

```
SELECT *  
FORM Personel  
WHERE sosy_g_no  
IN (SELECT Per_s_g_no  
FROM Parça,Proje,Çalışma  
WHERE pr_no=proj_no AND  
proj_no=proje_no AND par_no=24);
```

Buradaki içteki SELECT komutu parça, proje ve çalışma tablolarını proje numaraları üzerinde (proje numaraları bu tablolarda sıra ile pr_no, proj_no ve proje_no adı ile yer almaktadır) birleştirerek elde edilen genişletilmiş tablodan sadece parça no'su 24 olan satırdaki personel sosyal güvenlik numaralarını (pers_s_g_no) çıkarmakta ve sonuçta yukarıdaki örnek data için

per_s_g_no
527241
527625

değerlerini elde etmektedir.

Dış SELECT komutu ise, personel tablosundan bu sosyal güvenlik numaralarına sahip personel aşağıdaki tabloda görüldüğü gibi listelenecektir:

sicil	sosy_g_no	ad	soyad	dog_tar	bol_no
247	527241	Hasan	Okan	04/07/62	4
1148	527625	Mert	Caner	04/08/70	5

Benzer şekilde aşağıdaki sorunun çözümü de iç içe SELECT komutları ile gerçekleştirilebilir.

SORU: Fatih'te oturan personelin çalıştığı projelerin adları ve yerlerini listeleyiniz.

ÇÖZÜM: SELECT proje_ad,yer
FROM Proje
WHERE proj_no IN(SELECT proje_no
FROM Personel,Çalışma
WHERE sosy_g_no=Per_s_g_no
AND adres LIKE '%Fatih%');

SONUÇ:

proj_ad	yer
1	İstanbul
3	Ankara

çıktısı elde edilecektir.

4.5 UNION Sözcüğü

UNION sözcüğü küme birleşimi işlemi görür. İki ayrı SELECT komutunun sonucunda elde edilen tabloların birleşimi işlemini gerçekleştirir. Aşağıdaki soru ve cevabı olan SELECT komutları bu konuda bir fikir vermektedir.

SORU: Adı Ahmet ve soyadı Caner olan kişi ya da kişileri, işletmenin yürüttüğü projelerde çalışan bir kişi (sıradan bir personel ya da bölüm yöneticisi) olarak bulunduran projelerin isimlerini ve projelerin yürütüldüğü yerleri listeleyniz.

ÇÖZÜM: (SELECT proj_no,yer
FROM Proje,bölüm,Personel
WHERE bl_no=bölüm_no AND
y_sos_g_no=sosy_g_no AND
ad="Ahmet" AND soyad="Caner")
UNION (SELECT proj_ad,yer
FROM Proje,Çalışma,Personel
WHERE proj_no=proje_no AND
per_s_g_no=sosy_g_no AND ad="Ahmet" AND soyad="Caner");

UNION sözcüğü ile, iki ya da daha çok SELECT 'in sonucu olan tabloların küme birleşimi işlemine tabi tutulması için iki koşul gereklidir.,

1-) SELECT komutları sonucunda elde edilecek tablolar aynı sayıda kolon içermelidirler.

1. SELECT 2. SELECT
Sonuç Tablosu Sonuç Tablosu

--	--	--	--	--

--	--	--	--	--

5 Kolon 5 Kolon

2-) Sonuç tabloların karşılıklı olarak kolonları ayrı veri tipi ve aynı genişlikte olmalıdır.

4.6 ANY Sözcüğü

Aşağıdaki örnek soru çerçevesinde bu sözcüğün SELECT komutu içindeki etkisini açıklayacağız.

SORU: Satış bölümünde çalışan personelin her hangi birinden daha düşük maaş alan ve mühendislik bölümünde çalışan kişileri listeleyniz.

ÇÖZÜM: SELECT *
FROM Personel
WHERE maas<ANY
(SELECT maas
FROM Personel
WHERE bol_no=2) AND
bol_no=1;

Bu çözümün eşdeğeri olan ifade ise şöyledir.

```
SELECT *  
FROM Personel  
WHERE maas<(SELECT MAX (maas)  
FROM Personel  
WHERE bol_no=2) AND bol_no=1;
```

Burada satış bölümü kodu 2 ve mühendislik bölümü kodu ise 1 olarak kabul edilmiştir. İkinci çözüm ifadesinden de kolayca anlaşılacağı gibi, iç içe SELECT ifadesinde, içteki SELECT sorgulaması sonucu, ikinci bölümde (Satış) çalışan personelin içinde en yüksek maaş alan kişinin maaşı bulunmakta, dıştaki SELECT ise, mühendislik bölümünde, bu maaştan düşük olan maaşa sahip kişileri listelemektedir.

Buradaki düşünce tarzı şöyledir:

Mühendislik bölümünde çalışan ve satış bölümündeki en yüksek maaştan düşük maaş alan kişi “SATIŞ BÖLÜMÜNDEKİ HER HANGİ BİR MAAŞ’TAN DÜŞÜK OLMA” koşulunu sağlayacaktır. Eğer mühendislik bölümündeki kişinin maaşı, satış bölümündeki en yüksek maaştan daha yüksek olsa (veya ona eşit olsa) doğal olarak bu koşul sağlanmayacaktır.

Personel tablosu aşağıdaki verileri içeriyorsa,

Personel

sicil	ad	soyad	bol_no	maas
117	Ali	Can	1	7000000
247	Hasan	Okan	1	6000000
348	Ayşe	Pekcan	2	50000000
548	Akın	Pekol	1	4000000
1148	Mert	Caner	2	12000000
1215	Beril	Şen	2	5000000

Yukarıdaki SELECT komutları sonucu (ANY ile ya da eşdeğeri ile)

sicil	ad	soyad	bol_no	maas
117	Ali	Can	1	7000000
247	Hasan	Okan	1	6000000
548	Akın	Pekol	1	4000000

tablosu elde edilecektir.

ANY (her hangi bir) sözcüğü yerine, tamamen eşdeğeri olan SOME sözcüğü de kullanılabilir.

4.7 ALL Sözcüğü

“Hepsi, tamamı” anlamındaki bu sözcük, SELECT komutu içerisinde belirli bir koşulu sağlayan bir grup datanın tamamınca sağlanan koşullarla ilişkili olarak kullanılır. Aşağıdaki soru konuda açıklayıcı olacaktır:

SORU: Satış bölümünde çalışan ve mühendislik bölümündeki personelin hepsinden daha fazla maaş alan personeli listeleyiniz. Bu örnekte de satış bölümü kodu 2 ve mühendislik bölümü kodu 1 olarak alınırsa aşağıdaki SELECT grupları çözüm teşkil edecektir.

ÇÖZÜM:

1. Alternatif

SELECT *

FROM Personel

WHERE maas>ALL(SELECT maas

FORM Personel

WHERE bol_no=1)

AND bol_no=2;

2. Alternatif

SELECT *

FROM Personel

WHERE maas>(SELECT MAX(maas)

FROM Personel

WHERE bol_no=1)

AND bol_no=2;

Personel tablosu aşağıdaki verileri içeriyorsa

Personel

sicil	ad	soyad	bol_no	maas
117	Ali	Can	1	7000000
247	Hasan	Okan	1	6000000
348	Ayşe	Pekcan	2	50000000
548	Akın	Pekol	1	4000000
1148	Mert	Caner	2	12000000
1215	Beril	Şen	2	5000000

Yukarıdaki 1. Alternatif ve 2. Alternatif olarak belirtilen, her iki SELECT komutları grubu da aşağıdaki sonucu verecektir.

SONUÇ:

sicil	ad	soyad	bol_no	maas
348	Ayşe	Pekcan	2	5000000
1148	Mert	Caner	2	12000000

4.8 EXISTS Operatörü

“Var, mevcuttur” anlamındaki bu sözcük, SQL’de boolean (lojik, mantıksal) operatördür. İçteki SELECT komutunun sorgulanması sonucunda, en az bir tablo satırı üretilmişse EXISTS operatörü TRUE (doğru) değerini, hiçbir tablo satırı üretilmemişse, EXISTS operatörü FALSE (yanlış) değerini üretir.

EXISTS operatörü, AND, OR ve NOT gibi diğer mantıksal operatörlerle birlikte kullanılabilir. Aşağıdaki soru ve çözümü bu mantıksal operatörle ilişkili bir fikir vermektedir:

SORU: Numarası 27 olan parçayı satan satıcılarla ilişkili tüm bilgileri listeleyiniz.

