T.C.
TARIM VE ORMAN BAKANLIĞI
Personel Genel Müdürlüğü

Unvan Değişikliği Sınavı
Ders Notu

Mühendis
(Bilgisayar)

Uyarı: Bu doküman çeşitli kaynaklardan faydalanılarak oluşturulmuş bir derlemedir. Hiçbir suretle özgün bir kitap özelliği taşımamaktadır. Sadece ilgili konularda bilgi edinme amaçlı olarak kullanılması için bu doküman oluşturulmuştur. Kesinlikle başka çalışmalarda dipnot olarak gösterilemez.

GÖREV ALANLARI VE ATAMA YAPILACAK GÖREVİN NİTELİĞİNE İLİŞKİN KONULAR

- İŞLETİM SİSTEMLERİ VE UYGULAMALARI
- JAVA PROGRAMLAMA DİLİ
- SQL PROGRAMLAMA DİLİ

Giriş

İşletim sistemleri konusu, bilgisayar bilimleri kapsamındaki en temel konulardan birini oluşturmaktadır. İşletim sistemleri, bilgisayar sistemlerinin gelişmesine paralel olarak gelişme göstermiştir. Çünkü, yeni gelişen bilgisayar mimarisi, yeni istekler ve ihtiyaç duyulan güvenliğe göre işletim sistemleri gelişmiştir. Bu nedenle, kullanıcı ile bilgisayar arasında bir köprü görevi yürüten ve donanıma en yakın yazılım birimi olan işletim sisteminin ayrıntılarını incelemeye geçmeden önce, bir bilgisayar sisteminin yapısını genel olarak ele almak gerekmektedir.

Bir bilgisayar sisteminin genel olarak 4 bileşeni vardır.

1.Donanım (İşlemci (CPU), belek ve I/O üniteleri gibi)

2.Sistem Yazılımları

a) İşletim Sistemi Yazılımları (Windows, Linux, Unix, Mac OS gibi)

b) Aygıt Sürücüler

3. Derleyiciler, Uygulama Yazılımları

4. Kullanıcı Yazılımları (Kullanıcıların geliştirdikleri yazılımlar)

Bu noktada şu hususu açıklamak gerekir ki CPU (Central Processor Unit) bilindiği gibi bir bilgisayar sisteminin en temel bileşeni olup, aynı şekilde bilgisayar sistemlerindeki disk, yazıcı, disket, terminal (ana makineye bağlı, sıradan uç kullanıcılar) vs. gibi I/O (Input/Output) üniteleri donanım (hardware) kısmı olmaktadır.

Yazılım (software) ise, hem bilgisayar sistemini oluşturan donanım birimlerinin yönetimini hem de kullanıcıların işlerini yapmak için gerekli olan programlardır. Yazılım olmaksızın bir bilgisayar sistemi, bir takım elektronik kartlar, kablolar ve mekanik bazı parçalardan ibaret bir cihazdır. Bir bilgisayar sistemi, üzerine işletim sistemi (Operating Systems) ve onun üzerine de diğer yazılımların yüklenmesi ve çalıştırılmasından sonra gerekli işlevleri yerine getirebilmektedir.

Bilgisayar yazılımları genel olarak 2 ana grupta incelenebilir.

- Sistem Yazılımları (System Software)
- Uygulama Yazılımları (Application Software)

Sistem Yazılımları (System Software); bilgisayarın kendisinin işletilmesini sağlayan, işletim sistemi, derleyiciler (compilers) (Yazılım programında, yazılan programı makine diline çeviren program) gibi yazılımlardır.

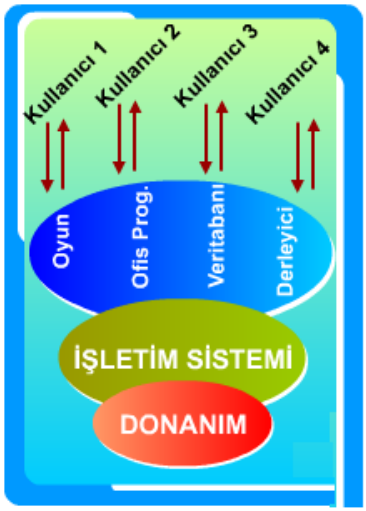
Uygulama Yazılımları (Application Software); bu kullanıcıların işlerine çözüm sağlayan örneğin çek, senet, stok kontrol, bordro, kütüphane kayıtlarını tutan programlar, bankalardaki müşterilerin para hesaplarını tutan programlar vs. gibi yazılımlardır.

Bütün sistem programları içinde en temel yazılım işletim sistemidir ki, bilgisayarın bütün donanım ve yazılım kaynaklarını kontrol ettiği gibi, kullanıcılara ait uygulama yazılımlarının da çalıştırılmalarını ve denetlenmelerini sağlar.

Modern bir bilgisayar sistemi, bir veya birden fazla işlemci (yada diğer bir söylemle "CPU"), gerçek bellek (RAM), saatler, terminaller, diskler, bilgisayar ağı (network) birimleri, yazıcı üniteleri, CD sürücüsü, disket ve teyp üniteleri gibi I/O ünitelerinden oluşmaktadır. Doğal olarak bir bilgisayar sistemi oldukça karmaşık bir yapıdadır.

Programcıları, donanımın bu karmaşık yapısından etkilenmemelerini sağlamak ve disk gibi donanım ünitelerinin nasıl çalıştıklarını anlamak zorunda bırakılmamaları için, donanımın üzerine ilave edilen yazılımların katmanlar şeklinde (layered system) oluşturulmaları ve bu sayede çok daha kolay bir şekilde, sistemin bütün parçalarının yönetilebilmesi şeklinde bir yapılanma, uzun yıllar önce geliştirilmiş bir yaklaşımdır.

Bu yapının en alttaki katmanı donanımı oluşturmaktadır. En alttaki katman, fiziksel üniteler, entegre devreler, kablolar, power (elektrik destek) üniteleri, disket sürücüleri, disk üniteleri ve diğer benzeri donanım birimlerinden oluşmaktadır. Bu katmanın mimari yapısı ile ilgilenmek ve bunları çalışma prensiplerini geliştirmek elektronik mühendislerinin işidir.



Bir bilgisayar sisteminin ikinci katmanında yer alan işletim sisteminin temel işlevi, donanımın karmaşıklığını kullanıcıya yansıtmamak ve daha elverişli ortam hazırlayıp, kullanıcının kolayca işini yapmasını sağlamaktır.

İşletim sistemi olan bir bilgisayarda, kullanıcının tek yapması gereken çalıştırmak istediği programın adını klavyeden yazıp ENTER tuşuna basmak veya simgesine (icon) tıklamaktır. Program dosyasının disket sürücüdeki yerinin bulunması, sürücüyü denetleyen kontrol devreleriyle gerekli görüşmeleri yapıp kafanın gerekli hareketleri yapmasını sağlayarak kayıtların belleğe aktarılması işinin sağlıklı bir şekilde yapılması, tamamen işletim sisteminin sorumluluğundadır.

İşletim Sistemi üzerinde yer alan bazı yazılımlar "Uygulama Yazılımları" olarak anılır. Örneğin derleyiciler (compilers) ; yazdığımız programı makine diline çeviren ara program, editörler (editors), yararlı programlar (utility) ; virüs temizleyen programlar gibi gerçek iş için yardımcılardır, veritabanı yönetim sistemleri (database management system) ve bilgisayar ağı

yazılımları (network software) yine birer sistem yazılımlarıdır. Ancak bu yazılımlar İşletim Sisteminin kendi öz parçaları değildir.

Özet olarak İşletim Sistemi, aynı zamanda donanım üreticisi olan veya yalnızca yazılım geliştiren özel bir firma tarafından yazılıp pazarlanan ve bir bilgisayar sisteminin donanım ve yazılım kaynaklarını kontrol eden ve kullanıcılarında kendi çözümlerini geliştirebildikleri ortamı hazırlayan bir sistem yazılımıdır.

İşletim Sistemi Tanımı

En büyüğünden en küçüğüne, bütün genel amaçlı bilgisayarlarda çalışan programlar, bir işletim sistemine gereksinim duyarlar. Bu yüzden bilgisayarlarda herhangi program çalıştırılmadan önce İşletim Sistemi ile programların ana belleğine (RAM) yerleştirilmeleri gerekir. Bu işlem genellikle bilgisayar ilk açıldığı zaman otomatik olarak yapılır ve HD' deki İşletim Sistemi ana belleğe yüklenir.

Bir işletim sisteminden beklenen hizmet, donanım ve yazılım kaynaklarının uyumlu ve verimli bir şekilde birlikte işletilmesidir. Örneğin kullanıcı Cobol veya Pascal dili ile geliştirdiği uygulama programını, bir bilgisayar sisteminde çalıştırabilmesi için, uygulama programı ve verilerini yazabileceği bir disk ünitesi, verilerini yazdıracağı yazıcı ünitesi, bu programı işletecek işlemci (CPU) ve gerçek bellek gibi donanım birimlerinin yanı sıra derleyici (compiler), yükeyici (loader) ve network yazılımları gibi yazılım birimlerine de ihtiyaç vardır.

İşletim sistemini, bir bilgisayar sisteminde kullanıcı ile iletişim kurarak, donanım ve yazılım nitelikli kaynakların kullanıcılar arasında adil bir biçimde paylaştırılmasını ve donanım ile yazılım birimlerinin etkin bir biçimde kullanılmalarını sağlayan sistem programları topluluğuna denir.

İşletim Sistemi, bilgisayar donanımı ile bilgisayar kullanıcısı arasında bir arayüz (interface) görevini gören programlar topluluğudur. Bu programlar topluluğunun genel amacı, bilgisayar kullanıcılarına programlarını çalıştırabilecekleri ortamı yaratmak ve bilgisayar sisteminin etkin ve verimli olarak kullanılmasını sağlamaktır.

Bütün bu işletim sistemleri yapısal ve işleyiş açısından bazı farklılıklar gösterirler. İleride işletim sistemleri türleri incelenirken yapısal ve işleyiş farklılıklarına ayrıca değinilecektir. Genel olarak işletim sistemi:

- Kullanıcı ile donanım arasında bir arabirim yada başka bir söyleyişle köprüdür. Kullanıcı ve uygulama programlarıyla, donanım arasındaki iletişimi sağlar.
- Bir koordinatördür. Karmaşık işlemlerin bilgisayar sisteminde önceden belirlenmiş bir sırada yapılmasını sağlar.
- Bir kaynak paylaştırıcısıdır. Merkezi işlem birimi, ana bellek ve giriş-çıkış birimleri gibi bilgisayar kaynaklarının, kullanıcılar arasında paylaşımını sağlar.

- Bir gardiyandır. Sistemin bütün kaynaklarını ve kullanıcıları yakından izleyerek yetkisiz erişimleri önler. Böylece sistemin bilgi giriş ve çıkışını sürekli denetim altında tutarak güvenliğini sağlar.
- Bir saymandır. Bilgisayar sistemindeki kaynakların kullanım oranlarını gelecekte sistem üzerindeki planlarda kullanılmak üzere izler. Böylece, sistemin birimlerinin verimli hizmet verip vermediğinin ve değiştirilmesinin gerekip gerekmediği konulara ışık tutar.
- Bir hizmetçidir. Kullanıcıya, alt düzeydeki donanımın işleyişini hissettirmeden daha arkadaşça bir ortam hazırlarlar. Ayrıca bilgileri ikincil bellekte belli bir düzen içinde saklayarak kullanıcının kolayca erişimini sağlar.

Bu verilen özelliklerin hepsi birden, bir işletim sisteminde bulunmak zorunda değildir. İşletim sistemleri, üzerinde çalışacağı donanıma uygun yazıldıkları için, donanımın özelliklerine göre yukarıdaki fonksiyonlardan bazılarını içermeyebilir. Örneğin, tek kullanıcılı bir bilgisayar sisteminin işletim sistemi, bu özelliklerden sadece birkaçını içerir.

Bilgisayar Sistem Yapısı

Modern ve genel amaçlı bir bilgisayar sistemi, işlemci (CPU) ve belleği paylaşmak için bir omurgaya (common bus) bağlanmış bulunan bir çok ünite kontrol biriminden (device controllers) oluşur.

