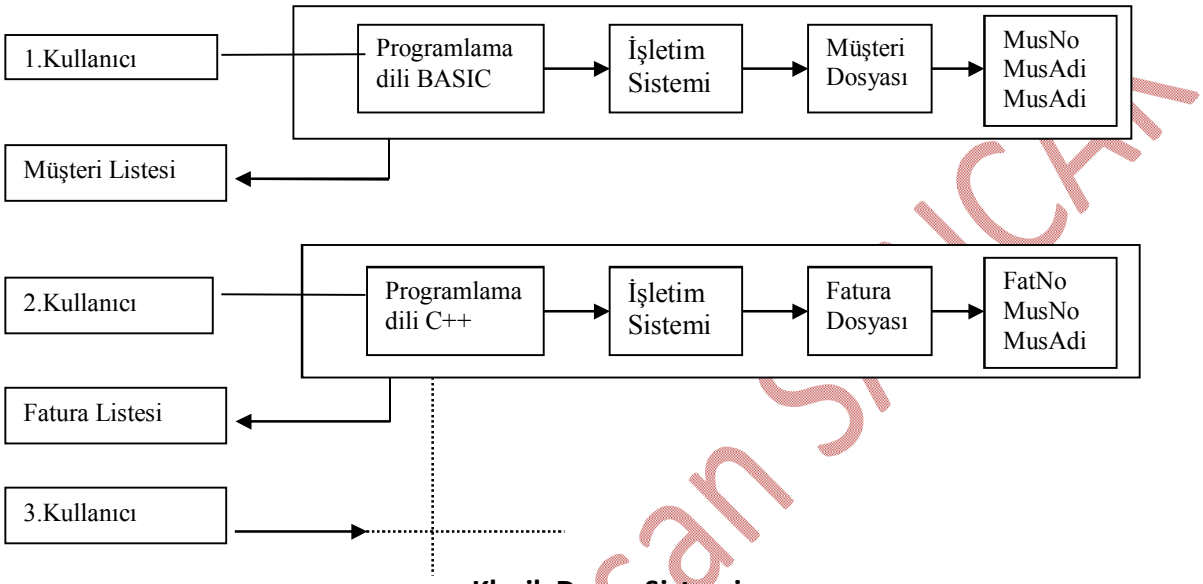
Klasik Dosya (File) Sistemi

Bir işletmenin günlük faaliyetlerini sürdürebilmesi için işletmenin çeşitli konuları ile ilgili olarak çok miktarda bilgi depolanması gerekebilir. Bilgisayarın işletme uygularında ilk kullanılmaya başlandığı dönemlerde bu tip bilgileri depolamak için klasik dosya sistemi kullanılmıştır. Klasik dosya sisteminin en temel özelliği dosyalar ile bu dosyaları işleyecek uygulama programlarının birbirine bağımlı olmasıdır.

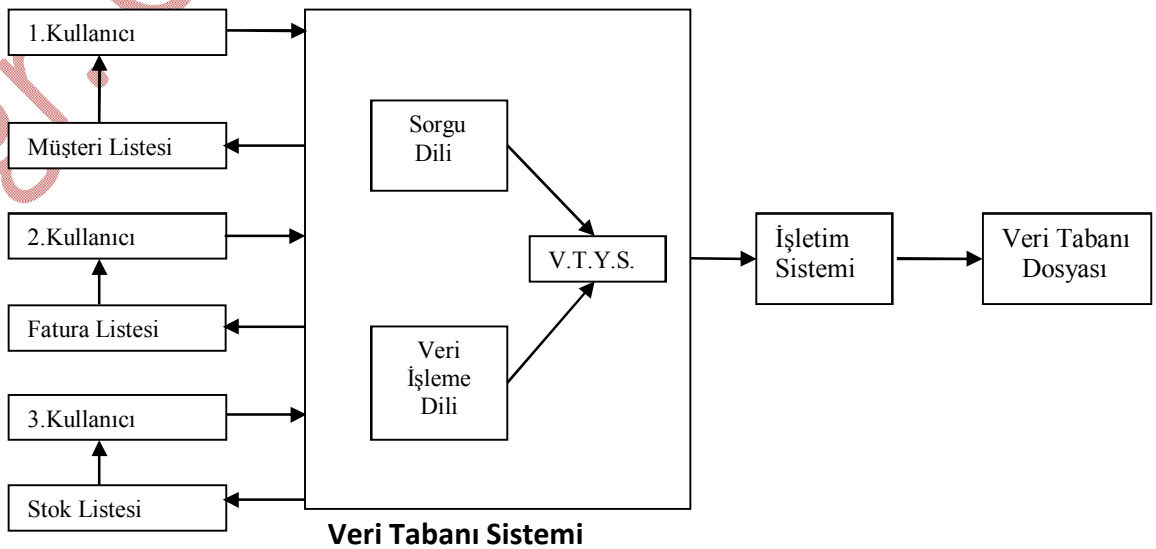


Klasik Dosya Sistemi

Klasik dosya sisteminde n tane ayrı dosya n tane ayrı program tarafından işlenmektedir. Her program rapor üretme esnasında ayrı ayrı işletim sistemi ile yüz yüze gelmektedir.

Veri Tabanı Sistemi

N tane ayrı program tarafından ihtiyaç duyulan tüm veriler tek bir mantıksal dosya içerisine yerleştirilmiştir. Böylece klasik dosya sisteminden farklı olarak her uygulama programı için ayrı bir dosya tutulması yerine bir veri tabanı dosyası kullanılmaktadır.



Veri Tabanı Sistemi

Veri tabanı sisteminde kullanıcıların istediği raporları üretebilmesi için üç yöntem vardır. Sorgu dilini kullanarak, veri işleme dilini kullanarak ve yüksek seviyeli dillerin veri tabanı ile etkileşimini kullanarak. Kullanıcı bu üç yöntemden birini kullanarak küçük bir uygulama programı geliştirip veri tabanı sisteminden istenilen bilgiler alınır.

Klasik Dosya Sisteminin Sakıncaları

- **Veri tekrarı:** Klasik dosya sisteminde, ayrı veriler çeşitli dosyalar içinde tekrar tekrar yer alabilmektedir. Buda aynı verilerin tekrar edilmesine neden olur.
- **Çoklu güncelleme:** Aynı veri birden fazla dosyada tekrar ettiği için bir veride güncelleme yapılacağı zaman aynı veriyi kullanan diğer dosyalarda da aynı güncellemenin yapılması gerekir. Aksi takdirde veri bütünlüğünün kaybolmasına neden olur. Bunun sonucunda birbiri ile uyumsuz ve çelişen raporlar üretilmesi söz konusu olabilir.
- **Bellek hacmi israfı:** Aynı verinin birden fazla dosyada tekrar tekrar kullanılması, kullanılan bellek alanın da israfa yol açacaktır.
- **Erişim dili:** Klasik dosya sisteminde kullanılan programa göre dosyaya erişim farklılıklar gösterebilir. Standart bir dil kullanımı söz konusu değildir.

Veri Tabanı Sisteminin Yararları

Veri tabanı sistemi, klasik dosya sisteminin belirtilen sakıncalarını ortadan kaldırarak aşağıdaki yararları sağlar:

- Veri tekrarları ya ortadan kaldırılır ya da en aza indirgenir.
- Veri bütünlüğü yani belirli bir konu ilişkili verinin sistemde, farklı noktalarda hep aynı şekilde tutulması (çelişkilerin oluşmaması) özel bazı yöntemlerle kolayca kontrol edilir.
- Bellek alanının israfı, veri tekrarları en aza indirildiği için önlenir.
- Veri tabanı sisteminde standart bir sorgu dili (SQL) kullanmak mümkündür.
- Güvenlik ve gizliliğin istenilen düzeyde sağlanmasına elverişlidir.

Veri Tabanı Sisteminin Sakıncaları

- Veri tabanı sisteminin kurulumu ve bakımı klasik dosya sisteminden daha pahalı ve zordur.
- Veri tabanı sistemi içinde, bazı bileşenler iyi tasarlanmadığı durumlarda, bir bütün olarak ciddi sistem başarısızlıklarına yol açabilir.