Gerekli SELECT komularını yazabilmek için, sistemde satıcı adlı tabloda ve par_sat (parça ve satıcısı ile ilişkili bilgileri içeriyor) adlı tabloda aşağıdaki kolon ve verilerin bulunduğunu varsayalım:

Satıcı

satıcı_n	adı	adres
1	Ali Akın	İstanbul
2	Ayşe Can	İstanbul
3	Akın Peker	Ankara
4	Can Ozan	İzmir
5	Mert Ak	Antalya

Par_sat

sat_no	parca_n	miktar
1	1	200
1	27	300
2	27	250
2	42	115
3	1	500
3	2	56
4	15	800
5	18	900

ÇÖZÜM:

```
SELECT *
FROM Satıcı
WHERE EXISTS (SELECT *
              FROM Par_sat
              WHERE sat_no=satıcı_n AND parca_n=27);
```

Aşağıdaki sonuç elde edilir:

satıcı_n	adı	adres
1	Ali Akın	İstanbul
2	Ayşe Can	İstanbul

Yukarıda da belirtildiği gibi, iç SELECT’te WHERE’i izleyen koşulu sağlayan satırlar, Par_sat içinde bulunduğça (mevcut iseler), EXISTS operatörü TRUE (doğru) değeri verecek bu durumda dış SELECT’in WHERE koşul kısmı doğru olacağı için, o satıcı ile ilişkili bilgiler, satıcı tablosunda listelenecektir.

4.9 NOT EXISTS İfadesi

EXISTS’te belirtildiği gibi, EXISTS mantıksal operatörü, NOT, AND ve OR mantıksal operatörleri ile birlikte de kullanılır.

NOT EXISTS şeklinde kullanılması, EXISTS’te açıklanan yapının tamamen tersi bir etki yaratır; yani SELECT komutunun sorgulanması sonucu koşulu sağlayan hiçbir tablo satırı bulunamazsa, dıştaki SELECT için WHERE’i izleyen koşul kısmı TRUE (doğru) olarak kabul edilir ve SELECT icra edilir.

SORU: Numarası 27 olan parçayı satmayan satıcılar kimlerdir? (EXISTS operatöründe kullanılan Satıcı ve Par_sat tablolarındaki verilerin aynen geçerli olduğunu varsayalım)

ÇÖZÜM:

```
SELECT *
FROM Satıcı
WHERE NOT EXISTS (SELECT *
                  FROM Par_sat
                  WHERE sat_no=satıcı_n
                  AND parca_n=27);
```

SONUÇ:

satıcı_n	adı	adres
3	Akın Peker	Ankara
4	Can Ozan	İzmir
5	Mert Ak	Antalya

NOT EXISTS ifadesinin kullanımı ile ilişkili olarak diğer bir örnek de aşağıdaki soru ile ilişkili olarak verilmiştir:

SORU: İşletmenin yönettiği projelerde kullanılan parçalardan, tümünü de bulunduran ve İstanbul’da faaliyet gösteren satıcılarla ilişkili bütün bilgileri listeleyiniz.

ÇÖZÜM: SELECT *
FROM Satıcı
WHERE NOT EXISTS (SELECT *
FROM Parça
WHERE NOT EXISTS
(SELECT *
FROM Par_sat
WHERE sat_no=satıcı_n AND
parca_n=par_no))
AND ADRES LIKE ‘%İstanbul%’;

Bu ifade şu şekilde yorumlanabilir.

İstanbul’daki satıcılar içinde, satmadıkları bir parça mevcut olmayan satıcıları liste!

4.10 Select Komutu İçinde Except Sözcüğü

EXCEPT sözcüğü, iki tablo arasında küme farkı (difference) işlemini gerçekleştirir. Tablo 1 ve Tablo 2 aşağıdaki gibi olsun.

Tablo 1	Tablo 2
Sütun 1	Sütun1
a	a
b	b
c	c
d	

Tablo 1 EXCEPT

Tablo 2

Tablo 1 ile Tablo 2’de yukarıdaki işleme göre (Tablo 1 EXCEPT,Tablo2) işlem sonucu elde edilecek tabloda, Tablo 1’de bulunup Tablo 2’de bulunmayan veriler mevcut olacaktır. Bu işlem SQL’de

Tablo 1 EXCEPT Tablo 2 şeklinde ifade edilir ve sonuçta elde edilen fark tablosu aşağıdadır:

Tablo 1 EXCEPT Tablo 2

Sütun 1
d

Aşağıdaki soru ve çözümü bu sözcüğün kullanımı konusunda daha iyi fikir verecektir:

SORU: Satış bölümündeki personel adlarından, mühendislik bölümünde bulunmayanları listeleyiniz: (Satış için bol_no 1, mühendislik için bol_no 2 olduğunu varsayalım.)

ÇÖZÜM: SELECT * FROM
(SELECT ad FROM Personel
WHERE bol_no=1
EXCEPT
SELECT ad FROM Personel
WHERE bol_no=2);

4.11 Select Komutu İçinde Intersect Sözcüğü

İki tablo arasında küme kesişimi (intersection) işlemini gerçekleştirir.

Tablo 1	Tablo 2
Sütun 1	Sütun1
x	p
y	q
z	x
	y

şeklinde ise;

TABLO 1 INTERSECT TABLO 2 işlemi sonucunda

Sütun 1
x
y

tablosu elde edilecektir. (Küme kesişimi işleminde, her iki kümede mevcut olan müşterek elemanlar alınır.)

SORU: Hem Ankara'daki hem de İstanbul'daki projelerde görev alan bölümleri listeleyiniz.

ÇÖZÜM: SELECT * FROM
(SELECT bl_no FROM Proje
WHERE yer LIKE '%Ankara%'
INTERSECT
SELECT bl_no FROM Proje
WHERE yer LIKE '%İstanbul%');

Proje tablosundaki bilgi aşağıdaki gibi olsun:

proj_ad	proj_no	yer	bl_no
otoyol-1	2	Ankara-Türkiye	1
ekspers-2	4	İstanbul-Türkiye	1
subway-2	16	Bakü-Azerbeycan	2
otoyol-16	17	Ankara-Türkiye	4
köprü-5	21	İstanbul-Türkiye	4
köprü-17	13	Roma-İtalya	1
geçit-5	18	İstanbul-Türkiye	7

Yukarıdaki SELECT komutu sonucunda bulunan tablolar ile son adımda

bl_no	bl_no
1	1
4	4
	7

işlemi sonucunda

bl_no
1
4

tablosu elde edilecektir.

4.12 Bir Select Komutunun Sonucunu Ayrı Bir Tablo Olarak Saklamak

Bir SELECT komutunun sonucu olarak elde edilecek bilgileri, geçici bir tablo olarak saklamak mümkündür. Bunu gerçekleştirebilmek için, SELECT komutunun sonuna SAVE TO TEMP Tabloadı şeklinde bir ifade eklemek gerekmektedir.

SORU: Bayan personeli Bayan adlı bir tablo içinde saklayınız.

ÇÖZÜM: SELECT *
FROM Personel
WHERE cinsiyet=.F. SAVE TO TEMP Bayan;

cinsiyeti belirleyen cinsiyet alanında .F. simgesinin bayanları temsil ettiği varsayımı ile, personel adlı tablodaki bayan personel bayan adlı tabloya geçici olarak saklanacaktır.

Bayan adlı tabloya, SELECT komutu, bütün şekilleri ile uygulanabilir. Örneğin, bayanlar içinde maaşı 1000000 TL'den fazla olanlar listelenmek istenirse

SELECT *
FROM Bayan
WHERE maas>1000000;

ifadesi kullanılır.

Burada üretilen tablonun geçici değil kalıcı olması istenirse KEEP sözcüğü eklenmelidir:

```
SELECT *  
FROM Personel  
WHERE cinsiyet=.F.  
SAVE TO TEMP Bayan KEEP;
```

BÖLÜM 5 VIEW (BAKIŞ) OLUŞTURMAK

5.1 Temel Tablo (Base Table) ve Bakış (View) Kavramı

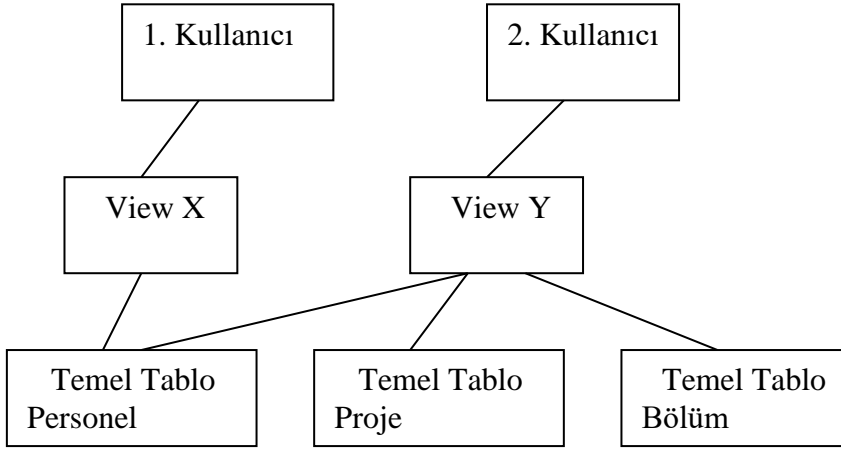
Bu bölüme kadar oluşturulan ve kullanılan tablolar, veri tabanı ya da SQL açısından temel tablolar (base tables) adını alırlar. Bu tablolar fiziksel olarak veri tabanı ortamında mevcut olan tablolardır.

