VERİTABANI SİSTEMLERİ 2

1-TEMEL KAVRAMLAR 2
1.1 Veri Nedir? 2
1.2 Genel Kavramlar 2
1.3 Veri Tabanı Nedir? 4
1.4 Veri Erişim Teknolojileri 7
2-VERİ VE VERİ MODELLERİ 8
2.1 Model Nedir? 8
2.2 Veri Modeli 8
2.3 Başlıca Veri Modelleri 9
3-VERİTABANI TÜRLERİ 13
3.1 Yerel Veritabanları 13
3.2 İstemci/Sunucu Veritabanları 13
3.3 Tek-Katmanlı, İki-Katmanlı ve Çok-Katmanlı Veritabanları 13
4- VERİ TABANI NESNELERİ 14
4.1 Tablolar 14
4.2 Görünümler 14
4.3 Eşanlam 14
4.4 İndeksler 15
4.5 Sıralar 15
4.6 Veri Tipleri 15
4.7 Varsayılanlar 15
4.8 Kurallar 15
4.9 Saklı Prosedürler 16
4.10 Tetikleyiciler 16
4.11 Sorgular 16
4.12 Joining (İlişkilendirme) 16
5- VERİ TABANI TASARIMI VE NORMALİZASYON 16
5.1 Veri Tabanı Tasarımı 16
5.2 Veri Tabanı Normalizasyonu 18
6-STRUCTURED QUERY LANGUAGE (SQL) 19
6.1 Veri İşleme Dili (Data Manipulation Language – DML) 23
6.2 Veri Tanımlama Dili (Data Definition Language – DDL) 49
6.3 Veri Kontrol Dili (Data Control Language – DCL) 63
6.4 Tabloların Bağlanması 66

VERİTABANI SİSTEMLERİ

1-TEMEL KAVRAMLAR

1.1 Veri Nedir?

Veri (Data) ve Bilgi (Information) terimleri sık sık birbirinin yerine kullanılmaktadır. *Veri* kavramı kimi yerde olayları veya yerleri, kimi yerde insanları veya diğer nesneleri ilgilendiren gerçeklerdir. Dolayısı ile veri kavramı her alan için farklılıklar gösterebilir. Biz bilgisayar ortamında veri kavramını yerine göre, bilgisayarın bir manyetik disk üzerinde (örn. Hard disk), yarı iletkenden oluşmuş bir hafıza biriminde (örn. RAM) veya bir veritabanı içerisinde işlenebilecek durumda bulundurulan kayıtlar olarak kabul edeceğiz. Veri çeşitli şekillerde işlenmeye veya tasfiye edilmeye hazır durumda olan fakat ilk bakışta faydasız ve anlamsız gibi görünen bir sürü kaydı gösterir.

Bilgi kavramı işlenmiş ve kullanıcı için yararlı ve kullanışlı olan verilere denmektedir. Örneğin öğrenci bilgilerinin tutulduğu bir veritabanı düşünelim. Bu veritabanından "Bilgisayar Teknolojisi ve Programlama" bölümü öğrencileri içinden "Veri Tabanı Yönetim Sistemleri" dersinden bütünlemeye kalan öğrencilerin listesi, veritabanları dersini veren ders hocası için çok kullanışlı ve önemli bir bilgiyi göstermektedir.

1.2 Genel Kavramlar

Varlık

Varlık (Entity), veritabanımızda bilgilerini tutmak istediğimiz şeyler, nesneler, yerler, olaylar veya kişilerdir. Varlık bir ürün, bir bilgisayar veya bir müşteri gibi somut bir nesne olabileceği gibi banka hesapları, öğrenci notları veya uçak tarifeleri gibi soyut bilgilerde olabilir.

Varlık Sınıfı

Varlık sınıfı ise benzer karakteristiklere sahip varlıkların bir araya gelerek oluşturduğu bir kümedir. Bu kümelere örnek olarak Müşteriler, Öğrenciler, Hastalar verilebilir. Varlık sınıfı yerine bazen Varlık kümeleri veya Varlık Tipleri gibi ifadeler de kullanılmaktadır.

Öznitelik

Bir öznitelik (attribute) kayıt için belirlenen bir varlık özelliğidir. Örneğin Öğrenciler isimli bir varlık sınıfını göz önüne alalım. Bu sınıfın öznitelikleri Şekil:1.1 deki gibidir.

Şekil: 1.1

Öznitelik

Öğrenciler ⟶ Varlık sınıfı

Öğrenci No	00118001
Adı	M.Fatih
Adresi	Alacatlı M. Adilcevaz
Telefon No.	0434 654 1234
Doğum Yeri	Manisa
Cinsiyeti	Erkek

Varlık

Her bir varlık sınıfı için 1, 5, 10 veya daha fazla öznitelik tanımlanabilir. Bir varlık sınıfındaki her bir varlık, kendisini sınıftaki diğer varlıklardan ayıran en az bir özniteliğe sahip olmalıdır. Tek olan bir varlık, ID (Identifier) olarak çağrılır. Bu o varlığa ait tek olan bir kimlik bilgisi veya numarasıdır. Örneğin, bir işçiye ait sosyal güvenlik numarası veya bir öğrenciye ait okul numarası tektir.

Metadata

Metadata, bir veritabanı içerisindeki verilerin kendisi olmayıp bu veriler hakkındaki bilgilerdir. Örneğin, tablolardaki alanlara ait özellikler veya indeksler gibi bilgiler. Bu bilgiler veritabanı yöneticisi tarafından veya bu sistem üzerinde geliştirme çabalarını yürütecek kişiler tarafından kullanılır. Metadata bir organizasyonun veri sözlüğünde veya klasöründe saklanır. Veritabanı içerisinde saklanmaz.

Entity Sınıfı	Kayıt Tanımları	Kayıt
Öznitelikler (Attributes)	Veri Öğesi tanımları	Veri öğesi

Şekil 1.2

Gerçek dünyadaki her varlık sınıfı için metadata dünyasında bir kayıt tipi tanımlanır. Her bir attribute özelliğine karşılık ise metadata dünyasında bir veri öğesi tanımlanır.

Veri Öğesi (Data Item)

Veri Öğesi bir birime karşılık gelir. Veritabanı içerisindeki en küçük veri birimine karşılık gelen isimdir. Örneğin şekil 1.1 de verilen Öğrenciler isimli varlık sınıfı için belirtilecek Öğrenci_no, Dogum_tarihi, Bolum ifadelerinin her biri birer veri öğesine karşılık gelir.

Veri Öğesi birçok yerde veri elementi, alan veya öznitelik olarak çağırılmaktadır. Biz veri öğesi için daha çok “Alan" terimini kullanacağız.

Veri Kümeleme

Veri Kümeleme, tek bir isim ile anılan veri öğeleri koleksiyonudur. Bu daha çok Delphi de kullanılan ADT (Abstract Data Type) tipindeki alanlara benzemektedir. Örneğin KİMLİK ile ifade edilen bir veri kümeleme AD ve Soyad gibi iki veri öğesinden oluşabilir. Yine adres ile çağrılan bir veri kümeleme, CADDE, SEHIR, ULKE, POSTAKODU gibi veri öğelerini kapsayabilir.

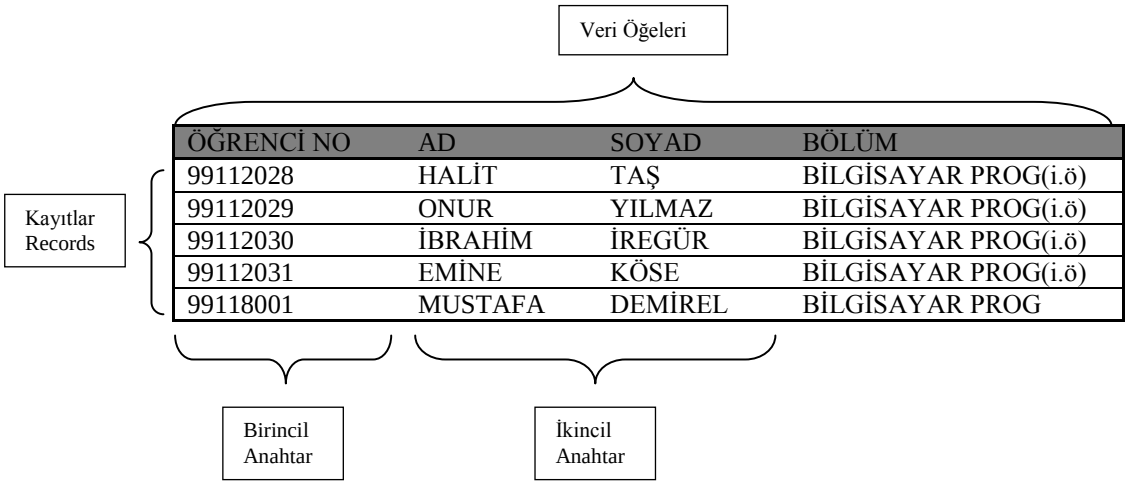
Kayıt

Kayıt ise veri kümeleme veya veri öğelerinin bir isim altında toplanmasına denir. Her varlık sınıfı için mutlaka bir kayıt tipi tanımlanır. Her bir kayıt için metadata tanımlanır ve veri sözlüğünde kataloğu hazırlanır. Bir kayıt için metadata; kayıt ismi, tanımı, uzunluğu, veri öğeleri veya veri kümeleme ile birincil ve ikincil anahtarlar şeklindeki bilgileri içerir.

Anahtar (Key)

Anahtar, bir kaydı tanımlamak için kullanılan veri öğesidir. E-R Modelini açıklarken genel olarak iki tür anahtar kullanılmaktadır: Süper Anahtar (superkey) ve Aday Anahtar (candidate key). Süper anahtarlar bir varlığı tek olarak tanımlar ve bir veya birden fazla veri öğesinden oluşabilir. Süper anahtar olmayan ancak anahtar olabilecek durumdaki veri öğelerine aday anahtar denir. Örneğin "Ad", "Okulno", "tlf" gibi veri öğeleri birer aday anahtardır. Ancak "ad" veri öğesi birincil anahtar için iyi bir seçim değildir. Çünkü ad bilgisi birçok kişide aynı olabileceğinden varlığı tek olarak tanımlamaz.

Anahtarlar ile iki varlık arasında bir ilişki kurulabilmektedir. İlişki kurmak için iki anahtar tipi kullanılmaktadır: birincil(primary) ve ikincil (secondary) anahtar.



ÖĞRENCİ NO	AD	SOYAD	BÖLÜM
99112028	HALİT	TAŞ	BİLGİSAYAR PROG(i.ö)
99112029	ONUR	YILMAZ	BİLGİSAYAR PROG(i.ö)
99112030	İBRAHİM	İREGÜR	BİLGİSAYAR PROG(i.ö)
99112031	EMİNE	KÖSE	BİLGİSAYAR PROG(i.ö)
99118001	MUSTAFA	DEMİREL	BİLGİSAYAR PROG

Şekil 1.3

Birincil (primary) anahtar, bir kaydı tek olarak tanımlamak için kullanılan bir veri öğesidir. Örneğin bir eğitim kurumundaki öğrenci kaydı için ogrenci_no bilgisi tek olacağından bu öğe bir birincil anahtar yapılabilir.

İkincil (secondary) anahtar, normalde bir kaydı tek olarak tanımlamaz fakat aynı özellikleri taşıyan bir kaydın numarasını veya genelde ilişkilerde kullanılan kayıt numarasını tanımlar. Ana/Ayrıntı yapısındaki bir ilişkinin Ayrıntı kısmındaki anahtara Yabancı Anahtar da denmektedir.

1.3 Veri Tabanı Nedir?

Veri Tabanı

Veritabanı, birbiriyle ilişkisi olan verilerin tutulduğu, kullanım amacına uygun olarak düzenlenmiş veriler topluluğunun mantıksal ve fiziksel olarak tanımlarının olduğu bilgi depolarıdır. Veritabanları gerçekte var olan ve birbirleriyle ilişkileri olan nesneleri ve ilişkilerini modeller.

Veri tabanı; banka üniversite, okul, seyahat şirketi, hastane, devlet dairesi gibi bir kuruluşun çalışıp işleyebilmesi için gereken uygulama programlarının kullandığı operasyonel çok çeşitli verilerin toplamıdır. Ticari bir şirket için müşteri bilgileri, satış bilgileri, ürün bilgileri, ödeme bilgileri vb. Okul için öğrenci bilgileri, açılan dersler, okula kaydedilmiş öğrenciler, öğretmen bilgileri vb. Hastane için hasta bilgileri, doktor bilgileri, yatakların doluluk ve boşluk bilgileri, teşhis-tedavi bilgileri, mali bilgiler vb. Kullanılan çok çeşitli operasyonel verilere örnek olarak gösterilebilir.

Belirli bir konu hakkında toplanmış veriler bir veritabanı programı altında toplanırlar. Bu verilerden istenildiğinde; toplanılan bilgilerin tümü veya istenilen özelliklere uyanları görüntülenebilir, yazdırılabilir ve hatta bu bilgilerden yeni bilgiler üretilerek bunlar çeşitli amaçlarla kullanılabilir. Veri tabanı ile ilgili bazı tanımlamalar listelenmiştir.

- **Veri tabanı sistemi;** veri tabanlarını kurmayı, yaratmayı, tanımlamayı, işletmeyi ve kullanmayı sağlayan programlar topluluğudur. Veri Tabanı Yönetim Sistemi ismi ile de anılırlar.
- **Veri tabanının tanımlanması;** veritabanını oluşturan verilerin tip ve uzunluklarının belirlenmesidir.
- **Veri tabanının oluşturulması;** veri için yer belirlemesi ve saklama ortamına verilerin yüklenmesini ifade eder.
- **Veri tabanı üzerinde işlem yapmak;** belirli bir veri üzerinde sorgulama yapmak, meydana gelen değişiklikleri yansıtmak için veri tabanının güncellenmesi ve rapor üretilmesi gibi işleri temsil eder.
- **Verinin bakım ve sürekliliği;** veri tabanına yeni kayıt eklemek, eskileri çağırmak ve gerekli düzenleme, düzeltme ve silme işlemlerini yapma gibi işlemlerin gerçekleştirilmesini ifade eder. Veri tabanı yönetim sistemi aynı zamanda verinin geri çağırılabilmesini de sağlar.
- **Veri tabanını genişletme;** kayıtlara yeni veri eklemek ve yeni kayıtlar oluşturmak gibi işlemleri ifade eder.

Bir veritabanından beklenen özellikler, verilerin korunması, onlara erişilmesini sağlaması ve başka verilerle ilişkilendirilmesi gibi temel işlemleri yapabilmesidir. Veritabanı sayesinde veriler bir merkezde toplanarak herkesin bu verilere yetkileri ölçüsünde erişmesi, düzeltmesi, silmesi veya görebilmesi sağlanabilir. Böylece veri girişinde ve veriye erişimde etkinlik ve güvenirlilik sağlanır.

Neden Veritabanı Kullanılır?

Bilgisayar ortamında verilerin tutulması, saklanması ve erişilmesinde günümüze kadar değişik yöntem ve yaklaşımlar kullanılmıştır. Bu yaklaşımlardan biri olan Geleneksel Yaklaşım verilerin ayrı ayrı dosyalarda gruplanma yaklaşımını kullanmaktadır. Veritabanı programlarından önce verileri saklamak için programlama dillerinde sıralı ve rastgele dosyalama sistemleri kullanılırdı. Bu sistem; birbiriyle ilgili olan ve aynı gruba dâhil olan verilerin bir dosyada tutulması yöntemine dayanmaktadır. Verilerin artması, verilere aynı anda erişilmesi ve aynı anda erişilen verilerin erişenlere göre düzenlenmesi gibi ihtiyaçlar arttıkça geleneksel yaklaşım yetersiz kalmıştır. Veri tabanı yaklaşımının yararları aşağıda sıralanmıştır.

- Ortak verilerin tekrarının önlenmesi; verilerin merkezi denetiminin ve tutarlılığının sağlanması
- Veri paylaşımının sağlanması
- Fiziksel yapı ve erişim yöntemi karmaşıklıklarının, çok katmanlı mimarilerle kullanıcılardan gizlenmesi
- Her kullanıcıya yalnız ilgilendiği verilerin, alışık olduğu kolay, anlaşılır yapılarda sunulması
- Sunulan çözümleme, tasarım ve geliştirme araçları ile uygulama yazılımı geliştirmenin kolaylaşması
- Veri bütünlüğü için gerekli olanakların sağlanması, mekanizmaların kurulması
- Güvenlik ve gizliliğin istenilen düzeyde sağlanması

- Yedekleme, yeniden başlatma, onarma gibi işletim sorunlarına çözüm getirilmesi

Veri Tabanı Yönetim Sistemi Nedir?

Veritabanı Yönetim Sistemi (VTYS), yeni bir veritabanı oluşturmak, veritabanını düzenlemek, geliştirmek ve bakımını yapmak gibi çeşitli karmaşık işlemlerin gerçekleştirildiği birden fazla programdan oluşmuş bir yazılım sistemidir. Veritabanı yönetim sistemi, kullanıcı ile veritabanı arasında bir arabirim oluşturur ve veritabanına her türlü erişimi sağlar. Veritabanı yönetim sistemi programları; fiziksel hafızayı ve veri tiplerini kullanıcılar adına şekillendirip denetleyen ve kullanıcılarına standart bir SQL ara yüzü sağlayarak onların dosya yapıları, veri yapısı, fiziksel hafıza gibi sorunlarla ilgilenmek yerine veri giriş-çıkışı için uygun ara yüzler geliştirmelerine olanak sağlayan yazılımlardır. VTYS'de kullanıcılar, roller ve gruplar vardır ve bunlar verileri tutmak üzere birçok türde nesne ve bu nesnelere erişimleri düzenleme görevi yaparlar. Her bir kullanıcının veritabanı yöneticisi tarafından yapılan tanımlanmış belirli hakları vardır. Bu haklar verilebilir, verilmiş haklar arttırılabilir, kısıtlanabilir veya silinebilir.

Veri Tabanı Yönetim Sisteminin Sağladığı Yararlar

- Veri tek bir merkezde tutulur ve aynı veri her kullanılan değişik bilgisayarda tekrar tekrar tutulmaz.
- Eklenen, düzenlenen veya silinen bir bilgi merkezde değil de kişisel çalışılan bilgisayarlarda yapılırsa bu bilgi ile merkezdeki farklı olacaklarından veri tutarsızlığı oluşur. Aynı verinin değişik yerlerde birkaç kopyasının bulunması aynı zamanda bakım zorluğunu da beraberinde getirir.
- Veritabanı Yönetim Sistemi kullanılmayan sistemlerde veriye erişim sıralı yapılır ve birden çok kullanıcı aynı anda aynı veriye erişemezler. Erişmek istediklerinde ilk istekte bulunan erişir ve diğerleri onun işleminin bitmesini bekler. Hazırlanan programda sadece okumak amacıyla tüm kullanıcılara erişme hakkı verilebilse de hepsi işlem yapmak istediklerinde ilk kullanıcının işlemini yapmadan diğerlerine işlem yapma izni vermez. Bir VTYS'de ise verinin tutarlılığını ve bütünlüğünü bozmadan aynı veritabanlarına saniyede yüzlerce, binlerce erişim yapılabilir.
- Bir tabloda bir işlem yapıldığında buna bağlı diğer tablolarda da ilgili işlem yapılır. Örneğin bir okuldaki bir öğrencinin kaydı silinirse, öğrencinin not, harç ve diğer bilgileri de diğer tüm tablolardan silinmelidir.
- Verilerin kasıtlı veya yanlış kullanım sonucu bozulmasını önlemek için verilerin korunabildiği özellikler bulunur. Örneğin; veri tabanına girmek için kullanıcı adı ve şifre istenmesi, kişilerin sadece kendilerine verilen haklar ölçüsünde tablolar ya da tablo içinde belirli kolonlarla işlem yapabilmeleri gibi.
- Programcı, kullandığı verilerin yapısı, organizasyonu ve yönetimi ile ilgilenmeden veritabanının bunları kendinin koordine etmesi ve yönetmesidir. Veri bağımsızlığı, veritabanı yönetim sistemi programlarının en temel amaç ve özelliklerindendir.

1.4 Veri Erişim Teknolojileri

Verilere erişim için temel bazı teknolojiler kullanılmaktadır.

ODBC (Open DataBase Connectivity)

Açılımı açık veritabanı bağlanabilirliği olan ODBC, Microsoft'un veritabanlarına erişim için kullandığı ilk standartlardan biridir. Veritabanı uygulamaları geliştiren programcılar, bir işletim sistemi üzerinde veritabanı bağlantısını sağlamak üzere kendi veritabanı bağlantı modüllerini, daha bilinen bir ifade ile veritabanı bağlantı sürücülerini yazmaları gerekmektedir. Microsoft bu standart ile programcıları bu gibi sürücü yazılımları yazmaktan kurtarmak istemiştir. Bugün bildiğimiz meşhur DBMS'ler ODBC desteği sunmaktadır.

ODBC başarılı bir veri erişim teknolojisi idi, ancak programlaması çok zordu. Bu nedenle ODBC birçok revizyon geçirmiş ve Microsoft bir sonraki teknolojide nesne tabanlı fonksiyonlar üzerinde çalışmayı tercih etmiştir.

DAO (Data Access Object)

Microsoft'un ilk nesne tabanlı API fonksiyonları sunan teknolojisidir. Nesne modeline dayanan DAO, Microsoft Access 97 ile birlikte kullanılıyordu. DAO'yu halen bir Microsoft Access programı içinden kullanabilirsiniz. Eğer Microsoft Jet veritabanları ile çalışıyorsanız DAO iyi bir seçimdir. Çünkü bu veritabanları üzerinde DAO üstün bir performans sergiler.

RDO ve ODBCDirect

DAO nesneleri daha çok yerel veri kaynaklarına erişim için kullanılıyordu ve uzaktan erişimli ODBC veri kaynakları ile kullanılamıyordu. Bu nedenle Microsoft, Visual Basic program geliştiricileri için Uzaktan Erişimli Veri Nesneleri olarak ifade edebileceğimiz (Remote Data Object) RDO'yu bu sisteme ekledi. Ayrıca Access veritabanı geliştiricileri için ise ODBC veri kaynaklarına hızlı ve doğrudan erişim sağlayabilen ODBCDirect nesne modelini ekledi.

ADO (ActiveX Data Object)

İstemci uygulamaların veritabanı sunucusundaki veriye bir OLE DB sağlayıcısı aracılığıyla erişmesini sağlayan teknolojidir. Yüksek hızı, etkili, kolay ve esnek kullanımı ile Web tabanlı ve İstemci/Sunucu uygulamalarda tercih edilmektedir. ADO, veriye erişim için Microsoft'un COM mimarisini kullanmaktadır. Ancak COM mimarisi, ADO'nun Firewall sistemler üzerinden veriye erişmesinde problem çıkarmaktadır. Çünkü çok katmanlı uygulamalardan verinin katmanlardan iletimi için COM Marshalling gerekmektedir. Firewall yapısı da zaten sisteme erişmeye çalışan sistem seviyesindeki fonksiyonlara karşı tasarlanmıştı. Bu durum ADO kayıt kümelerinin Internet ortamında seyahatine engel teşkil etmektedir.

DAO nesne modeline dayanan ADO, Access 2000 sürümünden itibaren yeni standart olarak kullanılmaktadır. DAO, Jet veritabanlarında hala ADO'dan daha üstündür. Bu veritabanlarında DAO'yu kullanabilirsiniz. Ancak SQL Server veya Oracle gibi uzaktan erişim sağlayabilen veritabanlarına bağlamak için ADO'yu kullanın.

ADO.NET

Microsoft'un çıkardığı en son veri erişim teknolojisi ADO.NET'tir. ADO.NET, ADO'nun devamı veya geliştirilmiş bir üst sürümü değildir. ADO.NET, ADO'nun kullandığı COM mimarisi yerine .NET Framework ve XML tabanlı bir sistemi kullanmaktadır. Bu

mimari ile ADO.NET, Firewall sistemleri üzerinden veri iletimi sağlayabilmektedir. Çünkü Firewall sistemleri metin biçimindeki verilerin iletimine izin vermektedir ve ADO.NET'te verileri metin tabanlı XML biçiminde sunmaktadır.

BDE (Borland Database Engine)

Borland'ın kullandığı veri erişim teknolojisidir. Nesne yönelimli yapıya sahiptir ve BDE API olarak bilinen yaklaşık 200 kadar fonksiyon veya prosedür içermektedir. BDE kullanarak tasarladığınız veritabanı projelerinizi başka bir bilgisayara kurarken BDE kurulumunu da taşımanız ve bu kurulumu yeniden yapılandırmanız gerekmektedir. Sistem gereksinimleri ve harcadığı bellek miktarı ile hantal bir sistem olarak kalmıştır. Şu anda da Borland firması BDE üzerindeki geliştirme çalışmalarına son vermiştir.

2-VERİ VE VERİ MODELLERİ

2.1 Model Nedir?

Model kelimesi; isim, sıfat ve fiil olarak kullanılmaktadır. İsim olarak model, bir mimarın, bir binanın küçük ölçekli modelini oluşturması gibi temsili ifade eder. Sıfat olarak model, "model uçak", "model öğrenci" ifadelerinde olduğu gibi mükemmeliyetin veya idealin ölçüsünü ifade eder. Fiil olarak model ise, bir şeyin nasıl olduğunu ispat etmek, açıklamak, göstermek anlamındadır.

Bilimde Simgesel Model, Benzetim Modeli ve Sembolik Model kavramları vardır.

Simgesel Modeller, durumların büyük veya küçük ölçekli temsilleridir. Gerçek şeylerin uygun özelliklerini temsil ederler. Şekilleri, temsil ettikleri şeylere benzerler. Yol haritaları, hava fotoğrafları bu tip modellere örnek verilebilir.

Benzetim Modelleri, bazı durumlarda ise; haritada yükseltiler, yol genişlikleri gibi özellikler belirtmek gerekebilir. O zaman, renkler ve kontur çizgileri gibi bir takım açıklayıcı özelliklere ihtiyaç duyulur. Bu tip modeller Benzetim Modelleri olarak isimlendirilir.