Temel Kavramlar

- **Veri Babanı (DataBase):** En genel tanımıyla, kullanım amacına uygun olarak düzenlenmiş veriler topluluğudur. Müşteri adres defterleri, ürün satış bilgilerinin saklandığı dosyalar, öğrenciler ve öğrenciler ait harç ve not bilgileri gibi, personel bilgi dosyaları gibi bilgi düzenleri veri tabanlarına örnek olarak verilebilir. Belirli bir konu hakkında toplanmış veriler; bir veri tabanı programı altında toplanırlar. İstenildiğinde toplanan bilgilerin tümü veya istenilen özelliklere uyanları görüntülenebilir, yazdırılabilir hatta bilgilerinden yeni bilgiler üretilerek bunlar çeşitli amaçlarla kullanılabilirler. Veriler fiziksel hafızada Veri Dosyaları (Data Files) halinde saklanırlar. Dosya, bilgisayarların bilgileri birbirinden

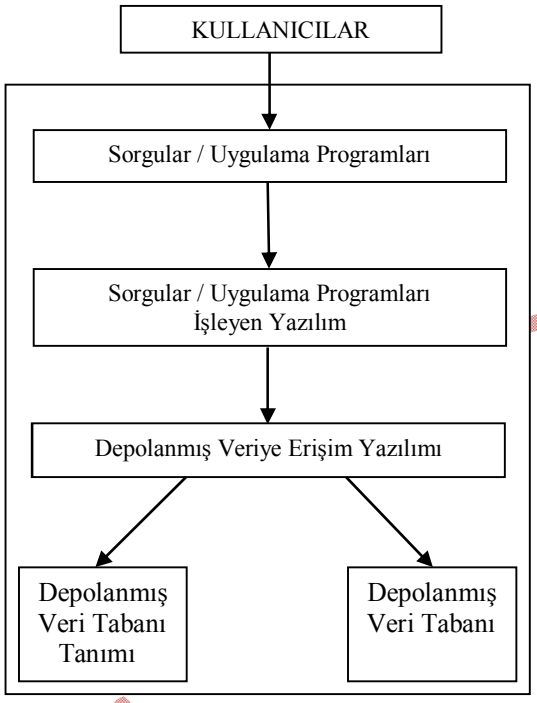
ayırarak saklamak için kullandığı temel bilgi depolama yapısıdır. Bir dosyada, birçok veri yer alabilir. Bir personel otomasyonu ele alınacak olursa, personel ile ilgili bilgiler, personelin çalıştığı birimler, meslekleri, aldığı maaş ile ilgili bilgiler aynı veri dosyasında ama farklı tablolar içerisinde yer alabilirler.

- **Tablo ve Elemanları:** Tablo verilerin satırlar (row) ve sütunlar (colomn) halinde düzenlenmesiyle oluşan veri grubudur. Veri tabanları bir veya daha fazla tablodan oluşurlar. Tablolar arasında ilişkiler düzenlenebilir. Tablonun satırlarındaki her bir bilgi kayıt (record), sütunlar ise alan (field) olarak isimlendirilir. Bir tabloda yer alan her bir kayıt bir satıra karşılık gelir. Örneğin personel listesi (yani personel tablosunu) ele alınacak olursa, her bir satırda bir personele ait bilgiler yer alır. Sütunlardaki alanlar (Field) ise yapılandırılmış bilginin her bir kısmını saklamak üzere yapılan tanımlamadır. Bir personele ait bilgilerin her biri sütunlarda tutulur. Personelin sicil numarası, adı, soyadı, çalıştığı birim, doğum tarihi gibi bilgilerin her biri bir sütun alanıdır. Her bir alan, yapılandırılmış verinin bir birimini tutmak üzere tanımlanır. Her bir sütunun adı ile birlikte diğer bilgilerinin (en fazla kaç birimlik bilgi bu hücrede saklanabilecek, ne tür bilgi saklanacak vs.) ortaya koyduğu tanıma alan denir. Veri tabanının en önemli bileşeni tablodur. Her veri tabanında en az bir tablo bulunur. Veritabanı işlemlerinde önce tablo/tablolar tanımlanır. Daha sonra tablolara kaydedilecek bilgilerin neler olacağı ve bu bilgilere ait özellikler tanımlanır. Personelin sicil numarası ve bunun sayılardan oluşması, personelin adı soyadı ve bunun harflerden oluşması gibi. Tanımlamalar bittikten sonra tablodaki bu alanlara ait gerçek bilgiler yazılır. Yazılan bu bilgiler tablolarda tutulur. Kayıt ile satır arasındaki temel fark, kayıt ile kastedilen yapının sütunlar hakkındaki bilgileri de içermesidir. Tablolara girilmiş bilgilerden belirli şartlara uyanların liste şeklinde alınmasına sorgu adı verilir. Tablolardan gerektiğinde sorgulamalar yapılabilir. Değişik amaçlara göre sorgular hazırlanarak tablodaki bilgilerin tümü, bir kısmı veya belirli şartı sağlayanların listesi alınabilir.
- **Veri Tabanı Sistemi:** Veri tabanlarını kurmayı, oluşturmayı, tanımlamayı, işletmeyi ve kullanmayı sağlayan programlar topluluğudur. Veri tabanı yönetim sistemi (VTYS) veya Database Management System (DBMS) ismi ile de kullanılabilir.
- **Veri Tabanı Yönetim Sistemi:** Yeni bir veri tabanı oluşturmak, veri tabanını düzenlemek, geliştirmek ve bakımını yaptırmak gibi çeşitli karmaşık işlemlerin gerçekleştirildiği birden fazla programdan oluşmuş bir yazılım sistemidir. Veri tabanı yönetim sistemi, kullanıcı ile veri tabanı arasında arabirim oluşturur ve veri tabanına her türlü erişimi sağlar.
- **Veri Tabanı Yöneticisi (Database Administrator):** Bir veri tabanı üzerinde her türlü yetkiye sahip olan kişidir. Veri tabanının tasarımı, üzerinde değişiklikler yapma, kullanıcılara izinler verme gibi fonksiyonlara sahiptir. Veri tabanı yöneticisi, gerçek bir kişi olabileceği gibi, bir gruba da yetki verilmiş olabilir.
- **Veri Tabanının Tanımlanması:** Veri tabanını oluşturan verilerin tip ve uzunluklarının belirlenmesidir.
- **Veri Tabanını Oluşturulması:** Veri için yer belirlemesi ve saklama ortamına verilerin yüklenmesini ifade eder.

- **Veri Tabanı Üzerinde İşlem Yapmak:** Belirli bir veri üzerinde sorgulama yapmak, meydana gelen değişiklikleri yansıtmak için veri tabanının güncellenmesi ve rapor üretilmesi gibi işleri teslim eder.

Veri Tabanı Sisteminin Kendini İçerme Özelliği

Bir veri tabanı sadece veri tabanının kendisini değil veri tabının tam bir tanımını da içerir. Bu tanım sistem kataloğunda saklanır. Sistem kataloğunda saklanan bu bilgilere META-DATA adı verilir.



Kullanıcılar veri tabanından istedikleri bilgiyi alabilmek için küçük uygulama programı ya da sorgu yazar. Bu uygulama programı ya da sorgu işlenerek depolanmış veri tabanına erişim yazılımı aracılığıyla veri tabanından istenilen bilgiler alınır. Veri tabanı isteminde veriler ile veri tabanı yapısal bilgileri birbirinden ayrı olarak saklanır.

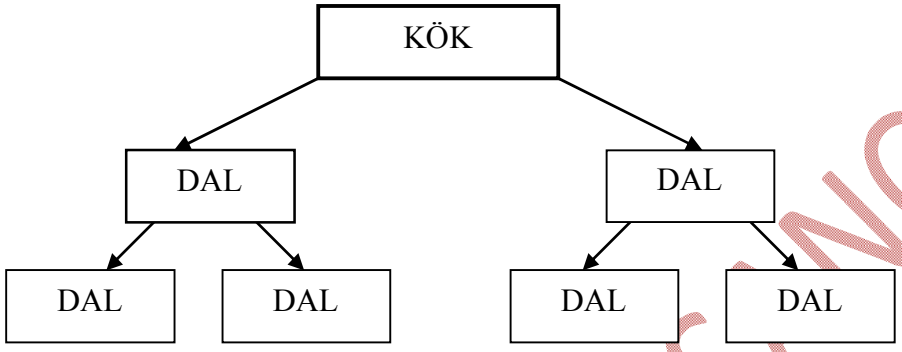
Program Veri Bağımsızlığı

Klasik dosya sisteminde tutulan dosyaların yapısı ve bu dosyalara erişim şekilleri uygulama programlarının içinde saklıdır. Dosyaların yapısındaki bir değişiklik uygulama programında değişikliğe neden olur. Veri tabanı sisteminde ise veriye erişim programları herhangi bir özel dosyadan bağımsız olacak şekilde yazılmıştır. Veri tabanı yapısı hakkındaki bilgi erişim programlarından ayrı olarak sistem kataloğunda saklanır. Bu nedenle veri dosyalarının yapısındaki bir değişiklik sadece sistem kataloğunda değişikliğe neden olur uygulama programları bu değişiklikten etkilenmez veri tabanının bu özelliğine program veri bağımsızlığı adı verilir.