Veri tabanı kavramı içinde teorik olarak mevcut olan view (bakış) terimi, farklı kullanıcıların veri tabanına nasıl baktıklarını ya da bakış açılarını anlatan bir terimdir. Bu anlamda bir kullanıcı, fiziksel olarak mevcut olan tabloların sadece bir kısmı ile ilgilenilebilir.

Temel tabloların tamamı ya da bir kısmı kullanılarak elde edilen ve fiziksel olarak veri tabanında mevcut olmayan sadece nasıl elde edilebileceğine dair bilginin saklandığı tablolara bakış (view) adı verilir. Bunlar için virtüel tablo terimide kullanılır.

SQL, bu tip tabloların oluşturulma ve kullanılmasına da olanak sağlayan komutlara sahiptir.

Şekil 5.1'de kavramsal olarak bir view'in nasıl oluşturulacağı gösterilmektedir.



Şekil 5.1 Kullanıcılar, Temel Tablolar ve View Oluşturma.

Örneğin, 1. Kullanıcı, Personel adlı tablonun sadece bayan personeli ile ilgilenmektedir. Bu nedenle, X adlı, sadece bayan personeli içeren bir view tablosu, personel adlı temel tablodan elde edilebilir.

2. Kullanıcı ise, örneğin 3. Bölümün yönettiği projelerde çalışan personel listesi ile ilgilenmektedir. O nedenle personel, proje ve bölüm adlı temel tabloları kullanarak, Y adlı bir view oluşturulabilir.

View oluşturmanın, veri tabanı ortamında aşağıda belirtilen faydaları vardır.

5.2 View Oluşturmanın Yararları

5.2.1 Veri güvenliği

Veri tabanı içinde bulunan tablolardaki bazı sütunlarda bulunan bilgilerin, herkes tarafından görülmesi istenmeyebilir.

Örneğin, personelin maaşlarının herkes tarafından listelenebilir olması mahsurlu olabilir. Bu durumda, Personel adlı temel (base) tablodan, persview adlı bir view oluşturulabilir.

```
CREATE VIEW persview  
AS SELECT sicil,sos_g_no,ad,soyad,dog_tar,adres,cinsiyet,bol_no,yon_s_g_n  
FROM Personel;
```

persview adlı view, herkesin kullanımına açık, Personel adlı temel (base) tablo ise, yetkili kişiler dışındakilere, erişilemez hale getirilirse, maaşların herkes tarafından erişilebilir bilgi olması önlenmiş olur.

Bir view'den bilgi listelenmesi temel tablodan bilgi listelenmesinden farklı değildir.

```
SELECT *  
FROM persview;
```

persview'den maaşlar hariç, tüm personel bilgileri listelenecektir.

Bir temel tablodan bir view oluşturulurken, temel tablodaki aynı sütun (alan) isimlerini kullanmak zorunda değildir. Örneğin, Parça adlı ve par_no, par_ad, pr_no, fiyat ve ağırlık adlı sütun (alan) isimlerini içeren tablo kullanılarak oluşturulan parview içinde, par_no yerine parc_no, fiyat yerine fiy ve ağırlık yerine ağır isimleri kullanılmıştır:

```
CREATE VIEW
Parview(parc_no,fiy,ağır)
AS SELECT par_no,fiyat,ağırlık
FROM Parça;
```

5.2.2 Sorgulamanın daha basit hale gelmesi

Karmaşık sorgulamalarda, bazı SELECT komutlarının sonuçları diğer SELECT komutlarınca kullanıldığında, sorgulamanın düzenlenmesinde yanlışlıklar yapma olasılığı artar.

Karmaşık sorgulamalar, VIEW özelliği kullanılarak daha basit hale getirilebilir. Burada temel fikir şudur: Madem ki bir view, bir sorgulama sonucu elde edilen bilgiyi (tabloyu) isimlendirerek elde edilen bir virtüel tablodur; o halde karmaşık SELECT komutu içinde, sonucu kullanılacak başka bir SELECT komutu kullanmak yerine, bu sonucu bir view olarak isimlendirerek, view adını kullanmak. Bazı durumlarda ise, işletmenin veri tabanı uygulamasında çok sık olarak sorulan karmaşık soruları bir view yapısı içinde saklayarak, daha sonra aynı tip sorgulamalar için bu view yapısını kullanarak daha basit ifadeler kullanmakta olasıdır.

SORU: Satış bölümünde çalışan personelin herhangi birinden daha düşük maaş alan ve mühendislik bölümünde çalışan kişileri listeleyiniz.

ÇÖZÜM:

```
SELECT *
FROM Personel
WHERE maas<ANY(SELECT maas
                FROM Personel
                WHERE bol_no=2) AND
bol_no=1;
```

(Satış bölümü kodu 2 ve mühendislik bölümü kodu ise 1 kabul ediliyor.)

Şimdi bu sorunun cevabı olan tablo bir view olarak saklanırsa:

```
CREATE VIEW S1view
AS SELECT *
FROM Personel
WHERE maas<ANY(SELECT maas
                FROM Personel
                WHERE bol_no=2) AND
bol_no=1;
```

bundan sonra aynı tip sorgulama için sadece

```
SELECT *
FROM S1view;
```

yazmak yeterli olacaktır.

5.2.3 Sadece view kullanılarak gerçekleştirilen sorgulamalar

Bir tablodan elde edilecek bilgiler için, iki kademeli işlem gerektiren sorgulamalarda, ilk adımda bir view oluşturup ikinci adımda esas sorgulamayı bu view yardımı ile gerçekleştirmek, çoğu kez kaçınılmaz bir durumdur.

Aşağıdaki soru ve bunun çözümü olan SQL ifadeleri bu konuda bir fikir verecektir:

SORU: Her bölümde, o bölümdeki ortalama maaştan daha yüksek maaş alanları listeleyiniz.

ÇÖZÜM: Bu sorunun cevaplandırılması için önce her bölümdeki ortalama maaşların bulunması gereklidir.

```
CREATE VIEW BOL_OR_VIEW(bol_no,ort,maas)
AS SELECT bol_no,AVG(maas)
FROM Personel
GROUP BY bol_no;
```

Daha sonra, yaratılan BOL_OR_VIEW yardımı ile (bu view, bölüm no'ları ve bölüm ortalama maaşlarını saklamaktadır) sorulan sorunun cevabı elde edilebilir:

```
SELECT *
FROM Personel
WHERE bol_no=BOL_OR_VIEW.bol_no
AND maas>ort_maas;
```

(Bu sorunun cevabını, şu ana kadar anlatılan diğer bilgilerle bulmaya çalışınız.)

5.2.4 Veri bütünlüğünün sağlanması

View oluşturma esnasında CHECK sözcüğünün kullanılması ile, o view'i oluştururken sağlanması gereken koşulların, daha sonra view içine veri ekleme ya da değişiklik işlemlerinde de ihmal edilmesi engellenmiş olur.

Örneğin aşağıdaki gibi bir VIEW oluşturulsun:

```
CREATE VIEW UST_PER_VIEW
AS SELECT * FROM Personel
WHERE maas>25000000,
WITH CHECK OPTION;
```

Burada,maaşı 25000000'un üstünde olan personelden oluşan bir UST_PVIEW adlı view oluşturulmuştur. Daha sonra bu view içine

```
INSERT INTO UST_PER_VIEW
VALUES(27521,'27865427','Ayşe'
'Okan',{01/05/62},'Cumh. Cad. 46-Taksim',
.F.,13000000,1,'27651112');
```

komutu ile maaşı 13000000 olan bir personel eklenmek istendiği zaman şu hata mesajı alınacaktır.

Error: not enough non-null values

Eğer CHECK opsiyonu kullanılmıyorsa hata mesajı alınmadan bu veri view içine yüklenecekti. Bir tablo ya da view üzerinde veri ekleme, güncelleme ve silme işlemleri bir sonraki bölümde incelenecektir.

BÖLÜM 6 TABLOLARDAKİ DEĞİŞİKLİK YAPMAK

6.1 Tabloya Veri Ekleme

SQL'de, mevcut bir tabloya veri eklemek için kullanılacak olan komut INSERT komutudur.