Sembolik Modellerde, temsil edilen şeylerin özellikleri sembollerle ifade edilir. Böylece, bir grafik ile gösterilen ilişki, bir eşitlik olarak da ifade edilebilir. Bu tip modellere Matematiksel Modeller de denilmektedir.

Bu üç tip modelden benzetim modeli, soyut ve geneldir. Matematiksel model ise en soyut ve en genel modeldir. Üzerinde düzenleme yapılabilmesi daha kolaydır. Simgesel modellerin ise anlaşılması diğerlerine göre daha kolaydır. Bilişim sistemlerinin oluşturulması için kullanılan veri modelleri, benzetim modelleri ve sembolik modellerdir.

Bir bilişim sistemi kullanıcısı, özellikle bir karar verici, kendisini sonsuz denebilecek kadar bilgi karşısında bulur. Bir bilişim sistemi modeli, gerçek bilgi kümesinin alt kümesini oluşturur ve onun daha basit bir şeklidir. Bu şekil, işlenebilmeye imkân verir ve bunu kullanarak elde edilen çözüm veya cevap, gerçek hayatta uygulanmaya çalışılır. Model, var olan bilgi yığınına bir düzen getirmeyi, hatta bir yapı oluşturmayı amaçlar. Tek bir model yoktur. Var olan bilgi yığınına, uygulanan farklı modeller doğal olarak farklı yorumlar getirir.

2.2 Veri Modeli

Bir veri modeli, verinin hangi kurallara göre yapılandırıldığını belirler.

2.2.1 Yapılar

Soyutlama, küme ve ilişki veri yapılarının temel unsurlarıdır. Detayları gizleme ve genel üzerinde yoğunlaşma yeteneği olan soyutlama veriyi yapılandırma ve görüntüleme işlemini yapar ve veri kategorilerini elde etmek için kullanılır. Yapılar düzgün şekilde tanımlanmış veri gruplarıdır. Kendiside aynı zamanda bir küme olan ilişki, iki nesne arasındaki ilişkiyi

gösteren bir tip olarak kümelerin toplanmasını ifade eder. Örneğin, ÖĞRENCİ ve OKUL arasında bir NOT ilişkisi vardır.
Veri yapısı oluşturulurken, verideki nesneler ve onlar arasındaki ilişkiler tablo ile temsil edilir. Veri tabanlarında uygulanabilecek genel kayıt ilişkilendirme tipleri şu şekilde sıralanabilir:
Bire bir ilişkiler(one to one relationships): Aralarında bir ilişki olan iki tablo arasında, tablolardan birindeki asıl anahtar alanın kayıt değerinin, diğer tablodaki sadece bir kayıtta karşılığının olması durumunu gösteren ilişki tipi. **Örnek:** bir öğrencinin doğum yeri bilgisinin doğum yerleri tablosundaki bir şehre karşılık gelmesi gibi.
Tekil çoklu ilişkiler (one to many relationships): Aralarında bir ilişki olan iki tablo arasında, asıl anahtar alanın kayıt değerinin, diğer tablodaki birden fazla kayıtta karşılığının olması durumunu gösteren ilişki tipi. **Örnek:** Bir öğrencinin birden fazla almış olduğu derse ve bu derse ait vize, final ve sınav sonuçları gibi. Bir öğrenciye karşılık birden fazla ders notu.
Çoğul tekli ilişkiler (many to one relations): Aralarında bir ilişki olan iki tablo arasında, tablolardan birindeki bir kaydın değerinin, asıl anahtar alanın olduğu diğer tabloda, birden fazla kayıtta karşılığının olması durumunu gösteren ilişki tipi.
Çoklu ilişkiler(many to many relations): Aralarında bir ilişki olan iki tablo arasında, tablolardan herhangi birindeki herhangi bir kaydın, diğer tablodaki birden fazla kayıt ile ilişkilendirilebildiği ilişki tipi.

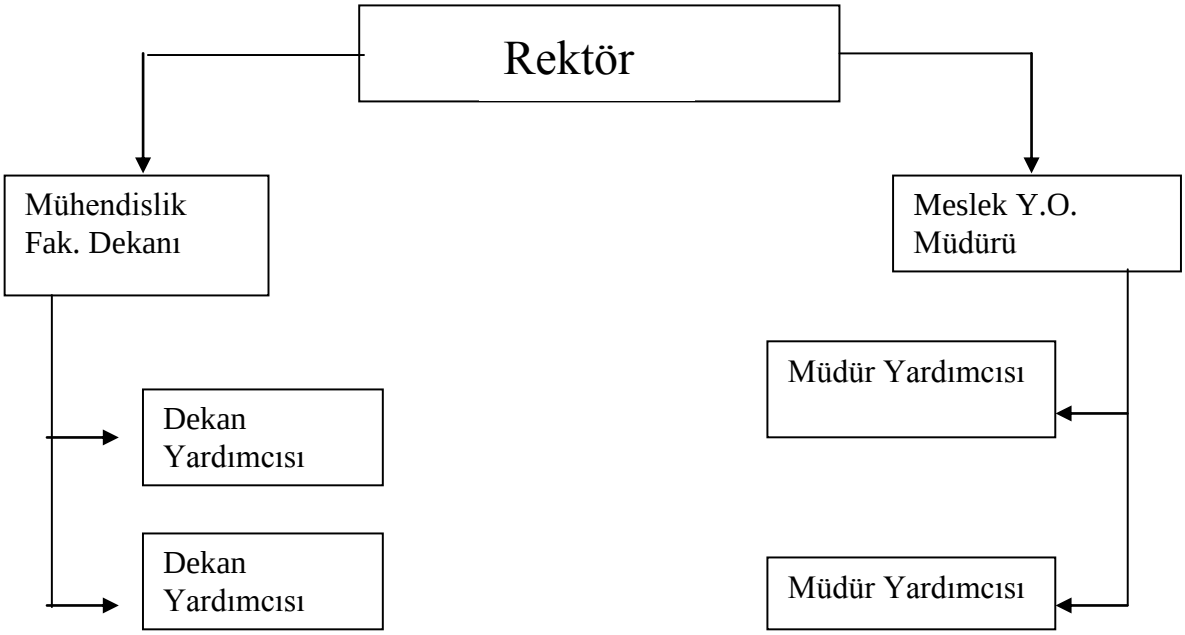
2.2.2. Kısıtlar
Veriler üzerindeki sınırlamalara kısıt adı verilir. Kısıtlar, veri modellerinde bütünlük sağlamak için kullanılır. Örneğin, öğrenciye ait not bilgisinin 0-100 arasında olması gibi.

2.2.3 İşlemler
İşlemler, bir veri tabanı durumundan, bir başka veri tabanı durumu elde etmek için yapılan işlemlerdir. Bunlar, verinin çağrılması, güncellenmesi, eklenmesi veya silinmesi ile ilgili işlemlerdir. Bütünlük mekanizması, toplam fonksiyonları, veriye ulaşım kontrolleri gibi genel işlemler vardır. CODASYL tarafından yayınlanan bu mekanizmalara veri tabanı yöntemleri denir.

2.3 Başlıca Veri Modelleri
Veri modellemesi yapmak amacıyla farklı durumlara uygun olan ve birbiriyle farklı özellikler taşıyan pek çok veri modeli vardır. Veri modelleri aşağıdaki gibi sınıflandırılabilir.

2.3.1. Basit Veri Modelleri
Bilgisayarlarda veri işleme ihtiyacının ortaya çıkmasıyla, dosyalama sistemleri oluşturmak amacıyla kullanılmaya başlanan Hiyerarşik ve Şebeke veri modelleridir.

2.3.1.1 Hiyerarşik Veri Modelleri
Çoklu ilişkileri temsil edebilmek için, varlık tiplerinin gereksiz veri tekrarı yapmadan her ilişki için ayrı ayrı tanımlanmasına Hiyerarşik veri modeli denilir. Bu model, bir ağaç yapısına benzer. Model içerisindeki herhangi bir düğüm, altındaki n sayıda düğüme bağlanırken, kendisinin üstünde ancak bir düğüme bağlanabilir. Hiyerarşik yapının en tepesindeki düğüm noktasına kök denir ve bu düğümün sadece bağımlı düğümleri bulunur. Bu veri yapısını gösteren grafiğe de hiyerarşik tanım ağacı denir.

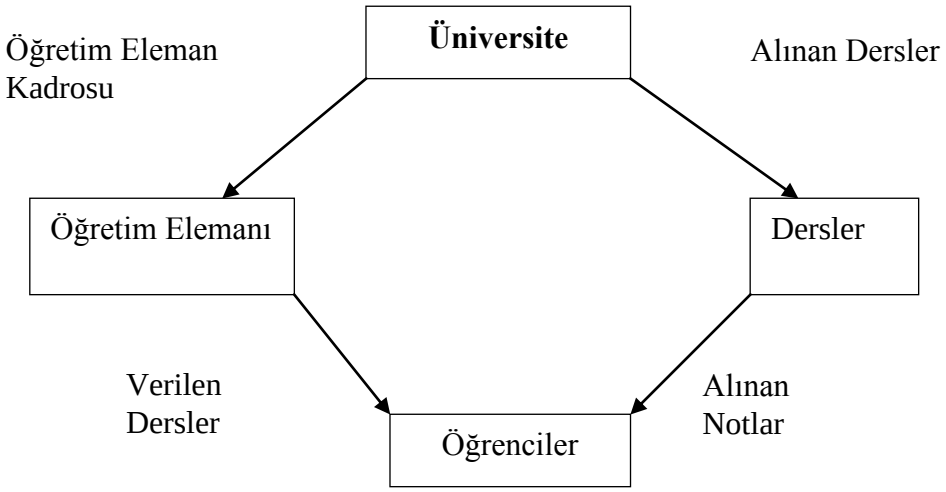


Hiyerarşik Tanım Ağacı

2.3.1.2. Şebeke Veri Modelleri

Tablo ve grafiklerden oluşan veri modeli Şebeke Veri Modelidir. Tablolar grafikteki düğümler olup varlık tiplerine karşılık gelirler. Grafiğin okları ise ilişkileri temsil eder. Özellikleri, 1971 yılında DBTG-CODASLY tarafından belirlenmiştir.

Kayıt tipi ve bağlantı olmak üzere iki ayrı veri yapılandırma aracı vardır. Kayıt tipleri varlık tiplerini, bağlantılar ise ilişki tiplerini belirler. Bu yapıyı gösteren grafiğe de veri yapısı grafiği adı verilir.



Şebeke Veri Yapısı Grafiği

Şebeke içinde bir eleman, herhangi başka bir elemana bağlanabilir. Hiyerarşik yapılardan farklı olarak, şebeke yapılarında bağlantı açısından herhangi bir sınırlama yoktur. Şebeke veri modelleri, düğümler arasında çoklu ilişkiler kurulamadığı için, kısıtlı bir veri modeli olarak kabul edilir. Hiyerarşik veri modelleri ise, daha da kısıtlı bir veri modelidir. Şebeke veri modelinde kullanılan işlemler, ilişkisel veri modelinde kullanılan işlemlerin benzeridir. Fakat şebeke veri modellerinde bağlantılar tarafından belirlenmiş ilişkiler dışında, kayıt tipleri arasında ilişki belirlenmez.

2.3.2 Geliştirilmiş Veri Modelleri

1960 ve 1970'li yıllarda hiyerarşik veri modeli üzerine geliştirilmiş VTYS, sonra şebeke veri modeli ile çalışan VTYS yaygın kullanım alanı bulmuştur. 1970'li yıllarda gelişmesini tamamlamış olan ilişkisel veri modeline dayalı VTYS 1980'li yıllarda ticari kullanıma girerek çok hızla yaygınlaşmaya başlamıştır.

Geliştirilmiş veri modelleri, Varlık-İlişki Veri Modelleri, İlişkisel Veri Modelleri ve Nesne Yönelimli Veri Modelleri olarak sıralanabilir.

2.3.2.1 Varlık-İlişki Veri Modelleri (Vİ Modeli)

Bir veri tabanı uygulamasında hakkında tanımlayıcı bilgi saklanabilen her şey varlık olarak kabul edilir. Varlık, bağımsızdır ve tek başına tanımlanabilir. Bir varlık, ev, öğrenci, araba gibi bir nesne ya da futbol maçı, tatil, satış gibi olaylar olabilir. En anlamlı şekilde kendi öznitelikleri tarafından temsil edilir. Örneğin, bir EV; özniteliğin kendisi tanımlayıcı bilgi içeriyorsa, onu varlık olarak tanımlamak gerekir. Örneğin, eğer evin malzemesi hakkında ek bilgi depolamak gerekiyorsa MALZEME' yi de varlık olarak sınıflamak gerekir.

Varlık-İlişki veri modelleri (Vİ), varlıklar ve aralarındaki ilişkiyi oklarla göstermek için kullanılan grafikler üzerine kurulmuş veri modelleridir. Tablo sütunları öznitelikleri temsil eden değişkenleri, satırlar ise özniteliklere ait verileri temsil eder.

Ticari veri tabanlarında yaygın olarak kullanılan veri modellerinden biridir. Şebeke ve hiyerarşik veri modelleri ile ortak noktaları vardır. Fakat veritabanı tasarım süreleri için kullanılmak maksadıyla geliştirildiklerinden bu iki modelin genelleştirilmiş şeklidir. Çoklu ilişki tiplerinin doğrudan modelde kullanılmasına izin verir. Bu modelde, kurum şeması kavramı söz konusudur. Bu şema, kurumun tüm verisinin görünümünü temsil ve fiziksel sınırlamalardan bağımsızdır. Temelde, Vİ veri modeli, veri tabanının mantıksal özelliklerinin bir dokümantasyonudur. Vİ modeline göre düzenlenen eri tabanının yapısı, **Varlık-İlişki Diyagramı** ile gösterilir.

Şebeke ve hiyerarşik veri modellerinde, sadece ikili fonksiyonel bağlantılara izin verilmektedir. Vİ veri modelinde ise, varlıklar arasında n adet ilişki tanımlanabilir. Bu ilişkiler, bire bir, fonksiyonel ve çoklu olabilir. Tekrar eden bağlantılar da kullanılabilir.

Vİ modeli 1976 yılında ilk olarak ortaya konulduğunda bir veri dili geliştirilmemişti ve CABLE (ChAin-Based LanguagE) dili geliştirilene kadar bilgi sorgulamaları küme işlemleri ile yapılıyordu. Vİ modellerinde en büyük avantajlarından biri, uzman olmayan kişiler tarafından da anlaşılabilecek yapıda olması ve üzerinde düzeltme işlemlerinin kolayca yapılabilmesidir. Bu açıdan belirli bir veri tabanı yönetim sistemine bağlı değiştir.

2.3.2.2. İlişkisel Veri Modelleri

İlişkiler ve onların temsilleri olan tablolardan oluşan ilişkisel veri modeli ilk olarak 1970 yılında Codd tarafından ortaya atılmıştır. İlişkisel veri modellerinde kullanılan tek yapılandırma aracı ilişkidir. Aşağıdaki örneklerde büyük harflerle yazılan ifadeler ilişki isimlerini, parantez içindeki ifadeler de tanım kümesi isimlerini göstermektedir.

UNIVERSITE (UniversiteKodu, UniversiteAdi, Adres)
OGRETIMELEMANI (OgrKodu, OgrAdi, OgrSoyadi, Bolum, Brans)
OGRENCI (OgrenciNo, OgrenciAdi, OgrenciSoyadi, Bolum, Adres)
DERSLER (DersKodu, DersinAdi, Kredisi, Yariyil)
OGRENCINOTLAR (OgrenciNo, DersKodu, Vize, Final)

Yukarıdaki satırlar, basit bir üniversite veri tabanının ilişkisel şemasını göstermektedir. İlişkisel şema, ilişki isimlerinin ve karşılık gelen tanım kümesi isimlerinin listesidir. Varlık tiplerini belirtmekte kullanılır.

Universite

UniversiteKodu	UniversiteAdi	Adres
3300	Mersin Üniversitesi	Mersin
0600	ODTU	Ankara
3400	Yıldız Üniversitesi	İstanbul

İlişkisel şema listesini oluşturan her bir satır, bir tablo olarak temsil edilir. Tablonun sütunları öznitelik olarak isimlendirilir. Örneğin, Universite tablosunun öznitelikleri; UniversiteKodu, UniversiteAdi ve Adres bilgisidir.

Tabloda veri tekrarı olmaması için her satır diğerlerinden farklıdır ve tabloda bütün özniteliklerin aynı değerleri aldığı ikinci bir satır bulunmaz. İlişki için bir anahtar kullanılması gerekir. Anahtar, bir satırı tek başına tanımlayabilen öznitelikler kümesidir. Anahtar kavramı, ilişkisel veri modelinde kullanılan önemli bir kısıttır.

Bu kurallar kullanılarak hazırlanan bir ilişkisel modelde, yine de belirsizlikler ve uyumsuzluklar bulunabilir. Bunları gidermek için de bir dizi düzgüleme işlemine gerek duyulabilir. Düzgülemek, veri tabanı tasarım prensiplerini yapısallaştırmayı amaçlar. İlişkiler ve öznitelikler arasındaki fonksiyonel bağımlılıkları düzenler. Birbirini takip eden beş işlemden oluşur. Fonksiyonel bağımlılık şu şekilde tarif edilebilir: "x ve y öznitelikleri arasındaki ilişki R ile gösteriliğinde, her bir x değerine bir tek y değeri karşılık geliyorsa, R'nin y özniteliğinin, R'nin x özniteliğine fonksiyonel olarak bağımlı olduğu söylenir."

Veri üzerinde yapılacak işlemler için, ilişkisel veri modellerinde üç tip dil kullanılır. Birincisi, matematikteki ilişkisel işlemlere dayanır. Bu tip dillere örnek olarak INGRES ve QUEL verilebilir. İkinci tip dil, görüntü yönelimleridir. Boşluk doldurma yöntemiyle çalışır. Örneğin, QBE (Query By Example) ve CUPID bu tür dillerdendir. Üçüncü tip dil, haritalandırma yönelimli dildir. Bu dip diller, bilinen bir özniteliğin kümesinin üzerinde, bir ilişki yoluyla haritalandırılması prensibiyle çalışır. Örneğin, yapısal sorgulama dili (SQL) bu tip bir veri dilidir. SQL ilerleyen bölümlerde detayları ile anlatılacaktır.

2.3.2.3 Nesne Yönelimli Veri Modelleri

Nesne yönelimli programlamanın başlangıcı, 1960'ların sonu ve 1970'lerin başı arasında geliştirilen simülasyon dili simula'ya kadar uzanır. Nesne yönelimli veri modelinde, bir sorgunun karşılığında mutlaka önceden tanımlanmış belirli bir nesne kümesi olması gerekir. Bir sorgunun sonucu olarak tesadüfî bir nesne kümesinin elde edilmesi mümkün değildir. Çünkü bütün nesnelerin, modelde önceden tanımlanmış olması gerekmektedir.

Genellikle soyutlama olarak anılan bu tip işlemler, şu başlıklar altında toplanabilir:
Sınıflandırma ve elemanlarına ayırma: Sınıflandırma, nesne yönelimli veri modeli yaklaşımının temelini oluşturmaktadır ve aynı özellik ve davranışlara sahip nesnelerin nesne sınıfları içinde gruplanması ile ilgilidir. Bir sınıftaki nesneler, o sınıfın tanımına göre tarif edilebilir. Böylece her nesneyi ayrı ayrı tarif etmeye gerek kalmaz. Elemanlarına ayırma ise sınıflandırma işleminin tersidir ve bir sınıf içinde farklı nesneler oluşturulması ile ilgilidir.

Tanımlama: Bu işlem hem soyut kavramların (sınıf), hem de somut kavramların (elemanlar), teker teker tanımlaması ile ilgilidir ve anahtar değerler yardımıyla yapılır.

Toplam: Nesneler arasındaki ilişkilerin daha üst düzeyde, bir toplam nesne (veya tip) tarafından temsil edilmesi ile ilgili bir soyutlama yöntemidir. Bu toplam tipe genellikle anlamlı bir isim verilir ve bu isim modelin başka yerlerinde, ona ait özellikler referans olarak verilmeden kullanılabilinir.

Genelleştirme: Aynı özelliklere sahip bir grup nesnenin, soysal nesne olarak temsil edilmesi ile ilgili bir soyutlama yöntemidir.

3-VERİTABANI TÜRLERİ

3.1 Yerel Veritabanları

Yerel veritabanları en basit veritabanı tipidir. Bu veritabanları tek bir bilgisayar üzerinde bulunurlar. Dolayısı ile yalnızca bir kullanıcı erişebilmektedir. Yerel veritabanlarına tek kullanıcılı veritabanları ya da Masaüstü veritabanları da denmektedir. Yerel veritabanları veri depolama ve veri üzerinde işlem için esnek ve basit çözümler sunarlarken bu veritabanlarının birçoğunun güncel sürümleri sunucu, internet ve geliştirme araçları ile desteklenmişlerdir.

3.2 İstemci/Sunucu Veritabanları

Bu tür veritabanı sistemlerinde, veritabanı bir dosya sunucu üzerinde bulunur ve veriler bu bilgisayar üzerinde saklanır. Bu bilgisayar İstemci/Sunucu veritabanının Sunucu kısmını belirtir. Bu veritabanına başka bilgisayarlar üzerinden erişen kullanıcılar ise İstemci kısmını belirtir. İstemci/Sunucu veritabanı kullanıcıları bir ağ ortamına yayılmış durumdadırlar. Bu şekilde her kullanıcı farklı konumlardan veritabanına aynı anda erişebilmektedir. Bu kullanıcılar sunucu üzerindeki veritabanı ile direkt olarak hemen hemen hiç uğraşmazlar. Bunun yerine kullanıcının çalışmakta olduğu bilgisayar üzerindeki uygulama ile veritabanına erişim sağlanır. Yerel bilgisayar üzerindeki bu uygulamalara istemci uygulamaları denir. İstemci/Sunucu mimarilerinde işletim sistemi seçimindeki yaklaşım, İstemci uygulamaların yer aldığı istemci bilgisayar üzerinde Windows 9X, ME, XP gibi işletim sistemlerinin bulunması, sunucu bilgisayar üzerinde ise sunucu işletim sistemlerinin bulunması şeklindedir. Çünkü sunucu işletim sistemlerinin güvenli olması, daha çok sunucu hizmetleri odaklı olması ve pahalı olması gibi nedenlerden dolayı bu yaklaşım tarzı seçilmektedir.

Avantajları arasında;

- Esneklik
- Performans
- Ölçeklenebilirlik,

Dezavantajları arasında;

- Özel bilgi ve eğitim gerektirmesi,
- Pahalı olmaları,
- Son kullanıcılar için zorluklar içermeleri

bulunmaktadır.

3.3 Tek-Katmanlı, İki-Katmanlı ve Çok-Katmanlı Veritabanları

Tek-Katmanlı veritabanı, kayıtlar üzerinde herhangi bir değişikliğin anında meydana geldiği veritabanıdır. Bu veritabanını kullanan uygulamalar veritabanına direkt bağlantı

sağlayabilmektedir. Çünkü program ve veritabanı dosyaları aynı bilgisayar üzerinde bulunmaktadır. Yerel veritabanları Tek-Katmanlı veritabanı olarak ta geçmektedir.

İki-Katmanlı veritabanında ise istemci uygulama veritabanı sürücülerini kullanarak veritabanı sunucusuna erişim sağlar.

Çok-Katmanlı veritabanı modelinde client uygulama, veritabanı için bir veya daha fazla uygulama sunucusu ile bağlantı kurar. Çok-Katmanlı veritabanları 3 Katmanlı veritabanları olarak ta çağrılmaktadır. Çünkü bu modelde Çok-Katmanlı uygulama aşağıdaki gibi 3 bölüme ayrılmıştır:

- **Client Application (İstemci Uygulama):** Kullanıcının kendi bilgisayarında bir kullanıcı ara birimi sunar.
- **Application Server (Uygulama Sunucusu):** Bir ağ ortamında bütün istemcilerin erişebileceği merkezi bir konumda bulunur ve istemcilerin ihtiyaçlarını karşılamak üzere ortak veri servisleri sunar.
- **Remote Database Server (Uzaktan Erişimli Veritabanı Sunucusu):** İlişkisel Veritabanı Yönetim Sistemini sağlar.

Bu modelde istemci uygulama veritabanına doğrudan değil de aradaki katmanda bulunan uygulama sunucusu üzerinden bağlanır.

4- Veri Tabanı Nesneleri

4.1 Tablolar

Tablolar bir veritabanı içindeki en önemli nesnelerden biridir. Çünkü bir veritabanında veya ilişkisel veritabanlarında veriler tablolarda saklanır. Bir veritabanı birden fazla tablo içerebilir. Tablolar ise sınırsız sayıda satır (kayıt) içerirler. Tablolar iki boyutlu bil bilgi kümesini temsil eder: Satırlar(rows) x Kolonlar(column).

Tablolarda satırların sırası keyfidir. Herhangi bir kurala göre dizilmezler. Ancak satırları anahtar olarak belirtilen bir kolona göre sıralamak mümkündür.

Tablolardaki kolonlar belli bir isim adı altında ve bir sıra numarası ile belirtilmişlerdir. Kolonlar genellikle alan olarak ta isimlendirilir.

SQL komutları, kolon sıraları ve isimleri ile çalışmaktadır.

4.2 Görünümler

Görünümler sanal tablolardır. Tablolarda saklanan verileri görüntülemek için alternatif bir yoldur. Görünümler bir tablodaki tüm verileri veya bu verilerin bir bölümünü kullanıcılara sunabilir. Bu özelliğinden dolayı herkesin ulaşması ve üzerinde değişiklik yapmasını istemediğimiz verileri saklayabiliriz. Çünkü görünümler veri içermezler, sadece görsel olarak başka noktadaki bir veriye işaret ederler. Görünümler iki veya daha fazla tabloyu birleştirerek veya bağlayarak, bu tablodaki verilerin ortak olarak sunabilmesine imkân verir.