Veri Modelleri

Hiyerarşik Veri Modeli

Veri tabanları için kullanılan ilk modeldir. Bu veri tabanı tipi, ana bilgisayar ortamlarında çalışan yazılımlar tarafından kullanılmaktadır. IBM tarafından çıkarılan IMS bu türde en çok kullanılan yazılımdır. Hiyerarşik veri tabanları uzun bir geçmişe sahip olmalarına rağmen, PC ortamına uyarlanmış olanı bulunmamaktadır. Hiyerarşik veri tabanları, bilgileri bir ağaç (tree) yapısında saklarlar. Kök(root) olarak bir kayıt ve bu köke bağlı dal(branch) kayıtlar olarak hiyerarşik veri tabanının yapısı Şekilde gösterildiği gibidir.

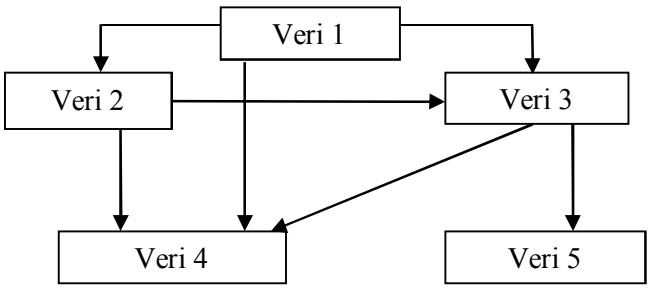


Ağ Veri Modeli

Hiyerarşik veri tabanlarının yetersiz kalmasından dolayı; 1960'ların sonunda toplanan Conference on Data System Languages (CODASYL) isimli bilimsel konferanstaki veri tabanı çalışma grubu (Database Task Group – DBTG) bilim adamlarının ortak çalışması sonucu ortaya konulmuş bir veri tabanı türüdür. Ağ veri tabanları verileri, ağaçların daha da gelişmiş hali olan graflar (ağacın kendisi de özel bir graftır) şeklinde saklarlar ve bu yapı en karışık yapılardan biridir.

İlişkisel Veri Modeli

1970'lerin başında E.F.Codd tarafından geliştirilmiştir. Bu sistemde veriler tablo şeklinde saklanırlar. Bu veri tabanı yönetim sisteminde; veri alışverişi için özel işlemler kullanılır. Bu işlemlerde tablolar operandlar olarak kullanılır. Tablolar arasındaki matematiksel bağıntılarla (ilişkilerle) temsil edilen ilişkiler belirtilir. Günümüzde hemen hemen tüm veri tabanı yönetim sistemleri ilişkisel veri modelini kullanırlar. Bu model, matematikteki ilişki teorisine ("the relational theory") dayanır. İlişkisel veri modelinde (Relational Data Model) veriler basit tablolar halinde tutulur. Tablolar, satır ve sütunlardan oluşur. Sütunlar bilgi alanlarını, satırlar ise bilgilerin içeriğini belirlerler.



Nesneye Yönelik Veri Modeli
Nesneye yönelik programlama tekniğinde ortaya çıkmış bir veri modelidir. Nesneye yönelik veri tabanı, C + + gibi nesneye dayalı bir dille (OOPL) yazılmış olan ve yine C + + gibi nesneye dayalı (OOPL) bir dille kullanılan veri tabanı anlamına gelir.

Veri Tabanı Yönetim Sistemlerinin Sınıflandırılması
Veri Modeline Göre

1. Hiyerarşik
2. Ağ
3. İlişkisel
4. Nesneye Yönelik

Kullanıcı Sayısına Göre

1. Tek kullanıcılı
2. Çok kullanıcılı olmak üzere ikiye ayrılır.

Günümüzde PC tabanlı VTYS' ler de dâhil olmak üzere tek kullanıcılı veri tabanı yönetim sistemi pek kalmamıştır. Çok kullanıcılı ise sınırsız kullanıcı veya belirtilen sayı kullanıcı olarak iki kategoride satılır ve kullanılırlar.

Fiziksel Konumuna Göre

1. Merkezi: Veri tabanı fiziksel olarak tek bir merkezde konumlandırılmış ise buna Merkezi veri tabanı adı verilir.
2. Dağıtılmış: Veri tabanının çeşitli parçaları çeşitli fiziksel merkezlere dağıtılmış ise buna dağıtılmış veri tabanı adı verilir. Dağıtılmış veri tabanı sistemlerinde tüm merkezlerde aynı veri tabanı sistemi kullanılıyor ise buna Homojen dağıtılmış, farklı merkezlerde farklı veri tabanı sistemleri kullanılıyor ise buna Heterojen dağıtılmış veri tabanı sistemi adı verilir.

Maliyetine Göre
Maliyetlerine göre çok çeşitli veri tabanı sistemleri mevcuttur. Ücretsiz olarak kullanılabilen veri tabanları olduğu gibi, küçük ölçekli düşük ücretli veri tabanları ve büyük ölçekli işletmeler için tasarlanmış çok yetenekli ve yüksek ücreti veri tabanı sistemleri de mevcuttur.

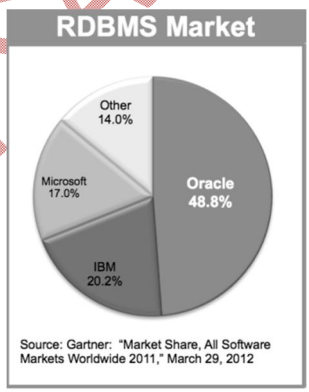
Veri Tabanı Yönetim Sistemi Programları

- MS SQL Server, bir orta ve büyük ölçekli VTYS' dir. ANSI SQL'e eklentiler yazmak için T-SQL'i destekler.
- Oracle, daha çok yüksek ölçekli uygulamalarda tercih edilen bir VTYS' dir. ANSI SQL'e eklentiler yapmak için PL/SQL geliştirilmiştir.
- Sysbase, bir orta ve büyük ölçekli VTYS' dir. ANSI SQL'E eklentiler yazmak için T-SQL komutlarını destekler. Ülkemizde daha çok bankacılık ve kamusal alanlarda tercih edilmektedir.
- Informix, bir orta ve büyük ölçekli VTYS' dir.

- MySQL, genellikle Unix – Linux temelli Web uygulamalarında tercih edilen bir VTYS' dir. Açık kod bir yazılımdır. Küçük – orta ölçeklidir. Özellikle Web için geliştirilmiş bir VTYS' dir.
- Postrage SQL, bu da MySQL gibi açık kod bir VTYS' dir.
- MS Access, çoklu kullanıcı desteği yoktur. İşletim sisteminin sağladığı güvenlik seçeneklerini kullanır. Bunun yanında belli sayıda kayda kadar (1000000 civarı) ya da belli bir boyutun (yaklaşık 250 MB) altına kadar bir sorun çıkartmadan kullanılabilecek bir küçük ölçekli VTYS' dir.
- Advantage, Türk programcılar tarafından geliştirilen bir orta ve büyük ölçekli VTYS' dir.
- DB/2, IBM'in framework'lere yönelik büyük ölçekli VTYS' dir.
- Bunların dışında daha birçok VTYS programı mevcuttur. VTYS' lerin Avrupa genelindeki yaklaşık olarak pazar payları Tabloda gösterilmiştir. En büyük Pazar payı IBM (DB/2) ile Oracle arasındadır. Hemen arkasından MS SQL Server, Informix ve Sybase gelmektedir. Yeni başlayanlar için; hangi VTYS'yi öğrenmem en iyisi olur sorusunu yanıtlamak gerekebilir. Ülkemizde insan kaynakları açısından en çok kalifiye eleman aranan VTYS Oracle ve arkasından da MS SQL Server gelmektedir.

VTYS'lerin Pazar Payları (Gartner, 2001)

VTYS	Yüzdesi (%)
IBM	37.8
Oracle	26.3
Microsoft	15.4
Informix	3.2
Sybase	3
Diğerleri	14.3



VTYS' lerin birçoğu ANSI SQL' in karşılayamadığı durumlarda kullanılmak üzere ek programlama komutları barındırırlar. Bu iş için MS SQL Server ve Sybase SQL Server Transact SQL (T-SQL) denilen komut takımlarını içerir. Oracle ise PL/SQL ile bu işe çözüm getirmiştir. Bu diller sayesinde, Stored (saklı prosedürler), Triggers, Fonksiyon gibi veri tabanları için vazgeçilmez olan nesneler yazılabilmektedir.

SQL(Structured Query Language – Yapısal Sorgulama Dili)

İlişkisel veri tabanı yönetim sistemlerinde tutulan verileri yönetmek sorgulamak için tasarlanmış özel amaçlı bir alt programlama dilidir.

SQL Edgar CODD tarafından 1970 li yılların başında geliştirilen ilişkisel veri modelini kullanan veri tabanı sistemlerinde verileri sorgulayabilmek için geliştirilmiştir.