Standart SQL'de, oluşturulan bir tabloya veri yüklemek için tek imkan INSERT komutudur. INSERT komutu ile, tabloya, belli bir anda, tek bir satır eklemek imkanı vardır. INSERT komutunun yazılış biçimi aşağıdaki gibidir.

```
INSERT INTO Tabloadı
(Sütunadı1,Sütunadı2,.....,Sütunadı n)
VALUES (değer1,değer2,.....,değer n);
```

Örneğin, Personel tablosuna, sicil no'su 275 olan personel ile ilişkili bilgiler aşağıdaki gibi bir INSERT komutu ile yüklenebilir:

```
INSERT INTO Personel(sicil,sosy_g_no,ad,soyad,
dog_tar,adres,cins,maas,bol_no,yon_s_g_n)
VALUES ('275','27652418','Ali','Caner',{01/05/62},'Fatih-İstanbul',T.,27000000,2,'876215342');
```

Karakter türü verilerin ' ' sembolleri arasında yüklendiğine diğer veriler içinse buna gerek olmadığına dikkat ediniz. Burada, tabloya tüm kolonlarla ilgili veri yüklendiği için, istenirse kolon isimleri ihmal edilebilir.

Standart SQL'deki INSERT komutunun, belli bir anda, tabloya, tek bir satırı yüklemesine karşılık, birçok SQL gerçekleştiriminde, yığın halinde veri yükleyen hizmet programlarından (utility) faydalanmak imkanı da vardır.

Ayrıca, INSERT komutunun bu şekli ile tabloya veri yüklemek pratikte tercih edilebilecek bir şekil değildir. (Her tablo satırı için bir INSERT komutu kullanılıyor.) Daha kullanışlı olan yol,verilerin kullanıcının zorlanmayacağı bir ekran düzeni ile klavyeden yüklenmesi daha sonra bunların INSERT ile tabloya yerleştirilmesidir.

SQL'de ekrandan interactive bilgi girişi ve ekran tasarımı sağlayacak komutlar yoktur. Fakat SQL'in bir veri tabanı yönetim yazılımının (dbase, Foxpro, Oracle) ya da daha üst düzey dilinin (C, Pascal, Cobol v.b) interactive bilgi girişine uygun komutları kullanarak bu işlemi arzu edilen kalitede gerçekleştirmesi mümkündür. SQL'in diğer dillerle etkileşimi bir sonraki bölümde incelenmiştir.

6.2 Tablo Satırlarını Silme

Bir tablonun satırlarını silmek için gerekli komut DELETE komutudur. Satır silme koşullu ya da koşulsuz olarak gerçekleştirilebilir.

```
DELETE FROM Tabloadı;
```

ÖRNEK: DELETE FROM Personel;
25 Rows Deleted

Bu komut ile Personel tablosundaki tüm satırlar silinecektir. 25 Rows Deleted mesajı ile, o anda tabloda bulunan 25 satırın silindiği bildirilmektedir.

Koşula bağlı olarak satır silmeyi gerçekleştirmek için, DELETE komutuna WHERE sözcüğü eklenmeli ve bunu izleyen ifade koşulu göstermelidir.

ÖRNEK: DELETED FROM Personel
WHERE bol_no=2;
5 Rows Deleted

Bu komut ile, 2 numaralı bölümdeki personelin tümü tablodan silinecektir. 5 Rows Deleted mesajı ile de, o anda 2 numaralı bölümde çalışan 5 personele ait satırların silindiğini belirtmektedir.

Aşağıdaki örnekte ise maaş alanı boş olmayan tüm personel silinecektir.

DELETE FROM Personel
WHERE maas IS NOT NULL;
25 Rows Deleted

6.3 Tablo Satırlarındaki Verilerde Değişiklik Yapma-Güncelleme İşlemi

Tablo satırlarında güncelleme yapmak için SQL'de UPDATE komutu kullanılır. DELETE komutunda olduğu gibi, UPDATE komutunu da koşullu ya da koşulsuz olarak kullanmak mümkündür. Koşul belirtilmemişse, belirtilen değişiklik tüm tablo satırları üzerinde gerçekleştirilir. Koşul belirtildiği takdirde, sadece koşulu sağlayan satırlar üzerinde değişiklik gerçekleştirilir. UPDATE komutunun yazılış biçimi aşağıdaki gibidir.

Koşulsuz ise,

UPDATE Tabloadı
SET Kolonadı1=değer1,Kolonadı2=değer2,.....,Kolonadı n=değer n;

Koşullu olduğu takdirde,

UPDATE Tabloadı
SET Kolonadı1=değer1,Kolonadı2=değer2,.....,Kolonadı n=değer n
WHERE Koşul;

Aşağıdaki UPDATE komutunun kullanılışı ile ilgili örnekler verilmiştir.

SORU: Tüm personelin maaşlarına %12 zam yapma işlemini gerçekleştiriniz.

ÇÖZÜM: UPDATE Personel
SET maas=maas*1.12;

SORU: 5 inci bölümde çalışan kişilerin maaşlarına %35 zam yapan UPDATE komutunu yazınız.

ÇÖZÜM: UPDATE Personel
SET maas=maas*1.35
WHERE bol_no=5;

SORU: 2 inci bölümün yürüttüğü projelerde kullanılan tüm parçaların fiyatlarına %7 zam yapan UPDATE komutunu yazınız.

ÇÖZÜM: UPDATE Parça
SET fiyat=fiyat*1.07
WHERE pr_no IN (SELECT proj_no
FROM Proje
WHERE bl_no=2);

SORU: Sicil numarası 27265421 olan personelin bölüm numarasını 5 olarak değiştiren ve maaşına %14 zam yapan UPDATE komutunu yazınız.

ÇÖZÜM: UPDATE Personel
SET bol_no=5,maas=maas*1.14
WHERE sicil='27265421';

6.4 Tablonun Yapısında Değişiklik Yapma (Alter Table Komutu)

ALTER TABLE komutu ile bir tablonun yapısında değişiklik yapmak mümkündür. Standart SQL'de bu değişiklikler, tabloya yeni bir kolon ekleme (ADD sözcüğü yardımı ile) ve mevcut bir kolonun özelliklerini değiştirme (MODIFY komutu ile kolon genişliğini değiştirme ya da kolondaki verinin NULL ya da NOT NULL özelliğini değiştirme) şeklindedir.

Standart dışına çıkan bir çok SQL gerçekleştiriminde ise ayrıca tablodan bir kolon silme (DROP), mevcut bir kolonun adını değiştirme (RENAME) ya da tablonun adını değiştirme (RENAME TABLE) özellikleri de, ALTER TABLE komutu içinde mevcuttur.

6.4.1 Mevcut bir tabloya bir kolon ekleme

ALTER TABLE komutu içinde ADD sözcüğü kullanılarak, mevcut tabloya bir satır eklenebilir.

Mevcut bir tabloya, yeni bir kolon eklenirken, o kolon içindeki verinin türü, uzunluğu ve bu kolondaki verinin boş bırakılıp bırakılmayacağı (NULL veya NOT NULL) özellikleri de belirtilir.

SORU: Personel tablosuna, işe başlama tarihini belirten yeni bir kolon ekleyiniz.

ÇÖZÜM: ALTER TABLE Personel
ADD is_bas_tar DATE;

Yeni eklediğimiz is_bas_tar alanı içinde veri yüklü olmayacağı için boş olacak yani NULL değerler taşıyacaktır. Eğer ADD is_bas_tar DATE NOT NULL; şeklini kullansaydık, bu kolon satırları gene boş olacaktı; fakat bu kolon ile ilişkili yeni boş değerler eklenmek istendiğinde, buna müsaade etmeyecekti (INSERT komutu ile). ADD sözcüğü ile aynı anda birden çok kolon eklenebilir.

6.4.2 Mevcut bir tablonun kolonlarında değişiklik yapma (modify komutu)

Mevcut bir kolon üzerinde değişiklik yapma, değişken uzunluklu bir veri tipine sahip olan kolonun genişliğini artırma ile sınırlıdır. Bu anlamda, kolon genişliğini azaltma ya da veri tipini değiştirme mümkün değildir.

Bu işlem için MODIFY sözcüğü ALTER TABLE komutu içinde kullanılır.

SORU: Daha önce Proje adlı tabloda VARCHAR(15) olarak tanımlanmış olan yer adlı alanı, 25 olarak genişleten SQL komutunu yazınız.

ÇÖZÜM: ALTER TABLE Proje
MODIFY yer VARCHAR(25);

Aynı anda birden çok kolon üzerinde değişiklik yapılabilir. Yukarıda belirtildiği gibi, tabloda daha önce tanımlanmış bir tür (type) başka bir türe çevrilmez. Örneğin DATE'i MODIFY komutu ile CHAR, ya da INT olan bir alanı VARCHAR şekline dönüştürmek mümkün değildir.