Bir bilgisayar sistemi açıldığında yani akım verildiğinde (power on) veya "Boot" edildiğinde, bir başlatma programına gereksinim vardır. Bu başlatma programı, sistemin bütün birimlerini başlama pozisyonuna getirir.

Bu başlatma programı işletim sistemine nasıl yükleyeceğini bilmelidir ve işletim sisteminin çalışmasını başlatabilmelidir. Bunu gerçekleştirebilmek amacıyla da İşletim Sisteminin çekirdeğinde (Kernel) yer almalı ve onu belleğe yerleştirmelidir. Sonra işletim sistemi ilk işi (proses) işletmeli ve bazı işlevlerin tamamlanmasını beklemeye başlamalıdır. Beklediği böyle bir işlev (olay) donanım veya yazılımdan kaynaklanacak bir kesinti (interrupt) olabilir.

En basit anlamıyla Kesinti (interrupt), işletim sisteminin o sırada yapmakta olduğu işi bırakıp, kesintiyi yaratan işe (proses) anahtarlanmasıdır. Kesintiler, bir bilgisayar mimarisinin önemli bir parçasını oluşturur. Her bilgisayar tasarımı kendi kesinti mekanizmasına sahiptir. Fakat birkaç fonksiyon geneldir.

İŞLETİM SİSTEMLERİNDE TEMEL KAVRAMLAR

a) Proses (Process)

Bir işletim sisteminde anahtar kavram Proses' dir. Bir proses temel olarak "çalıştırılmakta olan bir program" dır. "Çalıştırılabilir bir program", programın verileri, program sayacı, ve diğer bölümlerinden oluşan bir "veri yapısı" şeklindeki çatıdır.

Proses, yukarıda da belirtildiği gibi, bir "programın işletimi" ne verilen isimdir. Bir "kaynak program" durgun bir komutlar dizisi şeklinde bulunurken, proses bu komutlar dizisinin işletilmesi anındaki durumuna verilen isimdir. Kişisel bilgisayarlarda (PC), genellikle ortam tek kullanıcılı olmasına rağmen, zaman zaman işletim sistemine ilişkin prosesler de işletime alınmaktadır. Ancak yine de bu bilgisayarlarda çalışan işletim sistemlerinin bazılarının (MS-DOS) gibi tek iş düzeni (monoprogramming), bazıları ise kullanıcının kendisine ait farklı programları aynı anda işletime alabilmeleri nedeni ile (Windows işletim sistemi gibi) çok görevli (multitasking) özelliği taşıdığı söylenebilir.

Çok kullanıcılı olan, (multiuser) ve çok iş düzeni (multiprogramming) uygulanan sistemlerde ise, aynı anda birden çok işin işletilmesi zorunluluğu, CPU, bellek ve diğer sistem kaynaklarının bu işler (prosesler) arasında paylaştırılmasını gerektirir. Bu sistemlerde bu nedenle proses işletimi daha karmaşık bir hal alır.

b) Dosyalar (Files)

İşletim Sisteminin temel bir fonksiyonu, disklerin, çevre üniteleri vs. ile ilgili özelliklerini tutmaktır. Dosya (file) yaratmak, okumak veya yazmak için sistem çağrılarına ihtiyaç vardır. Bir dosya okunmadan önce mutlaka açılmalıdır. Dosyalar ile ilgili bilgiler " Dizinler (Directory)" şeklinde bir yapıdır.

Prosesler ve dosyalar hiyerarşik (iç içe dallanmış) bir yapıdadır. Ancak, proselerdeki hiyerarşi, dosyalardaki kadar derin ve kalıcı değildir. Proseslerin hiyerarşik yapıdaki yaşamları en fazla birkaç dakika sürerken dosyaların hiyerarşik durumdaki yapıları yıllarca sürebilir.

c) İş (Job)

Kullanıcıların, bilgisayar sisteminde bağımsız bir bütün olarak ve belli bir sıra dahilinde işlenmesini istedikleri hizmetler kümesine "İş (Job)" denilebilir. Bilgisayarın sistemlerine gönderilen işler, bir veya birden fazla programın ayrı ayrı işletileceği alt adımlardan oluşabilir. İşler genellikle adımların art arda uygulanacağı biçimde düzenlenir. Her adım, bir öncekinin sonuçlanması üzerine işletime girer.

d) İstemci / Sunucu (Client/Server)

Modern İşletim Sistemlerin de genel eğilim, çekirdek (kernel) (DOS' daki Command.com gibi düşünülebilir) en düşük düzeye indirip kullanıcıları etkileyen utility (yardımcı program) leri

zenginleştirmektir. Örneğin, bir dosyadan bir blok bilgi okumak için bir istek talebi olsun. Bu durumda istemci proses' i (client process), dosya sunucusuna (file server) bir istem gönderir. File server işi yapar ve sonucu işlemciye gönderir.

Bu model de Kernel (Çekirdek) istemcilerle sunusular arasında iletişimi sağlar. İşletim sistemini, "file server", "proses server", "memory server" gibi parçalara bölmek yönetimi daha kolaylaştırmıştır. Örneğin bir yazılım hatası (bug) sebebiyle sistemdeki "file server" in çalışmaz duruma gelmesiyle, dosya servisi durur ama sistemin tümü çökmemiş olur.

e) Terminal (Sonda Bulunan)

Modern İşletim Sistemlerinde, istemci konumunda olan ve son uç olarak bulunan sistemlerdir. Fakat bu sistemler, iki türlüdür. Bunlardan birisi şu an kullanmakta olduğumuz şekli ile olandır. Yani, kendi işletim sistemini kullanarak istemci konumunda olanlardır. Diğeri ise, sistemi olmayan yani sadece monitör ve klavyeden oluşan sistemlerdir. Bunlara Dumb Terminal (aptal terminal) denir ve bunlar kendi içinde, özel kartla küçük bir server' a bağlı olarak çalışır ve istemci durumunda bulunur. Örnek olarak bankalardaki memurların kullandığı bilgisayarları gösterebiliriz.

f) Boot (Yeniden Başlatma)

İşletim sisteminin yaptığı işler bitirilip veya kayıtları tutularak yarıda kesilip işletim sisteminin tamamen kapatılması veya elektriğinin kesilip yeniden verilmesi ve işletim sisteminin yeniden başlatılmasıdır.

g) Uyandırma çağrısı

Bir bilgisayar çalışmaya başladığında harekete geçen ilk program, sistem donanımlarının uygun bir şekilde çalışıp çalışmadığını kontrol eden bilgisayarın ROM'unda (Read Only Memory/Sadece Okunabilir Bellek) muhafaza edilen komut setidir. Bu ilk açılıştaki bilgisayarın kendi kendini testi (POST/Power On Self Test) sırasında, işlemci, bellek ve BlOS'ta (Basic Input Output Systems, temel giriş çıkış sistemleri) hatalar olup olmadığı kontrol edilir ve sonuç özel bir bellek alanına kaydedilir. POST işlemi başarılı bir şekilde tamamlandığında, ROM'da yüklü olan yazılım (bazen firmware de denir), bilgisayarın disk sürücülerini etkinleştirmeye başlayacaktır. Çoğu modern bilgisayarlarda, bilgisayar sabitdisk sürücüsünü etkinleştirdiğinde, işletim sisteminin ilk parçasını bulur: Ön yükleyici (bootstrap loader).

Tek bir işlevi olan bu küçük programa ön yükleyici (bootstrap loader) denir: Ön yükleyici bellek içine işletim sistemini yükler ve onun çalışmaya başlamasına izin verir. En temel biçimde ön yükleyici, küçük sürücü programlarını arayüzleriyle birlikte kurar ve bilgisayarın çeşitli alt sistem donanımlarını kontrol eder. İşletim sisteminin tuttuğu belleğin bölümlerini, kullanıcı bilgilerini ve uygulamaları kurar. Bilgisayarın uygulamaları ve alt sistemleri arasında iletişim trafiğini ayarlayan

çok sayıda sinyali, flag'leri ve semaforları tutan veri altyapılarını oluşturur. Ardından da, bilgisayarın kontrolünü işletim sistemine bırakır.

İŞLETİM SİSTEMİNİN BAŞLICA ÖZELLİKLERİ

Bir işletim sistemi bir anda yalnızca bir kullanıcının bilgisayarı kullanmasına izin veriyor ve birden çok kullanıcının kullanmasına izin vermiyorsa, böyle bir işletim sisteminden bir grup çalışanın ortak kullanım sağlaması mümkün olmaz. Buna benzer olarak örneğin bir kullanıcı aynı bilgisayar üzerinde birden fazla programı aynı anda işletime almak istiyorsa, o işletim sisteminde "çok görevlilik" (Multitasking) özelliğinin bulunmasını aramalıdır.

İşte bunun gibi işletim sistemlerinin bir kısmında bulunan bir kısmında bulunmayan çeşitli özellikler, özellikle endüstride bir işletim sisteminden bahsedilirken üzerinde en fazla konuşulan hususları oluşturmaktadır. Bu nedenle endüstrideki günlük konuşma dilinde çok geçen bazı kavramlar incelenecektir.

a) Batch Processing (Yığın İşleme)

İşletim Sistemine, okutulan programlar (Spooling sahası) denilen ve disk üzerindeki özel bir alanı kapsayan bölüme sıra ile ve arka arkaya okutulup biriktirilmeye ve sonra bu saha da derlenmek ve çalıştırılmak için bekleyen programların sıra ile derleme ve çalıştırılma işlemine tabii tutulması yöntemine geçildi. İşte bu yöntem "yığın işlem" in (Batch Processing) temellerini atmış oldu.

Yığın İşlem, bilgisayar sistemlerinin daha verimli kullanılmalarını sağlayarak, iş başına düşen sistem giderlerini azaltmıştır. Ancak bu olumlu yönünün yanı sıra 2 önemli sakıncası vardır. Bunlardan ilki iş yönetiminin durgun ve iş denetim dilinin olanakları ile sınırlanmış olmasıdır. Kullanıcı işletimde oluşan hataları çözümlemek için işin sonuçlanıp sonucun kendisine dönmesini beklemek zorundadır. Yani, işletim kullanıcının tamamen kontrolü dışındadır. İkinci sakınca, çoğu işletim ortamında işler sonuçlanmış olsalar bile çıktıların kullanıcıya ulaşması saatler sürebilmekte, buda verimliliği azaltmaktadır.

b) İnteractive Processing (Etkileşimli İşlem)

Etkileşimli işlem kullanıcılara, işlerini dinamik biçimde yönetme, çalıştırılan programların sonuçlarını doğrudan elde edip, her an müdahale edebilme olanağı sağlayan çalışma türüne ilişkin bir özelliktir. Bu çalışma türünde kullanıcılar, bir işin çalışma süreci boyunca işe, monitör ve klavye vasıtası ile her an müdahale edebilmektedirler. Yani bir başka söylemle, ekran başında oturan bir kullanıcının bilgisayara bir komut vermesi ve o komuta bilgisayardan yanıt alması türünde, bir nevi karşılıklı konuşma yapar gibi çalışma biçimine "Etkileşimli İşlem" denir.

Bu tanımdan da anlaşılacağı gibi, kullanıcılar program geliştirme, metin dosyaları oluşturma, program derleme ve test etme, veri tabanı sorguları işletme, bilgisayar ağı komutları girme, internet servislerini kullanma gibi kısa süreli işlerini Etkileşimli İşlem olarak yürütürler.

c) On Line Processing (Çevrim İçi İşlem)

"On Line" işlem, otomasyon (bankacılık işlemi gibi) uygulamalarında verilen sisteme sunuluş biçimini tanımlayan bir terimdir. Eğer veriler bilgisayar sistemine doğrudan bir biçimde ve işin sahibi tarafından bizatihi aktarılıyorsa yapılan uygulamaya On Line Processing denir.

Bu tür çalışma biçiminde bilgisayar sistemine bağlı uç birimlerde (başka illerdeki banka şubesindeki çalışanlar gibi), mönüler aracılığı ile belirli bir otomasyon projesine yapılması istenilen bir işin gerçekleştirilmesi amacı ile veriler girilir. İşlemde bilgisayar sistemi tarafından anında uygulanır.