SQL ilk olarak 1970 yıllarda IBM firmasının IMS sistemi için Donald D. Chamberlin ve Raymond F. Boyce tarafından geliştirilmiştir. O yıllarda dilin ismi SEQUEL (Structured English Query Language) olarak adlandırılmıştır. Daha sonraları SQL adını almıştır. İlk olarak 1986 yılında ANSI ve 1987 yılında ISO tarafından standardize edilmiş ve sonraki yıllarda ek özellikler eklenerek standardize işlemi devam etmiş ve günümüze gelmiştir. SQL bugün tüm veri tabanı sistemleri için kullanılabilen ortak bir dil haline gelmiştir.

Son derece esnek ve kullanımı kolay bir dildir.

Operatörler

1. **Aritmetik Operatörler**

- + Toplama İşlemi
- - Çıkarma İşlemi
- * Çarpma İşlemi
- / Bölme İşlemi
- % Mod Alma İşlemi

2. **Karşılaştırma Operatörleri**

- = Eşit mi?
- < > veya ! = Eşit değil mi?
- > Büyük mü?
- < Küçük mü?
- >= Büyük veya eşit mi?
- <= Küçük veya eşit mi?
- BETWEEN Aralık Sorgulaması
- LIKE Joker Karakter Sorgulaması
- IN Bir Alana ait Değer Sorgulaması

3. **Mantıksal Operatörler**

- AND Birden fazla şart ifadesinde Mantıksal VE işlemi
- OR Birden fazla şart ifadesinde Mantıksal VEYA işlemi
- NOT Şart ifadesinde elde değeri tersine çeviren DEĞİL işlemi

Çeşitli Veri Tipleri

1. **Karakter Veri Türleri**
   - CHAR: Karakter ve alfa sayısal tipinde verileri saklamak için kullanılır. Standart bir veri tipidir.
     Kullanımı ➔ CHAR(Uzunluk)
   - VARCHAR: Karakter ve alfa sayısal tipinde verileri saklamak için kullanılır. Standart bir veri tipi değildir. Belirtilen uzunluk verinin alabileceği maksimum uzunluktur. Verinin gerektirdiği kadar uzunluk kullanılır.
     Kullanımı ➔ VARCHAR(Uzunluk)
   - NCHAR: Karakter ve alfa sayısal tipinde verileri saklamak için kullanılır. Standart bir veri tipi değildir. Dillerin kendine has karakterlerini gösterebilmek için kullanılan uluslararası karakter kümesini destekler. Tek karakter bellekte 2 Byte yer kaplar.
     Kullanımı ➔ NCHAR(Uzunluk)

2. **Sayısal Veri Türleri**
    - INTEGER: Tam sayı türündeki verileri saklamak için kullanılır. Bir çok veri tabanı isteminde 4 Byte yer kaplar.
    - SMALLINT: Tam sayı türünden daha küçük değerdeki verileri saklamak için kullanılır. Standart bir veri tipi değildir. Bir çok sistemde 2 Byte yer kaplar.
    - DECIMAL(X,Y) – REAL(X,Y) – NUMERIC(X,Y) : Ondalıklı sayıları saklamak için kullanılan veri tipleridir. X verinin alabileceği toplam basamak sayısını Y ise ondalık basamak sayısını belirtir.
    - FLOAT(X,Y) : Ondalıklı sayı tipindeki verileri saklar. X ve Y değerleri üstteki veri tipleriyle aynıdır. Veri Bilimsel formatta saklar.

3. **Tarih Zaman Veri Türleri**
    - DATE: Tarih türündeki verileri saklamak için kullanılan veri tipidir.
    - TIME: Zaman türündeki verileri saklamak için kullanılan veri tipidir.
    - DATETIME: Tarih ve Zaman türündeki verileri birlikte saklamak için kullanılan veri tipidir.
    - TIMESTAMP: Tarih ve Zaman türündeki verileri birlikte saklamak için kullanılan veri tipidir.

4. **Mantıksal Veri Türleri**
    - LOGICAL: Evet/Hayır, Doğru/Yanlış, Açık/Kapalı gibi sadece iki farklı değer alabilen verileri saklamak için kullanılır.

Uygulama Veri Tabanı

PERSONEL									
SICIL_NO	**SOS_GUV_NO**	**ADI**	**SOYADI**	**DOG_TAR**	**ADRES**	**CINSIYET**	**BOL_NO**	**BRUT_MAAS**	**YON_SOS_GUV_NO**
CHAR(5)	CHAR(10)	CHAR(20)	CHAR(20)	DATE	CHAR(100)	CHAR(1)	CHAR(5)	REAL	CHAR(10)

BOLUMLER		
BOLUM_NO	**BOLUM_ADI**	**YON_SOS_GUV_NO**
CHAR(5)	CHAR(20)	CHAR(10)

CALISMA		
PER_SICIL_NO	**PROJE_NO**	**SAAT**
CHAR(5)	CHAR(5)	INTEGER

PROJE			
PROJE_NO	**PROJE_ADI**	**BOLUM_NO**	**YER**
CHAR(5)	CHAR(20)	CHAR(5)	CHAR(20)

PARCA				
PARCA_NO	**PARCA_ADI**	**PROJE_NO**	**FIYAT**	**AGIRLIK**
CHAR(5)	CHAR(20)	CHAR(5)	REAL	REAL

SATICI		
SATICI_NO	**ADI_SOYADI**	**ADRES**
CHAR(5)	CHAR(40)	CHAR(100)

PARCA_SATICI		
PARCA_NO	**SATICI_NO**	**MIKTAR**
CHAR(5)	CHAR(5)	INTEGER

PERSONEL : İşletmede çalışan personellerle ilgili bilgilerin tutulduğu tablo.
BOLUMLER: İşletmede var olan departmanlarla ilgili bilgilerin tutulduğu tablo.
CALISMA: Kimin hangi projede ne kadar saat çalıştığı bilgisinin tutulduğu tablo.
PROJE: Projelerle ilgili bilgilerin tutulduğu tablo.
PARCA: Üretilen parçalarla ilgili bilgilerin tutulduğu tablo.
SATICI: Üretilen parçaları satan pazarlamacılarla ilgili bilgilerin tutulduğu tablo.
PARCA_SATICI: Hangi satıcının hangi parçadan ne kadar sattığı bilgisinin tutulduğu tablo.

SQL Komut Yapısı

SQL deyimleri veri tabanları üzerinde çeşitli işlemleri yerine getirir. Veri tabanından sorgulama yapmak için SELECT, ekleme yapmak için INSERT güncelleme yapmak için UPDATE, silme yapmak için DELETE, yeni tablo oluşturmak için CREATE TABLE gibi komutlara sahiptir. Bu komutlar, işlevlerine göre şu şekilde ayrılır:

- DDL (Data DefinitionLanguage): Veri Tanımlama Dili
- DML (Data Manipulation Language) : Veri İşleme Dili
- DCL (Data Control Language): Veri Kontrol Dili

Veri Tanımlama Dili (DDL)

SQL Server içinde veri tabanı, tablo ve kullanıcı tanımlı veri tipleri gibi nesneler oluşturmak ve bunları yapılandırmak için kullanılır. Temel komutları aşağıdaki şekildedir:

Temel Komutlar	Açıklama
CREATE	Nesne oluşturmak için kullanılır.
ALTER	Nesneler üzerinde değişiklik yapmak için kullanılır.
DROP	Nesneleri silmek için kullanılır.

Veri İşleme Dili (DML)

Veri tabanı içindeki veriler ile ilgili işlemler yapılmasını sağlar. Temel komutları aşağıdaki şekildedir.

Temel Komutlar	Açıklama
SELECT	Veri tabanındaki verileri seçmeyi sağlar.
INSERT	Veri tabanına yeni veriler eklemek için kullanılır.
UPDATE	Veriler üzerinde değişiklik(güncelleme) yapmak için kullanılır.
DELETE	Veri tabanından veri silmek için kullanılır.

Veri Kontrol Dili (DCL)

DCL, bir veri tabanı ile ilişkili kullanıcıları ve rollerin izinlerini değiştirmek için kullanılır. Diğer bir deyişle verilere erişim yetkilerini düzenlemede kullanılır. Temel komutları aşağıdaki şekildedir.

Temel Komutlar	Açıklama
GRANT	Bir kullanıcının verileri kullanmasına ve SQL komutlarını çalıştırmasına izin verir.
DENY	Bir kullanıcının verileri kullanmasını kısıtlar.
REVOKE	Daha önce yapılan tüm kısıtlama ve izinleri iptal eder.

SQL KOMUTLARI

CREATE TABLE komutu: Veri tabanında tablo tanımlamak için kullanılan komuttur.

Kullanım Şekli ➔ CREATE TABLE Tablo_Adı(Alan_adı Veri_Tipi, Alan_Adı Veri_Tipi, …..)