6.4.3 Mevcut bir tablodan bir kolon silme (drop komutu)

Mevcut bir tablodan, bir kolon silmek için, ALTER TABLE komutu içine DROP sözcüğü eklemek gerekecektir. Örneğin, Personel tablosundan, is_bas_tar kolonunu silmek için

```
ALTER TABLE Personel  
DROP is_bas_tar;
```

komutunu kullanmak gerekir.

Aynı anda birden çok kolon silinebilir. Bu durumda, DROP komutu içinde bunları virgüllerle ayırmak gerekir.

```
ALTER TABLE Personel  
DROP is_bas_tar,yon_s_g_n;
```

Personel tablosundan işe başlama tarihi ve yönetici sosyal güvenlik numarası alanları silinmiştir.

Bir tablodan bir kolon silindiği takdirde, bu tablo kullanılarak üretilmiş VIEW'lerdeki ilgili kolonlar da otomatik olarak silinir. İndeks alanı olarak tanımlanmış alanların tablodan silinmesi, sistem tarafından kabul edilmez; önce indeks özelliğinin iptal edilmesi gerekecektir.

6.5 Bir Tablonun Adını Değiştirme – Rename Table Komutu

Mevcut bir tablonun adını değiştirmek için, ALTER TABLE komutu içinde RENAME TABLE ifadesi kullanılmalıdır. Örneğin Personel tablosunun adını elemanlar olarak değiştirmek istersek aşağıdaki komutu kullanmak gerekecektir.

```
ALTER TABLE Personel  
RENAME TABLE elemanlar;
```

6.6 Mevcut Bir Tablonun Bir Kolonunun Adını Deęiřtirme – Rename Komutu

Mevcut bir tablonun, bir kolonunun adını deęiřtirmek için, ALTER TABLE komutu içinde RENAME sözcüęü kullanılmalıdır. Örneęin, Personel tablosunda maas alanını, br_maas olarak deęiřtirmek için ařaęıdaki komutu kullanmak gerekir.

```
ALTER TABLE Personel  
RENAME maas br_maas;
```

6.7 Mevcut Bir Tablonun Tümüyle Silinmesi – Drop Table Komutu

Bir tablonun tümünü silmek için DROP TABLE komutu kullanılmalıdır. Örneęin, Proje adlı tablonun silinmesi için ařaęıdaki komut gereklidir:

```
DROP TABLE Proje;
```

Veri tabanından bir tablo, DROP TABLE komutu ile silindięi takdirde, bu tablodan üretilmiř bütün VIEW'ler, bu tablodan üretilmiř eř tablolar, tablo üzerindeki indeksler ve tablo için konulmuř bütün öncelikler de sistemden silinir.

6.8 Bir Tabloda Yapılan Deęiřikliklerin İptali – Rollback ve Commit Komutları

ROLLBACK komutu ile, veri tabanında, kullanıcının veri tabanında çalışmaya başlamasından itibaren yaptıęı tüm deęiřiklikleri ya da en son kullanılan COMMIT komutundan sonra yapılan tüm deęiřiklikleri iptal etmek mümkündür.

```
ROLLBACK;
```

Komutu girildikten sonra, tablodan kolon silme, kolon güncelleme, tablonun tümünü silme, view silme gibi deęiřiklik işlemlerinin tümü iptal edilerek önceki duruma dönülecektir.

COMMIT komutu ise, kullanıcının veri tabanına baęlandıęı andan itibaren ya da kullanılan en son COMMIT komutundan sonraki yukarıda bahsedilen türde bütün deęiřikliklerin kalıcı olarak veri tabanına aksettirilmesini ve saklanmasını saęlar.

```
COMMIT;
```

komutu ile o ana kadar gerçekleştirilen bütün deęiřiklikler sistemde kalıcı olarak yerleřecektir. Bu konuda ayrıntılı bilgi için hareket yönetimi bölümüne bakınız.

6.9 View'ler Üzerinde Ekleme, Silme, Deęiřiklik İşlemleri

VIEW'ler üzerindeki ekleme, silme ve deęiřiklik işlemleri esas itibarı ile tablolar üzerinde yapılan benzer işlemlerden çok farklı deęildir. Fakat VIEW'ler üzerinde bu tip işlemlerin gerçekleştirilmesinde bazı kısıtlamalarda mevcuttur. Ařaęıdaki hususların belirtilmesinde fayda vardır:

Bir view'in güncellenebilir nitelikte olması için, bir birleřtirme (join) işlemi sonucunda üretilmemiř olması gerekir. Bařka bir deyiřle, CREATE VIEW komutunda FROM sözcüęünü izleyen kısımda sadece tablo adı bulunmalıdır.

View içindeki hiçbir kolon bileřik (aggregate) fonksiyonlarca üretilmiř olmamalıdır. (MAX, SUM v.b) View'in üretildeęi SELECT komutunda DISTINCT, GROUP BY ya da HAVING sözcüklerini içeren parçaların yerine getirilmiř olmamalıdır.

Bu kořulları saęlamayan view'ler sadece okunabilir (Readonly) özellikteki view'lerdir ve üzerlerinde herhangi bir deęiřiklik yapılamaz.

6.9.1 View içine satır ekleme

Daha önceden oluřturulmuř Px adlı view, ad, soyad ve maas alanlarını içermiř olsun. Bu view, güncellenebilir nitelikte ise, ařaęıdaki INSERT komutu ile, aynen tablolarda olduęu gibi kendisine bir satır eklemek mümkün olacaktır:

```
INSERT INTO Px  
VALUES ('Ali', 'Çakır', 1200000);
```

Daha önceden, VIEW oluřturulurken, CHECK OPTION alternatifini kullanılmıřsa, bu takdirde, ekleme esnasında, VIEW'i oluřturan kořul ihlal ediliyorsa, sistem eklemeye müsaade etmeyecek ve hata mesajı verecektir.

SORU: Personel adlı tablodan, maaşı 20000000 TL'yi aşan personeli alarak, UST_PER_VIEW adlı bir view oluşturunuz.

ÇÖZÜM:
CREATE VIEW UST_PER_VIEW
AS SELECT FROM Personel
WHERE maas>20000000
WITH CHECK OPTION;

Şimdi UST_PER_VIEW içine

```
INSERT INTO UST_PER_VIEW  
VALUES (37261,'34268152','Beril',  
'Caner',{01/04/64},'Kadıköy'.F.,  
14000000,2,'37624158');
```

komutu ile maaşı 14000000 olan bir kişi eklenmek istendiğinde, bu komut kabul edilmeyecek ve aşağıdaki hata mesajı alınacaktır:

Error:Not enough non-null values

Eğer CHECK opsiyonu kullanılmazdı, hata mesajı verilmeksizin bu satır, view içine eklenecekti.

6.9.2 View içinden satır silme

Güncellenebilir bir view içinde satır silme işlemi, tablolardan satır silme işlemi ile aynı şekilde gerçekleştirilir. Örneğin 6.9.2'de oluşturulan UST_PER_VIEW içinden, maaşı 2500000'den az olan kişiler silinmek istenirse

```
DELETE FROM UST_PER_VIEW  
WHERE maas<2500000;
```

komutunu kullanmak yeterli olacaktır.

6.9.3 View satırları üzerinde güncelleme işlemi

Güncellenebilir view'lerde güncelleme işlemi tablolardakinin aynıdır. Örneğin UST_PER_VIEW adlı view'de sicili 27251 olan kişinin maaşının 37000000 olarak değiştirmek için

```
UPDATE UST_PER_VIEW  
SET maas=37000000  
WHERE sicil=27251;
```

komutunu kullanmak uygun olacaktır.

6.9.4 Bir view'i silmek

Tabloların silinmesine benzer şekilde, sistemde oluşturulan bir view, DROP VIEW komutu ile silinebilir.

```
DROP VIEW UST_PER_VIEW;
```

Bir view'in silinmesi ile, o view'e bağlı olarak oluşturulmuş diğer bütün view'ler ve bu view ile ilişkili önceliklerin de tümü silinmiş olacaktır.

BÖLÜM 7 İNDEKS OLUŞTURMA ve KULLANMA

7.1 İndeks Oluşturmanın Amacı

Bir indeks, veri tabanı ortamında bir tablo ya da bir view gibi bir nesnedir ve ilişkili olarak kullanıldığı tablo ya da view'deki satırların, indeksleme alanı (key field (anahtar alan)) olarak kullanılan kolondaki verilere göre sıralanmış biçimde işleme sokulmasını (listeleme ya da arama işlemi) sağlar.