4.3 Eşanlam

Bu türden nesneler ile bir tablo veya görünüm için takma ad niteliğinde başka bir isim kullanılabilir. Eşanlam bir tabloyu temsil eder. Dolayısıyla bu şekildeki bir nesne ile kullanıcıların tablo üzerinde bazı haklara kavuşması sağlanabilir veya bazı haklardan mahrum edilebilir. Başka kullanıcıların bir tabloya erişmesi için eşanlam tanımlamak uygun bir yöntemdir. Eşanlam nesnesi tablo üzerinden yapılan sorgulamalarda da kolaylık sağlar. Bu nesneler gerçekte bir tablo veya veri içeren bir nesne olmadıklarından hafızada yer kaplamazlar.

4.4 İndeksler

İndeksler veriye erişimi hızlandıran veritabanı nesnelerindendir. Bir kitabın sonunda verilen indeks bölümlerine benzerler ve aranılan veriye hızlı bir şekilde konumlanmayı sağlar. Çünkü indeksler, tablolarda saklanan veriye işaret eden sıralı işaretçiler içerir. SQL Server Veritabanı Yönetim Sistemi işaretçi bırakarak tanımlanan indeksler dışında bir de kümelenmiş indeks kullanır.

Clustered indeksler: Tablodaki kayıtları fiziksel olarak sıralar. Bir tablo sadece bir clustered indekse sahip olabilir.

Non-Clustered indeksler: Fiziksel olarak bağlı değil, tablodaki veriye işaretçiler konarak indeksleme yapılır. İşaretçiler kendileri sıralı haldedirler. Bir tablo sahip olduğu her kolon için bir indeks içerebilir. Non-Clustered indeks tablodan ayrı bir nesnedir.

4.5 Sıralar

Birbirinden farklı tamsayı değerler üreten nesnedir. Bu nesne bazı veritabanı programlarında tablo içerisinde başka bir benzeri olmayan tamsayı değerlere ihtiyaç duyulduğunda kullanılmaktadır. Örneğin tabloda birincil anahtar olarak tanımlanmış bir alana otomatik olarak ve birbirinden farklı tamsayı değerler bu nesne ile atanabilir. Sıra nesnesi istenilen bir sayıdan başlatılabilir veya artım değeri belirlenebilir.

4.6 Veri Tipleri

Veri tipleri tablolardaki kolonlara girilebilecek veri tiplerini belirler veya tanımlar. Bir veri tipi tablodaki her kolon için belirtilmelidir. Örneğin, bir kolon için nümerik bir veri tipi belirtilmiş ise bu alanda sayılar tutulacaktır, harfler ve karakterler değil.

Her veri tipi byte cinsinden belirli bir uzunluğa sahiptir ve bu veri tipindeki veriler hafızada bu uzunluk kadar yer kaplarlar. Bir veri tipi 2 byte uzunluğunda ise bu veri tipinden 65536 tane sayı gösterilebilir. Gösterilebilecek en küçük sayı 0(sıfır), en büyük sayı ise 65535' tir.

4.7 Varsayılanlar

Veri girişi esnasında bir kolona değer girilmez ise kolona daha önceden belirlenen bir değer otomatik olarak girilir. Default'lar tablodaki herhangi bir kolona atanabilir, fakat atanacak veri kolon veri tipi ile aynı veya uyumlu olmalıdır. Default gerektiğinde üzerine yazılabilir özelliktedir. Örneğin, evrak takiplerinin tutulduğu bir veritabanında evrakların geliş tarihlerini varsayılan olarak o günün tarihini atayabiliriz. Böylece her yeni gelen evrak kaydedilirken Tarih hanesine o günün tarihi otomatik olarak yazılacaktır, fakat bu tarih istenildiğinde değiştirilebilecektir.

4.8 Kurallar

Kurallar, tabloya girilebilecek veriyi kontrol eden veritabanı nesneleridir. Veritabanı tasarımcıları girilebilecek kötü ve uyumsuz verileri engellemek için bu nesneleri kullanır. Örneğin öğrenci notlarının girildiği bir alana 0 ile 100 arasındaki sayılar haricinde veri girişi yapılır ise kullanıcı uyarılabilir.

4.9 Saklı Prosedürler

Birçok görevin otomatize edilmesini sağlayan esnek ve güçlü veritabanı nesneleridir. Saklı prosedürler, veritabanlarında yerleşik olarak gelen ve belli bir görevi yerine getiren prosedür yapısında komutlardır. Özellikle İstemci/Sunucu yapıdaki veritabanlarında performans artışı sağlarlar. Çünkü işlemci tarafından istemde bulunulan bir işlem, sunucu tarafında hazır olarak bulunan saklı prosedürler tarafından yerine getirilir ve istemciye sadece işlemin sonucu gönderilir. Bu şekilde ağ trafiği de azaltılmış olur.

4.10 Tetikleyiciler

Özel bir saklı prosedür tipidir ve bir tabloda belirli olayların gerçekleşip gerçekleşmediğine göre çalışır. Örneğin bir tabloya veri eklenmesi, silinmesi veya güncellenmesi esnasında otomatik olarak ateşlenir. Ateşlenen bir tetikleyici bir saklı prosedürün çalışma anını belirler.

4.11 Sorgular

Bir sorgu veritabanı programınıza ait bir komuttur. Bu komut belirtilen kriterler doğrultusunda veritabanınızdaki tablolardan verileri seçerek alır ve belirtilen biçimde size sunar. Tablolardan alınan bu veriler bilgi amaçlı veya daha değişik işlemlerde kullanılmak üzere işlenebilir. Bu veriler bilgisayar ekranına, bir yazıcıya veya bir dosyaya gönderilebilir.

4.12 Joining (İlişkilendirme)

İki veya daha fazla tabloyu birlikte sorgulama işlemine join ismi verilir. İlişkisel veritabanının en temel özelliği birden fazla tablo ile birlikte işlem yapabilmektir. Bu sayede verilerin tekrarlanması önlenmiş olur ve sonuçta veri yönetimi kolaylaşır. Örneğin, personel işlerinde personele ait bilgiler bir tabloda, personelin her ay aldığı maaş bilgileri diğer tabloda tutuluyor olsun. X sicil numaralı kişinin ad, soyad ve maaş bilgilerinin listelenmesi gibi bu iki tablodaki bilgilere de bir tek sorgu sonucu olarak ihtiyaç duyulabilir.

5- Veri Tabanı Tasarımı ve Normalizasyon

5.1 Veri Tabanı Tasarımı

5N, tasarım aşamasında hangi şartlara uygun tasarım yapılması gerektiğini belirleyen kurallardır. Bazen bu kuralların uygulanmadığı veya vazgeçildiği durumlar olabilir ancak, veritabanında saklanacak veriler artarak veritabanı büyüdükçe bu kuralların daha sıkı uygulanması gerekir.

Bir veritabanı ile proje yapılırken işin en önemli aşaması veritabanının tasarlanmasıdır. Başlangıçta yanlış tasarlanan bir veritabanı ile yapılan projede sonradan yapılacak düzenlemelerle geri dönüş yapılamayabilir. O nedenle veritabanı tasarımı yapılırken 5N maddelerine uyularak yapılması gerekir.

1. **Nesneler Tanımlanır:** Nesne, çeşitli özellikleri olan bir varlıktır. Herhangi bir projede de öncelikle nesneler tanımlanır. Birkaç proje için örnek verecek olursak,
   Kütüphane Sistemi: Kitap, üyeler, türler, ödünç hareketleri
   Okul Sistemi: Öğrenciler, öğretmenler, dersler, derslikler, notlar
   Personel Sistemi: Çalışanlar, meslekler, çalışılan birimler, maaşlar, izinler
2. **Her nesne için bir tablo oluşturulur:** Her nesne için bir tablo oluşturulur ve her bir tabloya içereceği veriyi en iyi anlatan bir isim verilir. Tablo oluşturma işi, bir kâğıt

üstünde sembolik olarak gösterilebilir veya doğrudan Veri Tabanı Yönetim Sistemi üstünden oluşturulabilir.

3. **Her bir tablo için bir anahtar alan seçilir:** Veritabanındaki herhangi bir veriye erişilmeden önce tabloya erişilir. Veritabanında; üzerinde en çok işlem yapılan nesneler tablolardır. Bu aşamaya kadar hangi tabloların oluşturulacağı ve her bir tablonun içinde hangi bilgilerin saklanılacağı belirlendi. Bu aşamada, tabloda yer alacak her bir kaydı bir diğerinden ayırabilecek bir sütuna ihtiyaç duyulur. Örneğin bir öğrenci seçilmek istenildiğinde, bu öğrenciyi diğerlerinden ayırt edebilecek olan öğrenci numarası birincil anahtar olarak belirlenebilir. Öğrenci; numarası ile diğer öğrencilerden ayırt edilebilir.
4. **Nesnelerin her bir özelliği için tabloya sütun eklenir:** Tablo adları tanımlandıktan ve anahtar adları belirlendikten sonra, tablodaki nesnelerin her bir özelliği için bir alan eklenir. Örneğin, personel tablosu için Personelin sicil numarası, adı, soyadı, e-mail adresi, mesleği, çalıştığı birim, maaş gibi bilgiler için sütun tanımlamaları yapılır.
5. **Tekrarlayan nesne özellikleri için ek tablolar oluşturulur:** Tabloda veri tekrarı olacaksa hata yapılıyor demektir ve eldeki tablonun en az bir tabloya daha ayrılması gerekir. Her projeye uyacak bir veritabanı tasarım tekniği olmadığından tasarım belli veritabanı tasarım ve Normalizasyon kurallarına göre düşünülüp tasarlanır. Öğrenci ve öğrenciye ait ders notları için veritabanı tasarımını düşünelim. Tabloda aşağıdaki sütunların var olduğunu kabul edelim.

ÖğrenciNo	Dersin adı	Vize notu	Final notu	Ortalama

Bir öğrenci aldığı dersten başarılı olursa vize ve final notu yazılarak ortalaması hesaplanır ve sorun yaşanmaz. Ama öğrenci bu dersten başarısız olursa bu dersi yeniden almak zorundadır. Yeniden aldığı bu derse ait ders notlarının nereye yazılacağının düşünülmesi gerekir. Eski notlarının da kalması gerektiği düşünüldüğü bu durumda tablo şöyle olacaktır:

Öğrenci No	Dersin adı	Vize notu	Final notu	Ortalama	Vize notu	Final notu	Ortalama
03101001	Bilgisayar	37	40		45	48	

Tabloda iki adet not yazılabilecek alan vardır. Peki, ama öğrencinin dersi ikiden fazla kere tekrar etmesi gerekirse ne olacak? Bu durumda yeni sütun alanları mı eklemek gerekecek? Tabloya 3 tane not yazma alanı eklendiğinde dersi bir kere alan öğrenciler için 2. ve 3. alanlar boş kalacak. Bu her öğrenci için değişebilecek bir durum olduğu için tablo tasarımında bu mantıkla düşünmek doğru değildir ve bu şekilde tasarım yapılamaz.

Ayrıca; tabloda tanımlanan her sütun alanı, bu alana hiçbir bilgi yazılmasa bile Hard diskte yer kaplayacağı için; kullanılmayıp boş geçildiğinde de diskte tanımlanan bu alanlar boşuna kullanılmış olacaktır. Dolayısı ile diskte de boş yere disk alanı işgal edilmiş olacağından tabloda gereksiz sütun alanlarının tanımlanmaması gerekir.

6. **Anahtar Alana Bağlı Olmayan Alanlar Belirlenir:** İlişkisel veritabanında, tablodan herhangi bir tek kayda erişmek için mutlaka bir farklı özellik sağlanmalıdır ve bu özellik de anahtar alan tarafından sağlanır. Ancak bazen, anahtar alan ile aynı satırda yer aldığı halde, anahtar alan ile birebir ilişkisi olmayan bir alan yer alabilir. Bu türden alanların elimine edilip ayrı tablolara ayrılması gerekir. Örneğin, ödünç tablosu ele alınacak olursa, ödünç verilen her kitap için ödünç alanın adresi de belirlenmek istenirse, bu ödünç tablosuna yazılamaz. Çünkü ödünç tablosunun birincil anahtarı oduncno'dur ve bu alan, ödünç verme işlemi ile ilgilidir. Oysa ödünç alanın adresi ödünç alan kişinin kendisine bağlı bir özelliktir. Bu kişinin her aldığı kitap için adresini tekrar yazmaya gerek yoktur. Aynı şekilde otomasyon içerisinde başka

yerlerde de bu kişinin adres bilgilerine muhtemelen ihtiyaç duyulabilir çünkü adres, üyenin bir özelliğidir. Ödünç verilen kitabın adresi öğrenilmek istenildiğinde, üyeler adında bir tablo daha açılıp, burada herkesin adres bilgisi tutulmak zorunda kalınır. Ödünç tablosunda ise, oduncalan bilgisi olarak, üyeler tablosunun birincil anahtar alanına bir bağlantı içermesi daha doğru olur.

7. **Tablolar arasındaki ilişkiler tanımlanır:** Tanımlanan tablolardaki alanların birbirleri ile olan ilişkileri tanımlanır. Örneğin öğrenci bilgilerinin tanımlandığı tablo ile öğrenci notlarının tanımlandığı tablo arasında bir ilişki kurulur ve bu ilişki öğrenci numarası ile olur. Öğrencinin hangi dersine ait not bilgisi yazılacaksa ders kodları ve adlarının tanımlandığı tablo ile öğrenci notları tablosu arasında bir ilişki vardır. Bu ilişki notlar tablosundaki ders kodu ile ders kodları tablosundaki dersin kodu alanları arasında yapılır.
   İlişkili her iki tablo bir birincil alan ve bir yabancıl anahtar alan üstünden birbirine bağlanır.
   Farklı tablolardaki iki alan aynı veriyi tutuyorsa, iki alana da aynı ismi vermek, karışıklığa yol açabilir gibi görünse de aslında daha düzgün bir yapının ortaya çıkmasını sağlar. OgrenciNo alanı ogrenci tablosunda da notlar tablosunda da öğrenci numarası olarak tanımlanabilir. Bu alanlardan birine OgrenciNo, diğerine NotlarOgrenciNo ismi verilmesi sorgu ve program yazılımında isim karışıklıklarına neden olabilir. En önemlisi de her alan için her tabloda farklı isimlerin kullanılması değişkenlerin isimlerinin akılda tutulmasını ve daha sonraki tablolar üzerinde işlem yaparken işlemleri zorlaştırır. Tablo ve alanlar kullanılacağı zaman her seferinde ilgili alanın hangi isimle kaydedildiğine bir listeden bakmak zorunda kalınır. Çünkü büyük bir veritabanı projesinde 250'den fazla tablo bulunabilir. Her tabloda da birçok alanın bulunacağı dikkate alındığında her alana ait isimlerin akılda tutulması mümkün olmamaktadır. Birden fazla tabloda olan alanlar için; aynı ismi kullanmak bu zorluğu ortadan kaldıracaktır. En mantıklısı her ikisinde de OgrenciNo ismi verilmesidir.

5.2 Veri Tabanı Normalizasyonu

Bir tablo içerisinde yer alacak kaydın nelerden oluşmasına karar vermeye yarayan düzenlemelere normalizasyon kuralları denilir. Kabul görmüş 5 normalizasyon kuralı vardır :

1. **Normalizasyon Kuralı:**
   Bir satırdaki bir alan yalnızca bir tek bilgi içerebilir. Birden fazla notu olan öğrenci için not1, not2 ve not3 diye alanların açılması ile bu kurala uyulmamış olunur. Böyle bir durumda, ayrıca notlar tablosu oluşturularak veritabanı tasarımı ve normalizasyon kuralına uyulmuş olur.

2. **Normalizasyon Kuralı:**
   Bir tabloda, anahtar olmayan her alan, birincil anahtar olarak tanımlı tüm alanlara bağlı olmak zorundadır. Örneğin, notlar tablosuna DersinAdı gibi bir alan eklenirse, bu sadece dersin adı ile ilgili bir bilgi olur ve DersinKoduna bağlı bir nitelik olmaz. Bu nedenle ders isimleri için ayrı bir tablo yapılarak sorun çözülebilir. Ya da anahtar alanın birden fazla alandan oluştuğu tablolarda, anahtar alanlardan sadece birine bağlı veriler tabloda yer almamalı, ayrı bir tabloya taşınmalıdır. Bunun terside geçerlidir. Yani iki ya da daha fazla tablonun birincil anahtarı aynı olamaz. Böyle bir durum söz konusu ise, bu iki tablo tek tabloya indirilmelidir.

3. **Normalizasyon Kuralı:**

Bir tablo için, anahtarı olmayan bir alan, anahtarı olmayan başka hiç bir alana bağlı olamaz. Örneğin, notlar tablosunda hangi not bilgisi için SınavTipi isimli bir alan eklenip burada Vize için V, Final için F ve Bütünleme için B kodlaması yapılırsa bu alan notlar tablosunun birincil anahtarı olan OgrenciNo alanına bağlı bir kodlama olmaz. Çünkü bu kodlama bir başka anahtarı olmayan alana bağlıdır. Bunun sonucunda da veritabanında, karşılığı olmayan kodlama yer almış olur. Sınav tipi bilgisini kodlu olarak tutan alan aslında Sınav tipi açıklaması olan başka bir alana bağlıdır. Bu ilişki başka bir tabloda tutulmalıdır. Bu durumda, Sınav tipi bilgilerini tutan yeni bir tablo açılması daha doğru olur. Bu kodlamayı program içerisinde yapmak; ileride yeni oluşabilecek bir kodu tanımlama ve düzenleme açısından ve programın pek çok yerinde yeniden değişiklikler yapma nedeniyle birçok sorun meydana getirir. Bu tablonun alanları da SınavTipiKodu ve SınavSekli olarak tanımlanabilir ve Notlar tablosunda SınavTipi adında bir sütun eklenip buraya V, F, B gibi kodların yazılması gerekir.

4. **Normalizasyon Kuralı:**
   Birincil anahtar alanlar ile anahtarı olmayan alanlar arasında, birden fazla bağımsız bire-çok ilişkisine izin verilmez. Örneğin, notlar tablosunda yer alan bir not hem vize, hem final hem de bütünleme notu olamaz. Bu durumda notlar tablosu nasıl tasarlanabilir? Not ile birlikte bu notun hangi sınav tipine ait olduğu sütunu tanımlanır ve bilgiler ona göre yazılır. 4. normal formu sağlamak için, her bağımsız bire çok ilişki için ayrı bir tablo oluşturulması gerekir.

5. **Normalizasyon Kuralı:**
   Tekrarlamaları ortadan kaldırmak için her bir tablonun mümkün olduğunca küçük parçalara bölünmesi gerekir. Aslında ilk 4 kural sonuçta bu işe yarar ancak, bu kurallar kapsamında olmayan tekrarlamalar da 5 normalizasyon kuralı ile giderilebilir.
   Örneğin, öğrencilerin okula kayıt olma şekilleri olan OSYM sınavı ile, Özel Yetenek Sınavı ile, af ile gibi bilgileri içeren bir öğrenci okula kayıt şekli alanı olabilir. Bu kayıt şekilleri de KayıtSekliKodu ve Kayıt SekliAdı alanını kapsayan başka bir tablo tutulabilir. Bu tabloda KayıtSekliKodu için 1 ve KayıtSekliAdı için OSYM sınavı gibi bilgiler yazılabilir. Böylelikle, kullanıcıların bu alana gelişi güzel bilgiler girmesi engellenmiş olur. Bu da sorgulama esnasında veriler arasında bir tutarlılık sağlar. Bu işlem sonucunda, tutarsızlıklara neden olabilecek ve sık tekrarlayan veriler başka bir tabloya taşınmış olur. Bu tablo için, veritabanı programlamada "look-up table" terimi kullanılır.
   Ancak, veritabanı normalizasyon kuralları, bir ilişkisel veritabanının tasarlanma aşamalarını değil de ilişkisel veritabanında yer alacak kayıtların ilişkisel veritabanı ile uyumlu olup olmadığını denetlemeye yöneliktir. İlişkisel bir veritabanı tasarımı aşağıdaki dört özelliği taşımalıdır.
   1. Veri tekrarı yapılmamalıdır.
   2. Boş yer mümkün olduğunca az olmalıdır.
   3. Veri bütünlüğü sağlanmalıdır.
   4. Veriler, aralarında bir ilişki tanımlanmaya müsait olmalıdır.

6-Structured Query Language (SQL)

SQL, ilişkisel veritabanları üzerinde veri tanımlama ve veri işleme için kullanılan standardize edilmiş bir dildir. Tam bir programlama dili değildir. Bir programlama dilinde olan döngü komutları veya şart ifadeleri bulunmaz. Veri tanımlamak ve veri sorgulamak için

gerekli komutlara sahiptir. Ancak diğer programlama dilleri içerisinde bu komutları kullanabilmek mümkündür.

SQL bundan 30 yıl kadar önce Dr. E.F. Codd tarafından geliştirilen ilişkisel veritabanı modeli ile gündeme gelmiştir. Daha sonraları IBM firması Veritabanı Yönetim Sistemleri üzerine geliştirdiği projelerde bir sorgu dili kullanmıştı. Bu sorgu dili STRUCTURED ENGLISH QUERY LANGUAGE kelimelerinin baş harflerinden oluşan SEQUEL idi. Sonraki tarihlerde STRUCTURED QUERY LANGUAGE (Yapısal Sorgu Dili) olarak adlandırıldı.

SQL komutları veritabanı yöneticisi tarafından verilere yönelik olarak icra ettirilir. Verinin ilişkisel modeli gereğince; **veritabanı** bir tablolar kümesi olarak algılanır, **ilişki** tablolardaki değerler tarafından gösterilir ve **veri** ise bir veya birden fazla tablodan elde edilen bir sonuç tablo olarak belirtilir.

KATILIMLI SQL (EMBEDDED SQL)

SQL dilinin bir programlama dili içerisinden kullanılmasına Katılımlı SQL denir. Katılımlı SQL komutları ise programlama dili içerisinde yazılan SQL komutlarıdır. İki tip katılımlı SQL vardır: static ve dynamik.

Static SQL komutunun kaynak biçimi, ana programlama dilinde yazılmış bir uygulama programı içine katılır. Static SQL komutu program çalıştırılmadan önce hazırlanır ve program çalıştırıldıktan sonra komutun kullanılmaya hazır biçimi devam eder.

Dynamic SQL komutları ise çalışma anında üretilir ve çalıştırılır.

Borland'ın Borland Database Engine (BDE) olarak geçen veritabanı motoru ise Local SQL fonksiyonları sayesinde bu verileri işler. Local SQL, ANSI-standart SQL dilinin bir alt kümesidir. Bunun gibi çeşitli ürünlerin standart SQL kümesini geliştirerek elde ettikleri SQL yapıları ve isimleri şöyledir:

VTYS (DBMS)	SQL Yapısı
MS SQL Server	**Transact SQL**
Oracle	**PL/SQL**
Paradox/dBASE	**Local SQL**
Advantage	**StreamlineSQL**

GENEL YAZIM KURALLARI

SQL dilinde ifadelerin yazımı tamamen esnektir. Birçok programlama dilinde olduğu gibi SQL dilinde de ifadelerin yazımında satır ve sütun serbestliği vardır. Yani bir komut istenilen satır ve sütundan itibaren yazılır.

```
SELECT * FROM PERSONEL WHERE AD='MUSTAFA';
```

Yukarıdaki örnekte olduğu gibi ifadenin tamamı aynı satırda yer alabilir. Aynı şekilde bu ifade bölünerek aşağıdaki gibi üç satır halinde de yazılabilir.

```
SELECT *
FROM PERSONEL
WHERE AD='MUSTAFA';
```

SQL ifadeleri büyük/küçük harf duyarlı değildir. Yukarıdaki örneklerdeki gibi bütün sözcüklerin büyük harfle yazılması gerekmez. Örneğin aşağıdaki ifade yukarıdakinin aynısıdır.

```
Select * from personel where ad=’MUSTAFA’;
```

Bu örnekte dikkat edilmesi gereken bir nokta vardır. İfade içinde kullanılan tüm sözcükler küçük yazıldığı halde ‘MUSTAFA’ ismi yine aynı bırakılmıştır. Çünkü ‘MUSTAFA’ ismi “personel” tablosunda bir veriye işaret etmektedir. Dolayısı ile bu veri kullanıcının girişine bağlı olduğundan SQL, arama işlemlerinde büyük/küçük karaktere duyarlıdır.

SQL ifadelerinin sonunda yer alan noktalı virgül “;” işareti ifadenin tamamlandığını SQL programına bildirir. Bu işaret Access ve Delphi gibi bazı programlarda zorunlu değildir.

Tablo İsimlerinin Kullanılması

Sorgu isimlerinde tabloların kullanımı, birçok veritabanı programında farklılıklar göstermektedir. Örneğin Access programında veritabanı bir dosya halinde saklanır. Bu nedenle bu programda hazırlayacağınız SQL sorguları ve tablolar aynı yerde tutulduğundan sorgularda tablo isimlerini direkt olarak yazabilirsiniz.

```
Select * from personel
```

Bazı veritabanı programlarında ise tablo isimlerini sorgu ifadelerinde tek tırnak veya çift tırnak içinde belirtebilirsiniz.

```
Select * from ‘customer.db’
```

```
Select * from ‘c:\program files\common files\borland shared\data\customer.db’
```

Üstteki sorguda olduğu gibi yol (path) ismi de verilebilir. Yukarıdaki iki kullanımı tek tırnak yerine çift tırnak ile de yapabiliriz. Tablo isimlerinde kullanılan yol yerine takma isim (alias) kullanılabilir. Local SQL tablo isimlerinde takma isim (alias) kullanımına da izin vermektedir.

```
Select * from “:dbdemos:customer.db”
```

İfadesinde **dbdemos** takma ismi **customer.db** tablosunun bulunduğu konuma işaret etmektedir.