PERSONEL									
SICIL_NO	**SOS_GUV_NO**	**ADI**	**SOYADI**	**DOG_TAR**	**ADRES**	**CINSIYET**	**BOL_NO**	**BRUT_MAAS**	**YON_SOS_GUV_NO**
CHAR(5)	CHAR(10)	CHAR(20)	CHAR(20)	DATE	CHAR(100)	CHAR(1)	CHAR(5)	REAL	CHAR(10)

Örnek:
```
CREATE TABLE PERSONEL ( SICIL_NO CHAR(5),
                        SOS_GUV_NO CHAR(10),
                        ADI CHAR(20),
                        SOYADI CHAR(20),
                        DOG_TAR DATE,
                        ADRES CHAR(100),
                        CINSIYET CHAR(1),
                        BOL_NO CHAR(5),
                        BRUT_MAAS REAL,
                        YON_SOS_GUV_NO CHAR(10)  );
```

NOT NULL İfadesi: Kayıt girişi sırasında belirli alan ya da alanların boş geçilmemesi gerekiyorsa yani o alanlara mutlaka bir veri girilmesi gerekiyorsa tablo tanımlanırken bu alanlara NOT NULL ifadesi eklenir ve alan kayıt girişi sırasında boş geçilmesi önlenir.

Örnek:
```
CREATE TABLE PERSONEL ( SICIL_NO CHAR(5) NOT NULL,
                        SOS_GUV_NO CHAR(10) NOT NULL,
                        ADI CHAR(20),
                        SOYADI CHAR(20),
                        …… )
```
Tablo yukarıdaki gibi tanımlanırsa SICIL_NO ve SOS_GUV_NO alanlarına kullanıcı mutlaka bir veri girmek zorundadır.

INSERT Komutu: Veri tabanında tablolara yeni bir kayıt eklemek için kullanılır.

Kullanım Şekli ➔
```
INSERT INTO Tablo_Adı
        VALUES (Değer1 , Değer2, ……..)
```
Yukarıdaki kullanımda tablodaki tüm alanlara veri girişi yapılacaksa tablodaki alan sırasına uygun olarak değerler belirtilir.

Kullanım Şekli ➔
```
INSERT INTO Tablo_Adı (Alan_Adı1, Alan_Adı2, ……)
        VALUES (Değer1 , Değer2, ……..)
```
Yukarıdaki kullanımda tablodaki tüm alanlara veri girişi yapılacaksa belirtilen alan sırasına uygun olarak değerler belirtilir.

SELECT Komutu: Verilere erişmek için en sık kullanılan komuttur. Herhangi bir sorgulama sonucunda listelenecek alanların seçildiği komuttur. Tablolardaki belirli alanlar sorgu sonucunda gösterilecek ise alan adları tek tek belirtilir, tablolardaki tüm alanlara ait bilgiler listelenecekse tüm alan adlarını ifade etmek için * operatörü kullanılabilir.

Kullanım Şekli ➔ SELECT Alan_Adı, Alan_Adı, …

FROM Komutu: Herhangi bir sorgulamada istenilen sonuçlara ulaşmak için gerekli olan tabloların belirtildiği komuttur.

Kullanım Şekli ➔ FROM Tablo_Adı, Tablo_Adı, …

SELECT ve FROM komutlarla sorgularda birlikte kullanılan komutlardır. SELECT komutuyla sorgu sonucunda listelenecek alanları FROM komutuyla sorguda kullanılacak tabloları belirtiriz.

Örnek: İşletmede çalışan personellerin adı, soyadı ve maaş bilgilerini listeleyen SQL komutunu yazınız?

```
SELECT ADI, SOYADI, BRUT_MAAS
FROM PERSONEL ;
```

WHERE Komutu: Sorgulamada belirli şart ya da şartlara uyan kayıtlar sorgulamak istendiğinde kullanılan komuttur. Sorguda tek şart ifadesi varsa karşılaştırma operatörleriyle şart ifadesi belirtilir, birden çok şart ifadesi varsa şart ifadeleri uygun olan mantıksal operatörlerle birleştirilir.

Kullanım Şekli ➔ Tek şart → WHERE Şart_İfadesi
Birden Çok Şart → WHERE Şart_İfadesi1 AND/OR Şart_İfadesi1 …

Örnek: Maaşı 1500 ve yukarısı olan personellerin sicil numarası, adı, soyadı ve maaş bilgilerini listeleyen SQL komutunu yazınız?

```
SELECT SICIL_NO, ADI, SOYADI, BRUT_MAAS
FROM PERSONEL
WHERE BRUT_MAAS >= 1500 ;
```

IN Operatörü: Belirli bir alan ait değerleri sorgulama sonucuna dahil etmek için WHERE komutu içinde kullanılan operatördür. Operatörün önüne NOT operatörü eklenirse tam tersi işlem görür yani belirtilen değerleri sorgu sonucuna dahil etmez.

Kullanım Şekli ➔ WHERE Alan_Adı IN (“Değer1”, “Değer2”, …)

Örnek: Çalıştığı bölüm numarası BOL01, BOL02 ya da BOL03 olan personellerin adı, soyadı ve çalıştıkları bölüm numarasını bilgilerini listeleyen SQL komutunu yazınız?

```
SELECT ADI, SOYADI, BOL_NO
FROM PERSONEL
WHERE BOL_NO IN (“ BOL01”, “BOL02”, “BOL03” ) ;
```

BETWEEN Operatörü: Bir alana ait belirli aralıktaki değerleri sorgu sonucuna dahil eden WHERE komutu operatörüdür. Operatörün önüne NOT eklenirse tam tersi işlevi yerine getirir.

Kullanım Şekli ➔ WHERE Alan_Adı BETWEEN Değer1 AND Değer2

Örnek: Maaşları 1500 ile 2000 TL arasında olan personellerin sicil numarası, adı ve soyadı bilgilerini listeleyen SQL komutunu yazınız?

```
SELECT SICIL_NO, ADI, SOYADI
FROM PERSONEL
WHERE BRUT_MAA BETWEEN 1500 AND 2000 ;
```

LIKE Operatörü: Joker karakterler vasıtasıyla belirli bir grup veriyi sorgu sonucuna dahil eden WHERE komutu operatörüdür. Operatörün önüne NOT eklenirse tam tersi işlevi yerine getirir.

Kullanım Şekli ➔ WHERE Alan_Adı LIKE “İfade”

Eşleşme türü	Örnek	Eşleşme (True verir)	Eşleşme yok (False verir)
Birden fazla karakter	a*a	aa, aBa, aBBBa	aBC
	ab	abc, AABB, Xab	aZb, bac
Özel karakter	a[*]a	a*a	aaa
Birden fazla karakter	ab*	abcdefg, abc	cab, aab
Tek karakter	a?a	aaa, a3a, aBa	aBBBa
Tek hane	a#a	a0a, a1a, a2a	aaa, a10a
Karakter aralığı	[a-z]	f, p, j	2, &
Aralık dışı	[!a-z]	9, &, %	b, a
Rakam değil	[!0-9]	A, a, &, ~	0, 1, 9
Birleşik	a[!b-m]#	An9, az0, a99	abc, aj0

Eşleşen karakter	Microsoft Access SQL	ANSI SQL
Tek bir karakter	?	_ (alt çizgi)
Sıfır veya daha fazla karakter	*	%

Örnek: Adı A' dan E' ye kadar olan herhangi bir harfle başlayan ve soyadında S harfi olan personellerin adı ve soyadı bilgilerini listeleyen SQL komutunu yazınız?

```
SELECT ADI, SOYADI
FROM PERSONEL
WHERE ADI LIKE "[A-E]*"  AND  SOYADI LIKE " *S*" ;
```

ORDER BY Komutu: Sorgulama sonucunda listelenen kayıtları belirli alan yada alanlara göre artan ya da azalan şekilde sıralamak için kullanılan komuttur. Belirtilen alan için artan (Sayısal Veri için 0→9 , Karakter Veri için A→Z) sıralama olacaksa ASC deyimi eklenir. Belirtilen alan için azalan (Sayısal Veri için 9→0 , Karakter Veri için Z→A) sıralama olacaksa DESC deyimi eklenir. Herhangi bir alan için ASC ya da DESC deyimi belirtilmemiş ise varsayılan olarak o alan için artan sıralama yani ASC işlemi uygulanır.

Kullanım Şekli ➔ ORDER BY Alan_Adı ASC / DESC, ….