Örneğin bankacılık uygulamalarında müşteriler tarafından bankamatik cihazlarından geçekleştirilen para çekme, para gönderme, borsa işlemleri yapma gibi değişik bankacılık işlemlerine ilişkin veriler telefon hatları aracılığı ile doğrudan uygulamanın yürütüldüğü bilgisayar sistemine ulaşıyorsa yürütülen uygulama "On Line" işlemdir.

d) Off Line Processing (Çevrim Dışı İşlem)

Off Line Processing, On Line İşlemlerin bir noktaya kadar uygulanıp daha sonrasında Batch Processing olarak yürütüldüğü uygulamalardır denilebilir. Belli bir mekan içinde bulunan bilgisayar sitemine veriler doğrudan girilmek suretiyle belli bir süre On Line olarak yürütülen otomasyon projesinde, biriktirilen veriler bir süre sonra asıl bilgisayar sistemine topluca aktarılarak işlenirse bu tür uygulamalara Off Line Processing adı verilir.

Örneğin, bir şehirdeki fabrikanın departmanlarındaki terminallerinden bir takım satış, envanter, sipariş gibi veriler fabrikanın merkezindeki bilgisayar sisteminde anında işlenirler. Akşama kadar girilen bu veriler daha sonra, fabrikanın bilgisayar sisteminden bir başka şehirdeki genel müdürlük binasında bulunan asıl bilgisayar sistemine aktarılırsa, yürütülen bu uygulama biçimine Off Line İşlem adı verilmektedir.

e) Serial Processing (Seri İşleme)

Kişisel bilgisayar için kullanılan çoğu tek kullanıcılı (Single User) işletim sistemi, temel olarak Serial Processing yapmaktadır. Bu özellik, kullanıcının, komutları yada çalıştırmak istediği programların isimlerini klavye aracılığı ile arka arkaya girmesi yoluyla gerçekleşir. Kullanıcının yapmak istediği işleri bilgisayar ortamına birbiri ardı sıra aktarması işlemi Seri İşleme olarak anılır.

Örneğin, kişisel bilgisayarlardaki Ms-Dos işletim sisteminin bir kullanıcı kullanırken doğal olarak bir Seri İşlem uygulamaktadır. Çünkü, kullanıcı bir program çalıştırıyorsa bir sonraki yapmak istediği işi ancak bu programın çalışması tamamlanıp bittikten sonra uygulayabilecek,

ondan sonrakini de ikincinin tamamlanıp bitmesinden sonra ancak yine yöneltebilecektir. Böylelikle yapılmak istenen işler kullanıcı açısından birbiri ardı sıra seri olarak bilgisayara yöneltilebildiği için bu tür bir kullanım biçimi seri işleme olarak adlandırılır.

f) Monoprogramming (Tek İş Düzeni)

Monoprogramming yani tek iş düzeni, bir anda, bir işletim sisteminin yalnızca bir programı çalıştırabilmesidir. Bu yöntemde kullanıcı, CPU' yu tümü ile tek başına kullanmaktadır. İşletimde oluşan hatalar, başka bir kullanıcıya yansımayacağı için, korunma önlemleri yalnızca İşletim Sistemi ile kullanıcı arasında ön görülür. Ancak, verimlilik düzeyi düşük bir özelliktir.

Bugüne kadar endüstride yerleşmiş olan PC' ler de örneğin MS-DOS işletim sistemi ortamında bir muhasebe paket programının çalıştırılması gibi uygulamalar bu türdendir

g) Time-Sharing Systems (Zaman Paylaşımı)

İşletim Sisteminde zaman paylaşımı, genel program geliştirme ortamına ek olarak, bilgisayar destekli tasarım ve metin işleme (test Processing) sistemlerinde yaygın olan, Multiprogramming ve Multiuser özelliklerini kapsayan bir yaklaşımdır. Multiuser sistemlerin başlıca özelliklerinden bir tanesi, özellikle Time-Sharing desteği sayesinde de iyi bir yanıt süresi (response-time) göstergesi sağlamasıdır. İşletim sisteminin bu özelliği sayesinde, her kullanıcı, Mainframe sisteme tümü ile yalnız kendisi sahipmiş gibi çalışsa da, aslında time-sharing özelliği sistem kaynaklarını eşit bir şekilde kullanıcılara paylaştırma amacını taşır.

Bu yaklaşımda programlara belli zaman aralıklarında CPU' yu kullanma hakkı verilir. Bu sürenin sonunda da program, (ya da kullanıcı) tekrar CPU kullanma sırasının kendisine gelmesini beklemesi için, bir bekleme kuyruğuna koyulur. Zaman paylaşımlı sistemlerde bellek yönetimi, birlikte çalışan programların birbirlerinden izolasyonunu ve bellek korunmasını iyi bir şekilde sağlar.

h) Multiprogramming (Çok İş Düzeni)

Çok kullanıcılı bilgisayar sisteminde, bir çok farklı kullanıcılara ait işler aynı anda işletime alınabiliyorsa, bu işletim sistemi ortamına "Multiprogramming" yada çok iş düzeni denir. Multiprogramming başlangıçta, CPU' nun boş olarak beklediği süreleri değerlendirmek için tasarlanmıştır. Sistemde çalışan bir kullanıcıya ait herhangi bir iş, bir Giriş/Çıkış (I/O) veya başka bir nedenle beklemeye geçtiğinde, CPU' nun başka bir kullanıcının programını işletmeye tahsis edilmesini (atanması) ve böylece bu pahalı birimden daha fazla yararlanılması amaçlanmıştır

Multiprogramming genellikle ana bilgisayarlarda kullanılan işletim sistemlerinde olabilecek bir özelliktir. Bu özellik Multiuser özelliğinin de olmasını gerektirir. Burada örneğin tek bir CPU bulunan sistem üzerinde çalışan işletim sistemi, bu sisteme aptal (Dumb) terminaller vasıtası ile

erişen kullanıcıların programlarını aynı anda işletime alır ve her kullanıcının programına çok kısa sürelerle CPU' yu kullandırarak bütün kullanıcıların programları aynı anda çalışıyormuş gibi olur.

Bir bilgisayarda belli bir anda CPU ancak bir kullanıcının programını çalıştırır. Yani, sistemde örneğin 25 kullanıcı varsa ve bunların hepsi kendi programlarını çalıştırıyorsa, multiprogramming ortamında bunların hepsi işletime alınır, fakat çalıştırma ile kast edilen CPU' nun o sırada, yani çok kısa bir zaman süresi için (4' er milisaniye gibi) bunlardan yalnızca sırası gelen bir programı işletmesi anlatılmaktadır.

i) Multitasking (Çok Görevlilik)

Multitasking, bir işletim sisteminde bir kullanıcının, birden fazla sayıda prosesini aynı anda işletime alınabilmesi özelliğidir. Yani multitasking, bellekteki birkaç prosesi veriyi aynı anda işlemesi ve işlemci ile I/O ünitelerinin de bunlar arasında aynı anda kullandırılması ortamının yaratılmasıdır. Ancak bir bilgisayar sisteminde, işletim sisteminin kendisine ait birden fazla proses' in aynı anda çalıştırılması, bu sistemde "multitasking" özelliği olduğunu göstermez. Bu nedenle bir işletim sisteminde multitasking özelliği, ancak bir kullanıcının birden fazla sayıdaki kendi prosesi aynı anda işletebiliyorsa vardır.

Bir çok uygulamanın (programın) aynı anda çalıştırılmasıdır. Bunun sağlanması için, görevler (uygulamalar) kısa zaman dilimleri içinde işlemcide çalıştırılır. Bu zaman dilimlerinin oldukça küçük zaman dilimleri olması nedeniyle yapay da olsa bir eş zamanlılık söz konusu olur (İşlemci aynı anda iki işi yapamaz).

Bütün görevlerin toplam bitim süresi bakımından iki sistem arasında fark yoktur. Tek farklılık yukarıda anlatılan örnekte belirtilen avantajdan kaynaklanır. Kısa görevler daha çabuk biter ve kendisinden önce gelen uzun görevleri beklemez. Windows işletim sisteminde birden çok pencere açmak gibi.

j) Multiuser systems (Çok Kullanıcılı Sistemler)

Multiprogrammingi destekleyen işletim sistemleri, genellikle çok sayıda kullanıcının sistemi çeşitli amaçlarla kullanmalarını sağlar ki, bu sistemlere çok kullanıcılı sistemler (Multiuser System) denir. Bu özellik sayesinde her kullanıcı sisteme ayrı bir terminalden ya da bir bilgisayar ağına bağlı kendi bilgisayarından kendisine ait hesabını (userid) şifresi ile birlikte girerek sisteme erişmiş olur. Bu nedenle multiuser sistemler kullanıcı seviyesinde daha yüksek bir güvenlik (security) ve koruma (protection) mekanizmaları sağlamaya ek olarak kullanıcının sistem kaynaklarını kullanma düzeylerini (accounting) saptamaya ve izlemeye yarayan mekanizmalar içerir.

Buradan anlaşılmaktadır ki, bir işletim sisteminin multiuser özelliği varsa, o sistem genellikle multiprogramming de desteklenmektedir.

İşletim Sistemlerinin Temel Görevleri

Tek kullanıcılı ve tekli programlama olanağı sağlayan işletim sistemlerinde, bütün kaynaklar (Merkezi işlem birimi, RAM bellek, disk-disket ve yazıcı gibi) tek bir kişi ve tek bir programca paylaşılmaktadır. Çoklu programlama yapan bütün işletim sistemleri belli fonksiyonları yerine getirmek üzere tasarlanmıştır. Birden fazla program aynı anda çalıştırıldığı ve işlemci herhangi bir anda sadece bir işlemi çalıştırabildiği için, işletim sistemi işlemciyi bu işlemler arasında paylaştırabilmelidir. Ayrıca, bir işlem çalıştırılırken ana bellekte (RAM) olması gerektiğinden, çoklu programlama yapan işletim sistemi, ana belleğin çalıştırılan işlemler arasında paylaştırılmasını sağlamalı ve birbirlerinin bölümlerine erişimlerini engellemelidir. Bulara ek olarak, giriş,çıkış birimlerine erişimi kolaylaştırmalı ve bunları işlemler arasında paylaştırmalıdır. Ayrıca, kullanıcıların çalışmalarını ikincil bellekte (disk, disket vb) saklayabilmelerine olanak veren dosyalama sistemi sağlamalıdır. Bütün bunlardan önce, bu servislerin kullanıcıya sunulmasında kabuk (=shell) yada komut çeviricisi (=command interpreter) adlı programdan yararlanılır. Bu program, kullanıcı ile işletim sistemi arasında bir iletişim aracıdır. Yani, işletim sistemini dış dünyaya açılan kapısıdır. Kabuk kullanıcıdan gelen istekleri alır ve bunları işletim sistemine ileterek yapılmasını sağlar.

Yukarıda özetlendiği gibi, bir işletim sistemi, kullanıcılara bilgisayarda dört çeşit destek sağlamalıdır.

1. İşlem Yönetimi

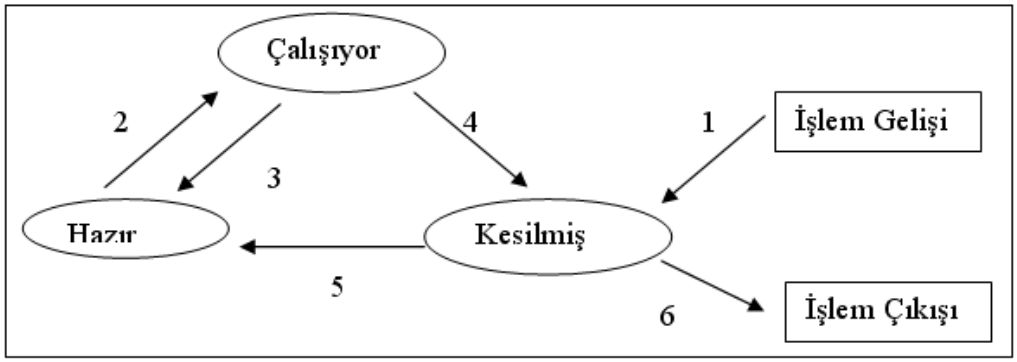
İşlem, bir programın belli bir veriyle çalıştırılmasıdır. Bilgisayar çalışırken, işlemci herhangi bir anda sadece bir işlemi çalıştırabilir. Çoklu programlamada birden fazla işlem olduğu için, çalıştırılan işlem dışındaki işlemler beklerler. İşlemci, her işleme belli bir zaman dilimi ayırır ve devamlı çalıştırdığı işlemi değiştirir. Fakat, bilgisayar çok hızlı çalıştığı için, bu değişimler fark edilemez. Sanki her işlem aynı anda çalışıyormuş gibi görünür.