Kolon İsimleri

Local SQL sorgu ifadelerindeki bir veya birden fazla kelime ile verilen kolon isimlerini çift veya tek tırnak arasında belirtilmesine izin vermektedir.

```
Select * c.”ulke ismi” from “customer.db” c
```

Tarih Biçimlerinin Kullanılması
Tarih (date) ve zaman (time) biçimindeki bilgilerin kullanımında yine tırnak işaretlerinden yararlanılır.

```
Select * from “orders.db” where SaleDate=”7/7/1988”
```

Mantıksal Verilerin Kullanımı
Mantıksal türdeki verilerin literal değerleri TRUE veya FALSE, tırnak işareti ile veya tırnak işareti olmadan kullanılabilmektedir.

```
Select * from “orders.db” where paid=true
Select * from “orders.db” where paid=”false”
```

NULL Değerler
Veritabanı sistemlerinde bir tablo alanı içerisinde eksik bilgi veya bilinmeyen bir bilgi olduğunu göstermek için NULL değerlerden faydalanılır. Bu değerler o tablo alanı içerisindeki verinin tipinden bağımsızdır. NULL değeri tablo alanında kullanılan boşluk karakteri, sıfır sayısal değeri, sıfır uzunluğundaki veya boş bir karakter katarı değildir. Kısacası bilinmeyen veri tipini göstermek için NULL değerler kullanılır.

İfadeler
Bir değer döndüren herhangi bir dizilime İFADE denir. SELECT veya FROM gibi sözcükleri izleyen herhangi bir dizilim ifadedir. Örneğin, aşağıdaki dizilim personel tablosundaki maas alanının içerdiği değerleri döndüren bir ifadedir.

```
Select maas from personel;
```

Aşağıdaki select komutunu izleyen ad, soyad ve adres ve from sözcüğünü izleyen öğrenci dizilimleri birer ifadedirler.

```
Select ad,soyad,adres from öğrenci;
```

Aşağıda where sözcüğü ile yapılan ifadedir.

```
Where ad=’Ahmet’;
```

Bu ifade bir şart ifadesidir. Şart ifadeleri geriye iki değer döndürür. TRUE veya FALSE. Burada ad=’Ahmet’ şartı ‘=’ karşılaştırma operatörüne bağlı olarak TRUE veya FALSE olacaktır.

Koşul İfadeleri
Veritabanından bir grup veriyi veya sadece bir veriyi çekip almak istediğimizde daima koşul ifadelerine ihtiyaç duyarız. SQL de koşul ifadeleri where sözcüğü ile belirtilir. **Ad=’Ahmet’** bir koşul ifadesidir ve veritabanında sadece belli bir veri kısmına işaret eder. Veritabanınızdan maaşı 1000 TL nın üzerinde olan personeli almak için maas>1000 gibi bir koşul ifadesi kullanılacaktır.

6.1 Veri İşleme Dili (Data Manipulation Language – DML)

DML kategorisindeki komutlar şunlardır.

Komut	İşlevi
Select	Seç. Tablolardan belirtilen kriterlere uyan kayıtları döndürür.
Delete	Sil. Bir tablodan bir veya birden fazla kaydı siler.
İnsert	Ekle. Bir tabloya bir veya birden fazla kayıt ekler.
Update	Güncelle. Bir tablodaki bir veya birden fazla kayıt üzerinde değişiklik yaparak bu kayıtları günceller.

Bu komutlar haricinde yine bu komutlar ile beraber kullanılan değişik işlevlere sahip komut sözcükleri de bulunmaktadır. Bu komutlar ile beraber kullanılan diğer sözcükler ise aşağıdaki gibidir.

Komut	İşlevi
Where	Sorgu çıktılarında sadece belirtilen kriterlere uyan kayıtların listelenmesini sağlar.
Order by	Sorguların çıktısının hangi kolona göre sıralanacağını belirler.
Group by	Satırların hangi kolona göre gruplandırılacağını belirtir.
Having	Tek değer çıktı döndüren toplam fonksiyonları için kriterlerin belirlenmesini sağlar.

SEÇME SORGUSU (SELECT)

Veritabanı içindeki tablolardan veri alır. Seçim anlamına gelen select burada da olduğu gibi bir tablodan koşula bağlı olarak veya olmayarak veri seçer. SQL dilinde en çok kullanılan komutlardan birisidir. En basit yapısı şöyledir.

```
Select tablo_kolonları from tablo_ismi;
```

Yukarıda da görüldüğü gibi select komutu from sözcüğü ile birlikte kullanılmaktadır. From sözcüğü ile veri alınacak kaynak ismi belirtilir. From sözcüğü kullanılmaz ise verinin alınacağı yer bilinmeyeceğinden select komutu çalışmayacaktır.

Select komutu ile diğer sözcükleri kullanırken aşağıdaki sıra takip edilmelidir.

```
SELECT KOLON_LISTESI
FROM TABLO_ISIMLERI
WHERE KOSUL_İFADELERİ
GROUP BY KOLON_İSİMLERİ
HAVING GRUPLANDIRILMIŞ_VERİLER_İÇİN_KOŞUL
ORDER BY KOLON_LİSTESİ;
```

Ogrenci Tablosu

OgrenciID	OgrNo	Ad	Soyad
1	99118001	Ayhan	PEKER
2	99118003	Fatih	KÜÇÜKBALTACI
3	99118004	Durmuş	KARADAŞ
4	99118005	Kürşat	POTUR

5	99118006	Bülent	SARITOPRAK
6	99118007	Fatih	AYDIN

Bu tablodaki verileri koşulsuz olarak listeleyen komut şöyledir:

Select OgrenciID, OgrNo, Ad,Soyad **from** ogrenci;

Bu komut şöyle bir çıktı verir.

OgrenciID	OgrNo	Ad	Soyad
1	99118001	Ayhan	PEKER
2	99118003	Fatih	KÜÇÜKBALTACI
3	99118004	Durmuş	KARADAŞ
4	99118005	Kürşat	POTUR
5	99118006	Bülent	SARITOPRAK
6	99118007	Fatih	AYDIN

Çıktıdan da görüleceği gibi koşulsuz ve tüm alanların listelenmesi ile "ogrenci" isimli tablodaki veriler aynen alınmıştır. Bu işlemi şu komutlada yapmak mümkündür:

Select * from ogrenci;

Bu komut ile de yine "ogrenci" isimli tablodan bütün kolonlar alınıp listelenmektedir. Kolon isimleri yerine joker karakter olan asteriks (*) sembolü kullanılmıştır.

Şimdide yine ogrenci tablosundan kolon seçimi yapalım:

Select Ad **from** ogrenci;

Komutu yanda ki çıktıyı verir.

Ad
Ayhan
Fatih
Durmuş
Kürşat
Bülent
Fatih

Select komutunun diğer kullanımlarından biri şöyledir:

Select ogrenci.* **from** ogrenci;

Bu sorguda kolonun bulunduğu tablo ismi kolon isminin baş tarafına eklenir. Bu tür bir uygulama select komutunda iki veya daha fazla tablo kullanıldığında listelenecek kolonları hangi tablodan alınacağını belirtmek açısından yararlıdır.

Birden fazla tablo kullanımında komutun genel yazım şekli şöyledir:

Select tablo1.*, tablo2.*,..... **from** tablo1, tablo2

İki tablomuz olan bir durumda çıktının nasıl olacağını görmek için aşağıdaki örneği inceleyelim.

Ogrenci

OgrNo	Ad	Soyad
99118001	Ayhan	PEKER
99118003	Fatih	KÜÇÜKBALTACI
99118004	Durmuş	KARADAŞ
99118005	Kürşat	POTUR
99118006	Bülent	SARITOPRAK
99118007	Fatih	AYDIN

Dersler

Kodu	DersAdi	DersKrd
PC101	İşletim Sistemleri	4
PC103	Basic Prog.Dili	5
PC105	Paket Prog. Uyg.1	2

Bu iki tabloyu birden seçen sorgu aşağıdaki gibidir.

Select ogrenci.*, dersler.* **from** ogrenci, dersler;

OgrNo	Ad	Soyad	Kodu	DersAdi	Ders Krd
99118001	Ayhan	PEKER	PC101	İşletim Sistemleri	4
99118001	Ayhan	PEKER	PC103	Basic Prog.Dili	5
99118001	Ayhan	PEKER	PC105	Paket Prog. Uyg.1	2
99118003	Fatih	KÜÇÜKBALTACI	PC101	İşletim Sistemleri	4
99118003	Fatih	KÜÇÜKBALTACI	PC103	Basic Prog.Dili	5
99118003	Fatih	KÜÇÜKBALTACI	PC105	Paket Prog. Uyg.1	2
99118004	Durmuş	KARADAŞ	PC101	İşletim Sistemleri	4
99118004	Durmuş	KARADAŞ	PC103	Basic Prog.Dili	5
99118004	Durmuş	KARADAŞ	PC105	Paket Prog. Uyg.1	2
99118005	Kürşat	POTUR	PC101	İşletim Sistemleri	4
99118005	Kürşat	POTUR	PC103	Basic Prog.Dili	5
99118005	Kürşat	POTUR	PC105	Paket Prog. Uyg.1	2
99118006	Bülent	SARITOPRAK	PC101	İşletim Sistemleri	4
99118006	Bülent	SARITOPRAK	PC103	Basic Prog.Dili	5
99118006	Bülent	SARITOPRAK	PC105	Paket Prog. Uyg.1	2
99118007	Fatih	AYDIN	PC101	İşletim Sistemleri	4
99118007	Fatih	AYDIN	PC103	Basic Prog.Dili	5
99118007	Fatih	AYDIN	PC105	Paket Prog. Uyg.1	2

Sonuçta bu şekilde bir çıktı elde etmiş oluruz. Kayıt sayısı "ogrenci" tablosundaki kayıt sayısı ile "dersler" tablosundaki kayıt sayısının çarpımı kadardır.

Kolon Başlığının Değiştirilmesi (as)

Select komutu ile elde edilen çıktılarda, kolon isimleri varsayılan olarak tablodaki kolon başlığının ismini alır. Tablonun tasarım anında kolon isimleri işlem açısından daha kısa kelimeler ile belirtilebilir. SQL komutları çıktısında bu kolon isimlerini daha anlamlı ifadeler ile değiştirebiliriz. Bunun için kolon isminden sonra bir boşluk bırakarak yeni kolon ismini belirtebilir veya "as" sözcüğünü ekleyerek bu sözcüğü takiben yeni kolon ismi belirtebiliriz. En sade yazım şekli şöyledir:

Select kolon_ismi as yeni_kolon_ismi **from** tablo_ismi;

Veya

Select kolon_ismi yeni_kolon_ismi **from** tablo_ismi;

Yeni kolon ismi birden fazla kelimeden oluşuyorsa, kelimeler arasına yukarıdaki satırda olduğu gibi alt çizgi koyabilir veya yeni kolon ismini çift tırnak içerisine alabilirsiniz.

Select kolon_ismi "yeni kolon ismi" **from** tablo_ismi;

As sözcüğü bazı VTYS'lerinde tablo isimlerini belirtmek içinde kullanılmaktadır. Örneğin bir sorgu birden fazla tablo üzerinden sorgulama yapıyor ise sorgu sonucundaki alanların hangi tablodan olduğunun belirtilmesi için kullanılabilir. Bu kullanımdaki as sözcüğünün sorgu sonucuna bir etkisi yoktur.

Select per."sicilno", per. "ad", per."soyad" **from** "personel.DB" as per

Bu sorguda personel.DB tablosu per sözcüğü ile temsil edilmiştir.

Çıktıda Literal Değerlerin Kullanılması
Çıktıda her kayıta bulunmasını istediğiniz sabit bir ifade için apostrof (") operatörü kullanılır. Her kayıtta bulunacak ifade iki apostrof işareti arasına ve kolon isimleri kısmına yazılır.

Select kolon1, 'sabit ifade', …… **from** tablo;

Bu sorguda geçen 'sabit ifade' şeklindeki metinler literal değer olarak adlandırılırlar. Aslında veritabanında olmayan bir ifadeyi bile sorgu ile varmış gibi çıktımıza ekleyebiliriz.

OgrNo	Ad	Soyad	Bolum
010701031	Nuran	Akkoç	Bilgisayar
010701032	M.İlke	Kocaduru	Turizm
010751002	Selda	Şekersoy	İşletmecilik
010751003	Mehmet Fatih	Buyun	Muhasebe
010751004	Burcu	Karataş	Muhasebe

Bu tablodan ogrno ve bolum kolonlarının yanına MYO yazısını ayrı bir kolonmuş gibi yazdıralım.

Select ogrno, bolum, 'MYO' okul **from** ogrenci;

Çıktısı:

OgrNo	Bolum	Okul
010701031	BİLGİSAYAR	MYO
010701032	TURİZM	MYO
010751002	İŞLETME	MYO
010751003	MUHASEBE	MYO
010751004	MUHASEBE	MYO

Sayısal Bilgi Çıktılarının Düzenlenmesi
Sorgu sonucunda elde edilen sayısal bilgi alanları üzerinde aritmetiksel işlemler gerçekleştirmek mümkündür. Örneğin personel isimli tablodan personele ait maaşların %15

zamlı şekli listelenebilir. Bunun için maaş kolonundaki değerlere maaşın %15'i ilave edilerek listelenecektir.

Select PerNo, Ad, Maas+Maas*15/100 **as** Zamlı_Maas **from** personel; veya
Select PerNo, Ad, (Maas+Maas*15/100) **as** Zamlı_Maas **from** personel; veya
Select PerNo, Ad, Maas+Maas*1.15 **as** Zamlı_Maas **from** personel;

Tekrarlı Kayıtların Ortadan Kaldırılması (distinct)

Bir sorgu sonucunda çıktıda aynı kayıtların bir defa yazılması gereken durumlar olabilir. Örneğin öğrenci bilgilerine ait bir tablodan öğrencilerin kayıtlı oldukları bölümleri listelemek isteyebilirsiniz. Aşağıdaki "ogrenci" isimli tablodan öğrencilerin okudukları bölümleri listeleyelim.

OgrNo	Ad	Soyad	Bölüm
96157015	Hüseyin	Babaoğlu	Turizm ve Otelcilik
96157017	Nurhan	Delice	Büro Yönetimi ve Sekreterlik
96157001	Güven	Toptaş	Turizm ve Otelcilik
96643003	Emel	Yakıcı	Büro Yönetimi ve Sekreterlik
96643004	Mahmut	Çelebi	Büro Yönetimi ve Sekreterlik
97133002	Rıdvan	Nas	Bilgisayar Programcılığı
97133003	Nevriye	Özcan	Bilgisayar Programcılığı
97133004	Ayhan	Ölmez	Bilgisayar Programcılığı

Bu tablo için "**Select** bolum **from** ogrenci " dersek aşağıdaki çıktı listelenir.

Bölüm
Turizm ve Otelcilik
Büro Yönetimi ve Sekreterlik
Turizm ve Otelcilik
Büro Yönetimi ve Sekreterlik
Büro Yönetimi ve Sekreterlik
Bilgisayar Programcılığı
Bilgisayar Programcılığı
Bilgisayar Programcılığı

Bu tablodan bölümlerin sadece birer kere listelenmesi için aşağıdaki sorgu yazılmalıdır.

Select distinct bolum **from** ogrenci;

Bölüm
Turizm ve Otelcilik
Büro Yönetimi ve Sekreterlik
Bilgisayar Programcılığı

Tekrarlı kayıtları kaldırmak için **distinct** sözcüğü kullanılır.

Bir Şarta Bağlı Olarak Sorgulama (Where)

Bir tablodan veri sorgulama işlemi için select komutu kullanılıyordu. Ancak select komutu en sade kullanımı ile tablodaki bütün kayıtları listeler. Bir tablodan belli bir şartı sağlan kayıtların listelenmesi işlemi ise SQL dilinin mantığını vermektedir. Bir şarta bağlı olarak sorgulama ise **where** sözcüğü ile yapılmaktadır. En sade yazım şekli şöyledir:

Select kolon_isimleri **from** tablo_isimleri **where** şart_ifadesi;

Bu yazımda şart ifadesi ile belirtilen ifadeye uyan tüm kayıtlar listelenecektir.

- **Select** * **from** personel **where** maas>1000;→ sorgusu personel tablosunda maası 1000 den yüksek olan personelin kayıtlarının listelenmesini sağlayacaktır.
- **Select** * **from** ogrenci **where** ad='Ahmet'; →bu sorgu ogrenci tablosunda adı Ahmet olan kişilerin listelenmesini sağlayacaktır.

Sorgu Sonuçlarının Sıralanması

Sorgu komutları sonucundaki verilerin belli bir alana göre alfabetik olarak artan-azalan sırada veya büyükten küçüğe, küçükten büyüğe doğru sıralamak mümkündür. Bu işlem için "order by" sözcüğü kullanılmaktadır. En sade yazım şekli şöyledir.

Select kolon_listesi **from** tablo_ismi **order by** sıralamanın_yapılacağı_kolon_ismi;

Örneğin aşağıdaki ifade "personel" tablosunu bir şarta bağlı olmaksızın tüm kolonlarını listelemektedir. Fakat bu listeleme sonucundaki verileri personel adına göre alfabetik olarak artan sırada listeler.

Select * **from** personel **order by** ad;

Aşağıdaki ifade, Ogrenci tablosundaki bütün öğrenci bilgilerini ad ve soyad alanlarına göre artan sırada, yani A'dan Z'ye doğru listeleyecektir. Order by komutundan sonra verilen ilk alan en öncelikli alandır. Veriler ilk önce bu alana daha sonra diğer alanlara göre sıralanacaktır. Aynı ada sahip öğrenciler soyadları dikkate alınarak sıraya konacaktır.

Select ad, soyad, ortalama **from** Ogrenci **order by** ad, soyad;

Bütün öğrencilerin öğrenci numarası, adı ve soyadı bilgileri ortalama puanlarına göre azalan sırada listelenecektir. Listenin en başında birinci olan öğrenci yer alacaktır.

Select ogrno, ad, soyad **from** Ogrenci **order by** ortalama **desc**;

Bir alana göre artan olarak sıralamak için **asc**, azalan olarak sıralamak için **desc** ifadeleri kullanılmaktadır.

Gruplandırarak Sorgulama (Group By)
Bir tablodaki veriler üzerinde gruplandırarak işlem yapılabilir. Örneğin bir öğrenci tablosundan bölümlere göre başarı ortalaması hesaplanabilir. Gruplama işlemi için "group by" sözcüğü kullanılır. En sade yazım şekli şöyledir.

Select kolon_ismi **from** tablo_ismi **group by** kolon_ismi;

SİLME SORGUSU (DELETE)

Tablodan bir veya birden fazla satırı (kaydı) silmek için kullanılır. En sade yazım şekli şöyledir:

Delete from tablo_ismi;

Yukarıdaki ifade "tablo_ismi" ile belirtilen tablodaki bütün kayıtları silerek tabloyu boşaltır. DELETE komutu ile silinen veriler ROLLBACK komutu ile geri alınabilmektedir.
Bir tablodaki kayıtların tamamını silmenin bir yolu da TRUNCATE komutunu kullanmaktır. Bu komut bir tablo veya cluster'dan bütün kayıtları siler. Ancak bu komutun geri dönüşü yoktur. Bu nedenle kullanırken dikkatli olmak gerekir. En sade kullanımı aşağıdaki gibidir.

Truncate table ogr;

Tablodan sadece belirli kayıtları silmekte mümkündür. Bunun için where sözcüğü ile silinecek kayıtları sınırlandırabiliriz. Belli bir şarta uyan kayıtların silinmesi için kullanılan format şöyledir:

Delete from tablo_ismi **where** şart;

Personel isimli tablodan maaşı 1000 TL altındaki personeli çıkaran ifadeyi yazalım.

Delete from personel **where** maas<1000;

Veritabanı tablolarında silinen kayıtlar tamamen tablodan çıkarılmaz. Bu kayıtlar zamanla tabloyu şişirmekte ve gerçek boyutundan farklı göstermektedir. Dolayısıyla zaman zaman silinen bu kayıtların tablodan tamamen çıkarılması gerekir. Bu işlem için veritabanı yönetim sistemleri değişik yöntemler kullanmaktadır.

EKLEME SORGUSU (INSERT)
Tabloya bir veya daha fazla yeni kayıt eklemek için INSERT komutu kullanılır. Komutun yazım şekli şöyledir:

Insert into tablo (kolon_listesi) **values** (yeni_kayıt_veriler)

Yukarıdaki ifade tablo ile belirtilen tablonun kolon_listesindeki kolonlara yeni_kayıt_verileri alanındaki değerleri atar. Eğer ifadede kolon_listesi belirtilmez ise values alanındaki yeni_kayıt_verileri kısmına girilen veriler tablodaki kolon sırasında göre kaydedilir. Aşağıda görülen personel isimli tabloya yeni bir veri girelim.

SicilNo	Ad	Soyad	DTarihi	Adres	İli	Maas
9912014	Ülkü	Yalçın	01/02/1980	Yukarı M 2. Sokak	Ankara	1250
9812010	Kutbettin	Gündüz	24/04/1980	Aşağı m.	Van	2200
9812015	Hatice	Karakuş	04/05/1984	Dumlu M 34	Konya	1800
9812016	Mehmet	Kaya	06/07/1981	Kanık M Bahçe	Erzurum	3500
9812017	Fuat	Cünedi	08/10/1979	Yurd M 5. Bahçe	Niğde	2100

Tabloda 7 alan bulunmaktadır. Her alan için verilen yeni kayıt aşağıdaki gibidir.

Insert into personel (SicilNo, Ad, Soyad, Dtarihi, Adres, İli, Maas) **values** ('98112007', 'Emin', 'Tüylüce', '12/10/1970', 'Fatih M 32. sokak', 'Edirne', 4500);

"ogrenci" tablosundaki bölümü 'bilgisayar' olan öğrencilere ait tüm bilgileri "ogrenci" tablosu ile aynı yapıdaki "bilg_ogrenci" tablosuna eklemek için insert komutunu aşağıdaki gibi kullanabiliriz. ("bilgisayar" bölümünün 2 nolu bölüm olduğu varsayılmıştır.)

Insert into ogrenci (ogrno, ad, soyad, dtarihi, bno) select (ogrno, ad, soyad, dtarihi, bno) from ogrenci where bno=2;

GÜNCELLEŞTİRME SORGUSU (UPDATE)

Tablonun bir veya birden fazla kolonundaki veriyi yine tablo üzerinde değiştirerek günceller. Değişikliğe uğrayacak kolonlar "set" sözcüğünden sonra belirtilir ve birden fazla değişiklik için aralarında virgül kullanılır. Yazım şekli şöyledir.

Update tablo **set** kolon=yeni_değer;

Birden fazla kolonda değişiklik yapmak için;

Update tablo **set** kolon1=yeni_deger1, kolon2=yeni_deger2,.....;

Aşağıdaki ifade "personel" tablosundaki bütün personelin ilini "Eskişehir" olarak değiştirir.

Update personel set ili="Eskişehir";

Personel tablosundaki maas alanına %15 zam yapan güncelleme ifadesini yazalım:

Update personel **set** maas=maas+maas*15/100;

Güncelleme ifadelerinde değişikliklerin yapılacağı kolonu where ifadesi ile sınırlandırmak mümkündür. Bunun için aşağıdaki yapı kullanılır:

Update tablo **set** kolon=yeni_deger **where** şart;

OPERATÖRLER

SQL ifadelerinde aritmetiksel (+ ,-, /, *) ve mantıksal (AND, OR, NOT) operatörler kullanılmaktadır. Aritmetiksel operatörler genelde sorgu sonucunda hesaplanmış alanlar elde etmek için ve select ile from arasında kullanılır. Mantıksal ifadeler ise daha çok karşılaştırma ve şart ifadelerinde where sözcüğünden sonra kullanılır. Kullanılan operatörler kategorilere göre aşağıdaki gibidir;

Tipi	Operatörler
Aritmetik	+, -, *, /
Karşılaştırma	<, >, =, <>, >=, =<
Mantıksal	AND, OR, NOT
Karakter Katarlarını Birleştirme	||
SQL Operatörleri	Between, in, like, is null, is not null, some, any, all

Aritmetiksel Operatörler

Aritmetiksel operatörler toplama (+), fark alma (-), bölme (/) ve çarpma (*) işlemlerinde kullanılır. Örneğin maaş verilerinin 1.1 ile çarpılması;

Select ad, soyad, maas*1.1 **from** personel;

Operatörlerin öncelik sıraları *, /, +, - şeklindedir. Çarpma ve bölme işlemlerinin önceliği aynıdır. Aynı şekilde toplama ve çıkarma işlemlerinin önceliği de aynıdır. Aynı öncelikli operatörlerin kullanılması durumunda öncelik sırası soldan sağa doğrudur. İşlemleri parantez içerisine alma öncelik sıralarını değiştirir.

Karşılaştırma Operatörleri

İki değeri birbiriyle kıyaslamak için karılaştırma operatörleri kullanılır. Karşılaştırma operatörleri; Eşittir (=), eşit değildir (!=, ^=, <>), büyüktür (>, !<, ^<), büyük eşittir (>=), küçüktür (<,!>, ^>), küçük eşittir (<=).
Aşağıdaki satırda "ad" alanındaki değer ile "Ali" değeri karşılaştırılmaktadır.
Select ogrno, ad, soyad **from** ogrenci **where** ad='Ali';

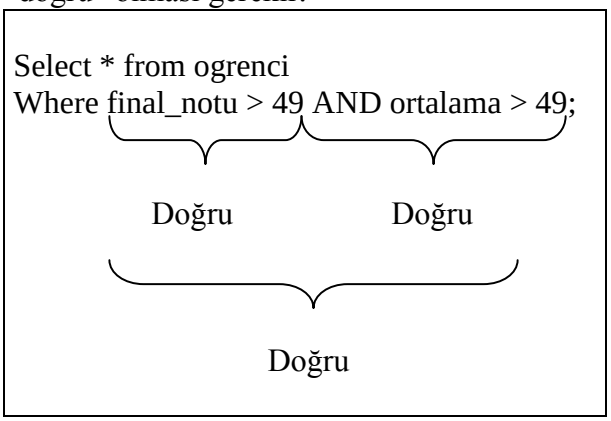
Aşağıdaki satırda iki sayısal değerin karşılaştırılması verilmiştir. Karşılaştırma operatörü olarak (>) büyüktür operatörü kullanılmıştır.
Select sicilno, ad, soyad **from** personel **where** maas>6500;

Mantıksal Operatörler
Mantıksal operatörler tek where sözcüğünde bir veya birden fazla koşul ifadelerini bağlamak için kullanılır. Bu operatörler iki koşul ifadesinden elde edilen boolean değerleri yine karşılaştırarak yeni bir boolean değer üretir. Operatörlerin çeşitli boolean değerlere karşılık verdiği sonuçlar aşağıdaki gibidir:

Koşul 1	Koşul 2	Koşul1 AND koşul2
yanlış	yanlış	yanlış
yanlış	doğru	yanlış
doğru	yanlış	yanlış
doğru	doğru	doğru

Koşul 1	Koşul 2	Koşul1 AND koşul2
0	0	0
0	1	0
1	0	0
1	1	1

Yukarıdaki tabloda verilen AND operatörü matematiksel olarak çarpma gibi işlem görür. Dolayısıyla üretilecek çıktının "doğru" değerine sahip olması için her iki koşul ifadesinin de "doğru" olması gerekir.