Örnek: İşletmede çalışan erkek personellerin adı, soyadı ve maaş bilgilerini maaş alanına göre artan soyadı alanına göre azalan sırada listeleyen SQL komutunu yazınız?

```
SELECT ADI, SOYADI, BRUT_MAAS
FROM PERSONEL
WHERE CINSIYET="E"
ORDER BY BRUT_MAAS ASC, SOYADI DESC ;
```

Aritmetiksel İşlemler

SQL komutlarında SELECT ifadesi içinde aritmetiksel işlemler gerçekleştirilip yapılan işlemin sonucu ayrı alanda listelenebilir. Bunun için herhangi bir sayısal değer içeren alanları +, -, *, / aritmetik operatörleriyle işlemek sokmak yeterlidir.

Örnek: İşletmede çalışan erkek personellerin adı, soyadı ve maaş bilgileri ile maaşlarının %25 fazlasını listeleyen SQL komutunu yazınız?

```
SELECT ADI, SOYADI, BRUT_MAAS, BRUT_MAAS*1.25
FROM PERSONEL
WHERE CINSIYET="E"
```

AS Deyimi: Sorgu sonucunda listelen alanlara yeni başlıklar vermek için kullanılan ifadedir. SELECT komutu içinde kullanılır. Mevcut bir alan ismi de değiştirilebilir herhangi bir aritmetik işlem sonucunda listelenen alana da başlık ismi verilebilir.

Örnek: İşletmede çalışan kadın personellerin adı, soyadı ve maaş bilgileri ile maaşlarının %25 fazlasını listeleyen SQL komutunu yazınız?

```
SELECT ADI, SOYADI, BRUT_MAAS, BRUT_MAAS*1.25 AS YENI_MAAS
FROM PERSONEL
WHERE CINSIYET="K"
```

Kümeleme Fonksiyonları
SELECT komutu içinde alanlara ait çeşitli işlemleri gerçekleştirip sonuç olarak tek değer döndüren fonksiyonlardır.

SUM Fonksiyonu: Belirtilen sayısal alana ait değerlerin toplamını döndüren fonksiyondur.
Kullanım Şekli ➔ SELECT SUM(Alan_Adı), ….

AVG Fonksiyonu: Belirtilen sayısal alana ait değerlerin ortalamasını döndüren fonksiyondur.
Kullanım Şekli ➔ SELECT AVG(Alan_Adı), ….

MAX Fonksiyonu: Belirtilen sayısal alana ait değerlerin en büyüğünü döndüren fonksiyondur.
Kullanım Şekli ➔ SELECT MAX(Alan_Adı), ….

MIN Fonksiyonu: Belirtilen sayısal alana ait değerlerin en küçüğünü döndüren fonksiyondur.
Kullanım Şekli ➔ SELECT MIN(Alan_Adı), ….

COUNT Fonksiyonu: Belirtilen alana ait kayıtların ya da değerlerin sayısını döndüren fonksiyondur. Parametre olarak alan adı belirtildiği gibi * işareti de kullanılabilir.
Kullanım Şekli ➔ SELECT COUNT(Alan_Adı), ….

STDEVP Fonksiyonu: Belirtilen sayısal alana ait değerlerin (grup verisinin) standart sapmasını döndüren fonksiyondur.
Kullanım Şekli ➔ SELECT STDEVP(Alan_Adı), ….

STDEV Fonksiyonu: Belirtilen sayısal alana ait değerlerin (grup örneğinin) standart sapmasını döndüren fonksiyondur.
Kullanım Şekli ➔ SELECT STDEV(Alan_Adı), ….

VARP Fonksiyonu: Belirtilen sayısal alana ait değerlerin (grup verisinin) varyansını döndüren fonksiyondur.
Kullanım Şekli ➔ SELECT VARP(Alan_Adı), ….

VAR Fonksiyonu: Belirtilen sayısal alana ait değerlerin (grup örneğinin) varyasını döndüren fonksiyondur.
Kullanım Şekli ➔ SELECT VAR(Alan_Adı), ….

FIRST Fonksiyonu: Sorgu sonucunda istenilen şartları sağlayan tablodaki ilk kaydın belirtilen alana ait değerini döndüren fonksiyondur. Belirtilen alan sayısal olmak zorunda değildir. Alanda değer olarak ne varsa o değeri döndürür.
Kullanım Şekli ➔ SELECT FIRST(Alan_Adı), ….

LAST Fonksiyonu: Sorgu sonucunda istenilen şartları sağlayan tablodaki son kaydın belirtilen alana ait değerini döndüren fonksiyondur. Belirtilen alan sayısal olmak zorunda değildir. Alanda değer olarak ne varsa o değeri döndürür.
Kullanım Şekli ➔ SELECT LAST(Alan_Adı), ….

Örnek: İşletmede çalışan kadın personellere ödenen toplam maaş miktarını, ortalama maaş miktarını, en yüksek maaş miktarını, en küçük maaş miktarını, kadın personellerin sayısını, maaşlarının standart sapmasını ve varyasını, tablodaki ilk kadın personelin ve son kadın personelin adlarını listeleyen SQL komutunu yazınız?

```
SELECT SUM(BRUT_MAAS) AS TOPLAM_MAAS,
        AVG(BRUT_MAAS) AS ORTALAMA_MAAS,
        MAX(BRUT_MAAS) AS EN_YUKSEK_MAAS,
        MIN(BRUT_MAAS) AS EN_KUCUK_MAAS,
        COUNT(*) AS PER_SAYISI,
        STDEVP(BRUT_MAAS) AS STANDART_SAPMASI,
        VARP(BRUT_MAAS) AS VARYANSI,
        FIRST(ADI) AS ILK_PER_ADI,
        LAST(ADI) AS SON_PER_ADI
FROM PERSONEL
WHERE CINSIYET=”K”
```

GROUP BY Komutu: Sorgulama işleminde verilerin belirli alan ya da alanlara göre gruplandırılması için kullanılan komuttur. Herhangi bir alana göre gruplandırma işlemi yapıldığında belirtilen alanın her farklı değeri farklı bir grup oluşturarak bunlar üzerinde kümeleme fonksiyonları ile işlemler gerçekleştirilebilir.

Kullanım Şekli ➔ GROUP BY Alan_Adı, …..

Örnek: İşletmede var olan bölümlerdeki personel sayılarını, bölümlerin ortalama maaşlarını bölüm numaralarıyla birlikte listeleyen SQL komutunu yazınız?

```
SELECT COUNT(*) AS PER_SAYISI,
        AVG(BRUT_MAAS) AS ORT_MAAS,
        BOL_NO
FROM PERSONEL
GROUP BY BOL_NO;
```

HAVING Komutu: Gruplandırılmış veriler üzerinde şart ifadesi belirtmek için kullanılır. Gruplandırılmış verilerde şart ifadesi belirtilirken SUM, AVG, MAX, MIN ve COUNT fonksiyonlarından en az birisinin mutlaka kullanılması gerekir. WHERE komutu tek tek kayıtlar üzerinde şart sınaması yaparken, HAVING grup verileri üzerinde şart sınaması yapar. HAVING ve WHERE komutları aynı sorgu içinde kullanılabilir. Ancak HAVING komutunun kullanılabilmesi için verilerin gruplandırılmış olması gerekir.

Örnek: Personel sayısı 3 ten fazla olan bölümlerinbölüm numarası ve personel sayılarını listeleyen SQL komutunu yazınız?

```
SELECT BOL_NO, COUNT(*) AS PER_SAYISI
FROM PERSONEL
GROUP BY BOL_NO
HAVING COUNT(*)>3;
```

Örnek: Sadece kadın personellerin kayıtlarını dikkate alarak bölümlerdeki ortalama maaş değeri 1000 TL nin üstündeki bölüm numaralarını ve ödenen toplama maaş miktarını listeleyen SQL komutunu yazınız?

```
SELECT BOL_NO, SUM(BRUT_MAAS) AS TOPLAM_MAAS
FROM PERSONEL
WHERE CINSIYET=”K”
GROUP BY BOL_NO
HAVING AVG(BRUT_MAAS)>1000;
```

ALL, DISTINCT, DISTINCROW, TOP n, TOP n PERCENT İfadeleri

SELECT iadesinde sonra kullanılırlar, sorgu sonucunda görüntülenecek kayıtların tamamının tekrarsız kayıtların ya da baştan itibaren belirli sayıda kayıtların listelenmesinde kullanılır.

ALL: SELECT için varsayılan ifadedir. Sorguda herhangi bir ifade(DISTINCT, TOP n vs..) belirtilmezse varsayılan olarak bu ifade kullanılmış olur. Sorguda işlenen ve şartları sağlayan tüm kayıtların listelenmesini sağlar.

DISTINCT: Sorgu sonucunda tekrar eden kayıtların görüntülenmesini engeller. Tekrar eden kayıtlardan sadece bir tanesini gösterir.

```
SELECT BOL_NO
FROM PERSONEL
```

sorgusunda benzer 18 kayıt listelenirken,

```
SELECT DISTINCT BOL_NO
FROM PERSONEL
```

sorgusu ile tekrar eden kayıtlar görüntülenmez değerleri farklı 5 kayıt listelenir.