İşlem yönetimini sağlayan işlem yöneticisi, çalıştırılmayı bekleyen bütün işlemleri bir sıraya sokar ve belli bir anda hangi işlemin çalıştırılması gerektiğine karar verir. İşlemler, herhangi bir anda üç durumdan birinde olabilir. Bunlar:

a) Hazır: İşlem, kullanıcı tarafından gönderilmiştir, ana belleğe yerleştirilmiştir. Ve hazır kuyruğunda çalıştırılmayı beklemektedir.

b) Çalışıyor: İşlemcinin o anda çalıştırdığı işlemin bulunduğu durumdur.

c) Kesilmiş: İşlem, giriş-çıkış birimlerinin veri transferi yapmasını bekliyor durumdadır. Bu transfer biter bitmez, hazır durumuna geçer.



Şekil 1 İşlem Yönetimi

Şekil 1'de bir işlemin durumlarını ve durum değiştirmelerini özetler. Bu şekildeki durum değiştirmeleri şöyle açıklanabilir. (1) İşlem harekete geçirilmiştir ve kesilmiş durumda belleğe yüklenmektedir. (2) İşlemci boşalmıştır. Hazır kuyruğunda sıra, bu işleme gelir ve işlem hazır durumundan çalışıyor durumuna geçer. (3) İşlem kendisine ayrılmış zamanı bitirmiş, sıra başka işleme gelmiştir. İşlem, çalışıyor durumundan hazır durumuna geçer ve hazır kuyruğundaki yerini alır. (4) İşlemin beklediği giriş-çıkış işlemi yüzünden kesilmiştir. (5) İşlemin beklediği giriş-çıkış işlemi tamamlanmıştır. İşlem hazır kuyruğuna geçer. (6) İşlem biter.

Bir işlemin yaşamını şöyledir: kullanıcıdan gelen bir işlem, önce belleğe yüklenir. Daha sonra, diğer işlemlerle beraber işlemciyi paylaşır. Bu arada, herhangi bir giriş çıkış işlemi olursa, bunu bekler ve bittiği zaman işlemciyi paylaşmaya devam eder. İşlem sona erdiği zaman sistemden ayrılır.

2. Bellek Yönetimi

İşletim sisteminin bellek yönetimi ile ilgilenen kısmı olan bellek yöneticisinin görevi, ana belleğin hangi kısımlarının kullanılıp hangilerinin kullanılmadığını izlemek, işlemlere gerektiğinde bellek ayırıp, işleri bittiğinde bunu geri almak, ana bellekte çalışan işlemleri tutmak için yer kalmadığında, ikincil bellek ve ana bellek arasında işlem transferini yapmak ve işlemlerin diğer işlemlere ayrılmış olan bellek alanlarına erişimlerini önlemektir.

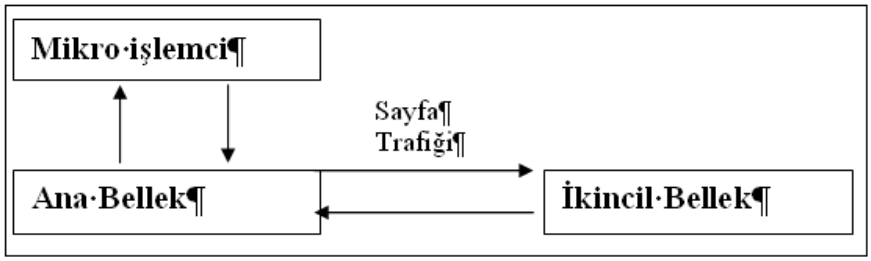
Bellek yöneticilerinin iki türlü çalışma yöntemi vardır. Birisinde işlem, ikincil bellekten ana belleğe transfer edilir ve bitimine kadar ana bellekte kalır. Bu sistemler basittir, genelde tekli programlama sağlar ve yavaş çalışırlar. Diğeri, oldukça karışıktır ve çoklu programlamada üstün performans sağlarlar. Bu tip sistemlerde, işlemler başlangıcından bitimine kadar ana bellekte kalmaz, gerektiğinde ikincil belleğe geri yazılıp, tekrar ana belleğe alınabilirler. Böylece, ikincil bellek ve ana bellek arasında devamlı işlem transferleri yapılır.

Bu yöntemin çeşitli şekilleri vardır. Bir yöntemde, çalıştırılacak işlem ana bellekte olmadığında ve ana bellekte yer kalmadığında, herhangi bir işlem ana bellekten ikincil belleğe geri yazılır ve ana bellekte açılan yere yeni çalıştırılacak işlem alınır. Eğer ana bellekte yer varsa, işlem,

bellek yöneticisi tarafından boş bir kısma yerleştirilir. Diğer yöntemler, çok önemli bir olay olan sanal belleğe olanak tanırlar.

Eğer, işlem ana belleğin kapasitesinden büyükse ne olacaktır ? işte bu sorunun yanıtı, sanal bellektir. Buna göre, bir işlemin tamamının değil, sadece o anda çalıştırılan bölümünün ana belleğe alınması yeterlidir. Böylece, daha küçük kapasiteli bellekte, daha fazla işlem olabilir ve bellek kullanım verimi artar. Bu yöntemde, ikincil bellek (disk, disket vb.) yardımıyla ana belleğin kapasitesi artırılmıştır.

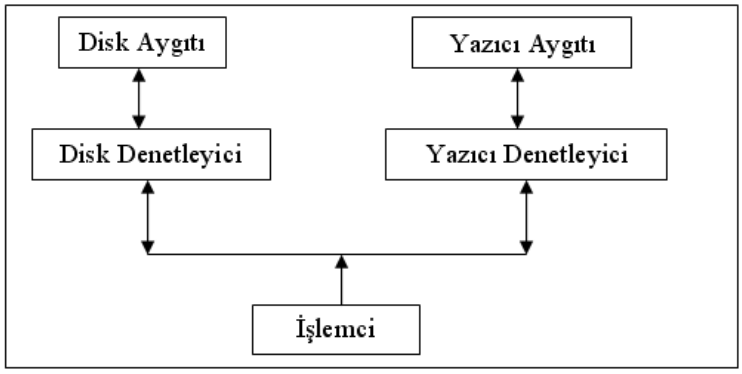
Sayfalama, sanal bellek olayının gerçekleştirildiği bir yöntemdir.bu yöntemde işlem, uzunluğu sabit, sayfa adı verilen kısımlara ayrılmıştır. İşlemcinin gereksinim duyduğu sayfa ana bellekte bulunamazsa, ikincil bellekten yüklenir. Böylece, ana bellekle ikincil bellek arasında bir sayfa trafiği olur. (Şekil 2)



3. Giriş-Çıkış Birimleri Yönetimi

İşletim sistemi programlarının en önemli görevlerinden biri de, giriş-çıkış birimlerinin denetimi ve yönetimidir. Bir bilgisayar sisteminde, bütün giriş,çıkış birimleri donanım açısından değişik özelliklere sahip olduğundan, her biri ayrı yazılıma gereksinim duyarlar. Bu yazılımlara aygıt sürücü (device driver) adı verilir. Sisteme yeni bir giriş-çıkış birimi eklendiği zaman, eğer işletim sisteminde o bileşen (aygıt) için gerekli yazılım yoksa, işletim sistemi o aygıtla iletişim kuramaz. Bu yüzden, birim için gerekli yazılımında sisteme eklenmesi gerekir. Örneğin, bilgisayara yeni bir CD yazıcı eklenmek isteniyorsa, ilgili aygıt için gerekli aygıt sürücü yazılımı da disket yada CD'den bilgisayar kurulmalıdır. Aksi halde aygıt düzgün çalışmayacak yada hiç çalışmayacaktır.

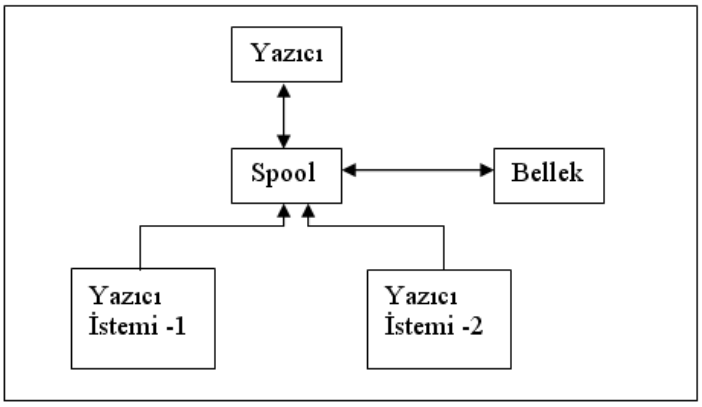
Giriş-çıkış birimleri genel olarak iki kısımdan oluşurlar. Bir mekanik, bir de elektronik kısım. Elektronik kısım denetleyici, mekanik kısım aygıt olarak adlandırılırlar. İşlemci, denetleyiciyle haberleşir, denetleyici de aygıta bilgi alır yada verir. (Şekil 3)



Şekil 3

Giriş-çıkış birimleri, kullanılışları bakımından ikiye ayrılırlar. İlki paylaşılabilir, diğeri ise paylaşılamayan birimlerdir. İlkine diski, ikinciye yazıcıyı örnek verebiliriz.

Paylaşılabilir birimler, aynı anda birden çok işleme hizmet verebilirler. Paylaşılamayan birimler ise, bir anda bir işlem tarafından kullanılabilirler. Eğer paylaşılamayan bir giriş-çıkış birimi olan yazıcı, iki işleme birden aynı anda yanıt verseydi, bir satır birinci işlemin istediğini, bir satır ikinci işlemin istediğini yazar ve yazılan şeyler hiçbir işe yaramazdı. Fakat, eğer sadece bir işleme hizmet verirse, diğeri yazıcıyı beklemek zorunda kalacak, bu da sistem performansını düşürecektir. Bu durumda işletim sistemi, ikinci işlemi bekletmemek için, ikincil belleğin yada ana belleğin bir kısmını onun yazıcıya göndermek istediği bilgileri saklamak için kullanır. Yazıcının işi bittiği zaman da, bu bilgileri, buradan yazıcıya yazması için gönderir. İşte işletim sistemi, paylaşılamaz konumdaki giriş-çıkış birimlerini, bu yolla paylaştırır. Buna SPOOL (simultaneous Peripheral Operation On Line = Eşzamanlı Bağlantı Çevre Birimleri İşlemi) (Kuyruk) adı verilir.

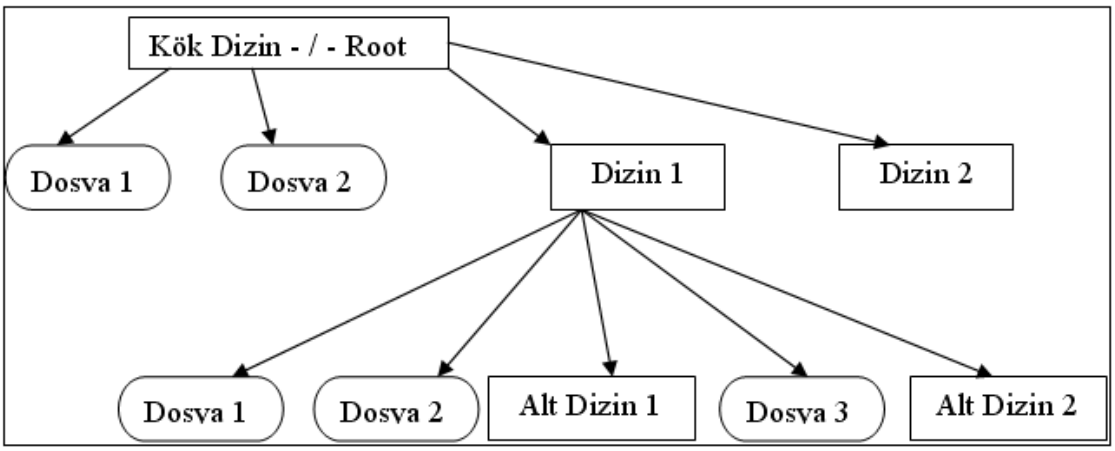


4. Kütük (Dosya ve Klasör) Yönetimi

İşletim sisteminin bu bölümü, kullanıcıya en yakın bölümüdür. Kütük (dosya), bilgilerin içinde saklandığı birimlerdir. Her işletim sistemi, kütüklere erişimi ve kullanımını sağlar. Kullanıcı, bu kütüklere erişebilmek için, ikincil bellekteki fiziksel yerini iz ve sektör olarak bilmek zorunda değildir, sadece kütüğün adını bilmesi yeterlidir. İşletim sistemi, bu adı alarak, bu kütüğün ikincil bellekteki fiziksel yerini bulur ve kullanıcıya sunar.