Yukarıdaki sorgu ifadesinde başarılı öğrenciler listelenmek istenmiştir. Bir öğrencinin başarılı sayılabilmesi, bu öğrencinin o dersin final notunun ve ortalamasının 49 dan büyük olması ile mümkündür. OR operatörüne ait doğruluk tabloları aşağıdaki gibidir.

Koşul 1	Koşul 2	Koşul1 ORkoşul2
0	0	0
0	1	1
1	0	1
1	1	1

Koşul 1	Koşul 2	Koşul1 OR koşul2
yanlış	yanlış	yanlış
yanlış	doğru	doğru
doğru	yanlış	doğru
doğru	doğru	doğru

NOT operatörüne ait doğruluk tabloları aşağıdaki gibidir.

Koşul	NOT Koşul
yanlış	doğru
doğru	yanlış

Koşul	NOT Koşul
0	1
1	0

Karakter Katarlarını Birleştirme

Belirtilen iki alandaki karakter katarlarını tek bir alan altında birleştirmek için || sembolü kullanılır.

Select ad **| |** soyad personel **from** personel;

Yukarıdaki satır ile "personel" tablosundan "ad" ve "soyad" alanları "personel" başlığı altında birleştirilmiştir. Aynı satırda birden fazla alan ve karakter katarı birleştirmekte mümkündür. Örneğin yukarıdaki ifadede ad ve soyad alanlarının arasına boşluk koymak için aşağıdaki ifade kullanılır.

Select ad **||** ' '**||** soyad personel **from** personel;

SQL Operatörleri

SQL ifadelerinde kullanılan operatörler şunlardır:

Sözcükler&Operatörler	İşlevi
Between	İki değer arasındaki verileri temsil eder.
Exists	Arama sorgusu veya bir altsorgu sonucunda dönen bir değerin var olup olmadığını tanımlar.
In	Bir verinin tabloda veya bir sayısal veri listesinde olup olmadığını tanımlar.
Like	Bir veri parçasının diğer bir veri içerisinde olup olmadığını test eder.
Is null	Bir veriyi boş veya NULL değer ile karşılaştırır.
Some/any/all	Karşılaştırmaların miktarını ölçer.

Between

Karşılaştırılacak bir değerin belirtilen bir aralığa düşüp düşmediğini tanımlar. En sade yazım şekli şöyledir.

Tablo_alani **between** değer1 **and** değer 2;

Yukarıdaki ifade Tablo_alani ile belirtilen alandaki değerin değer1 ve değer2 arasında olup olmadığını test eder. Ayrıca bu aralığında dışına düşen bir değer ise aşağıdaki gibi test edilir.

Tablo_alani **not between** değer1 **and** değer 2;

Yukarıdaki ilk ifadede between eğer tablo_alani içindeki değer, değer1 ve değer2 aralığına düşüyorsa veya değer1 veya değer2 değerlerine eşit olursa TRUE değerini döndürür. Eğer Tablo_alani değer1'den küçük veya değer2'den büyük ise between FALSE döndürür.

Finalnotu **between** 50 **and** 100;

Between BLOB olmayan veri tipleri ile kullanılabilir. Ancak karşılaştırılacak değerler ve aralık değerler aynı veri tipinde veya uyumlu veri tipinde olmalıdır. Birbiri ile uyumsuz veri tipleri için *cast* fonksiyonu kullanılabilir.

Select dtarihi **from** personel **where** (dtarihi **between** "01/01/1970" **and** "31/12/1979")

Yukarıdaki ifade doğum tarihi 70'li yıllara denk gelen personelin doğum tarihlerini listeler.
Between, belirtilen bir aralığa düşebilecek süreklilik gösterebilecek verilerin filtrelenmesinde oldukça kullanışlıdır. Eğer belli bir aralık süreklilik göstermeyen bir değer ile filtreleme yapılacaksa *in* kullanılmalıdır.

Exists
Bir alt sorguda bir değerin var olup olmadığını gösterir. Yazım şekli şöyledir.
Exists altsorgu

Exists bir altsorgudaki tablodan var olan değerlere bağlı olarak filtreleme yapmak için karşılaştırma olarak kullanılır. Eğer alt sorgu sonuç kümesinde en az bir kayıt mevcut ise exists sözcüğü true döndürür. Altsorgudan hiç kayıt çıktısı alınamaz ise exists sözcüğü false değerini döndürür.

Select s.sipno, s.musno **from** siparis s
Where exists(**select** m.müsno **from** musteri m, siparis s **where**(m.müsno=s.müsno));
Yukarıdaki ifade bir müşteriye ait sipariş numaralarını ve müşteri numarasını verir. Ancak bu altsorgu ile siparişi olan bir müşterinin varlığı kontrol edilir.
Exists karşılaştırma ifadelerinde not sözcüğü kullanılabilir.
Select * from personel **where not exists**(.....)

In
Bir veri kümesinde bir değerin var olup olmadığını gösterir. Yazım şekli şöyledir:
Tablo_alani **in** (veri_kümesi) veya tablo_alani **not in** (veri_kümesi)

Veri kümesi ile değişken veya sabit veriler kullanılabilir. Bir alt sorgu sonucundaki verileri alan değişken bir veri kümesi olabilir. Eğer sabit değerlerden oluşan bir veri kümesi kullanılacaksa her sabit değer arasında virgül kullanılır.

Final_notu in (50, 60) bu ifade Final_notu = 50 or Final_notu = 60 ifadesine eşittir.

Veri kümesindeki her değer or operatörü ile karşılaştırılır ve karşılaştırılan tablo alanının veri kümesindeki bir değere eşit olması şartın sağlanması için yeterlidir. Bu komutla birlikte karakter değerlerin karşılaştırılması için, değerler tırnak içine alınır.

Ad **in** ('Ahmet', 'Mehmet', 'Ali')

Like
Karşılaştırma ifadelerinde sıkça kullanılan sözcüklerden biriside *like* sözcüğüdür. Like sadece CHAR veri tipindeki veya uyumlu tipteki veri tipleri ile kullanılabilir. Bu sözcük iki karakter

katarını veya bir alt string karşılaştırır. Karşılaştırmalarda joker (*,_, ?) karakterlerden yararlanılabilmektedir. Yazım şekli şöyledir.
Değer **like** karşılaştırılacak_altdeğer

Örneğin aşağıdaki ifade personel tablosundan adı "ahmet" olan kayıtları listeler. Bu listede "ahmet can", "ahmet salih" gibi iki isimli kayıtlar yer almayacaktır. Karşılaştırma birebir yapılmaktadır.
Select * from personel **where** (ad like 'ahmet')

Birebir olmayan karşılaştırmalar için joker karakterlerden faydalanılır. Örneğin aşağıdaki ifade ismi "a" harfi ile başlayan personel kayıtlarını listeler.
Select * from personel **where** (ad **like** 'a%');
Bu listede "ahmet can", "ahmet", "ali" gibi "a" ile başlayan tüm personel yer alacaktır.

Diğer bir joker karakter altçizgi "_" karakteridir. Bu karakter karşılaştırmada sadece bir karakter için joker olarak kullanılır.

Select * from personel **where** (ad like 'c_n');

Yukarıdaki ifade ad kolonundaki verilerden birinci harfi "c" ve üçüncü harfi "n" olan tüm kayıtları listeler. Örneğin bu listede "can", "cin", "con" gibi veriler yer alabilir. Ancak "cem" gibi veriler yer alamaz.

Select * from personel **where** ad **like** _____
Yukarıdaki satırda like sözcüğünden sonra 5 tane alt çizgi yanyana kullanılmıştır. Bu sorguyla adı 5 harften oluşan personel listelenmiştir.
Is null
Bir kolonun NULL değer içerip içermediğini belirtir. Yazım şekli şöyledir;
Kolon_ismi is null;
Kolon_ismi is not null;

Örneğin şirket bilgisi eksik olan müşterilerin listesini almak için aşağıdaki ifade kullanılabilir.
Select * from customer **where** (company **is null**);

Some/any/all
Bir altsorgudan dönen çoklu verilerdeki bir kolon değerini başka bir kolon değeri ile karşılaştırır.
Kolon_ismi karşılaştırma_operatörü **some** (altsorgu)
Kolon_ismi karşılaştırma_operatörü **any** (altsorgu)
Kolon_ismi karşılaştırma_operatörü **all** (altsorgu)

FONKSİYONLAR
Fonksiyonlar, veriler üzerinde hesaplamalar yapmak, tarih verileri üzerinde değişiklikler yapmak, veri tipleri arasında dönüşüm yapmak gibi birçok amaç için kullanılan komutlardır. Bu komutlar şu kategorilere ayrılır.

Grup	Karakter	Sayı	Tarih	Veri tipi
Avg	İnitcap	abs	Add_months	Decode
Count	Instr	ceil	Last_day	Greatest
Max	Length	floor	Months_between	Least
Min	Lower	mod	Next_day	nvl
Stddev	Lpad	power	Round	To_char

Sum	Ltrim	round	To_char	To_date
variance	Rpad	sign	trunc	To_number
	Rtrim	sqrt		vsize
	Soundex	trunc		
	Substr			
	Translate			
	Trim			
	Upper			

Tek Değer Döndüren Fonksiyonlar
Grup fonksiyonları da denen bu tür fonksiyonlar çıktıda sadece bir tek değer döndürür ve çıktıları özet bilgi olarak kullanılır.örneğin bir tablodaki kayıt sayısı veya en yüksek ortalama gibi.

Fonksiyon	**İşlevi**
Avg	Bir kolondaki NULL olmayan bütün sayısal verilerin ortalamasını alır.
Count	Sorgu sonucunda listelenebilecek kayıtların sayısını verir.
Max	Bir kolondaki en büyük sayısal değeri döndürür.
Min	Bir kolondaki en küçük sayısal değeri döndürür.
Stddev	Bir kolondaki NULL olmayan bütün sayısal verilerin standart sapmasını verir.
Sum	Bir kolondaki NULL olmayan bütün sayısal değerlerin toplamını verir.
variance	Bir kolondaki NULL olmayan bütün sayısal verilerin varyansını verir.

Avg
Belirtilen bir kolondaki değerlerin ortalamasını alır. Yazım şekli şöyledir;
Avg (kolon_ismi)
Avg (ALL kolon_ismi)
Avg (DISTINCT kolon_ismi)

Avg fonksiyonu da sum fonksiyonu gibi sadece sayısal değerler ile çalışır. Sayısal olmayan değerler için cast fonksiyonu yardımıyla sayısal bir veri tipine çevrim yapılmalıdır.

Select avg(maas) **as** ortalama_maas **from** personel;

Count
Bir sorgu sonucu listelenebilecek kayıt sayısını döndürür. Yazım şekli şöyledir**.**
Count (*)
Count (kolon_ismi)
Count (ALL kolon_ismi)
Count (DISTINCT kolon_ismi)

Fonksiyonda asteriks(*) işareti kullanılması sayım işleminde bütün satırların dikkate alınmasını gösterir. Bundan başka sadece kolon ismi belirtilirse sadece o kolon dikkate alınır. Fonksiyonda ALL kullanılması * işareti gibi bütün satırların sayımını gösterir. DISTINCT ise ilgili kolondaki aynı kayıtların sadece bir kez sayılmasını gösterir. Eğer fonksiyonda DISTINCT belirtilmemişse varsayılan olarak ALL kabul edilir.

Ogrenci

OgrNo	**Ad**	**Soyad**	**Bolum**
98118027	Erdal	Yıldız	Bilgisayar Programcılığı
98112001	M.Salih	Bulut	İşletmecilik

98112002	M. Fatih	Yıldırır	Turizm ve Otelcilik
98112003	M. Emin	Portek	Bilgisayar Programcılığı
98112004	Özlem	Yorulmaz	Turizm ve Otelcilik
98112005	Salih	Eskioğlu	Bilgisayar Programcılığı
98112022			

Tablodaki kayıt sayısı ;

```
Select count (*) from ogrenci;
```

Çıktısı;

Count (*)
7

Tablodaki öğrenci sayısı;

```
Select count (ad) as Öğrenci_sayısı from ogrenci;
```

Veya

```
Select count (all ad) as ogrenci_sayısı from ogrenci;
```

Çıktısı;

Öğrenci_sayısı
6

Count fonksiyonu kolonlardaki NULL değerlerini sayım sonucuna eklemez.
Tablodaki bölüm sayısını veren ifade;

```
Select count (distinct bolum) as bölüm_sayısı from ogrenci;
```

Çıktısı;

Bölüm_sayısı
3

Distinct sözcüğü ile kolonlardaki tekrar eden verilerin sayılması engellenebilir.
Tabloda tamamlanmamış kayıt sayısı ;

```
Select count(*) as eksik_kayıt_sayısı from ogrenci where (ad is null);
```

Çıktısı;

eksik_kayıt _sayısı
1

Max

Belirtilen bir veri kümesindeki en büyük değeri döndürür. Veri kümesi bir tablo, filtrelenmiş veriler veya GROUP BY sözcüğü ile gruplandırılmış veriler olabilir. Yazım şekli, şöyledir;
Max (kolon_ismi)
Max (ALL kolon_ismi)
Max (DISTINCT kolon_ismi)

Max fonksiyonu BLOB veri tipine sahip olmayan bütün alanlar için kullanılabilir. Fonksiyondan dönen değer kolon_ismi ile belirtilen alandaki veri tipi ile aynı veri tipine sahiptir. Eğer CHAR veri tipinden bir alan üzerinde işlem yapılacaksa dönen değer yine aynı tipte olacak ve BDE'nin kullandığı dil sürücülerine bağlı bir sonuç üretecektir.

Select max(maas) **as** en_yuksek_maas **from** personel;

Min
Belirtilen bir veri kümesindeki en küçük değeri döndürür. Veri kümesi bir tablo, filtrelenmiş veriler veya GROUP BY sözcüğü ile gruplandırılmış veriler olabilir. Yazım şekli, şöyledir;
Min (kolon_ismi)
Min (ALL kolon_ismi)
Min (DISTINCT kolon_ismi)

Min fonksiyonu BLOB veri tipine sahip olmayan bütün alanlar için kullanılabilir. Fonksiyondan dönen değer kolon_ismi ile belirtilen alandaki veri tipi ile aynı veri tipine sahiptir. Eğer CHAR veri tipinden bir alan üzerinde işlem yapılacaksa dönen değer yine aynı tipte olacak ve BDE'nin kullandığı dil sürücülerine bağlı bir sonuç üretecektir. Karakter veri türlerinde genelde rakamlar harflerden önce gelmektedir. Rakamsal değerler bulunan karakter veri tipindeki bir alanda minimum değerler 0,1,2….. şeklinde gider.

Select min(maas) **as** en_düşük_maas **from** personel;

Stddev
Bir kolondaki NULL değere sahip olmayan bütün sayısal verilerin standart sapmasını hesaplar.

```
SQL > select Stddev (sal) from emp;

              STDDEV (SAL)
              --------------------
              1182,50322

SQL>
```

Sum
Belirtilen bir kolondaki bütün değerlerin toplamını hesaplar.
Sum (kolon_ismi)
Sum (ALL kolon_ismi)
Sum (DISTINCT kolon_ismi)

Sum fonksiyonu sadece sayısal değerler üzerinde işlem yapar. Sayısal olmayan değerler içim cast fonksiyonu yardımıyla sayısal bir veri tipine çevrim yapılmalıdır.

Select sum(maas) **as** toplam_maas **from** personel;

Sum fonksiyonu kolonlar üzerinde aritmetiksel ifadeleri destekler. Aşağıdaki gibi aritmetiksel işlemleri sum fonksiyonu gerçekleştirmek mümkündür.
Sum (kolon*5)
Sum (kolon)*5
Sum (kolon1 + kolon2)
Sum (kolon1 * kolon2)

Variance

Bir kolondaki NULL olmayan bütün sayısal verilerin varyansını hesaplar.

```
SQL> select variance (comm) from emp;

                VARIANCE (COMM)
                ---------------------------
                363333,333
SQL>
```

Group By Sözcüğünün Kullanımı
Bir tablodaki veriler üzerinde gruplandırma yapan GROUP BY sözcüğünün kullanımını görmüştük. Bu sözcüğün en yaygın kullanımı, tek değer döndüren fonksiyonlar ile olan kullanımıdır.

Select job, **avg**(sal) ortalama **from** emp **group by** job;

Having Sözcüğünün Kullanımı
Gruplar için koşul belirterek tek değer döndüren fonksiyonları kullanmak için HAVING sözcüğünden yararlanılır.

Select job, **max**(sal) **from** emp **group by** job **having count**(*) = 4;

Karakter Katarı Fonksiyonları
Bu kategoride karakter katarı türü veriler üzerinde işlem gören fonksiyonlar yer almaktadır.

İnitcap
Bir kelimenin veya bir cümlede geçen tüm kelimelerin baş harflerini büyük harfe çevirir.

```
SQL> select İnitcap (‘mustafa yusuf daşdemir’ ) “İLK HARFLER BÜYÜK” from dual;

İLK HARFLER BÜYÜK
-----------------------------
Mustafa Yusuf Daşdemir

SQL>
```

Instr
Belirtilen bir karakter veya karakter katarının, başka bir karakter katarındaki veya bir tablo verisindeki konumunu verir. Aşağıda ‘Oracle Developer’ karakter katarı içerisinde ‘c’ harfinin ve ‘dev’ karakter katarının konumları bulunmuştur.

```
SQL> select instr (‘Oracle Developer’, ‘c’) sırası from dual;

SIRASI
----------
   4

select instr (‘Oracle Developer’, ‘Dev’) sırası from dual;

SIRASI
----------
   8

SQL>
```

Length
Belirtilen bir karakter katarındaki veya tablo alanındaki verilerin karakter sayısını verir. Verilen karakter katarındaki boşluklarda sayılmaktadır.

```
SQL> select length ( ‘mustafa yusuf daşdemir’) “toplam karakter” from dual;

    Toplam karakter
    ---------------------
                  22

SQL>
```

Lower
Belirtilen bir karakter katarındaki veya bir tablo alanındaki verileri küçük harflere dönüştürür.

```
SQL>select Lower (‘oracle, DB2, Sql, sERvER’) “küçük harfler” from dual;

Küçük harfler
-----------------------------------------
oracle, db2, sql, server

SQL>
```

Lpad
Bu fonksiyon sabit karakter katarı veya tablo alanı için ikinci parametresinde belirtilen uzunluk kadar yer ayırır. Daha sonra verileri sağa hizalar ve solda kalan boşlukları üçüncü parametresinde belirtilen karakter veya karakter katarı ile doldurur.

```
SQL> select lpad (‘Oracle’, 10, ‘*’) doldur from dual;

Doldur
------------------
****Oracle

SQL>
```

Ltrim

Bu fonksiyon ikinci parametresinde belirtilen harfleri bir karakter katarının veya bir tablo alanındaki verilerin solunda arar. Bu harfleri bulunduran verilerden bu harfler silinir.

```
SQL>select Ltrim (‘Yüksekokul’, ‘yüksek’) buda from dual;
Buda
------------
Okul

SQL>
```

Rpad

Bu fonksiyon sabit karakter katarı veya tablo alanı için ikinci parametresinde belirtilen uzunluk kadar yer ayırır. Daha sonra verileri sola hizalar ve sağda kalan boşlukları üçüncü parametresinde belirtilen karakter veya karakter katarı ile doldurur.

```
SQL> select rpad (‘Oracle’, 10, ‘*’) doldur from dual;

   Doldur
   ------------------
   Oracle****

   SQL>
```

Rtrim

Bu fonksiyon ikinci parametresinde belirtilen harfleri bir karakter katarının veya bir tablo alanındaki verilerin sağında arar. Bu harfleri bulunduran verilerden bu harfler silinir.

```
SQL>select Ltrim (‘Yüksekokul’, ‘okul’) buda from dual;
Buda
------------
Yükse

SQL>
```

Soundex

Bu fonksiyon ile ingilizce de bazı kelimeler fonetik olarak temsil edilebilir ve bu şekilde farklı telaffuz edilebilen kelimeler kıyaslanabilir.

```
SQL> select empno, ename, job from emp
        where soundex (ename) = soundex ('skat');

EMPNO    ENAME       JOB
----------  ----------    ------------------
 7788              SCOTT       ANALYST

SQL>
```

Substr
Bir karakter katarından, belirtilen konumundan itibaren yine belirtilen miktar kadar karakter alınır. Genel yazımı şöyledir;

Substr (tabloalani, konum, miktar)

"tabloalani" ile belirtilen alandaki verinin, "konum" ile belirtilen yerinden itibaren "miktar" kadar alınarak yeni bir karakter katarı elde edilir.

```
SQL> select substr ('Meslek Yüksekokulu', 8,6) from dual;

Substr
-----------
Yüksek

SQL>
```

Translate
Bir karakter katarındaki veya bir tablo alanındaki veride geçen bir karakteri başka bir karakter ile değiştirir. Aşağıdaki örnekte "CASE" metninde "C" harfleri "K" harfleri ile yer değiştirmektedir.

```
SQL> select translate ('CASE', 'C', 'K') from dual;

      TRAN
      -----------
      KASE
SQL>
```

Değiştirilecek karakter birden fazla olabilir. Ancak sırası önemlidir.

Trim
Bir karakter katarının başından, sonundan veya hem başından hem sonundan belirtilen karakteri siler.

```
SQL> select trim(leading (‘O’ from ‘Oracle’) soldan from dual;

SOLDA
------------
racle

SQL>
```

```
SQL> select trim(trailing ‘e’ from ‘Oracle’) sağdan from
dual;
SAĞDA
----------
Oracl

SQL>
```

Upper
Belirtilen bir karakter katarındaki veya bir tablo alanındaki verileri büyük harflere dönüştürür. Aşağıda sabit karakter verilerin büyük harflere dönüştürülmesi verilmiştir.

```
SQL> select upper (‘oracle, DB2, Sql, sERvER’) “Büyük Harfler” from dual;

Büyük Harfler
--------------------
ORACLE, DB2, SQL, SERVER

SQL>
```

Sayısal Değerler İle İlgili Fonksiyonlar
Bu fonksiyonlar sayısal veriler üzerinde işlem yapmaktadır. Bu tür fonksiyonlar hem isimleri hem de işlevleri bakımından birçok programlama dilinde aynıdır.