Bir tablo, indekslenmiş ise, bu tablo içinde gerçekleştirilecek bir arama (search) ya da koşullu listeleme (SELECT komutu ile) işlemi çok daha hızlı biçimde gerçekleştirilebilecektir.

7.2 İndeks Yaratma

SQL'de bir tablo ile ilişkili olarak indeks yaratmak için gerekli komut CREATE INDEX komutudur. Komutun yazılış biçimi aşağıdaki gibidir:

```
CREATE INDEX indeks adı  
ON tabloadı (kolonadı 1,kolonadı 2,....,kolonadı n );
```

İndeksleme artan (ascending) ya da azalan (decending) şeklinde olabilir. Artan, alfabetik olarak A'dan Z'ye nümerik olarak küçükten büyüğe şeklindedir. Azalan ise bunun tersidir. Hiçbir özel sözcük kullanılmazsa indeksleme artan sayılır ya da alan adının yanında bir boşluktan sonra ASC sözcüğü kullanılırsa bu alana göre artan sıralama yapılacak demektir.

Herhangi bir alanın adının yanında DESC sözcüğünün kullanılması ise indekslemenin azalan olacağını gösterir. Komutun yazılış biçiminden anlaşılacağı gibi, aynı anda, birden çok alana göre indeksleme de yapılabilir.

7.2.1 Tek bir alana göre artan sırada indeksleme

İşletmede çalışan personeli maaşlarına göre artan sırada listelemek istersek, maas alanına göre bir indeks oluşturmalıyız.

```
CREATE INDEX pers_maas
ON Personel (maas);
Index created 127 Rows
```

127 satırlık personel tablosu ile ilişkili olarak maas alanına indeks anahtarı olarak kullanılan pers_maas adlı indeks oluşturulmuştur. Bu durumda

```
SELECT *
FROM Personel;
```

şeklindeki listeleme komutu sonucunda, personel tablosundaki tüm personel, maaşlarına göre sıralı olarak listelenecektir.

7.2.2 Tek bir alana göre azalan sırada indeksleme

İşletmede çalışan personeli maaşlarına göre azalan sırada (yüksek maaştan düşük maaşa doğru) listelemek istersek, maas alanına göre aşağıdaki şekilde oluşturmak gerekir.

```
CREATE INDEX pers_maas
ON Personel (maas DESC);
```

7.2.3 Birden fazla alana göre indeksleme

İşletmedeki personelin öncelikle adlarına göre, aynı ad da olanların soyadlarına göre, hem adı hem soyadı aynı olanların maaşlarına göre sıralanmış olarak listelenmesi istenirse aşağıdaki komut kullanılmalıdır:

```
CREATE INDEX p_ad_soy_m
ON Personel (ad,soyad,maas);
```

Bu durumda

```
SELECT *
FROM Personel;
```

komutu sonucunda, aşağıdaki şekilde sıralanmış tablo görüntülenecektir.

sicil	ad	soyad	maas	
11117	Ahmet	Caner	15000000
247	Ahmet	Deniz	27000000
645	Ahmet	Zoran	12000000
3871	Ali	Caner	26000000
15372	Ali	Caner	34000000
4246	Ali	Caner	65000000
16656	Ali	Şener	12000000
7216	Beril Arkan		18000000
.....

Burada, kolayca görüleceği gibi personel öncelikle adı alanına göre sıralanmış (Ahmet, Ali, Beril) aynı ada sahip olanlar soyadlarına göre sıralanmış (Ahmet ismindeki kişilerin soyadları olan Caner, Deniz, Zoran sıralaması gibi), hem ad hem de soyadları aynı olanların sıralanmasında ise maas alanı dikkate alınmıştır.

```
İndeks komutu
CREATE INDEX p_ad_soy_m
ON Personel (ad,soyad,maas DESC);
```

şeklinde yazılsa idi, tablodaki değerler

sicil	ad	soyad	maas	
11117	Ahmet	Caner	15000000
247	Ahmet	Deniz	27000000
645	Ahmet	Zoran	12000000
3871	Ali	Cenker	65000000
15372	Ali	Cenker	34000000
4246	Ali	Cenker	26000000
16656	Ali	Şener	12000000
7216	Beril Arkan		18000000
.....

şeklinde sıralanırdı. Bu durumda farklı olan ad ve soyad alanı aynı olan kişilerin maaşlarına göre, yüksek maaştan düşük maaşa göre sıralanmış olmasıdır. (maas DESC ifadesinden ötürü)

7.2.4 Unique sözcüğü

Bir tablo, seçilen bir sütuna (alana) göre indekslenirken, indeksleme alanı olarak seçilen sütundaki verilerin tekrarlanmasına müsaade edilmesi isteniyorsa, indeksleme yapılırken, CREATE INDEX komutu içinde UNIQUE sözcüğü kullanılmalıdır:

```
CREATE UNIQUE INDEX pers_sicil
ON Personel (sicil);
```

UNIQUE sözcüğünün etkisi, bu komuttan sonra, tabloda, aynı sicilden birden fazla tekrar olmasını engellemesidir.

```
Personel tablosunu
INSERT INTO Personel
VALUES(53768,'27241685','Ayşe','Şen',{01/04/63},'Kadıköy',
.F.,27000000,2,'34261578');
```

komutu ile sicil 53768 olan kişi eklenmek istendiği zaman, bu sicilden daha önce o tabloda mevcutsa, ekleme kabul edilmeyecek ve

```
-Error- data is not unique
-Hata- veri tekrarsız (tek) değildir.
```

şeklinde bir hata mesajı alınacaktır.

7.3 Mevcut Bir İndeksin Silinmesi

Bir tablo üzerinde tanımlanmış herhangi bir indeks, o tablonun veri tabanından silinmesi ile otomatik olarak silinecektir.

Tablo silinmeksizin, o tablo üzerinde oluşturulan indeksin silinmesi içinse, DROP INDEX komutu kullanılmalıdır.

```
DROP INDEX pers_in;
```

komutu ile

```
INDEX DROPPED
(İndeks Silindi)
```

mesajı alınacaktır. Böylece, Personel tablosu üzerinde oluşturulmuş pers_in adlı indeks, personel tablosu veri tabanında kaldığı halde silinecektir.

BÖLÜM 8 SQL DEYİM BLOKLARI

SQL deyimlerini işletirden bir grup deyimini bir arada işletmek gerekebilir. Bu olanak deyim bloklarıyla yapılır. Diğer bir olanak da IF, CASE ve WHILE gibi hem blok olarak hem de blok olmadan işletilecek deyimleri belli koşullara bağlamaktır.

8.1 BEGIN...END

BEGIN ve END deyimleri bir grup SQL deyimini bir blok içinde toplamayı sağlar.

Örneğin IF deyimini ile yalnızca bir deyim işletilip işletilmemesi sınanırken, BEGIN ve END bloğu ile bir grup deyim çalıştırılıp çalıştırılmaması sağlanır.

Örnek:

```
WHILE (SELECT AVG(adet) FROM titles) < 30
```

```
-- döngü başlat
```

```
BEGIN
```

```
    UPDATE siparis
```

```
        SET adet= adet* 2
```

```
END
```

BEGIN ve END deyimleri genellikle şu durumlarda kullanılır.

- **WHILE ile döngü yapıldığında.**
- **Bir CASE fonksiyonunun blok deyimini içermesi durumunda.**
- **IF ve ELSE deyiminde.**

8.2 DENETİM DEYİMLERİ

SQL dilinde programlama dilleri kadar olmasa da program akışını kontrol etmek için deyimler ve yapılar vardır. Bunların başında IF-ELSE, CASE ve WHILE yapısı gelir.

IF...ELSE

Bir deyim işletilmesini belli bir koşula bağlar.

Kullanımı:

```
IF ifade  
  { deyim }  
[ ELSE  
  { deyim} ]
```

Örnek:

Adet ortalamasının 20'den küçük olması durumunda çalıştırılacak deyimler:

```
IF (SELECT AVG(adet) FROM siparis) < 20  
BEGIN
```

```
    --işlemler  
END
```

```
ELSE  
BEGIN
```

```
    --işlemler  
END
```

CASE

Bir değere göre daha fazla alternatifi yerine getirmeyi sağlar.

Kullanım biçimi:

CASE değer

WHEN değer THEN işlem

WHEN değer THEN işlem

ELSE işlem

END

Tablodan Aktarma:

INSERT INTO yenitablo (alanlar..)