DISTINCTROW: Birden çok tablonun kullanıldığı sorgularda eşleşen kayıtlardan tekrar edenlerin görüntülenmesini engeller.

TOP n: ORDER BY komutu ile belirli bir alana göre sıralanmış kayıtlardan ilk n tanesinin görüntülenmesini sağlar.

Örnek: En yüksek maaş alan ilk 5 personelin ADI, SOYADI ve BRUT_MAAS bilgilerini listelemek için gerekli sorguyu yazınız?

```
SELECT TOP 5 ADI, SOYADI, BRUT_MAAS
FROM PERSONEL
ORDER BY BRUT_MAAS DESC;
```

TOP n PERCENT: Bu şekildeki kullanımda ise PERCENT ifadesi n ile belirtilen rakamın yüzde cinsinden değerlendirilmesini sağlar. Örneğin en yüksek maaş personellerin ilk %25’i için SELECT TOP 25 PERCENT yazılmalıdır.

BİRDEN FAZLA TABLOYU İLİŞKİLENDİREREK SORGULAMA

Şu ana kadar sorgulama işlemlerinde hep tek tablo üzerinde sorgulamalar gerçekleştirilmiştir. Ancak gerçek işletme uygulamalarında sorgulama yazılacağı zaman daha sık karşılaşılan ve daha güç olan sorgular birden çok tablonun kullanıldığı sorgulardır.

BİRLEŞTİRME (JOIN) İŞLEMİ

Birden fazla tablonun kullanıldığı sorgularda istenilen sonuçlara ulaşmak için sorguda kullanılan tabloların birleştirilmesi gerekebilir. Birleştirme işlemini gerçekleştirmek için birkaç yöntem vardır. Bizim kullanacağımız yöntem WHERE komutunda tabloların ortak alanlarını eşitleyerek birleştirme işlemi olacaktır.

İki ya da daha fazla tabloyu sorguda kullanabilmek için tabloların ortak alanlarından ikişer ikişer birleştirilip oluşturulan tek tablodan istenilen bilgiler listelenebilir. İki tablonun ortak alanlarını tespit edebilmek içinse 3 kural vardır. Birincisi alanların veri tipleri aynı olmalıdır. İkincisi alanların veri boyutları aynı olmalıdır, üçüncüsü ise içlerinde taşıdıkları bilgiler benzer olmalıdır yani bir alanda örneğin bölüm numarası bilgisi varsa ilişkilendirilecek diğer alanda da yine bölüm numaraları olması gerekir. Tabi veri tipleri ve boyutları da aynı olmak kaydıyla. Bu şartları sağlayan alanlar WHERE komutu içinde birbirine eşitlenerek aynı değere sahip olan kayıtlar aynı satırda birleşir ve tabloların birleştirilme işlemi gerçekleştirilir.

Uygulama veri tabanında kullandığımız PERSONEL ve BOLUMLER tablolarını birleştirmek için ortak alanlarını bulmamız gerekir. Bu alanlar

PERSONEL Tablosunun BOL_NO ve YON_SOS_GUV_NO alanları ile
BOLUMLER Tablosunun BOLUM_NO ve YON_SOS_GUV_NO alanlarıdır. Bu alanlardan birini kullanarak bu tablolar birleştirilebilir.

Örnek: İşletmede çalışan personellerin Adı, Soyadı ve çalıştıkları Bölüm Adı bilgilerini listeleyen SQL komutunu yazınız?
Bu sorguyu yazabilmek için iki tabloya ihtiyacımız var çünkü Personel Adı ve Soyadı bilgileri PERSONEL tablosunda Bölüm Adı bilgisi ise BOLUMLER tablosunda yer alan bilgidir. Bu iki tablonun ortak alanlarından birleştirilip istenilen sonuçların alınması için;

```
SELECT ADI, SOYADI, BOLUM_ADI
FROM PERSONEL, BOLUMLER
WHERE BOL_NO=BOLUM_NO
```

yazılırsa istenilen sonuçlar alınabilir sorgu ile ilgili başka şart ifadeleri varsa bunlarda WHERE komutunda belirtilir.

Dikkat edilmesi gereken diğer nokta ise sorguda belirttiğimiz alanlar birden fazla tabloda aynı isimde tanımlanmış ise hangi tablonun alanını kullandığımızı belirtmek için Tablo ismi de belirtilir. Örneğin yukarıdaki sorguda ortak alan olarak Bölüm Numarası değil de Yönetici Sosyal Güvenlik Numarası alanlarını kullanmış olsaydık alan ismi her iki tabloda da aynı olduğu için sorguyu aşağıdaki şekilde tablo ismi belirterek yazmamız gerekecekti.

```
SELECT ADI, SOYADI, BOLUM_ADI
FROM PERSONEL, BOLUMLER
WHERE PERSONEL.YON_SOS_GUV_NO=BOLUMLER.YON_SOS_GUV_NO
```

Örnek: SATICI1 adlı satıcının sattığı parçaların üretiminde görev alan personellerin Adı, Soyadı ve Maaş bilgilerini listeleyen SQL komutunu yazınız?
Sorguyu yazabilmek için PERSONEL, CALISMA, PARCA, PARCA_SATICI ve SATICI tablolarına ihtiyacımız var bu tabloları ikişer ikişer birleştirip istediğimiz sonucu alabiliriz.

```
SELECT ADI, SOYADI, BRUT_MAAS
FROM PERSONEL, CALISMA, PARCA, PARCA_SATICI, SATICI
WHERE PERSONEL.SICIL_NO=CALISMA.PER_SICIL_NO AND
        CALISMA.PROJE_NO=PARCA.PROJE_NO AND
        PARCA.PARCA_NO=PARCA_SATICI.PARCA_NO AND
        PARCA_SATICI.SATICI_NO=SATICI.SATICI_NO AND
        SATICI.ADI_SOYADI="SATICI1";
```

EŞDEĞER TABLO SORGULAMASI
Daha önceden tanımlanmış bir tablonun farklı isimlerde eşdeğerleri oluşturularak aynı sorguda kullanılabilir. Eşdeğer tablo oluşturma işlemi FROM komutunda gerçekleştirilir. Bir tablonun tamamen alanlarıyla verileriyle birebir aynı olan farklı isimlerde tablolar oluşturulur ve bu tablolar aynı sorguda kullanılır. Örneğin PERSONEL tablosunun A ve Be isminde iki adet eşdeğerini oluşturmak için FROM PERSONEL A, PERSONEL B şeklinde yazılabilir. Bunu belirttikten A ve B isminde sadece o sorgu için geçici iki tablo oluşturulur ve bu tablolar PERSONEL tablosunun aynısıdır. İsim olarak A ve B yerine farklı isimlerde verilebilir.

Örnek: İşletmede çalışan personellerin Adı Soyadı ve yöneticilerinin Adı Soyadı bilgilerini listeleyen SQL komutunu yazınız?

```
SELECT PER.ADI AS PERSONEL_ADI, PER.SOYADI AS PERSONEL_SOYADI
        YON.ADI AS YONETICI_ADI, YON.SOYADI AS YONETICI_SOYADI
FROM PERSONEL PER, PERSONEL YON
WHERE PER.YON_SOS_GUV_NO=YON.SOS_GUV_NO
```

Bu sorguda PER tablosunu Personel, YON tablosunu ise Yönetici tablosu olarak düşünüp uygun alandan birleştirdik ve istediğimiz sonuçlara ulaştık.

İÇ İÇE SORGULAMALAR
Bazı sorgularda sorgunun yapısı gereği iç içe SELECT komutlarının kullanılmasını gerektirir. Bu tarz sorgularda içteki sorguda elde edilen bilgiler dıştaki asıl sorguda istenilen sonuçlara ulaşmak için kullanılır.

Kullanım Şekli:

```
SELECT Alanlar
FROM Tablolar
WHERE Şartlar AND/OR Alan_Adı (IN, <, >,= ….)      ( SELECT Alan_Adı
                                                      FROM Tablolar
                                                      WHERE Şartlar ) …
```

Örnek: Fiyatı 25 ile 50 arasındaki parçaların üretiminde görev alan personellerin Adı Soyadı bilgilerini listeleyen SQL komutunu iç içe sorgulama yöntemiyle yazınız?

```
SELECT ADI, SOYADI,
FROM PERSONEL
WHERE SICIL_NO IN   ( SELECT PER_SICIL_NO
                      FROM CALISMA
                      WHERE PROJE_NO IN  ( SELECT PROJE_NO
                                           FROM PARCA
                                           WHERE FIYAT BETWEEN 25 AND 50 ) )
```

Yukarıdaki sorguda en içteki sorgu fiyatı 25 ile 50 arasında olan parçaların Proje Numarasını bir üst sorguya ortadaki sorgu o proje numaralı projelerde çalışan personellerin Sicil Numaralarını en dıştaki sorgu ise o sicil numaralı personellerin Adı Soyadı bilgilerini listeler.