Abs
Bir tablo alanındaki sayısal verinin veya bir sayısal değerin mutlak değerini verir.

```
SQL> select abs (-123), abs (123) from dual;

ABS(-123)          ABS(123)
---------------    ----------------
      123                123

SQL>
```

Ceil
Verilen bir sayısal değerden veya bir tablo alanındaki sayısal değerden büyük olan en küçük tamsayıyı verir. Değer tamsayı ise kendisine, ondalık sayı ise kendinden sonraki tamsayıya eşitlenir.

```
SQL> select ceil (4), ceil (9.0001), ceil (9.99) from dual;

CEIL(4)          CEIL(9.0001)          CEIL(9.99)
-----------      ------------------    ----------------
    4                 10                    10

SQL>
```

Floor
Verilen bir sayısal değerden veya bir tablo alanındaki sayısal değerden küçük olan en büyük tam sayıyı verir. Değer tamsayı ise kendisine, ondalık ise kendisinden önceki tamsayıya eşitlenir.

```
SQL> select floor (4) t1, floor (9.0001) t2, floor (9.99) t3 from
dual;
T1           T2             T3
---------    -------------  ------------
 4           9              9

SQL>
```

Mod
İki sayısal değerin veya iki tablo alanındaki değerin birbirine bölümü sonrasındaki kalanı verir.

```
SQL> select mod (7, 2), mod (-9, 2), mod (9, 3)  from dual;

MOD (7,2)       MOD(-9, 2)       MOD(9, 3)
-------------------------------------------------
       1               -1               0

SQL>
```

Power
İki parametre alan bu fonksiyon, ilk parametre değerinin ikinci parametre değeri kadar üssünü alır.

```
SQL> select power (2, 0), power (2, 2), power (2, 6)  from dual;

power (2, 0)       power (2, 2)        power (2, 6)
--------------     --------------      ----------------
     1                  4                   64

SQL>
```

Round

Sayısal bir değeri veya bir tablo alanındaki sayısal değeri verilen basamak sayısı kadar en yakın basamağa yuvarlar.

```
SQL> select round (4653.485, 2) y1, round (4653.481, -2) y2 , round
(294.44532, 1) y3, from dual;

       Y1              Y2              Y3
     ------------    -------------   ------------
      4653,49         4700            294,4

SQL>
```

Sign

Verilen sayısal değer veya bir tablo alanındaki sayısal değer, negatif ise -1, sıfır ise 0, pozitif ise +1 sayılarını döndürür.

```
SQL> select sign (-3492), sign (0), sign (239) from dual;

sign (-3492)        sign (0)        sign (239)
---------------     ----------      ------------
      -1                      0                 1

SQL>
```

Sqrt

Verilen bir sayısal değerin karekökünü bulur.

```
SQL> select sqrt(4), sqrt(625), sqrt (99) from dual;

sqrt(4)           sqrt(625)          sqrt (99)
---------         ------------       ----------
   2                   25                9.94987437

SQL>
```

Trunc

Sayısal bir değeri veya bir tablo alanındaki sayısal değeri verilen basamağa kadar kısaltır.

```
SQL> select trunc (234.5267, 2) b1, trunc (10/3) b2 from
dual;
   B1          B2
---------      ---------
234,52                3

SQL>
```

Tarih Verileri ile İlgili Fonksiyonlar
Add_months
Verilen bir tarihe ay ekler. İki parametreye sahiptir. İlk parametreye ikinci parametrede belirtilen sayı kadar ay ekler.

```
SQL>select sysdate from dual;

SYSDATE
--------------
23/12/2003

SQL> select add_months(sysdate,3) from dual;

Add Months
----------------
23/03/2004

SQL>
```

Last_Day
Ayın son gününü verir. Tarih verisindeki gün ayın son gününü gösterecek şekilde ayarlanır.

```
SQL>select sysdate, last_day(sysdate) from dual;

   Sysdate            Last_Day
---------------       ------------------
23/12/2003            31/12/2003

SQL>
```

Months_Between
İki tarih arasındaki ay sayısını verir. Ay sayısı pozitif veya negatif çıkabilir.

```
SQL> select months_between ('01.01.2003', '01.02.2003') from dual;

Months_Between
----------------------
              -1
SQL>
```

Next_Day

Bir tarihten sonra ismi verilen bir günün geldiği ilk tarihi hesaplar. Gün ismi yerine 1 ile 7 arasında bir sayı girilebilir.

```
SQL>select next_day(sysdate, 'Cuma') from dual;

              Next_Day
              -----------------
              26/12/2003
SQL>
```

Round

Round fonksiyonu iki parametre alır. İkinci parametreye 'month' değeri verilirse, birinci parametrede verilen tarihteki ayın ilk veya bir sonraki ayın ilk gününü, 'year' değeri verilirse birinci parametredeki tarihin ilk günü veya bir sonraki yılın ilk günü bulunur. 'month' parametresi için verilen tarihte gün 15 ve üzeri ise bir sonraki ayın ilk günü, 15'ten küçük ise aynı ayın ilk günü bulunur. 'Year' parametresi verilmiş ise ilk altı aya kadar yılın ilk günü, sonraki altı ay için gelecek yılın ilk günü bulunur.

```
SQL>select tarih, round(tarih,'month'), round(tarih, 'year') from
stok;
Tarih              Month              Year
---------------    -------------------    ---------------------
10/12/2003         01/12/2003         01/01/2004
20/12/2003         01/01/2004         01/01/2004
16/12/2003         01/01/2004         01/01/2004
01/01/2004         01/01/2004         01/01/2004
08/08/2004         01/08/2004         01/01/2005

SQL>
```

To_Char

Tarih verilerini biçimlendirir. DAY biçimi için günler Pazartesi,Salı,…… şeklinde, MONTH biçimi için aylar Ocak,Şubat,…….şeklinde, DD için rakamlar ise gün YYYYiçin 4 haneli yıl yazılır.

```
SQL>select to_char (sysdate, 'DD-MONTH-YYY DAY') bugün from  dual;

Bugün
------------------------
23-ARALIK-2003 SALI

SQL>
```

Trunc
Bu fonksiyon iki parametre alır. İkinci parametreye 'month' değeri verilirse birinci parametrede verilen tarihin ayının ilk günü, ikinci parametreye 'year' verilirse birinci parametredeki tarihin yılının ilk günü hesaplanır.

```
SQL>select tarih, trunc (tarih,'month'), trunc (tarih, 'year') from stok;
Tarih              Month              Year
---------------    -------------------    ---------------------
10/12/2003         01/12/2003         01/01/2003
20/12/2003         01/12/2003         01/01/2003
16/12/2003         01/12/2003         01/01/2003
01/01/2004         01/01/2004         01/01/2004
08/08/2004         01/08/2004         01/01/2004
SQL>
```

Değişik Veri Tipleri ile İlgili Fonksiyonlar
Bu bölümdeki fonksiyonların çoğu bütün veri tipleri ile çalışabilmektedir.

Decode
Bir tablo alanındaki verileri parametre olarak belirtilen veriler ile değiştirir. Fonksiyon büyük/küçük harf ayrımı yapmaktadır.

```
SQL> select job,  decode(job, 'Manager', 'Yönetici',
          'Analyst', 'Analist', 'Tanımlanmadı') iş from emp;

JOB                    İŞ
-------------------    ----------------------
Clerk                  Tanımlanmadı
Manager                Yönetici
Salesman               Tanımlanmadı
Analyst                Analist

SQL>
```

Greatest
Bir listedeki en büyük değeri döndürür.

```
SQL> select greatest(12, 23, 59, 239, 599, 450)
     "En büyük değer" from dual;

En büyük değer
------------------
               599

SQL>
```

Least
Bir listedeki en küçük değeri döndürür.

```
SQL> select least(12, 23, 59, 239, 599, 450)
     "En küçük değer" from dual;

En küçük değer
-------------------
             12
SQL>
```

Nvl
Bir tablo alanındaki NULL değeri, ikinci parametresi ile belirtilen değere çevirir. Özellikle aritmetiksel işlemlerde NULL değerlerin işlem sonucunu etkilememesi için bu değer ayarlanabilir. Yazılımı şöyledir;

```
SQL>Select sal, comm, sal+comm, sall+nvl(comm,0) from emp;

SAL        COMM       SAL+COMM          SAL+NVL(COMM,0)
---------- ---------- ----------------- ---------------------------
800                                              800
1600       300        1900                      1900
1250       500        1750                      1750
2975                                             2975
1500       0          1500                      1500

SQL>
```

To_Date
Verilen bir karakter katarını belirli bir biçimde tarih verisine çevirir. Karakter katarının yapısı ile çevrilecek tarih veri yapısının uyuşması gerekir. Eğer tarih biçimi belirtilmezse varsayılan biçim kabul edilir.

```
SQL> select  to_date('12.02.2004','dd.mm.yy') tarih  from dual;

      Tarih
     ----------
     12/02/2004
SQL>
```

To_Number
Rakam içerek karakteri sayıya çevirir.

select to_number('100') **from** dual;

Vsize
Belirtilen bir tablo alanının veya bir değerin Oracle tarafından kaç byte ile temsil edildiğini verir.

```
SQL> select vsize('Veritabanı Yönetim Sistemleri') from dual;

vsize
---------
   29
SQL>
```

6.2 Veri Tanımlama Dili (Data Definition Language – DDL)

DDL kategorisindeki komutlar şu şekildedir.

Komutlar	İşlevi
Create table	Yeni bir tablo oluşturur.
Alter table	Belirtilen bir tablodan kolon siler veya yeni bir kolon ekler.
Drop table	Varolan bir tabloyu siler.
Create index	Yeni bir indeks oluşturur.
Alter index	Varolan bir indeks üzerinde düzenleme yapar.
Drop index	Varolan bir indeksi siler.

SQL VERİ TİPLERİ
Veri tipleri tablolarda saklanan verilerin gösterimi için kullanılan değerler kümesidir. Veri tipleri ANSI/ISO SQL standartları ile tanımlanmıştır ve her VTYS bu standartlardaki veri tiplerini kullanmışlardır. Ancak bazı VTYS'ler bu veri tiplerine kullanıcılar açısından faydalı yeni veri tipleri de eklemişlerdir. Henüz standartlaşmamış bu tür veri tipleri ile VTYS'lerin kullandığı veri tipleri arasında farklılıklar görülebilmektedir.

Karakter Katarları (String'ler)
Tablolarda isim, soy isim, adres, açıklama gibi metin türü bilgilerin saklanacağı sütunlar için bu veri tipleri kullanılır. Bu veri tiplerinden bir kısmı hafızada sabit uzunlukta bir yer işgal ederken bir kısmı değişken uzunluklu olup hafızadaki miktarı belirleyebilme imkanı sunmaktadır. Veri tipi isminde VARYING ve VAR sözcüğü geçen veri tipleri değişken

uzunluklu, NATIONAL sözcüğü geçen veri tipleri ise ulusal karakter katarları için kullanılan veri tipleridir. Bu veri tiplerine, parametre olarak girilen sayıya kadar maksimum uzunlukta karakter katarı girişleri yapılabilir. Ancak karakter katarı bu uzunluktan kısa ise kullanılan karakter katarının uzunluğu kadar yer tahsis edilir.

Karakter Katarları

Veri Tipi	Açıklama
Char(uz) Character	Uz uzunluğu kadar uzunlukta değişmez uzunluklu veri tipi
Char Varying(uz) Character Varying(uz) Varchar(uz)	Maksimum uz uzunluğu kadar uzunlukta değişken uzunluklu veri tipi
Nchar(uz) National Char(uz) National Character(uz)	Uz uzunluğu kadar uzunlukta değişmez uzunluklu ulusal karakter katarları veri tipi
Nchar Varying (uz) National Char Varying (uz) National Character Varying (uz)	Maksimum uz uzunluğu kadar uzunlukta değişken uzunluklu ulusal karakter katarları veri tipi

Tamsayılar

Tamsayılar (integer) tipindeki veri tipleri öğrenci mevcudu, malzeme miktarı, tablolar arasındaki bağlantılarda kullanılan anahtar alanlardaki verileri içerir. Bu tür alanlarda saklanacak veriler üzerinde aritmetiksel işlemler yapılabilir. Karakter katarlarına nazaran daha geniş bir değer aralığına sahiptirler. bu nedenle hafızada karakter katarlarına göre daha fazla yer işgal ederler. "1" şeklindeki karakter olarak temsil edilen bir karakter, karakter katarı olarak saklanırsa hafızada 1 byte, tamsayı olarak saklanırsa hafızadan duruma göre 4 byte harcar. Ancak karakter katarı olarak saklanmış "1" verisi ile 1+1=2 gibi bir aritmetiksel işlem gerçekleştirilemez.

Tamsayılar

Veri Tipi	Açıklama
INT INTEGER	Değişmez uzunluklu tamsayı veri tipleri.
SMALLINT	Değişmez uzunluklu küçük tamsayılar.

Reel ve Kayan Noktalı Sayılar

Ondalık haneli değerlere sahip verileri saklamak için kullanılır. Örneğin KDV oranı, yüzdelik oranlar, parasal değerler.... gibi kesirli veriler. Birçok DBMS parasal verilerin saklanması için money veya currensy gibi veri tiplerini kullanır.

Reel ve Kayan Noktalı Sayılar

Veri Tipi	Açıklama
Numeric (p, s) Decimal (p, s) Dec(p, s)	Reel sayılar için kullanılan veri tipleridir.
Float (p) Real Double Precision	Kayan noktalı sayılar için kullanılan veri tipleridir.

BIT Katarları

"0" ve "1" rakamlarından oluşan "bit" verilerini saklamak için kullanılır.

BIT Katarları

Veri Tipi	Açıklama
BIT (uz)	Uz uzunluğu kadar uzunlukta değişmez bit veri tipi.
BIT VARYING (uz)	Maksimum uz uzunluğunda değişken uzunluklu veri tipi

Tarih/Zaman Veri Tipleri

Tarih ve saat verilerini saklamak için bu veri tipleri kullanılır. Genellikle doğum tarihi, işe başlama tarihi, şimdiki tarih, sipariş tarihi gibi bilgiler için kullanılır.

Tarih/zaman Veri Tipleri

Veri Tipi	Açıklama
Date Time (p) Timestamp (p) Internal	Tarih/Saat tipindeki veriler için kullanılan veri tipleri. Date "01.01.03", Time "01:06:09", Timestamp "01.01.03. 01:06:09", internal ise zaman aralıklarını belirtmek için kullanılır.

Grafik Veri Tipleri

Grafik türü veriler veya sıkıştırılmış video görüntüleri gibi verileri saklamak için bu tip kullanılır.

Grafik Veri Tipleri

Veri Tipi	Açıklama
Graphic (uz) Vargraphic (uz)	Uz tane 16 bitlik veri içeren değişken ve değişken olmayan grafik tipleri

TABLOLAR

Veritabanı üzerinde yeni bir tablo oluşturmak için DDL komutlarından CREATE TABLE komutu kullanılır. Bu komutun en sade yazım şekli şöyledir;

Create table tablo_adi
(kolon01 veri_tipi kısıtlamalar,
kolon02 veri_tipi kısıtlamalar,
kolon01 veri_tipi kısıtlamalar
.....);

Bu ifadenin bir parametresi olan tablo_adi için seçeceğiniz ismin bazı kurallara uyması gerekir.

- Tablo isimleri A-Z veya a-z arasındaki harf ile başlamalıdır.
- İsimlerde diyez (#), alt çizgi (_) veya dolar ($) gibi bazı özel karakterleri kullanabiliriz. Ancak slash (/), plus (+), asteriks (*), space (boşluk) gibi karakterleri kullanamayız. Mümkün olduğunca tablo isimlerinde özel karakter kullanmamaya gayret etmeliyiz.
- Tablo isimleri de büyük/küçük harfe duyarlı değildir. Musteri, MUSTERI, MuSteRi hepsi aynıdır.
- Tablo isimlerinin uzunlukları DBMS'ler tarafından kısıtlanmıştır. Bu kurallara dikkat etmeliyiz.
- Verilecek olan tablo ismi veritabanındaki başka bir tablo veya görünüm ismi ile aynı olmamalıdır.

Aşağıda 6 tane alana sahip personel isimli tablonun oluşturulması için kullanılan ifade bulunmaktadır.

Create table personel (perno int, Ad char(15), Soyad char(15), Adres char(64), Dogumtarihi date, Maas double precision);

Yaratılan personel tablosuna veri girişi yapmak için aşağıdaki ifade kullanılır.
Insert into personel (perno, ad, soyad, adres, dogumtarihi, maas)
Values (1, 'Yaşar', 'Daşdemir', 'orta mah. No:42/2', !01/01/1975', 8000);

Create table komutunun bir başka kullanım şekli ise var olan bir tablodan yeni bir tablo türetmen olayıdır. Bu şekilde var olan bir tablodaki alan isimleri ve alanlara ait veri tipleri aynen alınır.
Create table calisanlar
As select * from personel;

Eğer personel tablosundan sadece belirli alanlara sahip yeni bir tablo oluşturmak istiyorsak (*) yazan kısma istediğimiz tablo adlarını yazarız.
Create table P_Ad_Soyad
As select perno, ad, soyad **from** personel;

Çalışanlar ile personel aynı alanlara sahip olduğundan personel tablosunu silelim.
Drop table personel;

İlişkisel Bütünlük
İlişkisel veritabanlarında daha fazla birbiri ile ilişkili tablo kullanıldığından bu tablolardaki verilerin tutarlılığının sağlanması gerekir. Eğer bu sağlanmaz ise birçok tablo birbiri ile alakalısı olmayan değerler ile doldurulur ki, buda veri bütünlüğünü bozar. Veri bütünlüğü ise tablolar arasındaki ilişki ile sağlanmaktadır. İlişki kurulan tablolarda bağlantılardaki ilişki tiplerine İlişkisel Bütünlük denir ve bir ilişkisel veritabanı bu özelliği sağlar.

Anahtarlar (Keys)
Tablodaki bir kaydı ilişkisel bütünlük veya indeksleme amacıyla tanımlamak için kullanılan bir kolona veya kolon setine anahtar denir.

Ana ve Yabancı Anahtarlar
Bir tablodaki alan diğer bir tablodaki alana başvurduğu zaman başvuran kolon yabancı anahtar (Foreign Key) olarak, başvurunun yapıldığı tablodaki başvurulan kolon ise ana anahtar (Parent Key) olarak çağırılır. Örneğin; siparişler isimli bir tablodaki musno alanı yabancı anahtar, bu alan musteriler isimli diğer tablodaki musno alanına başvuruda bulunduğu için ana anahtar olarak çalışır. Yabancı anahtar siparişlerin mevcut olup olmayacağını kontrol için müşteriler tablosunda musno alanına başvurmaktadır. Bu alanda eşlenen bir değer bulunmadığı zaman "kaydı bulunmayan bir müşterinin siparişleri" diye bir durum engellenerek veri bütünlüğü korunur. İlişkisel bütünlüğü sağlanan tablolardaki başvurulan ve başvuran alanların isimlerinin aynı olma zorunluluğu yoktur. Fakat aynı tipte olması gerekir.
Bir alan yabancı anahtar olarak geçiyorsa bu alan bir tabloya başvuruda bulunmak için bağlıdır. Bu alandaki her değer diğer alandaki bir değer ile direkt olarak ilişkilidir. Yabancı anahtarın her değeri, ana anahtarın yalnızca bir değerine başvuruda bulunur. Bu şekilde ana anahtarda tek olan bir değere karşılık yabancı anahtarda birden fazla değer bulunabilir.

Veriler Üzerinde Yapılan Kısıtlamalar
Kısıtlamalar, tablonun kolonlarına girilecek veriyi bazı şartları sağlayacak şekilde kısıtlamak için yapılan tanımlardır. Kısıtlama yapılmadan kullanılan bir alan için varsayılan değerler kullanılır. Bir tabloya insert komutu ile kayıt eklerken boş geçilen kolonlar için bu kolona program tarafından otomatik olarak eklenen değerlere varsayılan değer denir. Birçok durumda NULL değer varsayılan olarak kullanılır.

İki temel kısıtlama tipi vardır: Tablo kısıtlamaları ve kolon kısıtlamaları. Bu iki kısıtlama arasındaki temel fark; kolon kısıtlamaları sadece tek olarak kolonlar üzerinde uygulanırken, tablo kısıtlamaları bir veya birden fazla kolon grupları için uygulanır.
SQL'de kısıtlamalar CREATE TABLE komutu ile tanımlanır. Bu komutta kolon kısıtlamaları, kolon tanımının sonunda yer alırken tablo kısıtlamaları bütün kolon tanımlamaları bittikten sonra en sondaki kolon tanımından sonra gelir. CREATE TABLE komutunun kısıtlamalar ile yazım şekli şöyledir.

Create table tablo ismi (
Kolon_ismi veri_tipi kolon_kısıtlamaları,
Kolon_ismi veri_tipi kolon_kısıtlamaları,
.....
Tablo_kısıtlaması (kolon_ismi, kolon_ismi.....) **) ;**

Create table ogrenci (
Ogrno integer not null,
Ad char(15) not null, } kolon kısıtlaması
İli char(20),
Bolum_no integer not null,
Unique (ogrno));

Eğer kısıtlamalarınıza Constraints sözcüğü ile bir isim verirseniz bu isim Constraints_name alanında yer alacak ve bu kısıtlamayı takip etmeniz daha kolay olacaktır.

```
SQL>Create table Constraints_deneme
      (no integer Constraints no_icin_nn_kisiti not null,
       Ad varcahar2(16) Constraints ad_icin_nn_kisiti not null) ;
SQL>
```

Kısıtlamalar varsayılan olarak aktif durumuna getirilirler. **Alter table** komut yardımı ile bu kısıtlamalar pasif veya aktif durumuna getirilebilir.

```
SQL>alter table constraints_deneme
      disable constraints ad_icin_nn_kisiti;
SQL>
```

Kısıtlamaların durumunu user_ constraints görünümünü sorgulayarak öğrenebiliriz.

```
SQL> select constraints_name, validated, status from user_ constraints;

constraints_name        validated          status
----------------------  -------------  --------------
Sys_c002724             Validaded          Enabled
No_icin_nn_kisiti       Validaded          Enabled
ad_icin_nn_kisiti       Not Validaded      Disabled
sys_C002717             Validaded          Enabled
.....

SQL>
```

Sadece ilgilendiğiniz bir tablodaki kısıtlamalar hakkında bilgi almak için şu sorgu ifadesi kullanılır.

```
SQL> select constraints_name, constraints_type from user_ constraints;
       Where table_name=' constraints_deneme';

constraints_name              c
-----------------------      -------
No_icin_nn_kisiti             c
ad_icin_nn_kisiti             c

SQL>
```

Alter table komutunu drop constraints sözcükleri ile birlikte kullanarak bir kısıtlamayı silebiliriz.

```
SQL> alter table constraints_deneme
       drop constraints ad_icin_nn_kisiti;
SQL>
```

NOT NULL Kısıtlamaları

CREAT TABLE komutu ile tanımlanan bir alana NOT NULL kısıtlaması koyarak NULL değerlerin girişine engel olunabilir.

```
Create table ogrenci
      ( ogrno        integer NOT NULL,
        Ad           char(10) NOT NULL,
       Sehir         char(10),
      Ortalama       real);
```

Yaratılan bu tablonun ogrno ve ad alanlarının boş geçilmesine engel olunmuştur. Bu alanlar için gereklilik şartı getirilmiştir.

UNIQUE Kısıtlamaları

CREATE TABLE komutu ile tanımlanan bir alana girilen verinin tablo bazında sadece bir değere karşılık gelmesi sağlanabilir.

```
Create table ogrenci
      ( ogrno        integer NOT NULL UNIQUE,
        Ad           char(10) NOT NULL UNIQUE,
       Sehir         char(10),
      Ortalama       real);
```

4 alana sahip öğrenci isimli bir tablo oluşturulmakta ve ogrno ve ad alanları indekslenerek bu alanlara tek kayıt girilmesi sağlanmıştır. Diğer bir yöntem ise tablo kısıtlaması kullanarak bir veya birden fazla alan üzerinde kısıtlama getirmektir.

```
Create table ogrenci
      ( ogrno        integer NOT NULL,
        Ad           char(10) NOT NULL,
       Sehir         char(10),
```

```
        Ortalama        real,
        bolumNo         integer NOT NULL,
        UNIQUE  (OgrNo, BolumNo)  );
```

Burada dikkat edilmesi gereken nokta, UNIQUE ile getirilen bir tablo kısıtlamasındaki her iki alan için NOT NULL kolon kısıtlamalarının hala kullanılır durumda olmasıdır.

PRIMARY KEY Kısıtlaması

Birincil anahtar olarak bilinen bu kısıtlama ile tablodaki bir kolona iki tane aynı olan değer girişi engellenir. Bu kısıtlama Unique ve Not Null kısıtlamalarının birleştirilmiş halidir.

```
Create table ogrenci
        ( ogrno         integer PRIMARY KEY,
          Ad            char(10) NOT NULL,
         Sehir          char(10),
        Ortalama        real,
);
```

Burada ogrenci tablosu birincil anahtara sahip ogrno alanı ile oluşturulmaktadır. Bu tabloda birden fazla birincil alan oluşturmak istersek;

```
Create table ogrenci
        ( ogrno         integer NOT NULL,
          Ad            char(10) NOT NULL,
         Sehir          char(10),
        Ortalama        real,
        PRIMARY KEY (ogrno, ad)
);
```

İki birincil anahtara sahip bir tabloda tek kayıt girişi yaparken bu iki anahtar birden dikkate alınır. Yani ogrno kolonunda iki tane "3" verisi bulunabilir, fakat bu iki "3" verisine karşılık ad kolonunda iki tane aynı veri bulunamaz. Tam terside geçerlidir ve ad alanında iki tane aynı veri bulunabilir, fakat bu veriler için ogrno alanındaki veriler farklı olmak zorundadır.

FOREIGN KEY ve REFERENCE Kısıtlaması

Bu kısıtlamalar ile bir kolonun değeri başka bir tablonun kolonundaki değeri referans gösterilebilir. Bu durum birbiriyle bağlı iki tablo arasındaki birincil anahtar-yabancı anahtar ilişkisini temsil eder. Yabancı anahtar kısıtlamaları ile ilgili olan ilişkisel bütünlük konusu bu tür kısıtlamaların gerekliliğine açıklık getirmişti. Bu kısıtlama diğer bir tablodaki anahtara bağlı olmayı gerektirir. Dolayısıyla yabancı anahtar içeren tabloyu ayrıntı tablo, bağlı bulunan tabloyu da ana tablo olarak belirtebiliriz.

```
SQL>Create table bolum (bno integer primary key,
        Ad varchar2(32) not null,
        Prg varchar2(64));
SQL>
```

Aşağıda öğrenci isimli ayrıntı tablo tanımı verilmiştir. Birincil anahtar olarak ogrno kolonu, yabancı anahtar olarak ise bno kolonu tanımlanmıştır. Bno kolonunun references sözcüğü ile bolum tablosunun bno alanına referans gösterildiğine dikkat edilmelidir.

```
SQL>Create table ogrenci (ogrno integer primary key,
    Okulno char (12) not null,
    Ad varchar2(16) not null,
    Soyad varchar2(16) not null,
    Bno integer not null,
    Foreign key (bno) references bolum (bno) );
SQL>
```

CHECK Kısıtlaması

Check kısıtlaması ile bir kolona girilecek değerler için koşul belirtilebilmektedir. Aşağıdaki tablo tasarımında Check kısıtlamasının değişik kullanımlarına dikkat edilmelidir. Kısıtlamalarda karşılaştırma sembolleri ve mantıksal operatörler kullanılabilmektedir.

```
SQL> Create table check_deneme (no integer not null,
    Vize integer check (vize>=0 and vize <=100),
    Final integer check (vize>=0 and vize <=100),
    Cinsiyet char(5) not null check (cinsiyet='erkek' or cinsiyet='kadın'),
    Bolum char(32) default 'bilgisayar'
    Check (bolum='bilgisayar' or bolum='buro' or bolum='turizm'
    Or bolum='muhasebe')
);
SQL>
```

Silme Kuralı

Tablolar arasında veri bütünlüğünün korunabilmesi için ana/ayrıntı yapısındaki tablolarda silme kuralı kullanılmalıdır. Bu kural referans gösteren yabancı anahtar tanımını takip eden satırlarda ON DELETE sözcüğü kullanılarak yapılmaktadır. Birçok veritabanında bu sözcükle birlikte 4 tane silme kuralı kullanılabilmektedir: Cascade, restrict, set default ve set null.

- Restrict: Ayrıntı tablosunda ana tablodaki kayıtlara referans gösteren kayıtların bulunması durumunda ana tablodaki kayıtların silinmesini engeller.
- Cascade: Ana tablodaki kayıtların silinmesi durumunda bu kayıtlara bağlı ayrıntı tablosundaki kayıtlarında silinmesini sağlar.
- Set Default: Ana tablodan silinen kayda bağlı olan ayrıntı tablosundaki kayıtlara yabancı anahtarın varsayıla değerinin verilmesini sağlar.
- Set Null: Ana tablodan silinen kayda bağlı olan ayrıntı tablosundaki kayıtların yabancı anahtarına Null değerinin verilmesini sağlar.

Tabloya Eklemeler Yapma

Varolan bir tabloya yeni kolonlar veya kısıtlamalar eklemek için alter table komutu kullanılır. Aşağıdaki kod ogrenci isimli tabloya meztarihi isimli tarih tipinde bir kolon eklemekte ve bu kolona veri girişini zorunlu hale getirmektedir.