Kütükleri takip edebilmek için, işletim sistemi dizin adı verilen listeler kullanır. Dizinde, her kütük için bir bilgi bloğu tutulur. Bu bilgi bloğunda, kütüğün ismi, tipi, büyüklüğü yanında ikincil bellekte nerede bulunduğu gibi, sadece işletim sisteminin kullanacağı bilgiler de bulunur. Her dizinin ayrıca alt dizinleri de olabilir. Böylelikle, ikincil bellekteki kütük sistemi kullanıcıya şekil 6'da ki gibi görünür. Şekilde, dikdörtgenler dizinleri, elipsler kütükleri simgelemektedirler.

Her işletim sisteminin kendine özgü kütük yapısı vardır. Ayrıca her işletim sistemi kütük ve dizinleri kendine özgü yöntemlerle ikincil belleklerde saklar. Örneğin DOS işletim sistemi, FAT 16 sistemini kullanırken, Windows 98 FAT 32 ve Win NT ise NTFS sistemini kullanmaktadır.



Şekil 5 Dosya ve Klasör Yönetimi

Java nedir, nasıl çalışır?

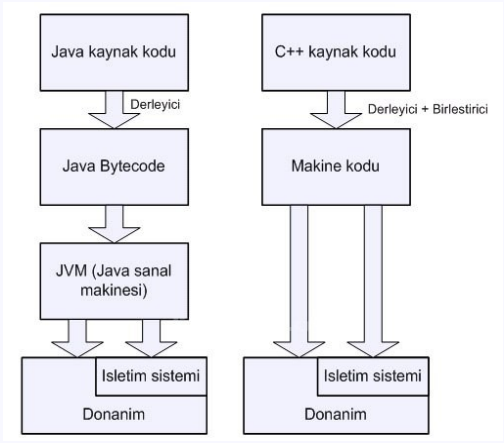
Java Sun microsystems mühendislerinden James Gosling tarafından geliştirilmeye başlanmış gerçek nesneye yönelik, platform bağımsız, yüksek performanslı, çok işlevli, yüksek seviye, interpreted(adim adim isletilen) bir dildir.

Java Temelleri

Java ilk çıktığında daha çok küçük cihazlarda kullanılmak için tasarlanmış ortak bir platform dili olarak düşünülmüştür. Ancak sonradan bakılmıştırkı platform bağımsızlığı özelliği C ve C++'tan çok daha ustun ve güvenli bir yazılım geliştirme ve isletme ortamı sunuyor, hemen her yerde kullanılmaya başlanmıştır. Su anda özellikle kurumsal alanda ve mobil cihazlarda son derece popüler olan Java, özellikle J2SE ile masaüstünde de gücünü arttırmayı hedefliyor. Java'nın ilk surumu olan Java 1.0 (1995) Java Platform 1 olarak adlandırıldı ve tasarlama amacına uygun olarak küçük boyutlu ve kısıtlı özelliklere sahipti. Daha sonra platformun gücü gözlendi ve tasarımında büyük değişiklikler ve eklemeler yapıldı. Bu büyük değişikliklerden dolayı geliştirilen yeni platforma Java Platform 2 adi verildi ama versiyon numarası 2 yapılmadı, 1.2 olarak devam etti

Bir java yazilimi su sekilde gelistirilir:

1. Programcı java kodunu yazar.
2. Bu kod bir java derleyicisi ile derlenir. Sonuçta bytecode adı verilen bir tür makine kodu ortaya çıkar. Platform bağımsızlıgını sağlayan şey bytecode'dir. Çünkü bir kere bytecode oluştuktan sonra yazılım tüm işletim sistemlerinde çalışabilir.
3. Bu bytecode Java virtual Machine (Java Sanal Makinesi) tarafından adım adım işletilir. Asağıda java ve C++ kodunun geçirdigi aşamalar gösterilmistir.



Java kodunun yazılması.

Java nesneye yönelik bir dil olduğundan tüm yazılım sınıflar ve nesneler üzerinden yürütülür. Sınıflar uygulamadaki nesnelerin tanımlandığı kod parçalarıdır. Java'da her bir sınıf bir dosya içerisinde yer alır. Dosyaların uzantıları .Java seklindedir. Dosya adi ise içinde tanımlanan sınıf ile aynidir. Örneğin, BenimSinif.Java gibi.

Derleme

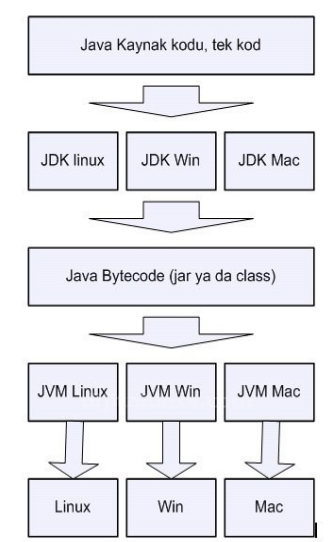
Derleyici kısaca herhangi bir editör ile yazılan Java kaynak kodlarını (yani .Java uzantılı sınıfların yer aldığı dosyaları) Java sanal makinesinin çalıştırabileceği bir tür makine dili (assembler) olan Bytecode'a dönüştürür. Bu dönüştürülen bytekod ise (.class) dosyaları içerisinde saklanır. Java kodunu derlemek için bir Java derleyicisine ve Java kütüphanelerine ihtiyacınız var. Su anda iki derleyici yaygın olarak kullanılmaktadır. Bir tanesi Sun'in SDK'si ile birlikte gelen javac. diğeri ise IBM'in acık kodlu derleyicisi jikes. Jikes, çok hızlı derlemesi ile ünlü olsa da en son Java yazılımlarını derlemek için javac'ye ihtiyaç duyulabilir. Ayrıca linux altında derleme yapmak için gnu lisansı ile geliştirilen acık kodlu Gnuj kullanılabilir. Derleyici ve kütüphanelerin bulunduğu (Java API) uygulamaya "J2SE SDK" adi verilmiştir.

Çalıştırma ve Java sanal makinesi.

Sanal makine donanımdan bağımsız yazılım geliştirme ihtiyacına cevap verme amacıyla geliştirilen bir teknolojidir. Java'nın temel felsefesi olan "bir kere yaz, her yerde çalıştır" sanal makine sayesinde var olmuştur. Sanal makineyi bir yönden bir tür hayali bir mikroişlemci gibi düşünebiliriz. gerçek tüm mikroişlemciler (Intel Pentium, AMD Athlon, Sun Sparc vs) belirli bir grup komutu işlemek üzere tasarlanmıştır. Bu komutlara işlemcinin komut kümesi adı verilir. Örneğin x86 komut kümesi gibi. tüm yazılımlar çalışabilmek için önce bu komut kümesine dönüştürülür, daha sonra işlemci bu komutları sıra ile gerçek işlemci komutlarına dönüştürüp isletir. Java Sanal makinesi de Bytekod komut kümesini tıpkı işlemci gibi adım adım işletir. Java'nın interpreted bir dil olarak adlandırılmasının nedeni budur. Bytekod ilkel işlemlerin yanında (ilkel işlemler, mikroişlemci seviyesi komutlardır, aritmetik işlemler, bit işlemleri, bellek ve yığın işlemleri vs.) sanal makinenin üzerinde çalıştığı işletim sistemine yönelik işlemler de barındırır. Bu sayede Java Virtual Machine yazıcı, seri porte, grafik, dosya servisi, ağ bağlantısı gibi yazılım ve donanım servislerine erişim yapabilir.

Java'nın doğrudan bytecode çalıştırması performansının düşük olabileceği izlenimini verebilir. Ancak, JVM tasarımı gecen 10 yılda çok değişmiş ve geliştirilmiştir. Su anda Java'nın performansı

çoğu alanda C++'in performansına yakin bir seviyededir ve işlemci hızı ve bellek miktarının her geçen yıl katlanarak artması ile performans konusu çoğu uygulamada artık ikinci planda kalmıştır. Aşağıda çeşitli platformlar için Java'nın çalışması basit blok şema olarak gösterilmiştir. Eğer söz konusu dil C++ olsaydı üç ayrı kod yada kodda üç ayrı #ifdef vs tanımı, üç ayrı obje dosyası oluşturulması gerekirdi ve çoğu uygulamada C++ projesi sadece tek platforma destek verirdi (win32 gibi)



Java ile ilgili yazılarda karsılaşabileceğiniz bazı kavramlar:

Hot Spot teknolojisi:

Java sanal makinesi HotSpot adi verilen özel bir teknolojiyi içinde barındırır. HotSpot yani sıcak nokta , bir yazılımda sürekli olarak tekrarlanan ve üzerinden geçilen kod bölümlerine verilen bir isimdir. Java sanal makinesi bir kod çalışmaya başladıktan sonra sıklıkla kullanılan kod bloklarını gözler ve bir süre sonra bu bytecode bloklarının çalışılan sistemdeki gerçek işlemci komut karşılıklarını bir tür cep belleğe yazar ve zaman ilerledikçe artik byte kod üzerinden değil doğrudan sistemin öz komutlarını kullanarak yazılımın o bölümlerini işletmeye baslar. Bu şekilde ciddi performans avantajı sağlanmıştır.

JIT

Java ilk çıktığında bytecode isletme hızı çok iyi değildi. yerine göre sistemin öz yazılımlarından 5-10 kat yavaş çalışıyordu. Bu nedenle bazı yazılım geliştirme şirketleri JIT yani Just-in-time compile, "anında derleme" araçları üretmeye başladılar. Yapılan şey byte kodu sanal makinenin kurulu olduğu gerçek sistemin diline anında derleme yaparak dönüştürmesiydi. Bu sayede performansta ciddi artışlar sağlandı. Ama 2000 yılından sonra HotSpot teknolojisinin gelişmesi ile JIT'in işlevi VM'icinde yer almaya başlamış, işlemci hızı ve bellek miktarının dramatik bicimde artması ile dış JIT yazılımları popülerliğini kaybetmiştir. Bugün halen bir kaç urun pazarda bulunsa da genellikle bu yöndeki ihtiyaç yok olmuş gibi gözükmektedir.

Java API

Java API, java yazılımlarında kullanılan yazılım kütüphanelerine genel olarak verilen isimdir. Java API ile disk, grafik, ağ, veri tabanı, güvenlik gibi yüzlerce konuda kullanıcılara erişim imkanı sunulur. Java API J2SDK'nin bir parçasıdır.