ANY Deyimi

İç içe sorgulamada dıştaki sorguda karşılaştırılan kayıtlar için belirtilen şartı içteki sorguda üretilen değerlerin herhangi birisinin şartı sağlayıp sağlamadığını kontrol eder.

Örnek: MUHASEBE bölümünde çalışan personellerin herhangi birinden daha yüksek maaş alan ve ARGE bölümünde çalışan personellerin Adı Soyadı ve Maaş bilgilerini listeleyen SQL komutunu yazınız?

```
SELECT ADI, SOYADI, BRUT_MAAS
FROM PERSONEL, BOLUMLER
WHERE BOL_NO=BOLUM_NO AND
        BOLUM_ADI=”ARGE” AND
        BRUT_MAAS > ANY ( SELECT BRUT_MAAS
                          FROM PERSONEL, BOLUMLER
                          WHERE BOL_NO=BOLUM_NO AND
                                BOLUM_ADI=”MUHASEBE” )
```

ALL Deyimi

İç içe sorgulamada dıştaki sorguda karşılaştırılan kayıtlar için belirtilen şartı içteki sorguda üretilen değerlerin hepsinin şartı sağlayıp sağlamadığını kontrol eder.

Örnek: ARGE bölümünde çalışan personellerin hepsinden daha yüksek maaş alan ve YÖNETİM bölümünde çalışan personellerin Adı Soyadı ve Maaş bilgilerini listeleyen SQL komutunu yazınız?

```
SELECT ADI, SOYADI, BRUT_MAAS
FROM PERSONEL, BOLUMLER
WHERE BOL_NO=BOLUM_NO AND
        BOLUM_ADI=”YÖNETİM” AND
        BRUT_MAAS > ALL  ( SELECT BRUT_MAAS
                           FROM PERSONEL, BOLUMLER
                           WHERE BOL_NO=BOLUM_NO AND
                                 BOLUM_ADI=”ARGE” )
```

UNION Deyimi
UNION sözcüğü iki ya da daha çok SELECT komutunun sonucunda elde edilen tabloların küme birleşimi işlemini gerçekleştirir.

Kullanım Şekli

```
( SELECT Alanlar
  FROM Tablolar
  WHERE Şartlar ) UNION ( SELECT Alanlar
                          FROM Tablolar
                          WHERE Şartlar ) …..
```

UNION sözcüğü ile iki ya da daha çok SELECT sonucunu küme birleşimi işlemine tabi tutabilmesi için 2 şart vardır.

1. SELECT komutları sonucunda elde edilen tablolar aynı sayıda alana sahip olmalıdır.

Tablo-1 Tablo-2

2. Sonuç tablolarının karşılıklı olarak alanlarının aynı veri tipi ve aynı veri boyutuna sahip olması gerekir.

EXCEPT Deyimi
Bu sözcük iki tablo arasındaki küme farkı işlemini gerçekleştirir.

Kullanım Şekli

```
SELECT * FROM ( SELECT Alan_Adı
                FROM Tablolar
                WHERE Şartlar   EXCEPT   SELECT Alan_Adı
                                         FROM Tablolar
                                         WHERE Şartlar )
```

Tablo-1 EXCEPT Tablo-2

Tablo-1
A
B
C
E
F

Tablo-2
A =
B
C
D
F

Tablo-1 EXCEPT Tablo-2 = | E |

INTERSECT Deyimi
Bu sözcük iki tablo arasındaki küme kesişimi işlemini gerçekleştirir.

Kullanım Şekli

```
SELECT * FROM ( SELECT Alan_Adı
                FROM Tablolar
                WHERE Şartlar    INTERSECT    SELECT Alan_Adı
                                              FROM Tablolar
                                              WHERE Şartlar )
```

Tablo-1

A
B
C
E
F

INTERSECT

Tablo-2

A =
B
C
D
F

Tablo-1 INTERSECT Tablo-2 =

A
B
C
F

GÜNCELLEME İşlemi
Tablo kayıtlarını güncellemek için UPDATE komutu kullanılır.

Kullanım Şekli

```
UPDATE Tablo_Adı
SET Alan_Adı=Değer, Alan_Adı=Değer, ….
WHERE Şartlar
```

Örnek: Projelerde görev almış personellerin maaşlarına %10 oranında maaş zammı yapmak için gerekli güncelleme komutunu yazınız?

```
UPDATE PERSONEL
SET BRUT_MAAS=BRUT_MAAS * 1.10
WHERE SICIL_NO IN ( SELECT PER_SICIL_NO
                    FROM CALISMA)
```

SİLME İşlemi
Tablolardan kayıt silmek için DELETE komutu kullanılır.

Kullanım Şekli

```
DELETE FROM Tablo_Adı
WHERE Şartlar
```

Örnek: Projelerde 70 saatten az görev almış personellerin kayıtlarını silmek için gerekli silme komutunu yazınız?

```
DELETE FROM PERSONEL
WHERE SICIL_NO IN ( SELECT PER_SICIL_NO
                    FROM CALISMA
                    WHERE SAAT<70 )
```

TABLONUN YAPISINDA DEĞİŞİKLİK YAPMAK

ALTER TABLE komutu ile tablonun yapısında değişiklik yapmak mümkündür. Tablonun yapısında yapılacak değişikliğe göre ALTER TABLE konutuna ilave olarak yapılacak değişiklikle ilgili yardımcı bir sözcük kullanılır.

Mevcut Bir Tabloya Alan Eklemek

Var olan bir tabloya yeni alan eklemek için ALTER TABLE komutu ile beraber ADD sözcüğü kullanılır.

Kullanım Şekli

```
ALTER TABLE Tablo_Adı
ADD Yeni_Alan_adı Veri_Tipi
```

Örnek: PERSONEL Tablosuna personellerin Telefon Numaralarını tutabilmek için yeni bir alan ekleyiniz?

```
ALTER TABLE PERSONEL
ADD TEL_NO CHAR(10)
```

Mevcut Bir Tablo Alanında Değişiklik Yapmak

Var olan bir tablo alanının veri tipi ve boyutunu değiştirmek için ALTER TABLE komutu ile beraber MODIFY sözcüğü kullanılır.

Kullanım Şekli

```
ALTER TABLE Tablo_Adı
MODIFY  Alan_adı  Yeni_Veri_Tipi
```

Örnek: PERSONEL Tablosuna personellerin Telefon Numaralarını tutabilmek tanımlanan alanın veri boyutunu 9 karakter olacak şekilde düzenleyiniz?

```
ALTER TABLE PERSONEL
MODIFY  TEL_NO CHAR(9)
```

Mevcut Bir Tablodan Alan Silmek

Var olan bir tablodan mevcut bir alanı silmek için ALTER TABLE komutu ile beraber DROP sözcüğü kullanılır.

Kullanım Şekli

```
ALTER TABLE Tablo_Adı
DROP  Alan_adı
```

Örnek: PERSONEL Tablosundaki personellerin Telefon Numaralarını tutulduğu TEL_NO alanını silmek için gerekli komutu yazınız?

```
ALTER TABLE PERSONEL
DROP TEL_NO
```

Mevcut Bir Tablonun Adını Değiştirmek

Var olan bir tabloya yeni bir ad vermek için ALTER TABLE komutu ile beraber RENAME TABLE komutu kullanılır.

Kullanım Şekli

```
ALTER TABLE Tablo_Adı
RENAME TABLE Yeni_Tablo_Adı
```

Örnek: PERSONEL Tablosunun adını ELEMANLAR olarak değiştirmek için gerekli SQL komutunu yazınız?

```
ALTER TABLE PERSONEL
RENAME TABLE ELEMANLAR
```

Mevcut Bir Tablo Alanının Adını Değiştirmek

Var olan bir tablo alanının adını değiştirmek için ALTER TABLE komutu ile beraber RENAME sözcüğü kullanılır.

Kullanım Şekli

```
ALTER TABLE Tablo_Adı
RENAME Alan_Adı Yeni_Alan_Adı
```

Örnek: PERSONEL Tablosunun BOL_NO alanının adını BOLUM_NO olarak değiştirmek için gerekli SQL komutunu yazınız?

```
ALTER TABLE PERSONEL
RENAME BOL_NO BOLUM_NO
```

Mevcut Bir Tabloyu Tümüyle Silmek

Var olan bir tablonun veri tabanından tamamen silinmesi için gerekli komut DROP TABLE Tablo_Adı şeklindedir.

Örnek: SATICI tablosunu veri tabanından silmek için gerekli SQL komutunu yazınız?

```
DROP TABLE SATICI
```