Alter table ogrenci add meztarihi date not null;

Tabloda mevcut bir kolonu silmek için alter table komutunun aşağıdaki yapısı kullanılır.

Alter table ogrenci drop meztarihi;

Varolan bir kolon üzerinde düzenleme yapmak için modify sözcüğü kullanılır. Aşağıdaki kod bakiye isimli kolonun yeni veri tipini belirlemektedir.
Alter table musteri modify bakiye number(7, -2);

Bir tablonun belli bir kolonuna varsayılan değer atamak için modify sözcüğü ile beraber default kullanılır.
Alter table ogr modify cinsiyet default 'E';

Set unused sözcüğü ile kullanılmamış bir veya birden fazla kolon işaretlenebilir.
Alter table musteri **set unused column** babaad;

Kullanılmamış olarak işaretlenmiş kolonlar drop sözcüğü ile silinebilir.
Alter table musteri drop unused columns;

Tablo Silme
Veritabanındaki istenmeyen tabloların silinerek çıkarılması için DROP TABLE komutu kullanılmaktadır. Tablo silme işlemi dikkatle yapılmalıdır. Çünkü silinen tablolar tekrar geri alınamamaktadır. Bu komut silinen tablo ile ilgili indeks, tetikleyici gibi nesneleri de siler.
Drop table silinecek_tablo;

Tablonun Yeniden Adlandırılması
Varolan tablolara yeniden isimler vermek için Rename komutu kullanılır.

Rename musteri to musteriler;

Tabloya Açıklama Ekleme
Bir tablo veya tablo kolonuna açıklama eklemek mümkündür. Bunun için COMMENT komutu kullanılmaktadır. Açıklamalar veri sözlüğündeki catolog ve syscatalog nesnelerinin REMARKS isimli kolonunda saklanır.
Comment on table ogrenci is 'ogrenci kimlik bilgileri';

İNDEKSLER

Dikkatli bir indeks seçimi SQL tarafından sağlanan where, order by ve group by sözcüklerinin işlevlerini hızlandırabilir. Tüm tabloyu sıralı olarak taramadan hızlı bir şekilde kayıt bulmanın en iyi yöntemlerinden birisi indeks kullanmaktır.
İndeksler için birkaç kural bulunmaktadır;

- Bir indeks hemen hemen her tablo için yarar sağlamaktadır.
- Daima bir birincil anahtar tanımlanmalıdır.
- Birincil ve yabancı anahtarlar ile indeks tanımlamayın.
- Tek bir tablo için 6 indeksten fazlasını tanımlamayın

İndeks oluşturmak için **Create index** komutu kullanılır. Bu komutun yapısı şöyledir.
Create index indeks_ismi **on** tablo_ismi (kolon_ismi1, kolon_ismi2.....);

Bu yapıdan görüldüğü gibi her indeks için bir isim tanımlanmakta ve bu isimdeki bir indeks bir veya birden fazla alanı kapsayabilmektedir.
Bir tablo için tanımlanan her indeks, o tabloyu indeksli alana göre sıraya dizecektir. İndeksler tanımlanırken eğer sıralama yönü olarak artan veya azalan belirtilmemişse varsayılan olarak artan sıralama dikkate alınır ve sıralanmış yeni kayıtları bir sorgu ile görebiliriz.

Eğer tanımladığınız indeksli alanda tekrarlı kayıtlara izin vermek istemiyorsanız Create index komutunu unique ile birlikte kullanabilirsiniz.
Create unique index ndxogrno on ogrenci (ogrno);

İndeksler Üzerinde Düzenleme
Varolan indeksler üzerinde düzenleme yapmak için Alter Index komutu kullanılır.
Alter index ndxogrno **rename** indeks_ogrencino;

Indeks Silme
Bir tabloda mevcut olan bir indeks Drop Index komutu ile silinebilir.
Drop index ogrenci.ndxdyeri;

GÖRÜNÜMLER

Kullanıcıların bir veritabanı tablosunun belli bir bölümünü görmelerine olanak tanıyan sanal tablolara görünüm denir. Sanal tablo denilmesinin nedeni bu tablolarda herhangi bir veri olmamasıdır. Sadece belirli bir tabloya bakış sağlayan bir pencere niteliğindedirler. Bu sanal tablolar sadece belirli tanımlar içerirler ve kaydedilmezler.
Bir görünüm kullanarak;

- Bir veya birden fazla tablonun belirli bölümlerini görüntüleyebilirsiniz.
- Tabloların ve görünümlerin bileşimlerinden oluşan bir bölümü görüntüleyebilirsiniz.
- Tablo üzerinde ekleme, silme, güncelleme yapabilirsiniz.

Görünümler iki temel amaç için kullanılırlar:

- Veri erişimini basitleştirmek
- Veri güvenliğini arttırmak

Görünümler herhangi bir veri içermemesine rağmen kolonlardan ve satırlardan oluşan normal bir tablo gibi görünürler.
Create view görünüm_ismi(gör_kolon1, gör_kolon2.....) **as** sql_ifadesi;
Bu ifadede görünüm_ismi bölümüne oluşturulmak istenen görünümün ismi yazılır. Görünümler ile gösterilecek veriler yine bir sorgu ifadesi sonucu elde edilir. Görünümün görüntüleyeceği kayıtlar sql_ifadesi bölümündeki bir sorgu ile alınır.
Select ad, soyad, dtarihi **from** ogrenci **where** bolum='bilgisayar programcılığı'
Bu sorgu ifadesi ile "bilgisayar programcılığı" bölümündeki öğrenci kayıtlarına sık sık ihtiyaç duyuyorsanız bunun için bir görünüm tanımlayabilirsiniz.
Görünümler varsayılan olarak as sözcüğünden sonra kullanılan kolon başlıklarını alırlar. Bunu değiştirmek için;
Create view bilgisayar (ogrenci_ad, ogrenci_soyad, ogrenci_dtarih)
As select ad, soyadi dtarihi **from** ogrenci **where** bolum='bilgisayar programcılığı';

Görünümleri birden fazla tablo üzerinde de kullanabilirsiniz. Bu tabloların birbiriyle bağlantılı olması da gerekmez. Bu iki tablodan bir alt sorgu yardımı ile sorgu yapabilirsiniz.
Create view bilgisayar
As select ogrno, ad, soyad **from** ogrenci
Where bolumno = (**select** bolumno **from** bolumler
where bolumadi='bilgisayar programcılığı';

Görünümler Üzerinde Düzenlemeler

Bazı kısıtlamalar ile görünümler üzerinde ekleme, silme ve güncelleme işlemlerini kullanabilirsiniz. Aşağıdaki kod görünümden belirlenen kayıtların silinmesini sağlar.
Delete from bilgisayar where ad=’salih’;
Aşağıdaki ifade Mustafa Sarıoğlu’nun öğrenci numarasını ‘98118020’ olarak günceller.
Update bilgisayar **set** ogrno=’98118020’ **where** ad=’Mustafa’ **and** soyad=’Sarıoğlu’;
Aşağıdaki ifade ise bilgisayar görünümünün gösterdiği tabloya yeni bir kayıt ekler.
Insert into bilgisayar (ad, soyad, dtarihi) **values** (‘Doğan’, ‘Güngör’, ‘01/05/1979’);

Görünümler üzerinde ekleme ve güncelleme işlemlerinin kontrolünü sağlayan bir seçenek daha bulunmaktadır: with check option.
Create view bilgisayar
As select ogno, ad, soyad **from** ogrenci
Where bolum=’bilgisayar programcılığı’
With check option
Bu seçenek görünümün gösterdiği tabloda yer alan ancak görünümde yer alamayan kayıtlar üzerinde ekleme ve güncelleme işlemlerini kontrol eder.
Insert into bilgisayar (ogrno, ad, soyad, bolum)
Values (‘00124001’, ‘Nurullah’, ‘Bilgeç’, ‘büro yönetimi’);
Eklenmek istenen bu kaydın girişi engellenir çünkü izim görünümümüz ‘bilgisayar programcılığı’ bölümündeki öğrencileri göstermektedir. Dolayısı ile bunun dışında bir öğrenci eklememize ya da güncelleştirmemize izin vermez.

Görünümlerin Silinmesi
Bir veritabanı üzerindeki görünümü silmek için şu yapı kullanılır.
Drop view görünüm_ismi;
Aşağıdaki ifade bilgisayar isimli görünümü silmektedir.
Drop view bilgisayar;

Güvenlik
Görünümler ile gelişmiş veri güvenliği sağlamak çok kolaydır. Bir kullanıcıya tablodaki tüm bilgileri değil, sadece kullanıcı için gereken bazı bilgileri görünümler ile vermek mümkündür. Örneğin personel bilgilerinin tutulduğu bir tablodaki tüm bilgileri her personele açmak güvenlik açısından sakıncalıdır. Çünkü bu tabloda personele ait sicil ya da maaş bilgileri bulunabilir.
Görünümler ile;

Tablodaki kolonlar seçilebilir.	Bu şekilde sadece belli kolonları içeren bir görünüm tanımlanabilir. Kullanıcılara sadece bu kolonlara erişim verilir ve kısıtlama getirilir.
Tablodaki satırlar seçilebilir.	Where veya having fonksiyonları ile tablo veya tablolardan sadece belirli kayıtların seçilmesi sağlanabilir. Bu şekilde kullanıcıların tablodaki her kayıta ulaşması engellenebilir.

SIRALAR
Sıralar otomatik artan sayılar üreten nesnelerdir. Sıra nesnesi oluşturmak için **Create sequence** komutu kullanılır**.**
Create sequence sıra_ismi
Start with başlangıç
Increment by artış
Minvalue min_değer
Maxvalue max_değer
Cache

Cycle;

Sıra_ismi Sıra nesnesinin adı.
Başlangıç Sıra nesnesinin başlangıç değeri. Bu değerden itibaren artım başlayacak**.**
Artış Artımın miktarını belirler. Kaçar kaçar artım verileceği bu değer tarafından belirlenir.
Min_Değer Sıra nesnesinin kullanacağı en küçük tamsayı değer.
Max_Değer Sıra nesnesinin kullanacağı en büyük tamsayı değer.
Cache Hafızada ön bellekleme için kaç tane sıra değeri kullanılacağını belirtir. Burada ayrıca NoCache isminde bir parametrede kullanılmaktadır. Bu ise sıra değerlerini belirtir. Cache veya NoCache parametrelerinin her ikisi de kullanılmaz ise Oracle varsayılan olarak bu parametre için 20 kullanır.
Cycle Sıra değerleri Max_Değer ile belirtilen sayıya ulaştıklarında tekrar Min_Değer ile belirtilen sayıdan başlayıp artış değeri kadar artım ile yeniden devam etmesini sağlar. Eğer bu parametre yerine NoCycle kullanılırsa Max_Değer sayısına ulaşan Sıra nesnesi artık sayı üretmeyecektir ve duracaktır. Varsayılan değer de budur.

Aşağıda seq_bolum isminde bir sıra nesnesi oluşturan kod verilmiştir. Bu nesne artım işlemine 1'den başlayacak ve birer birer artımlarla devam edecektir.

```
SQL> Create sequence seq_bolum
       Start with 1  increment by 1;
SQL>
```

```
SQL>Create sequence seq_sipno
Start  with 1000  increment by 10
Minvalue 1 maxvalue 100000
Cycle;
SQL>
```

Sıra nesnesinin ürettiği değerleri kullanmak için nextval ve curr val ifadeleri kullanılmaktadır. Sıra nesnesinin ilk değerini üretmesi için tanımlandıktan sonra sıra_ismi.nextval gibi bir tanım kullanın. Daha sonra sıra nesnesinin aktif değerini öğrenmek için sıra_ismi.currval ifadesini kullanabilirsiniz.

```
SQL> select seq_sipno.nextval from dual;

NEXTVAL
----------
1000

Select  seq_sipno.currval from dual;

CURRVAL
----------
1000
SQL>
```

Sıra nesnesinin değerini tabloya veri eklerken aşağıdaki gibi kullanabiliriz.
insert into musteri (no, tarih, urunno) **values** (seq_sipno.nextval, '01.04.2004', 13);

Sıra Nesnesini Düzenleme
Varolan bir sıra nesnesi üzerinde düzenleme yapmak için Alter Sequence komutu kullanılır.

SQL> alter sequence de increment by 10; SQL>

Sıra Silme
Varolan bir sıra nesnesini silmek için Drop Sequence komutu kullanılmaktadır.
Drop Sequence seq_sipno

Sıra Listesi
Sıra nesnelerinin listesi user_sequence veya user_objects görünümlerinde yer alır.

```
SQL>select sequence object_name from user_objects
                where object_type='sequence';

Object_Name
---------------
seq_harno
deneme
seq_bolum
SQL>
```

EŞANLAMLAR
Eşanlamlar, tablo, görünüm, sıra, saklı prosedür gibi birçok veritabanı nesneleri için kullanılan takma adlardır. Çok karmaşık ve uzun ifadeler ile örneğin uzak bir veritabanı nesnesine erişim sağlanıyorsa bu nesne için eşanlam tanımlanabilir. Eşanlamlar Create Synonym komutu ile tanımlanır.
Create synonym eşanlam_ismi for veritabanı_nesnesi_ismi;

Bu şekilde tanımlanmış eşanlamlar sadece aktif kullanıcı için geçerli olmaktadırlar. Eğer tüm kullanıcılar tarafından kullanılacak bir eşanlam tanımlamak isterseniz, eşanlam tanımına Public sözcüğünü eklememiz gerekir. Bu sözcüğün kullanılabilmesi için Create public synonym sistem ayrıcalığına sahip olunması gerekir. Birçok ayrıcalık ile birlikte bu ayrıcalık system yöneticisi hesabında mevcuttur.

Eşanlam Silme
Eşanlam nesnelerini silmek için Drop Synonym veya Drop Public Synonym komutları kullanılır.
Drop Public Synonym Blm;
Drop Synonym Blm;

Eşanlam Listesi
Eşanlamların listesi User_Synonyms görünümünden alınabilir.
Select synonym_name, table_name from user_synonyms
Select synonym_name, table_name from syn;

TETİKLEYİCİLER
Tetikleyiciler, görsel programlama dillerindeki olaylar yardımı ile çalıştırılan eylemlere benzetilebilir. Olaylar bazı hareketlerin meydana gelmesi sonucunda ortaya çıkıyor olabilir ve programcı tarafından belirlenen bir eylemi yerine getirirler. Tetikleyiciler ise DBMS'ler tarafından kullanılan eylemlerdir. Bu eylemler insert, delete ve update gibi hareketler tarafından tetiklenirler.

Create triger tetikleyici_ismi
Before/after detele or update or insert
On tablo_ismi
For each row
Begin
SQL ifadeleri veya PL/SQL kodları
End;

Tetikleyici delete, update ve insert işlemlerinin hepsinin gerçekleşmesi durumunda veya sadece birisinin gerçekleşmesi durumunda tetiklenebilir. Ancak bu noktada bize yine bir kontrol imkanı daha sunulmaktadır: **Önce**(Before) ve **Sonra**(After). Bu üç işlemden önce tetikleyicinin çalışmasını istiyorsak **before,** sonra çalışmasını istiyorsak **after** anahtar sözcüğünden sonra belirtilir. Eğer tetikleyicinin satır tetikleyicisi olarak tasarlanmasını istiyorsak **for each row** sözdizimini eklememiz gerekir. Satır tetikleyicileri, tetikleyici komutu tarafından etkilenen her satır için bir kez ateşlenir. Ayrıca kolonların değişen değerlerini elde etmek için **New** veya **Old** sözcükleri kullanılır. **New** kolondaki yeni değeri, **Old** ise eski değeri verir

```
SQL>Create table bolum (bno integer, ad varchar2(32), prog varchar2(64) );
SQL>Create table ogrenci (ogrno integer, ad char(16), soyad char(16), dtarih date, bno
integer);
SQL> insert into bolum values( 1, 'Bilgisayar', 'Tekn.Prog');
        insert into bolum values(2, 'Muhasebe', 'İkt.İd.Prog.');

SQL> insert into ogrenci values (200,'ünal','bay','02/08/1980',1);
SQL> insert into ogrenci values (300,'serap','bayrakçı','01/07/1985',2)
```

içermektedir. Tablolar arasında herhangi bir ilişkisel bütünlük sağlanmamıştır. Şimdi ayrıntılı tablodan otomatik olarak ilgili kayıtları silecek ayrıntı_sil isimli tetikleyicimizi tanımlayalım:

```
SQL>Create triger ayrıntı_sil
        Before delete on bolum
        For each row
        Begin
          Delete from ogrenci
          Where bno=:old.bno;
        End;
SQL>
```

Tetikleyicimiz bölüm tablosuna aittir ve bu tablodaki Silme olayından hemen önce meydana gelmektedir. Şimdi bu tetikleyicimizi tetikleyecek silme sorgusunu yazalım.

```
SQL> delete from bolum
        Where bno=1;
SQL>
```

Bu sorgu ifadesi ayrıntı_sil isimli tetikleyicinin çalışmasına neden olur. Bu tetikleyici ise 1 nolu bölümü silmeden önce ogrenci tablosuna giderek bu tablodaki 1 nolu bölüme kayıtlı öğrencileri siler ve daha sonra da 1 nolu bölümü bölüm tablosundan siler.

Tetikleyiciler Üzerinde Düzenleme
Tetikleyiciler üzerinde düzenleme yapmak için **Alter Triger** komutu kullanılır. Bu komut yardımı ile bir tetikleyici aktif veya pasif duruma getirilebilir. Aşağıdaki komut dizilimi ile

ayrıntı_sil isimli tetikleyici ilk önce pasif duruma getirilmiştir, daha sonra aktif duruma getirilmiştir.

```
SQL>alter triger ayrinti_sil disable;

SQL>alter triger ayrıntı_sil enable;
```

Bir tabloya ait tüm tetikleyicileri aktif veya pasif yapmakta mümkündür. Bunun için alter table komutunun **All Trigers** söz dizimi ile birlikte kullanılması gerekir.

```
SQL>alter table bolum disable all Trigers;

SQL>alter table bolum enable all Trigers;
```

Tetikleyici Silme
Varolan bir tetikleyiciyi silmek için **Drop Triger** komutu kullanılır.

```
Drop triger ayrinti_sil;
```

Tetikleyici Listesi
Tetikleyici listesi **User_Trigers**, **All_Trigers** ve **Dba_Trigers** görünümlerinden alınabilir.

```
SQL> select triger_name, triger_type from User_Trigers;

Triger_name              Triger_Type
-----------------------  -----------------------
Ayrinti_sil                before each row

SQL>
```

```
SQL> select triger_name, triger_evet from All_Trigers;

Triger_name              Triger_Evet
-----------------------  -----------------------
Ayrinti_sil                delete

SQL>
```

6.3 Veri Kontrol Dili (Data Control Language – DCL)

KOMUTLAR
DCL kategorisindeki komutlar şöyledir.

Komut	İşlevi
Create user	Yeni bir kullanıcı oluşturur.

Alter user	Varolan bir kullanıcı üzerinde değişiklik yapar.
Drop user	Varolan bir kullanıcıyı siler.
Grant	Bir kullanıcıya veya role ayrıcalıklar verir.
Revoke	Bir kullanıcıya veya role verilen ayrıcalıkları geri alır.

VERİTABANI GÜVENLİĞİ

Veritabanı güvenliği, kullanıcılara doğru ayrıcalıkların verilmesi ile mümkündür. Çok kullanıcılı bir ortamda kullanıma açılan bir veritabanı içindeki kayıtların her kullanıcıyı ilgilendirmeyeceği bir gerçektir. Bu nedenle veritabanı üzerinde bazı güvenlik ayarlarının yapılması gereklidir. Örneğin, öğrenci bilgilerinin tutulduğu bir veritabanı sistemini kullanan eğitim kurumunda öğrencilere de bazı bilgileri kullanma ayrıcalığı tanınabilir. Ancak bu ayrıcalık öğrencinin notlarının tutulduğu bir tabloya erişim olmamalıdır veya bu tabloya erişim verilebilir ancak tablo üzerinde ekleme, güncelleme ve silme ayrıcalıkları verilmeyebilir. Hangi veritabanı nesnesinin kim veya kimler tarafından hangi amaçlarla kullanacağı belirlenmeli ve yetkisiz kişilerin önemli verilere erişimi engellenmelidir.

Kullanıcılar (Users)

Veritabanı yönetim sistemlerinde veritabanı nesnelerini kullanan her kişiye veya istemci bilgisayara Kullanıcı denir. Bir kullanıcı bir tabloyu ilk defa oluşturup kullanmış ise bu kullanıcı bu tablonun Sahibi (Owner) olur ve bu tablo üzerindeki tüm haklara sahip olduğu gibi başka kullanıcıya da bu tablo üzerinde ayrıcalıklar tanıyabilir. Oracle gibi veritabanı yönetim sistemlerinde bir veritabanı kullanabilmek için mutlaka sisteme bir kullanıcı adı ve [illegible]e kullanıcıları listelemek için aşağıdaki komut kullanılır.

```
Select username, account_status from dba_users;
```

SQL'de yeni bir kullanıcı oluşturmak için **Create User** kullanılmaktadır:

```
Create User yasar Indentified by classmodule;
```

Kullanıcı ismi yasar ve şifresi classmodule olan bir kullanıcı oluşturduk. Varolan bir kullanıcı hesabı üzerinde düzenleme yapmak için **Alter User** komutu kullanılır. Aşağıda yasar isimli kullanıcının şifresi değiştirilmektedir.

```
Alter User yasar Identified by delphi;
```

Varolan bir kullanıcı hesabını silmek için ise **Drop User** komutu kullanılır.

```
Drop User yasar;
```

Roller (Roles)

Kullanıcıların veritabanındaki bazı fonksiyonları çalıştırabilmeleri için sahip olmaları gerekli ayrıcalık veya ayrıcalıklar kümesine rol denir. Örneğin veritabanına bağlanma, tablo oluşturma, tablo silme gibi işlemler birer ayrıcalık olup bu işlemler ayrı ayrı olarak veya bir rol altında birleştirilerek bir kullanıcıya verilebilir. Böylece bir kullanıcı bu işlemleri yerine getirmeye yetkili birisi olur. Yeni bir rol oluşturmak için **Create Role** komutu kullanılır.

```
Create Role tbl_ekle;
```

Bu ifade tbl_ekle isimli yeni bir rol tanımlar. Şu an bu role boştur. Bu role e bazı ayrıcalıklar atanması gerekir. Şimdi bu role bir tane nesne ayrıcalığı ekleyelim.

```
Grant insert on musteri to tbl_ekle;
```

Bu ifade tbl_ekle rolüne musteri tablosuna ekleme yapabilme ayrıcalığı atamaktadır.

```
Grant tbl_ekle to yasar;
```

Bu ifade tbl_ekle rolündeki ayrıcalıkları yasar isimli kullanıcıya atamaktadır. Kullanıcıya verilen rolü tekrar geri almak için aşağıdaki gibi bir ifade kullanılır.

```
Revoke tbl_ekle from yasar;
```

Bu ifade daha önce yasar isimli kullanıcıya verilen tbl_ekle rolünü kullanıcıdan geri alır.

Ayrıcalıklar

Bir veritabanı nesnesi üzerinde kullanıcının yapabileceklerine Ayrıcalık denir. Örneğin, kullanıcı bir tablo nesnesi üzerinde silme veya ekleme iznine sahip olabilir. Kullanıcıya verilen bu izinler birer ayrıcalıktır. Ayrıcalıklar bir kullanıcıya veya bir role verilebilir.

```
Grant ayrıcalıklar to kullanıcı ismi veya rol [with admin option];
```

...a verilen ayrıcalıkları ...seçeneğin dâhil olduğu Grant ifadeleri ile verilen ayrıcalıkları kullanıcı başka kullanıcılara da verebilir.

Sütunlar Üzerindeki Ayrıcalıklar

SQL, veritabanı nesnelerinin sütunları üzerinde de kullanıcılara haklar ve kısıtlamalar getirebilmektedir. SQL, insert, update ve references nesne ayrıcalıkları ile birlikte sütunlara erişim yetkisi verebildiği gibi kısıtlama da getirebilmektedir. Yetki verilmek istenen veritabanı nesnesine ait sütunlar ayrıcalık ismini takiben parantez içerisinde belirtilir.

```
Grant insert (ad, soyad) on musteri to yasar
```

Yukarıdaki ifade ile "yaşar" isimli kullanıcıya "musteri" tablosunun "ad" ve "soyad" alanları üzerinde ekleme yapabilme ayrıcalığı tanınmıştır.

Görünümler İle Güvenlik

Bir tablo üzerinde güvenlik yöntemlerini arttırmanın diğer bir yolu görünüm kullanmaktır. Görünüm bir tablonun tamamını içerebildiği gibi belirli sütunları veya satırları da içerebilir. Bu nedenle görünümler yardımı ile tablonun tamamı ya da belirli sütun veya satırları üzerinde kullanıcılara ayrıcalıklar tanımak mümkün olmaktadır. Örnek olarak "yasar" isimli kullanıcının "personel" isimli bir tablonun sadece ad, soyad, adres isimli sütunlarında tutulan bilgilere erişmesini sağlayalım. İlk önce bu sütunları "personel" isimli bir tablosundan çeken bir görünüm tanımlayalım.

Create view kisisel as select ad, soyad, adres from personel;

Şimdi bu görünüm üzerinde seçme ayrıcalığını "yasar" isimli kullanıcıya verelim.

Grant select **on** kisisel **to** yasar;

Yukarıdaki ifadede "yasar" kullanıcısının sadece "kisisel" görünümü üzerinde yetkisi olduğu görülmektedir. Kişisel görünümü personel isimli asıl tabloyu perdelemiştir.