SELECT (alan1, alan2, alan3,

CASE alan4 WHEN 'A' THEN '1' WHEN 'B' THEN '2' ELSE '3' END,

alan5 FROM eskitablo

WHERE isdate (tarih) <> 0

Örnek:

Tablolar arasında aktama:

İNSERT İNT0 KAMİL1

(KODU, ADİ, GRUBU, ADRESİ)

SELECT KODU, ADİ,

(CASE GRUBU WHEN 'A' THEN 'EMEKLI' WHEN 'B' THEN 'TERHIS' END)

,ADRESİ FROM KAMİL2

WHILE

SQL deyimlerinin döngü içinde yinelenmesini sağlar. WHILE ile belirtilen döngü koşulu yerine getirildiği sürece deyimler yerine getirilir.

Kullanımı:

WHILE ifade

{ deyim ya da blok }

[BREAK]

{ deyim ya da blok }

[CONTINUE]

Örnek: Satış adetleri 50 oluncaya kadar adet alanının artır.

```
WHILE (SELECT AVG(adet) FROM siparis) < 50
```

```
BEGIN
```

```
    UPDATE siparis
```

```
        SET adet= adet* 2
```

```
    SELECT MAX(adet) FROM siparis
```

```
    IF (SELECT MAX(adet) FROM siparis) > 50
```

```
        BREAK
```

```
    ELSE
```

```
        CONTINUE
```

```
END
```

```
PRINT 'adet değeri büyük'
```

BREAK ve CONTINUE kullanmadan:

```
WHILE (SELECT AVG(adet) FROM siparis) < 50
```

```
BEGIN
```

```
    UPDATE siparis
```

```
        SET adet= adet* 2
```

```
END
```

```
PRINT 'adet değeri büyük'
```

BREAK ve CONTINUE kullanmadan:

BÖLÜM 9 HAREKET YÖNETİMİ

9.1 Temel Kavramlar

Hareket yönetimi (transaction management) bir veri tabanı yönetim sistemindeki en önemli işlemlerden biridir. Hareket yönetiminin SQL komutları ile nasıl gerçekleştirilebileceği bu bölümün konusunu teşkil edecektir.

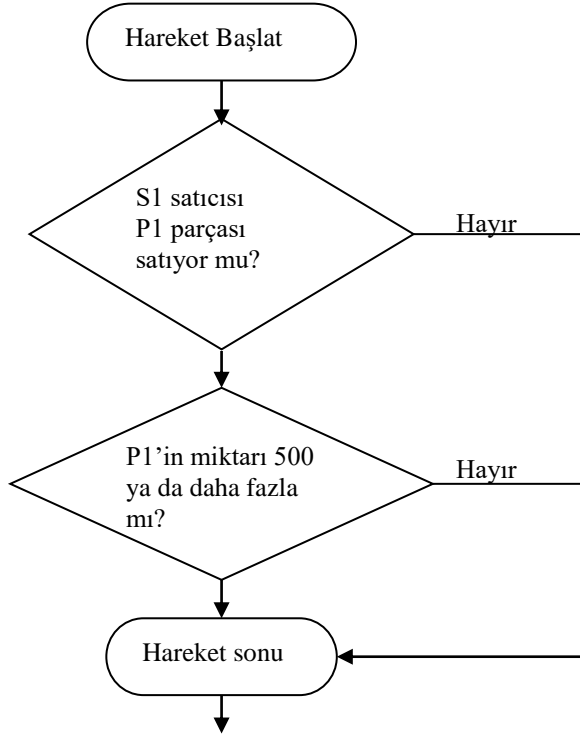
Hareket (transaction)

SQL'de bir hareket, 'veri tabanı üzerinde çeşitli fonksiyonları gerçekleştiren bir işlemler dizisi' olarak tanımlanır. Bu tanım çerçevesinde, işlemler dizisi, çok sayıda birbirini izleyen SQL komutlarından ya da SQL'in katılımlı kullanılması durumunda üst dil komutlarından oluşur.

Veri tabanında mevcut olan par_sat adlı tabloda, satıcı numarası (sat_no), parça numarası (parca_n) ve miktar (miktar) sütunları mevcuttur. Şimdi S1 satıcısından P1 parçasından 500 tane talep edilmiş olsun. Stoktan talep etme hareketi olarak nitelendirebileceğimiz bu hareket için aşağıdaki işlemlerin gerçekleştirilmesi gerekecektir:

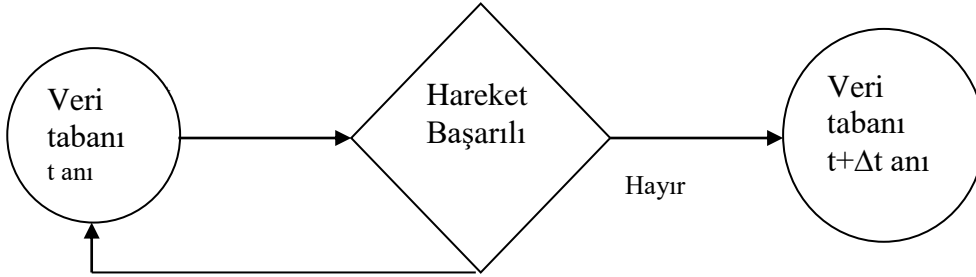
- 1) S1 satıcısının P1 parçasını satıp satmadığını kontrol etmek.
- 2) S1 satıcısı P1 parçasını satıyorsa, talep edilen miktarı, tablodaki miktarla mukayese ederek talebi karşılayıp karşılamadığını kontrol etmek.
- 3) Talep karşılanabiliyorsa, par_sat üzerinde S1 satıcısının P1 parçasının miktar alanından 500 eksilterek güncelleme yapmak.

Stok talebi hareketini aşağıdaki gibi bir akış diyagramı ile ifade etmek mümkündür:



Şekil 9.1 Hareket Akış Diyagramı

Hareket başarılı ise (istenilen değişiklikler yapılmış ise) veri tabanının yeni durumu harekettten önceki durumundan farklı olacaktır. Bunu aşağıdaki diyagram ile daha açık görmek mümkündür:



Şekil 9.2 Hareket ve Veri Tabanı Durumu İlişkisi.

Eski duruma dönme (recovery)

Bir hareket içinde herhangi bir işlem başarısızlıkla sonuçlanmışsa, o hareket içindeki işlemlerden bazıları tarafından veri tabanı üzerinde gerçekleştirilmiş değişiklikler iptal edilmeli ve veri tabanı hareketin başlamasından önceki duruma dönmelidir.

Aynı zamanda çok sayıda işleme (eşzamanlılık-concurrency)

Aynı zamanda birden çok hareket aktif halde olabilir. bu hareketlerden bazıları aynı anda veri tabanı üzerindeki aynı alanlara erişip değişiklik yapmak isteyebilir. Veri tabanı yönetim sisteminin bu durumu kontrol etmesi zorunludur.

9.2 Hareket Yönetiminin Gerçekleştirilmesi

Bir işlemler dizisinden oluşan bir hareketle, işlemlerin tümü başarılı ise veri tabanının durum değişikliğinin kalıcı hale getirilmesi, işlemlerden biri bile başarısız ise değişikliklerin reddedilerek veri tabanının harekettten önceki durumuna dönüştürülmesi şeklindeki hareket yönetimini gerçekleştirmek için iki yaklaşım vardır:


1) Standart olmayan yöntem

Bu yöntemde, hareketi oluşturan işlemlerle ilişkili komutlar BEGIN ve END TRANSACTION komutları arasına alınır.

```

BEGIN TRANSACTION;
  komut 1;
  komut 2;
  -
  -
  komut n;

```



Hareketi oluşturan komutlar

```

END TRANSACTION;

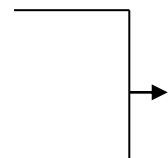
```

2) ANSI standardı
Buna göre aşağıdaki yapı kullanılacaktır:

```

komut 1;
komut 2;
-
-
komut n;
COMMIT

```



Hareketi oluşturan komutlar

Standart metod kullanıldığına göre, bir hareket, ilk rastlanılan SQL komutu ile başlar ve aşağıdaki komut ya da olaylardan biri ile karşılaşınca sona erer:

```

COMMIT
ROLLBACK
Herhangi bir veri tanımlama dili
(Data Definition Language-DDL)
komutu
  Hata koşulu
  Sistemden çıkılması (logoff)
  Sistemden normal olmayan çıkış

```

Bir işlemler dizisinden oluşan hareket sona erince, bir sonraki SQL komutu otomatik olarak devreye girer.

9.3 Commit ve Rollback Komutları

Bir hareketi oluşturan komutların sonunda COMMIT komutu kullanılmışsa, bu hareketin veri tabanı üzerinde oluşturduğu değişiklikler sistem tarafından kalıcı hale getirilir.

Hareketi oluşturan komutlar sonunda ROLLBACK komutu kullanılmışsa, gerçekleştirilen değişikliklerin tümü iptal edilecek ve veri tabanı, hareketten önceki durumuna dönecektir.