Çöp Toplayıcı (Garbage Collector)

Çöp toplayıcı Java'nın en belirgin özelliklerinden birisidir. C++, C gibi dillerin en büyük handikaplarından birisi dinamik bellek yönetimidir. Yazılımda işaretçi (pointer) kullanarak dinamik olarak bellek ayırdıktan sonra o bellek ile işiniz bittiğinde mutlaka ayrılan belleği bellek yöneticiye özel metodlar yardımıyla (delete, destructor vs.) iade etmeniz gerekir. Yoksa bellek sızıntısı (Memory Leak) oluşur ve bu bir süre sonra yazılımın ve işletim sisteminin beklenenden farklı davranmasına yol açar. Bundan dolayı, tüm büyük C ve C++ yazılımları az da olsa bellek sızıntısı içerir (işletim sistemleri dahil). Sızıntıların tespiti oldukça güçtür ve bulunması zor hatalara yol açar. Çöp toplayıcı sayesinde Java'da bir nesne oluşturulduktan sonra o nesne ile isiniz bittiğinde hiç bir şey yapmanız gerekmez. Sanal makine akıllı bir biçimde kullanılmayan bellek bölümlerini belirli aralıklarla ya da adaptif metotlarla sisteme otomatik olarak temizler ve sisteme iade eder. Bu isleme Çöp toplama, ya da garbage collection adı verilir. Çöp toplama sistemlerinin yapısı oldukça karmaşıktır ve geçen yıllar içinde büyük gelişmeler kaydedilmiştir. Çöp toplayıcının varlığı java'da bellek sızıntısı olmayacağı anlamına gelmez, ama bellek sızıntıları daha ender olarak ve farklı sekilerde karşınıza çıkar ve genellikle tedavi edilmesi daha kolaydır.

Jar:

Jar, aslında bir tür sıkıştırma formatıdır. Jar ile derlenen Java kodları ile oluşan yazılımın paketlenip taşınması kolay bir hale getirilir. Jar dosyaları temelde bytekod blokları içerir. Jar dosyaları genellikle kütüphane oluşturmada ya da uygun bicimde hazırlanırsa işletim sisteminden doğrudan çalıştırılabilecek bir şekilde kullanılabilir. (Executable jar, isletilebilir jar)

AWT ve Swing

AWT, ilk Java ile birlikte geliştirilen temel grafik arayüz oluşturma kütüphanesine verilen isimdir. Ancak Java 2 platformu ile birlikte AWT'nin yetersiz görülmüş ve çok daha geniş ve gelişmiş özelliklere sahip Swing kütüphanesi sisteme eklenmiştir. Özellikle çok platform destekleyen yazılımlarda kullanıcı arayüzü geliştirme aracı olarak Swing halen önemini korumaktadır.

SWT

SWT Swing'e bir alternatif olarak IBM tarafından geliştirilen bir gösterim sistemidir. Swing'den en büyük farkı çalıştığı işletim sisteminin grafik kütüphanesi ve komutlarını kullanmasıdır. Bu nedenle SWT uygulamaları Swing'e göre çoğu yerde daha hızlı ve işletim sistemindeki diğer uygulamaları andıran bir şekilde çalışmasını sağlar. Ancak yapı itibari ile SWT kullanımı Swing kadar efektif olamayabiliyor (özellikle olay mekanizması, tablo ve ağaç yapılarındaki yavaşlığı, ayrıca linux performansı ile SWT eleştirilmiştir.) Swingin Java 1.5 ile performans açığını kapatacağı iddia edilse de SWT'nin de artik Java camiasında kabul görmüş bir sistem olduğu aşikardır. SWT'nin dezavantajı ise Java'nın bir parçası olmamasıdır. Yani SWT uygulamaları SWT kütüphanesi ile birlikte dağıtılmaktadır. En bilinen SWT uygulaması ünlü Java yazılım geliştirme aracı Eclipse'tir. Bununla birlikte son yıllarda Swing ile profesyonel derecede arayüze sahip masaüstü yazılımları da ortaya çıkmıştır

Applet

Applet, uzaktaki sistem üzerinden indirilip internet tarayıcı üzerinde çalıştırılabilen Java uygulamalarina verilen isimdir. Java'nın son kullanıcılar tarafından tanınması applet sayesinde olmuştur dersek yanlış olmaz herhalde. Applet'ler sisteme zarar veremeyecek bir şekilde tasarlanmıştır ve bugün özellikle oyun sitelerinde halen yaygın olarak kullanılmaktadır. İçerisinde

applet olan bir sayfayı açmaya çalıştığınızda tarayıcınız otomatik olarak Java sanal makinesini çalıştırıp ekranın applet'e ayrılan bölümünde uygulamanın çalışmasını sağlar.

Java'nın Özellikleri

Java Basittir.

Java, kendisine yakın güçteki dillerin en basitidir. Örneğin, Java model olarak aldığı C++'tan çok daha kolaydır. C++'ın çok güçlü bir dil olduğu tartışılmaz. Ama **C++** ve **C#** çoğu bir dilde bulunması çok da şart olmayan karmaşık bir çok özellik içermektededir. Bütün bu özelliklerin öğrenilmesi çok zaman alır. Oysa Java gereksiz çok fazla özellik içermez. Ve Java'da bir özelliğin öğrenilmesi ve kullanılması çok kolaydır. Başka dillerde çok zor yazılan işlemler Java'da çok basit ifadelerle gerçekleştirilebilmektedir.

Java Nesneye Yöneliktir.

Java tamamen (bazı kişilere göre %99) **nesneye yönelik**'tir. Diğer bazı diller gibi nesneye yöneliklik sonradan dile eklenmemiştir, başından beri Java'da bulunmaktadır. Bir Java programında olabilecek her şey ya nesnedir ya da bir nesnenin parçasıdır. Java nesneye yönelik programlamayı sadece mümkün değil aynı zamanda kolay kılmıştır. Java'da bir nesnenin yapılması, kullanılması, geliştirilmesi, başka bir yere aktarılması çok kolaydır.

Java Dağıtıktır.

Java '**dağıtık**' bir dildir. 'Dağıtık' birden fazla bilgisayarda çalışan programların bir biriyle uyumlu çalışabilmesidir. Bir yazılım parçasının bir kısmının bir makinede diğerinin başka makinede aynı anda çalışması mümkündür. Bu yüzden Internet'in dilinin Java olduğu söylenmektedir. Günümüzde tek bir makine üzerinde çalışan, başka hiç bir yerle bağlantısı olmayan uygulama kalmamış gibidir. Bir çok işletme Internet üzerinden iş yapmaktadır. Birçok işletmede birden fazla makine birbirine bağlı olarak çalışmaktadır. Bu yüzden Java dağıtık programlama için en uygun çözümlerden biridir.

Java Sağlamdır.

Java sağlam bir dildir. Programlamadaki hataların çoğu daha yazılma aşamasında anlaşılabilmektedir. Yazılma aşamasında anlaşılmayanlar, programın çalışması esnasında yakalanabilmektedir. Bir '**exception**'la, programda hatanın ne olduğu, nerede olduğu ve hangi işlemi yaparken olduğu bile belirtilebilmektedir. Başka dillerin aksine çalışma esnasında bir Java programı "Bir hata oldu!" deyip çökmez. Bir çok durumda hataya rağmen program çökmeden çalışmaya devam eder. Hata olması durumunda da programı çalıştıranlar hatayı ayrıntılarıyla öğrenme olanağına sahip olur.

Java Güvenlidir.

Java güvenli bir dildir. Java diliyle virüs yapılamaz. Hiç bir virüs bir Java programına bulaşamaz. Bir Java programının yaptığı her hareket takip edilir. Kötü niyetli bir program, bir işlemi eğer izin verilmediyse yapamaz. Bu özellik, Internet gibi herkesin başkasının makinesine erişebildiği bir ortamda çok önemlidir. Java'yı **güvenlik** gereksinimi yüksek bir çok firma bu yüzden tercih etmektedir.

Mimarilere Yansızdır.

Sıradan kullanıcıların bildiğinin aksine dünya üzerinde bir çok işletim sistemi vardır. Hatta kullanıcın çoğunun iyi bildiği **Windows** işletim sistemi bazı bilgisayar alanlarında hiç kullanılmaz. **UNIX/Linux**, **Apple Mac**, IBM'in çeşitli işletim sitemleri dünya işletim sistemi pazarında büyük yer tutmaktadır. Birçok masaüstü kullanıcısının tek bildiği işletim sisteminin, bazı sektörlerde adı dahi geçmemektedir. Java'da yazılan bir program hemen hemen bütün

işletim sistemlerinde hiç değiştirmeye gerek duymaksızın çalışır. Diğer dillerde bu özellik yoktur. Hatta Windows'un bir versiyonunda çalışan program diğer bir versiyonda çalışmayabilmekte, bazen makinenin çökmesi gibi büyük sonuçlar bile doğurabilmektedir. Windows'u üreten Microsoft firmasının yazdığı programlar bile kendilerinin işletim sistemlerinin bazılarında çalışmamaktadır. Kendilerinin yazdığı belgelerde hangi işletim sistemlerinin hangi versiyonlarında hangi programlarının çalışmadığını belirtilmektedir. Bu durum bütün işletim sistemi ve bütün programlar için geçerlidir. Ürettiği yazılımların her platformda çalışmasını isteyen firmalar giderek daha çok Java'ya yönelmektedir.

Java Taşınabilirdir.

Java programları her ortamda aynı veya benzer bir şekilde çalışır. Her ortam/makine için ayrı bir program yazmaya gerek kalmaz. Programcı programın çalışacağı makinenin durumunu göz önüne almak zorunda kalmaz. Bir programın görüntüsü çalıştığı hey yerde hemen hemen aynıdır. C programları da hemen hemen her işletim sisteminde yeniden derlenmek suretiyle çalışabilir. Ama programcı bunun için hazırlık yapmalıdır. Bir işletim siteminde tamsayı -2^15 ile 2^15 arasında değer alırken diğerinde -2^31 ile +2^31 arasında alabilir. Ama Java bu farkları programcıya şeffaf kılar. Programcı her işletim sistemi ve her sürüm için ayrı bir program yazmak zorunda kalmaz.

Java Yorumlanır.

Java '**yorumlamalı**' bir dildir. Yani bir Java programının komutları, çalışırken makinenin anlayacağı formata çevrilir. Java'da bu **Java Virtual Machine** (**JVM**) tarafından yapılır. Bunun avantajı bir programın kullandığı standart kütüphanelerin programla birlikte taşınması zorunluluğunu ortadan kaldırmasındır. Bir yorumlayıcı herhangi bir ortamda varsa, bir dildeki standart her özellik o ortamda var demektir. Programla birlikte bu kütüphanelerin de taşınması gerekmez. Bu da bir Java programının bir makineden başka makineye indirilmesini çok hızlandırır. Çünkü sadece programcının yazdığı nesneler yolculuk yapar. Applet'lerin çalışma prensibi budur. Ana makinede bulunan applet, tarayıcı tarafından kullanıcının makinesine alınır ve çalıştırılır. Java ortamı, yani JVM tarayıcının içinde mevcutur.

Java Yüksek Başarımlıdır.

Diğer dillerde olmayan bir çok özelliğe sahip olmasına rağmen, Java'da bunun için fazla bir performans kaybı yoktur. Java'nın ilk versiyonlarında çalışan programlar diğer dillerde yazılan eşdeğerlerine göre elbette yavaştır. Ancak Java'nın gelişmesiyle birlikte Java bu farkı, üstün özelliklerinde vazgeçmek zorunda kalmaksızın kapatmaktadır. En son Java sürümüyle birlikte **JIT** (Just-In Time-Tam Zamanında Derleme) teknolojisi devreye girmiş bulunmaktadır. Bununla birlikte Java programları, diğer dillerde en iyi yazılmış programların hızını hemen hemen yakalayabilmektedir.

Java Çok Kanallıdır.

Java dili başından 'çok kanallıdır' (**multi-threaded**'dir). Çok kanallılık, bir programın aynı anda birden fazla işlemi yürütebilmesi demektir. Bir program herhangi bir şeyi beklerken arada başka bir işlemi gerçekleştirebilir. Beklenen olay gerçekleşince ilk işlem kaldığı yerden aynen devam ettirilir. Üstelik bunun için programcının fazla bir şey yapması da gerekmemektedir. Çok kanallılık bir çok dilde hiç yoktur. C++ gibi dillerde de dilin özelliği değil, ona sonradan eklenmiş kütüphanelerle kullanılabilmektedir. Ama Java'nın kendisi doğuştan çok kanallıdır. Bu yüzden çok kanallı program yapmak için en kolay dil Java'dır.

Java Dinamiktir.