Şimdi tablonun satırları üzerinde yapılabilen kısıtlamaya bir örnek verelim. Bu sefer "personel" tablosundan sadece "satis" bölümünde çalışan personelin bilgilerine seçme ve güncelleme yetkilerini verelim.

```
Create view satiskisisel as
Select * from personel
Where bolum='satis';
```

Bu görünüm personel tablosunu filtrelemiştir ve sadece belli bir bölümün kayıtlarını kullanıma açmıştır. Şimdi bu kayıtları "yusuf" isimli satış bölümü kullanıcısına seçme ve güncelleme ayrıcalığı verelim.

```
Grant select, update on satiskisisel to yusuf;
```

6.4 Tabloların Bağlanması

Birden fazla tablo arasında bir ilişki tanımlamak SQL sorgularının en önemli özeliklerinden birisidir. Tablolar arasındaki ilişki tabloların birbirleri ile belirli kurallar çerçevesinde bağlanması ile oluşturulur.
Bir sorgu ifadesinde birbirleri ile bağlanacak tablolar, From sözcüğünden sonra birbirinden virgül karakteri ile ayrılmış şekilde yer alır. Bağlantının yapılacağı kolonlar ise where sözcüğünden sonra bir şart ifadesi ile belirtilir. İki tablonun bağlantısı için en sade yazım şekli şöyledir.

```
Select kolon_listesi
From tablo1,tablo2
Where tablo1.kolon1=tablo2.kolon2;
```

Bu ifade ile tablo1 ve tablo2 arasındaki bağlantı, yine bu tablolara ait kolon1 ve kolon2 alanları üzerinden gerçekleştirilmiştir.

TAKMA AD (ALIAS)
Bir SQL ifadesinde, birden fazla tablo kullanımında Takma Ad denen yardımcı sözcüklerden sıkça faydalanılmaktadır. Takma Ad, burada tablo ismini temsil eden bir sözcüğü karşılık gelmektedir. Özellikle birden fazla tablo kullanan SQL ifadelerine Takma Ad kullanmak SQL ifadesinin okunurluğunu arttırmaktadır.
Takma Ad'lar bir sorgu ifadesinde From sözcüğünden sonra tanımlanır. Tanımlanması oldukça basittir: Tablo ismi yazılır ve bir karakter boşluk bırakılıp Takma Ad yazılır. Başka tablo kullanılacaksa ayıraç olarak virgül kullanılır.

```
Select alias1.kolon_listesi, alias2.kolon_listesi
From tablo1 alias1, tablo2 alias2;
```

Ayrıca as operatörü yardımıyla aynı ifade aşağıdaki gibi de kullanılabilmektedir.

```
Select alias1.kolon_listesi, alias2.kolon_listesi
From tablo1 as alias1, tablo2  as alias2;
```

Aşağıda ogrenci isimli tablodan öğrenci numaraları, öğrenci adı ve soyadı verileri sorgulanmaktadır. İfadede ogrenci tablosu tablo takma adı ile temsil edilmiştir.

```
Select tablo.ogrno, tablo.ad, tablo.soyad
From ogrenci tablo;
```

Bunun gibi tek tablo kullanan sorgularda takma ad kullanımı sorgu ifadesinde karmaşıklığa neden oluyorsa da takma ad kullanımı en çok birden fazla tabloya sahip sorgularda kullanılmaktadır.

BAĞLI TABLOLARDA İLİŞKİSEL BÜTÜNLÜK

Bu özellik veritabanlarına inşa edilmiş başarılı ilişkisel modeller için sık sık kullanılmaktadır. İki tablonun belirli bir kolon üzerinden birbirine bağlanması işlemine ilişkisel bütünlük denir.

İlişkisel bütünlük sağlanarak bağlantısı yapılan iki tablo arasında veri tutarlılığı sağlanır. Örneğim müşteri tablosunda müşteriye ait kişisel bilgiler yer alırken sipariş isimli tabloda musteri tablosundaki müşterilerin vermiş oldukları sipariş bilgileri yer alır. Eğer yan bir sipariş girişi tablosundaki musno

```
Select m.musno, s.sipno, s.tarih, s.miktar, s.tutar
From musteri m, siparis s
Where m.musno=s.musno;
```

BAĞLANTI TİPLERİ

Genel olarak VTYS'lerinde kullanımına rastlanan bağlantı tipleri ve bu tipler için kullanılan operatör listesi şu şekildedir.

Bağlantı	Operatör	İşlevi
Eşit bağlantı	=	İki tablo bağlanır ve bağlantı sonucunda eşleşmeyen satırlar bulunmaz.
Eşit olmayan bağlantı	Between, >, <, <=, >=	Ortak bir alana sahip olmayan iki tablo arasında kurulan bağlantıdır.
İç Bağlantı	İnner join	İki tablo bağlanır ve bağlantı sonucunda eşleşmeyen satırlar bulunmaz.
Dış bağlantı	Outer join	İki tablo bağlanır ve bağlantı sonucunda eşleşmeyen satırlar kalır.
Kartezyen		İki tablo bağlanır ve bağlantı sonucunda bir tablodaki her kayıt diğer tablodaki kayıtlar ile eşleşecek şekilde satırlar çoğalır. Bağlantı koşulu bulunmaz.
Kendisi (self) ile bağlantı		Bir tablonun yine kendisi ile bağlantısı.
Heterojen Bağlantı		Farklı veritabanlarındaki iki tabloyu bağlar.
Birleşim	union	Bir sorgunun sonucu ile diğer bir sorgunun sonucunu birleştirir. Küme operatörlerindendir.
Kesişim	intersect	Bir sorgunun sonucu ile diğer bir sorgunun sonucundaki ortak değerleri verir. Küme operatörlerindendir.
Fark	Minus	Bir sorgunun sonucundaki değerlerden diğer bir sorgunun değerleri çıkarılır. Küme operatörlerindendir.

Eşit Bağlantı

Bağlantılı sorgularda şart ifadesinde kullanılan eşitlik bağlantıları Eşit Bağlantı olarak adlandırılır. Bu bağlantıda iki tablo, ortak bir veya birden fazla alan üzerinden bağlanır.

Sonuçta bu alanlardaki değerlerin her biri eşit olan kayıtlar alınır ve bu şartı sağlamak için where sözcüğünden yararlanılır.

```
Select * from musteri m, siparis s where m.musno=s.musno;
```

Eşit bağlantı ile birden fazla tablo arasında bağlantı kurulabilir. Bunun için her tabloda karşılaştırmaya tabi tutulacak alanlar için where sözcüğü ile şart belirtilir.

```
Select * from bolumler.b, ogrenci.o, notlar.n
Where (b.bolumno=o.bolumno) and (o.ogrno=n.ogrno)
```

Eşit Olmayan Bağlantı

Ortak alana sahip olmayan iki tablo arasında between, >, <, >=, <= gibi operatörler ile kurulan bağlantıdır.

```
Select o.ogrno, o.ad, d.derece from ogrenci o , derece d
Where o.ortalama>=d.altsinir and
o.ortalama<=d.üstsinir;
```

İç Bağlantı

İki tablo arasında ortak bir kolon üzerinden yapılan bağlantıdır. Eşit bağlantı gibidir. Access veritabanı tabloları arasında sıkça kullanılmaktadır. Ortak kolonlardaki değerlerin eşit olduğu veriler bağlantı sonucundaki verileri oluşturur. Eşleşmeyen kayıtlar hariç tutulur. Kolonlardaki değerlerin karşılaştırılması için where sözcüğü yerine inner join ..... on kalıbında bir yapı içerisindeki on sözcüğü kullanılır. Örneğin aşağıda iki tablo musno ortak alanı üzerinden bağlanmıştır.

```
Select * from musteri
İnner join siparis on (musteri.musno=siparis.musno);
```

Birden fazla tablo arasında bağlantı kurmak için de inner join kalıbı kullanılabilir. Bu yapıda inner sözcüğü opsiyoneldir. Kullanılmasa da olur.

```
Select * from bolumler b
Join ogrenci o on (b.bolumno=o.bolumno)
Join notlar n  on (o.ogrno=n.ogrno)
```

Dış Bağlantı

İki tablo arasında ortak bir kolona göre bağlantı kurar, ancak bağlantı eşleşmeyen kayıtları da içerir. Oracle (+) operatörünü kullanır. Access bağlantı için outer join operatörünü kullanır. Yine eşitlik şartını vermek üzere on sözcüğü kullanılır. Bağlantıda ilk önce dikkate alınacak tabloyu belirtmek üzere outer join ifadesi left, right ve full sözcükleri ile birlikte kullanılır.

```
Select * from musteri m
Left outer join siparis s
On (m.musno=s.musno);
```

İfadede left kullanıldığı için outer join kalıbının solunda kalan tablodaki bütün satırlar sonuç veri kümesinde bulunacaktır. Eğer soldaki tabloda herhangi bir kayıta karşılık sağdaki tabloda herhangi bir kayıt yok ise sonuç veri kümesinde bu kayıtlar NULL değerleri içerecektir.

```
Select * from musteri m
Right  outer join siparis s
On (m.musno=s.musno);
```

İfadede right kullanıldığı için outer join kalıbının sağında kalan tablodaki bütün veriler dikkate alınacak ve sonuç veri kümesinde bu tablodaki bütün kayıtlar yer alacaktır. Eğer sağdaki tablodaki bir kayıt ile eşleşmeyen soldaki tabloda bir kayıt yok ise bu kayıt NULL değerler içerecektir.
Dış bağlantı ile kullanılan üçüncü sözcük ise full sözcüğüdür. Bu sözcük kullanılarak yapılan bağlantı sonrasındaki sonuç veri kümesinde tablolardaki bütün kayıtlar yer alır. Diğer tabloda karşılığı olmayan kayıtlar için NULL değerler içeren kayıtlar bulunur.

Kartezyen Bağlantı
Eğer iki tablo herhangi bir şart olmadan aynı select komutu ile bir ifadede kullanılır ise bu ifade Kartezyen bağlantıyı oluşturur. Bu tür bağlantıda çıktı daima iki tablodaki kayıtların çarpımı kadardır. Kartezyen bağlantı için kullanılan temel yapı aşağıdadır:

```
Select * from ogrenci, bolmler;
```

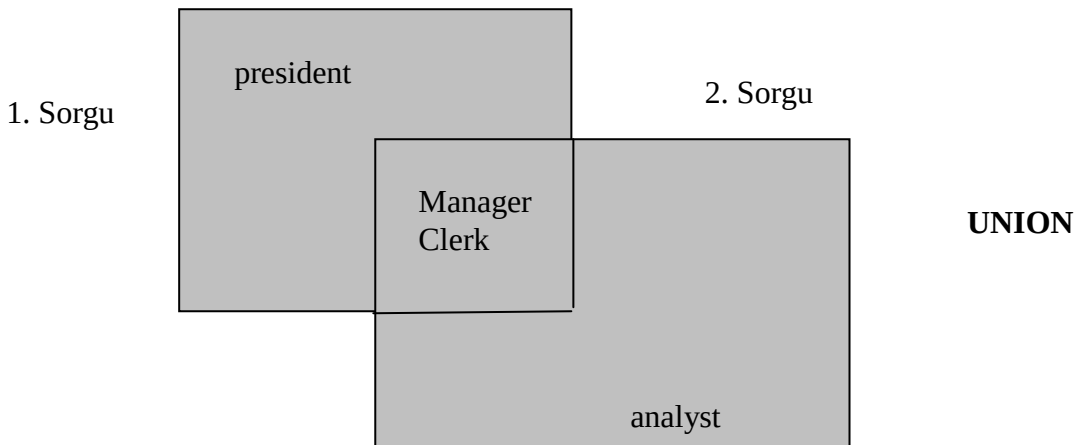
Bu ifadenin sonuç veri kümesindeki kayıt sayısı: Her kayıda karşılık ikinci tablodaki kayıt sayısı kadar kayıt düşeceğinden 5x3=15 kayıt elde edilir.

Kendisi ile Bağlantı
Bir tabloyu aynı veya farklı kolonları üzerinden bağlamak mümkündür. Bağlantı koşulunu belirtmek üzere aynı tablo için iki farklı takma ad belirlenir.

```
Select o1.ad, o2.ad, o1.ortalama from ogrenci o1, ogrenci o2
        Where o1.ortalama=o2.ortalama;
```

Bu ifade bize aynı ortalamaya sahip öğrenci çiftlerinin bütün kombinasyonlarını verir ve tekrar eden veri birden fazla olacaktır. Bu tekrarları ikinci bir koşul ile kaldırmak mümkündür.

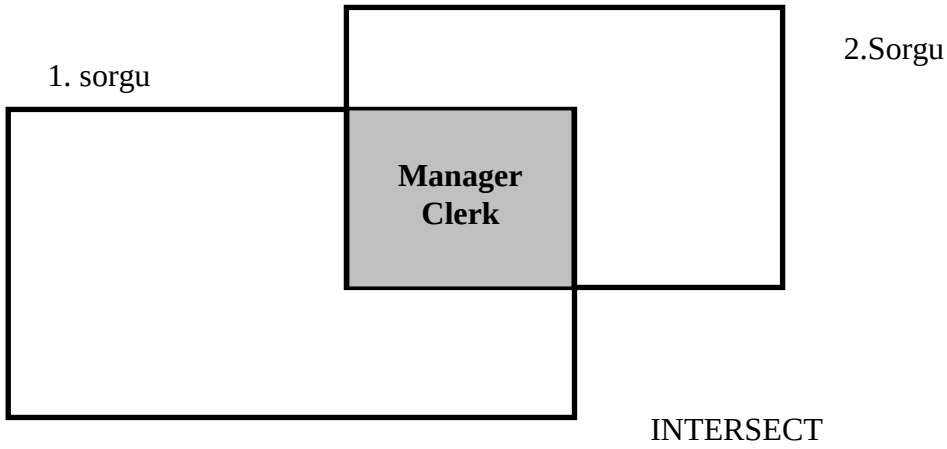
Birleşim (Union)
Bir sorgunun sonucuna diğer bir sorgunun sonucunu ekler. İki sorgu sonucunda listelenecek kolonların sayısının eşit ve benzer olması gerekmektedir.



```
Select kolon_listesi from tablo1
Union
Select kolon_listesi from tablo2
```

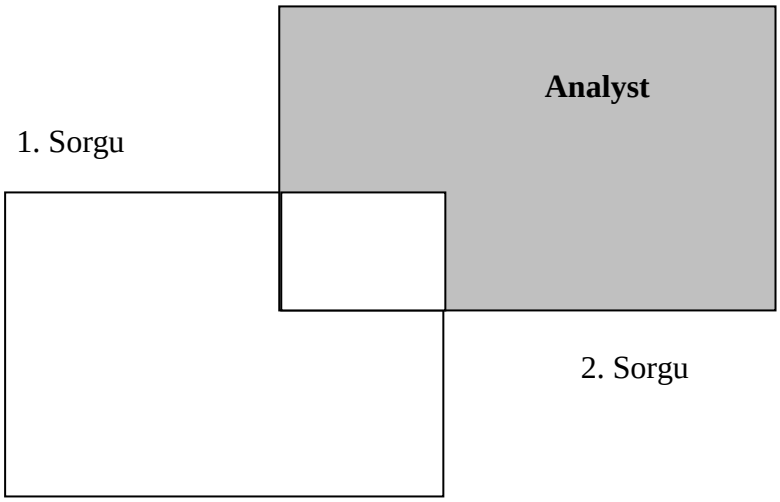
```
Select ogno, ad, soyad from ogrenci99
Union
Select ogrno, ad, soyad from ogrenci00;
```

Kesişim (Intersect)
Her iki sorgunun sonucundan elde edilen ortak değerleri almak için "intersect" kullanılır. İki sorgu sonucunda listelenecek kolnların sayısının eşit ve benzer olması gerekmektedir.




```
Select job from emp where deptno=10
Intersect
Select job from emp where deptno=20;
```

Fark (Minus)
Bir sorgunun sonucundaki değerlerden diğer sorgunun sonucundaki değerleri çıkartmak için "minus" sözcüğü kullanılır. İki sorgu sonucunda listelenecek kolonların sayısının eşit ve benzer olması gerekmektedir.



MINUS

```
Select job from emp where deptno=20
Minus
Select job from emp where deptno=10;
```

Heterojen Bağlantı
Farklı veritabanlarından alınan iki tablonun bağlanması. Örneğin aşağıdaki ifade ile dBASE ve Paradox tabloları state alanı üzerinden birbirine bağlanmıştır.

```
Select * from
":DBDEMOS:customer.db" c1, :BCDEMOS:clients.db" c2
Where c1.state=c2.state;
```

ALTSORGULAR
Altsorgu (SubQuery), bir SQL ifadesinde parantezler içinde verilen diğer bir select komutudur. Alt sorgunun çıktısını diğer sorgu girdi olarak alır ve bu durumda iki sorgu ifadesi iç içe kullanılır. Bir sorgu ifadesinde altsorgunun yeri şöyledir.

Ana Sorgu { Select / from / where (select…..from……where) — Alt Sorgu

Parantez içindeki içteki sorgu olarak belirtilir ve yukarıdaki ifadede ilk olarak bu sorgu çalıştırılır. Bu sorgunun ürettiği değerleri dıştaki sorgu veri olarak kullanır.
Örneğin "LED" isminde bir müşterinizin sipariş bilgilerine ulaşmak istiyorsunuz. Veritabanınızda müşterilere ait "müşteri" ve siparişlere ait "sipariş" isminde iki tablonuz bulunmaktadır. Müşterinize ait sipariş bilgileri "sipariş" tablosunda tutulmaktadır. İlişkisel bir modelde "sipariş" tablosunda müşterinizin ismi yer almaz. Genelde bu tablolar müşterinin tek olan "müşterino" alanı ile bağlıdır. Bu alanı kullanarak müşterilerinize ait bilgileri değişik yollarla bulabilmeniz mümkün. Fakat bu işi aşağıdaki gibi bir alt sorgu ile müşterinizin numarasını öğrenmeden yapabilmenizde mümkündür.

```
Select * from siparis
Where musno=(select musno from müşteri where ad='LED'
```

Yukarıdaki ifadede parantez içinde belirtilen içteki sorgu ismi "LED" olan bir müşteriye ait müşteri numarası döndürecektir. Dıştaki sorgu ise müşteri numarası içteki sorgunun döndürdüğü değer olan müşteriye ait siparişleri listeleyecektir.
İç içe sorgularla aradaki müşteri numarasını bilmeniz gerekmemektedir. Yukarıdaki iç sorgu bir değer döndürecektir. Çünkü "müşteri" isimli tabloda müşteri numarası tektir. İç sorguyu seçerken dikkatli olunması gerekir. İç sorgunun spesifik tek br değer döndürmesi gerekir. Birden fazla değer döndürüldüğünde sorgu hata verir.
Bir SQL ifadesinde altsorgular ve şart ifadeleri birden fazla olabilir. Yine bu ifadeler mantıksal operatörler ile birleştirilir. Birden fazla şart ifadesi ve alt sorgu içeren yapı aşağıdaki gibidir.

```
Select * from …….
Where şart_1 (altsorgu1)
And şart_2 (altsorgu2)……..;
```

Bir alt sorgu içinde diğer bir altsorgu da yer alabilir ve bu böyle devam edebilir. Bu şekilde iç içe bir alt sorgu yapısı meydana gelmiş olur.

```
Select * from …….
Where şart_1 (altsorgu)
And şart_2 (select * from….. where şart_3 (altsorgu)
```

Altsorgular ile Kümeleme Fonksiyonlarının Kullanımı
Tek değer döndüren kümeleme fonksiyonlarını alt sorgulu ifadelerde kullanmak mümkündür. Fakat tek değer döndüren kümeleme fonksiyonları ile Group By sözcüğü kullanılmamalıdır. Çünkü Group By ile tek değer döndüren fonksiyonlar yine işlevini yapacak fakat her grup için bir değer üretecektir.

```
Select avg(ort) from ogrenci
Group by bolum
Having bolum='Bilgisayar Programcılığı';
```

Altsorgularda IN Sözcüğünün Kullanımı
Alt sorgunun birden fazla değer döndürmesi halinde karşılaştırma yapmak için IN sözcüğü kullanılır. IN sözcüğü bir alan değerinin biden fazla değer ile karşılaştırılması amacıyla kullanılıyordu. Karşılaştırılan her değerden birinin şartı sağlaması durumunda TRUE sonucunu üretirdi. Bu nedenle mantıksal operatörlerden OR operatörünün işlevi gibi çalışır.

```
Select * from ogrenci
Where bolumno in
        (select bolumno from bolumler
           Where bolumadi in ( 'Muhasebe' , 'Turizm');
```

Altsorgular ile ANY ve ALL Kullanımı

Altsorgularda karşılaştırma ifadelerinin sol tarafı ile altsorgudan dönen değerin yer aldığı sağ taraf eşleşmelidir. Örneğin mantıksal operatörleri (>, =, <, <>…) içeren bir karşılaştırma kullanılıyorsa altsorgudan dönen değer tek olmalıdır.

```
Sayi > ( altsorgudan dönen tek sayı)
```

```
String = (alt sorgudan dönen tek karakter stringi)
```

Alt sorgular tek değer döndürmelidir.

```
Maas > (maas listesi)
```

Bu ifade yanlıştır. Bu ifadeyi SQL'in ANY sözcüğünü kullanarak, altsorgudan dönen her bir değer ile karşılaştırma ifadesinin solundaki değerin karşılaştırılması şekline sokabiliriz. Doğru ifade şöyledir;

```
Maas > ANY (maas listesi)
```

Bu ifade "maas değeri, maas listesindeki değerlerden en az birinden büyükse" anlamını taşır. ANY sözcüğü diğer mantıksal operatörlerle de kullanılabilir.

Altsorgudan dönen değerlerin birden fazla olması durumunda kullanılan bir diğer SQL komutu ise ALL sözcüğüdür. Yine ALL sözcüğü ile sol taraftaki değer altsorgunun döndürdüğü sağ taraftaki listede bulunan değerlerin her biri ile karşılaştırılır.

Maas > ALL (maas listesi) listedeki her değerden büyükse TRUE

İlişkisel Altsorgular

Daha önce kullandığımız alt sorgular çalıştırıldığında bir değer döndürmekteydi. Bu değer bir ilişkisel veritabanı yönetim sisteminin hafızasında karşılaştırma yapmak için geçici olarak tutulur. Ana sorgudaki şart hafızadaki bu değere göre kontrol edilir.

…….where maas = (select avg (maas)….)

X→ Alt sorgudan dönen değer

Altsorgudan dönen X değeri her kayıt için karşılaştırmaya tabi tutulacaktır.

Maas < X 1.kayıt için,
Maas < X 2.kayıt için

…….

Maas < X n.kayıt için

```
Select * from personel
Where maas = (select avg(maas) from personel) ;
```

```
Select * from personel where maas = 200000;
```

SQL tarafından kullanılan ikinci bir alt sorgu tipi ise ilişkisel sorgulardır. İlişkisel altsorgularda, sabit bir değer (X) döndürülmez. Bu sorgularda anasorgudaki bir değer altsorgu tarafından alınır ve bu değer ile yapılan işlem sonucu altsorgu tarafından farklı bir değer döndürülür.
İlişkisel altsorguları, yukarıdaki verileri kullanarak "aynı meslekten olan personel arasında ortalama maaşa eşit maaş alan personeli bulunuz?" sorusuna çözüm bularak anlamaya çalışalım. Bu sefer yapacağımız işlem bütün personelin ortalama maaşını hesaplayarak tek bir değer elde etmek değildir. Sadece aynı meslekten olan personelin maaşlarının ortalaması hesaplanacak ve meslek sayısı kadar ortalama hesaplaması yapılacaktır. Yani ifademiz aşağıdaki biçimde çalışmalıdır;

……..where maas = (select avg(maas)….)

X,Y,Z…→Altsorgudan dönen değerler

Burada altsorgudan dönen değerler bir dizi değer olmayıp, altsorgunun anasorgudan aldığı veriler sonucunda çalışma anında elde edeceği değerlerdir. Altsorgudan dönen değerler her kayıt için değil, sadece aynı mesleğe sahip kayıtlar için karşılaştırmaya tabi tutulacaktır. Dolayısı ile örneğin "DR" mesleği için altsorgu bir değer döndürecek daha sonra diğer meslekler için aynı şeyi tekrarlayacaktr.

Maas = X 1.mesleğe sahip kayıtlar için,
Maas = Y 2.mesleğe sahip kayıtlar için,

…….

Maas = Z n.mesleğe sahip kayıtlar için

Elimizde olan bilgiler ve bulmak istediğimiz bilgiler aynı tablo üzerinde bulunmaktadır. Aynı tablodaki iki alanı karşıaştırmak için takma adlardan yararlanmak gerekir. Bu şekilde içinde anasorgu ve altsorgu bulunan bir ifade de hangi alanın altsorgudan hangi alanın anasorgudan geldiğini anlayabiliriz.

```
Select * from personel ana
Where maas = (select avg(maas) from personel alt);
```

Yukarıdaki ifade için kullanılan personel tablosu, anasorgu için ana, alt sorgu için alt sözcükleri ile temsil edilmiştir.
Aynı meslekten olan personel arasında işlem yapacağımız için aynı tablodaki meslek alanını karşılaştıran şart ifadesi aşağıdaki gibi olacaktır.

……..where alt.meslek=ana.meslek
Aynı meslekten olan personel arasında ortalama maaşa eşit maaş alan personeli listeleyen ifadenin tamamlanmış şekli şöyledir;

```
Select * from personel ana
Where maas = (select avg(maas) from personel alt
                Where alt.meslek=ana.meslek);
```

Altsorgular ile EXISTS ve NOT EXISTS Kullanımı
Altsorgular ile kullanılan mantıksal ifadelerden biride exists ve bu ifadein olumsuzu not exists sözcükleridir. Mantıksal bir değer döndürmeleri nedeniyle şart ifadelerinde kullanılır. Sözcüklerin SQL ifadeleri içerisinde kullanımları şöyledir;

```
Where exists (altsorgu)
Where not exists (altsorgu)
```

Var olanlar ve olmanlar anlamına gelen sözcüklerin işlevleri oldukça basittir. Eer alt sorgu tarafından en az bir kayıt döndürülürse, exists ifadesi TRUE, altsorgu hiçbir değer döndürmezse FALSE değerini alır.

Altsorgu Sonuçlarını Aritmetiksel Olarak Kullanma
Altsorgu sonucunu normal bir değer gibi kabul edip üzerinde aritmetiksel işlemler apmak mümkündür. Aşağıdaki kod ile final notu genel ortalamanın yarısında düşük öğrenci kayıtları listelenmektedir.

```
Select * from ogrenci
Where finalnotu< (select avg (finalnotu)  from ogrenci)/2;
```