ÖRNEK:

```

UPDATE Personel
SET maas=27000000
WHERE sicil=27115;
INSERT INTO Bölüm (bölüm_ad,
bölüm_no,y_sos_g_n,y_is_b_tar)
VALUES ('halkla ilişkiler',7,'27141527',{01/05/93});
COMMIT

```

Bu örnekte, personel tablosunda sicil 27115 olan kişinin maaşı 27000000 yapılmakta ve bölüm adlı tabloya ise, halkla ilişkiler adlı yeni bir bölüm, bölüm numarası, yöneticinin sosyal güvenlik numarası ve yöneticisinin işe başlama tarihi yeni bir satır olarak yüklenmektedir.

Yukarıda açıklandığı gibi, COMMIT komutuna kadar görülen SQL komutları ANSI standart yönetimine göre bir hareket kabul edilecekler COMMIT ile bu komutlarca veri tabanında gerçekleştirilen değişiklikler kalıcı hale getirilecektir.

COMMIT yerine ROLLBACK kullanılsa idi, o takdirde yapılan değişiklikler iptal edilecek ve veri tabanında bu hareketten dolayı bir değişme görülmecekti.

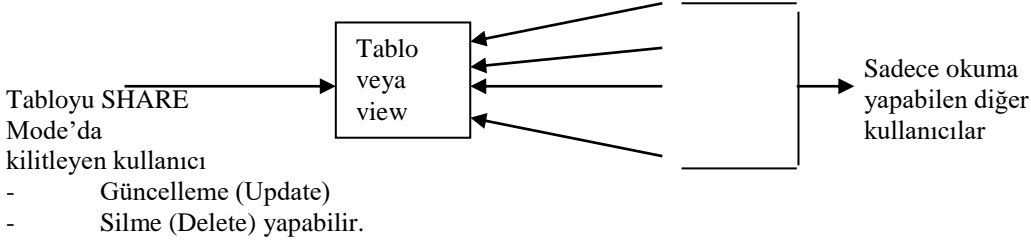
9.4 Eşzamanlı Erişimler İçin Kilitleme Yöntemi – Lock Table Komutu

Farklı veri tabanı yönetim sistemleri dolayısıyla bunlar içindeki SQL gerçekleştirmeleri, veri tabanı içindeki tablo sütunlarına aynı zamanda erişim ve güncelleme taleplerini farklı şekilde kontrol etmektedirler. Pekçok SQL gerçekleştirmesinde mevcut olan LOCK TABLE adlı SQL komutu, kullanıcıya, kısıtlı bir zaman dilimi içinde bir veya daha fazla tabloyu tek başına, özel olarak kullanma imkanı vermektedir. Bu komutun yazılış biçimi aşağıdaki gibidir:

LOCK TABLE tablo ya da view adı IN SHARE | EXCLUSIVE MODE

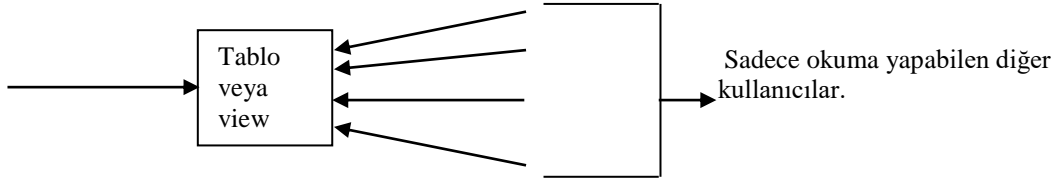
SHARE MODE seçeneği kullanılırsa, diğer kullanıcılar tablo ya da view üzerinde sadece okuma işlemi (SELECT) yapabilirler; silme ya da güncelleme yapamazlar.

SHARE ile kilitlemiş bir tabloyu aynı anda çok sayıda kişi kullanabilir. (Bir kullanıcı her türlü hakka sahip olabilir; diğerleri sadece okuma yapabilir. (Şekil 10.3))



Şekil 9.3 LOCK TABLE (Tablo Kilitleme) Komutunun SHARE MODE (paylaşımlı mod) da Kullanımı.

LOCK TABLE komutu, EXCLUSIVE MODE'da kullanılırsa, diğer kullanıcılar gene tabloda herhangi bir değişiklik yapamazlar; tabloda sorgulama yapabilirler fakat tablo üzerinde herhangi bir kilitleme (LOCK TABLE) işlemine girişmelerine müsaade edilmezler. (SHARE MODE ile aradaki en önemli fark budur (Şekil 10.4))



Tabloyu EXCLUSIVE MODE'da kilitleyen kullanıcı

Bunlar tablo üzerinde kilitleme işlemine de teşebbüs edemezler.

- Güncelleme (Update)
- Silme (Delete) yapabilir.

Şekil 9.4 LOCK TABLE (Tablo Kilitleme) Komutunun EXCLUSIVE MODE (Özel Mod) da kullanımı.

Her iki tür kilitleme komutunun da, tablo üzerindeki kilitleme etkisi, tabloyu kilitleyen kullanıcının bir COMMIT ya da ROLLBACK komutunu kullanması ile sona erer.

Sistem bir tabloyu kilitlemeye teşebbüs etmeden önce, o tablo üzerinde herhangi bir kullanıcının EXCLUSIVE MODE'da bir LOCK TABLE işlemi uygulayıp uygulamadığını kontrol eder.

9.5 Kilitleme Yönteminin Sakıncaları

Tablolar üzerinde kilitleme yönteminin uygulanmasının çeşitli sakıncaları olabilir.

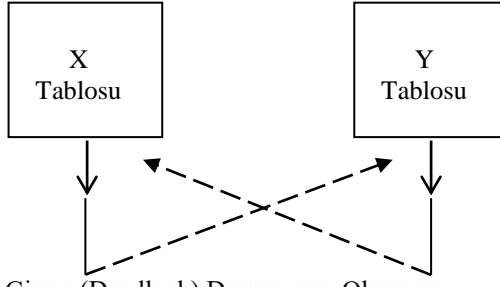
Bunlardan ilk başta geleni, sistemdeki bazı kullanıcıların bir veya daha fazla tablo üzerinde uzun süre kilitleme işlemi uygulayarak tabloları tekeline alması ve dolayısıyla sistemin genel performansını düşürmesidir.

İkinci ve çok daha ciddi olan sakınca ise çıkmaza girme (deadlock) olayının meydana gelmesidir.

Çıkmaza girme aşağıdaki şekilde oluşur:

A kullanıcısı X tablosunu EXCLUSIVE olarak kilitlemiştir. Bir sonraki işlem adımında ise Y tablosuna erişmek istemektedir. Aynı anda ise, B kullanıcısı Y tablosu üzerinde Y tablosu üzerinde EXCLUSIVE kilitleme yapmıştır ve X tablosuna erişmek istemektedir. (Şekil 9.5)

A, X'i
EXCLUSIVE
olarak
kilitlenmiştir.



B, Y'yi
EXCLUSIVE
olarak
kilitlenmiştir.

Şekil 9.5 Çıkmaza Girme (Deadlock) Durumunun Oluşması.

Çıkmaza girme durumu, herhangi bir müdahale olmazsa, sonsuza kadar bekleme ile sonuçlanacaktır.

Sistem tarafından, çıkmaza girme durumu, kullanıcılardan birini bir anlamda feda ederek ortadan kaldırılabilir.

ORACLE veri tabanı yönetim sisteminde, çıkmaza girme durumu sistem tarafından otomatik olarak tespit edilir ve sistem tarafından ortadan kaldırılır.

Ortadan kaldırma şu şekilde olur: Sistem, çıkmazı oluşturan komutları analiz ederek bunlardan birine ROLLBACK işlemi uygulayarak, o komutu kullanan kullanıcının yarattığı kilitlenmeyi ortadan kaldırır.

9.6 Otomatik Kilitleme

Bazı veri tabanı yönetim sistemleri, tablonun değişimine sebebiyet verecek herhangi bir komut (INSERT, UPDATE, DELETE) uygulandığı zaman, o tablo üzerinde otomatik olarak EXCLUSIVE kilitleme oluşturur. Bu kitleme herhangi bir COMMIT ya da ROLLBACK komutu ile ortadan kalkacaktır.

ÖRNEKLER:

```
LOCK TABLE X IN SHARE MODE;
```

X tablosu SHARE MODE'da kilitleniyor. Diğer kullanıcılar sadece okuma yapabilir.

```
LOCK TABLE Y IN EXCLUSIVE MODE;
```

Y tablosu EXCLUSIVE MODE'da kilitleniyor. Diğer kullanıcılar sadece okuma yapabilir ve Y tablosu üzerinde kitleme yapamazlar.