Java'da bir programda kullanılan birimlerin (kütüphaneler,modüller veya sınıfların) birbirine bağlanması çalıştırma anında yapılır. Buna 'sonradan bağlama' (**late binding**) denir. Kullanılan birimlerin iç yapısı değiştirildiğinde, bu birimleri kullanan programın değişmesi gerekmez. Yeter ki birimlerin dışarıdan çağırma şekilleri değişmesin. Oysa C++ gibi dillerde herhangi bir değişiklikte (bu bir modülün iç yapısında çağıranları ilgilendirmeyen çok ufak bir değişiklik bile olsa) herşeyin yeniden işleme tabi tutulması gerekir. Buna da 'erken bağlama' (early binding) denir. Java'da late-binding bu özellik olmasaydı, Java kütüphanlerindeki her hata düzeltme ve değişiklikte dünyadaki bütün programları yeniden işleme tabi tutmamız gerekirdi ki bu mümkün değildir.

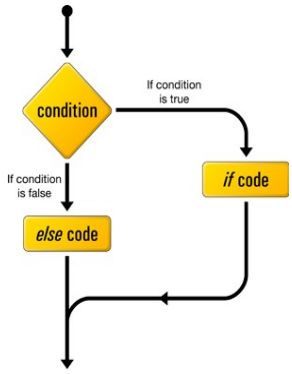
Kontrol Mekanizmaları

If-then-else Yapısı

Java'da bulunan if-then-else komutu, basit bir şekilde çalışır. if komutu içerisinde bulunan expression (ifade) true ise if altındaki blokda bulunan komutlar, eğer false ise else komutu altındaki komutlar çalıştırılır.

```
if(ifade)
{
   // ifade == true ise çalıştırılacak komutlar
}
else
{
   // ifade == false ise çalıştırılacak komutlar
}
```

if then else kontrol mekanizmasının en basit şekilde çalışma prensibi yukarıdaki gibidir. Aşağıda da Java'da bulunan if-then-else yapısının çalışma mekanizmasını gösteren akış diyagramı (flowchart) gösteriliyor.



Condition:Şartı Kontrol İf Code: İf Kodu Else Code: Else Kodu

Şimdi Java'daki if-then-else yapısını kullanarak, basit bir örnek geliştirelim. Örnek, okullarda kullanılan not sistemini hesaplayan basit bir uygulama. Geçme notunun 45 olduğu varsayılarak öğrenciye o dersten geçip kaldığını belirten bir mesaj gösterilir. Eğer öğrenci 45 ve üzeri aldı ise dersten geçildiğini, 45 den az bir not alındı ise o dersten kalındığını ekranda gösterir.

```
package org.javablog;

import java.util.Scanner;

public class Test
{
   private static final int GECME_NOTU = 45;
   public static void main(String[] args)
   {
      Scanner notAl = new Scanner(System.in);
      System.out.printf("%s :", "Notunuzu girin");
      int not = notAl.nextInt();
      if(not < GECME_NOTU)
      {
         System.out.printf("%s\n", "Bu dersten kaldınız. Daha çok çalışmanız gerekiyor");
      }
     else
      {
        System.out.printf("%s\n", "Bu dersten geçtiniz. Tebrikler");
      }
   }
}
```

if-then-else yapısını kullanarak iç içe if-else yapıları istenildiği kadar tanımlanabilir. Örneğin else komutundan sonra başka bir if-else yapısı kurgulanarak daha derin bir analiz yapılabilir.

```
if(ifade)
{
   //ifade == true komutları
}
else
{
   // false ise çalıştırlacak kodlar
   if(baskaifade)
   {
      //baskabirifade == true
   }
   else
   {
      //baskabirifade == false ise çalıştırlacak kodlar
      // gibi birçok if-else tanımlanabiliyor.
   }
}
```

Ancak Java, bu tür içiçe if-else yapılarının kodun okunmasını zorlaştırdığını ve hataya açık olduğunu bilen kişiler tarafından tasarlandığı için, daha basit bir yapı sunar. if else-if komutu.

Yukarıdaki kod bu yöntem sayesinde şu şekilde daha basit bir şekle indirgenebilir.

```
if (ifade)
{
   //ifade== true ise yapılcak kodlar
}
else if (ifade)
```

```
{
   //ifade== true ise yapılcak kodlar
}
else if (ifade)
{
   //ifade== true ise yapılcak kodlar
}
else
{
   //ifade== false ise yapılcak kodlar
}
```

Yukarıdaki kod ile, ifadeden dönen değerin if bloklarında true olup olmamasını kontrol ederek uygun if bloklarının altındaki hangi kod parçalarının çalıştırılacağı belirlenir.

Şimdi yukarıdaki if-then-else yapısındaki örneği daha da geliştirip, geçen öğrencilerin aldığı harf notunu da ekrana bastırıp daha detaylı bilgi verelim. Eğer öğrenci 45 üzeri not aldı ise dersten geçiyor ve 45 60 arası DD, 60 70 arası ise CC, 70 85 arası BB ve 85 100 arası AA notunu alıyor.

```
package org.javablog;

import java.util.Scanner;

public class Test
{
   private static final int GECME_NOTU = 45;
   public static void main(String[] args)
   {
      Scanner notAl = new Scanner(System.in);
      System.out.printf("%s :", "Notunuzu girin");
      int not = notAl.nextInt();
      if(not < GECME_NOTU)
      {
         System.out.printf("%s\n", "Bu dersten kaldınız. Daha çok çalışmanız gerekiyor");
      }
      else
      {
         System.out.printf("%s\n", "Bu dersten geçtiniz. Tebrikler");
         if(not <= 60)
         {
            System.out.printf("%s\n", "Notunuz DD");
         }
         else if(not <= 70)
         {
            System.out.printf("%s\n", "Notunuz CC");
         }
         else if(not <= 85)
         {
            System.out.printf("%s\n", "Notunuz BB");
         }
         else
         {
            System.out.printf("%s\n", "Notunuz AA");}
         }
```

```
} }
```

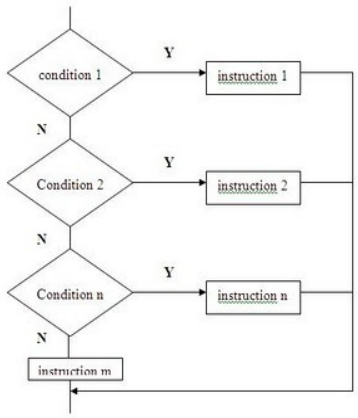
Switch Yapısı

Java kontrol mekanizmalarında switch adında başka bir yapı daha sunar. Bu komut, uzun if-else if-else gibi yapılarda ortaya çıkan karmaşıklığı daha iyi yönetmek için tasarlanmış başka bir mekanizmadır. switch yapısında bilinmesi gereken en önemli nokta, switch yalnızca byte, short, int ve char gibi primitive tipler ile enumerated tipler ile kullanılabilir.

```
switch(sayısalDeger)
{
   case 1 : //komutlar
   break;
   case 2 : //komutlar
   break;
   //..
   default : //komutlar
   break; // bu break şart değil
}
```

Basitçe switch, sadece sayısal değerleri kabul eden bir kontrol mekanizmasıdır. switch içerisine "string" şeklinde herhangi bir string değerini verip bunun üzerinde işlem yapamazsınız. Aynı şekilde boolean değer üreten bir expression verip yine buna uyan durumları switch ile kontrol edemezsiniz. switch yalnızca sayısal (int gibi) değerler kabul eder. Birde buna ek olarak, sayısal değerleri temsil eden primitive tiplerin özel sınıfları olan Integer, Character, Byte, Short gibi sınıflar ile de işlem yapılabiliyor.

Aşağıdaki şema switch kontrol ifadesinin çalışma prensibini gösteren akış diyagramıdır.



Condition:Şart İnstruction:Komut

switch kullanırken dikkat edilmesi gereken diğer nokta, her case ibaresinden sonra "break" komutunun kullanılması. Çünkü, switch ifadesine verilen değer herhangi bir case ile eşleştiğinde o durumdaki kodlar işlendikten sonra, switch komutunun sonlandırılması gerekmektedir. Bundan sonraki program kodu switch yapısının bittiği blokdan sonra devam eder. "default" ifadesinden sonra gelen break şart değildir.

Şimdi ayları gösteren basit bir program geliştirelim. Program, verilen sayısal değerin hangi ay ismine karşılık geldiğini göstererek ekrana bu bilgiyi basar. Örnek olarak 9 Eylül ayını gösterir.

```
package org.javablog;
```

```
import java.util.Scanner;

public class Test
{
   public static void main(String[] args)
   {
      Scanner ayAl = new Scanner(System.in);
      System.out.println("Ay numarasını girin :");
      int ay = ayAl.nextInt();

      switch(ay)
      {
         case 1: System.out.printf("%s\n", "Ocak");break;
         case 2: System.out.printf("%s\n", "Şubat");break;
         case 3: System.out.printf("%s\n", "Mart");break;
         case 4: System.out.printf("%s\n", "Nisan");break;
         case 5: System.out.printf("%s\n", "Mayıs");break;
         case 6: System.out.printf("%s\n", "Haziran");break;
         case 7: System.out.printf("%s\n", "Temmuz");break;
         case 8: System.out.printf("%s\n", "Ağustos");break;
         case 9: System.out.printf("%s\n", "Eylül");break;
         case 10: System.out.printf("%s\n", "Ekim");break;
         case 11: System.out.printf("%s\n", "Kasım");break;
         case 12: System.out.printf("%s\n", "Aralık");break;
         default: System.out.printf("%s\n", "Verdiğini numara geçersiz");break;
      }
   }
}
```

switch yapısında bulunan bir başka ifade “default” komutudur. Bu komut switch komutu içerisindeki ifade değerinin switch içerisindeki herhangi bir case ile eşleşememesi durumunda çalıştırılacak varsayılan değerdir. Örnekde, 12 dışında verilen bir değer ile eşleşen ay olmadığı için, 12 dışında verilen sayısal değer switch içerisinde otomatikman “default” etiketine düşecek ve burada belirlenen kodlar çalıştırılacaktır.

Döngüler (Loops)

Programlama dilleri ile uygulama geliştirirken, en çok kullandığımız işlemlerden biri, tekrarlanan operasyonları bilgisayarlara hızlı bir şekilde yaptırmaktır. Bu yüzden döngüler programlama konusunda çok önemli ve temel işlemlerden biridir.

Döngüler (Loops), hemen hemen her uygulamada ihtiyaç duyulur. Java’da bu tür döngü işlemlerini yapabilmek için birtakım komutlar sağlar. Bunlar while, do-while ve for döngü komutları olarak programcıların kullanımına sunulmuştur.

Döngü kullanımındaki amaç oldukça basittir. Verilen bir koşul, geçerliliğini sürdürdüğü sürece, döngü (loop) bloğu içerisindeki komutlar tekrar tekrar işleme alınır.

Java Döngüleri

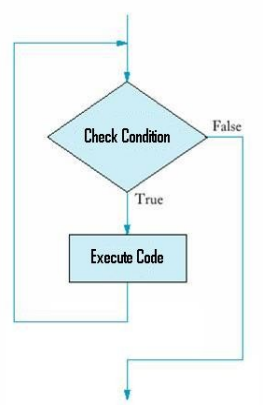
Java’da kullanılan döngü komutları, C ve türevi olan dillerde kullanılan döngü komutları ile hemen hemen aynıdır. Daha önce bu tür dillerden biri ile uygulama geliştirdi iseniz, Java döngülerinin sözdizimi ve kullanımı size çok tanıdık gelecektir.

while Döngüsü

Java'da while döngüsü, çok basit bir temel üzerinde tasarlanmıştır. while komutu içerisindeki expression (ifade) "true" değerini döndürdüğü sürece while döngüsü bloğu içerisindeki komutlar işleme alınır. Eğer expression, bir sonraki döngüde "false" değerini dönürür ise döngü bloğunun bittiği yerden program çalışmaya devam eder.

```
while(ifade)
{
   //döngü kodları
}
```

Aşağıdaki akış çizelgesinde (flow-chart) de while döngüsünün çalışma prensibini daha net bir şekilde görebilirsiniz.



Check Condition:Şartı Kontrol Et Execute Code:Kodu Çalıştır

while döngüsü kullanarak ayrıca, sonsuz bir döngü (infinite loop) kurarak, döngünün bir şekilde sonlanmasını önleyebilirsiniz. Bu bazı durumlarda oldukça ihtiyaç duyulan bir yöntemdir. Ya da, zaman zaman döngü koşulu döngü başlangıcında belirlenmemiş olabilir. Bu nedenden dolayı, döngüyü sonsuz bir döngü şeklinde ayarlayarak, döngü koşulunu blok içerisinde dinamik olarak belirleyerek, döngünün devam etmesini veya sonlanmasını sağlayabilirsiniz.

```
while(true)
{
   //döngü kodları
}
```

Şimdide while döngüsünün (while loop) kullanımına bir örnek verelim.

```
/**
Program 1 ile 100 arasındaki tüm çift sayıları
ekrana sırasıyla yazdırır
*/
package org.javablog;

public class Test
{
   public static void main(String[] args) {
      int i = 0;
      while(i < 100)
      {
         if(i % 2 == 0)
```

```
            {
               System.out.printf("%d ", i);
            }
            i++;
         }
      }
   }
```

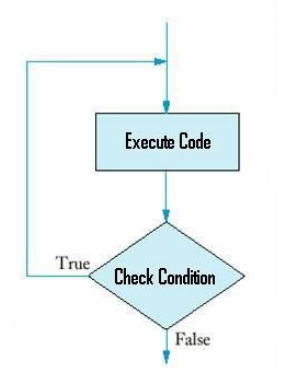
do-while Döngüsü

do-while döngüsü aslında while döngüsü ile aynı şekilde çalışır. Tek fark, programın içerisinde eğer bir while döngüsü var ise bu döngüye girilip girilmeyeceği while komutu içerisindeki ifadenin true olup olmamasına bağlıdır. Eğer ifade true değerini döndürmüyor ise, döngüye girilmeyerek, döngü bloğu bitimindeki komutlardan program devam eder.

Ancak, do-while döngü yapısında bu böyle değildir. Döngü, döngü koşulundaki değere bakılmadan mutlaka 1 kez çalıştırılır. Dönünün 2. veya daha fazla çalışıp çalışmayacağı, döngü koşuluna bakılarak eğer true dönüyor ise belirlenir.

```
do
{
   //döngü kodları
} while(ifade);
```

Aşağıdaki akış diyagramında do-while döngüsünün çalışma prensibi daha net olarak anlaşılabilir.



Check Condition:Şartı Kontrol Et Execute Code:Kodu Çalıştır

Şimdi de while döngüsü ile yazdığımız 1 den 100 e kadar olan sayılar arasında çift sayıları ekrana basan programı, do-while ile yapalım. Burda dikkat edilmesi gerek tek nokta, döngünün 1. çalışmasında while içerisindeki koşulun hiçbir şekilde önemi yoktur. Döngü mutlaka 1 kez çalıştırılacaktır.

```
package org.javablog;

public class Test
{
   public static void main(String[] args)
   {
      int i = 0;
      do
      {
         if(i % 2 == 0)
         {
            System.out.printf("%d ", i);
```

```
        }
      i++;
      } while(i < 100);
   }
}
```

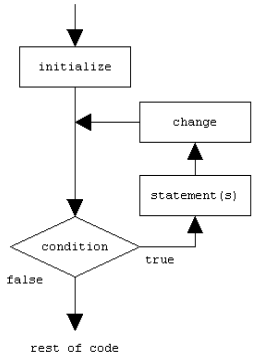
for Döngüsü

for döngüsü, while ve do-while döngülerine göre daha farklı bir kullanım alanına sahiptir. Programcılar, kendi belirledikleri koşul etrafında döngünün kabaca kaç defa döneceğine karar verirler.

for döngüsü 3 farklı yapı ile oluşturulur. Bunlar, döngüyü kullanıma hazırlama (initialization), koşul (condition) ve arttırma (increment). Arttırma bölümünde, döngü ifadesini oluşturan değişkenler güncellenerek döngünün çalışma yapısı güncellenir.

```
for(initialization; condition; increment)
{
   //döngü kodları
}
```

for döngüsünün çalışma prensibini gösteren şemada aşağıdaki gibidir. increment bölümü change ile ifade edilmiştir.



Condition:Şart Statement:Durum False:Yanlış True:Doğru Change:Değişme

while ve do-while döngüleri ile yapılan her türlü döngü işlemleri for ile de yapılabilir. for döngüsü döngüyü oluşturan ifadeleri tek bir satırda toplayarak, döngü yapısının okunmasını ve değişkenlerin güncellenmesi gibi komutların unutulmasında ortaya çıkabilecek hataların önüne geçebilmek için tasarlanan gelişmiş bir döngü yapısıdır.

```
//for döngüsü ile sonsuz döngü (infinite loop)
for( ; ;)
{
   // komutlar (statements)
}
```

Yine, 1 den 100 e kadar sayılar arasındaki çift sayıları ekrana bastıran programımızı bu kez for döngüsü ile yapalım.

```
package org.javablog;

public class Test
{
   public static void main(String[] args)
   {
```

```
    for (int i = 0; i < 100; i ++)
    {
       if(i % 2 == 0)
       {
          System.out.printf(" %d", i);
       }
    } // end of for
  } // end of main
} // end of class
```

görüldüğü gibi, değişken oluşturulması ve değişkenin (i) güncellenmesi gibi işlemler for döngüsünde tek bir satırda halledilerek döngü çok daha basit bir şekle indirgenmiştir.

1. **Veritabanı (Database) Nedir?**

Bir kurum veya kuruluşun birçok uygulamasında kullanılan, gereksiz yinelemelerden arınmış olarak, düzenli biçimlerde bilgisayar diskinde saklanan birbiriyle ilişkili veriler topluluğudur. Burada; "kurum ya da kuruluş ", bir okul, üniversite, banka, bir üretim şirketi, hastane, devlet kuruluşu, vb. olabilir. "Birbiriyle ilişkili veriler" bir kuruluşun çalışabilmesi, işleyebilmesi için kullanılan çok çeşitli verilerdir. Ticari bir şirket için müşteri bilgileri, satış bilgileri, ürün bilgileri, ödeme bilgileri, vb., okul için öğrenci bilgileri, açılan dersler, kimlerin kaydolduğu, öğretmen bilgileri, boş ve dolu derslikler, sınav tarihleri, vb., hastane için hasta bilgileri, doktor bilgileri, yatakların doluluk boşluğu, teşhis-tedavi bilgileri, mali bilgileri, vb …

2. Veritabanı Yönetim Sistemi (Database Management System)

Yeni bir veritabanı oluşturmak, veri tabanını düzenlemek, geliştirmek ve bakımını yapmak gibi çeşitli karmaşık işlemlerin gerçekleştirildiği birden fazla programdan oluşmuş bir yazılım sistemidir. Veri tabanı yönetim sistemi, kullanıcı ile veri tabanı arasında bir arabirim oluşturmaktadır ve veri tabanına her türlü erişimi sağlar. Veri tabanının tanımlanması: veri tabanını oluşturan verilerin tip ve uzunluklarının belirlenmesidir.

3. Neden Veritabanı

a) Gereksiz veri tekrarını önler:

Geleneksel veri depolamak için kullanılan dosya sistemlerinde, bazı veriler gereksiz şekilde bir çok kez tekrarlanır. Böylece milyonlarca kayıtın tutulduğu geleneksel dosya sistemlerinde dosya büyüklüğü gereksiz şekilde şişer.

b) Veri güvenliği sağlar

Bazı uygulamaları ürettiği verilerin güvenliğini sağlamak önemli konudur. Veritabanı kullanıcılarının veritabanının içerdiği tüm bilgilere kolayca erişmesi istenilen bir durum değildir. Örneğin, pazarlama bölümü uyglulamalarında çalışan bir kullanıcının, diğer personel özlük bilgilerine ulaşması engellenmelidir. Bunun gibi, her kullanıcının erişebileceği veriler ayrı ayrı tanımlanmalıdır. Veritabanı sistemlerinde kullanıcılara veritabanı üzerinde çeşitli yetkiler atanır ve bu yetkiler veritabanı üzerindeki veriler ile birlikte saklanır.

c) Çoklu kullanıcı erişimini sağlar

Veritabanı üzerindeki verilere aynı anda çok sayıda kullanıcı erişerek veriler üzerinde değişiklik yapabilir.

d) Aynı andaki erişimlerde tutarsızlıkları önler

Veritabanı uygulamalarında, veritabanı nesneleri başka başka uygulamalar tarafından paylaşılabilir. Veriler aynı anda farklı uygulamalar ve dolayısıyla farklı kullanıcılar tarafından aynı anda paylaşılabilir. Örneğin; bir elektronik eşya satan bir firmanın veritabanında ürün stoğunda 50 adet uydu alıcısı olduğunu varsayalım. İki farklı kullanıcıdan birinin aynı anda 25, diğerinin 30 adet uydu alıcı çıkışı yapmaya çalıştığını düşünelim. İşlem aynı anda yapıldığı için, 50 birimlik stoktan 55 birimlik çıkış yapılabileceği düşünülebilir. Ancak veritabanı yönetim sistemi buna izin vermez. Çıkışlar aynı anda yapılmasına rağmen, önce birincisini stoklardan çıkarır ve ikincisi için bir kontrol yaparak çıkışı önler.

4. Veri modelleme

Veri Tabanı Yönetim Sistemleri (VTYS) belirli bir veri modeline dayanır. Bir veri tabanı yapısının temelini veri modeli kavramı olusturmaktadır. Veriyi mantıksal düzeyde düzenlemek için; kullanılan kavramlar, yapilar ve islemler topluluguna “Veri Modeli” denir.

Şu ana dek birçok veri modeli gelistirilmistir. Bunlar

a) Hiyerarşik veri modeli
b) Ağ (network) veri modeli

c) İlişkisel veri modeli

d) Nesneye yönelik veri modeli

Sayılan bu veri modellerinin içinde en yaygın kullanılanı, ilişkisel veri modelidir. Günümüzde kullanılan VTYS'lerin hemen hemen tümü iliskisel veri modeline dayalidir. Son zamanlarda ortaya çıkan nesneye yönelik veri modeli, ilişkisel veri modeli ile birlikte bazı VTYS'lerde kullanılmaktadır.

5. E-R (Entitiy-Relationship) ilişkisel veri modeli

Veritabı tasarım işlemi birkaç değişik işlem aşamalarından oluşur. İlk olarak, veritabanı yapılacak kurum ya da kuruluşun ihtiyacı olan bilgilerin analizi yapılır. Bu süreçte, veritabanı tasarımcısı, veritabanını kullanıcıları ile görüşerek kullanıcı ihtiyaçlarını tespit eder. Veriler toplanıp sistem analizi yapıldıktan sonra, ilişkisel veri modeline göre veritabanı şeması oluşturulur. Bu şema, veritabanı kullanıcılarının ihtiyaç duydukları verilerin kısa açıklamasıdır. Bu şema içerisinde varlık tipleri ve ilişkilerin açıklaması bulunur. Daha sonra bu şemaya göre tablolar çıkarılır.

İlişkisel veri modeli türlerinden olan E-R (Entitiy-Relationship=Varlık-İlişki) modeli günümüzde yaygın şekilde kullanılmaktadır.

6. E-R modeli kavramları

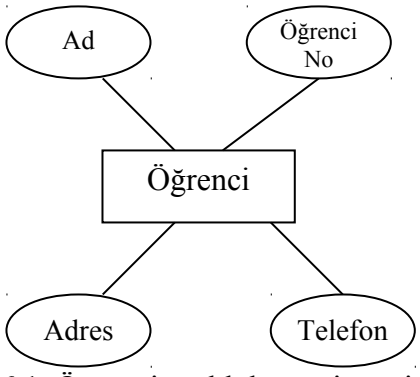
6.1. Varlık ve nitelikleri

Gerçek hayatta diğerlerinden ayırt edilebilen nesnelere varlık denir. Bir varlık, kişi, araba, ev veya çalışan gibi fiziksel nesneler olabileceği gibi, şirket, iş veya ders gibi fiziksel olmayan nesneler de olabilir. Her varlık kendisini tanımlayan kendisine has özelliklere sahiptir. Örneğin, bir çalışan varlığı, çalışan adı, yaşı, adresi, maaşı ve görevi özellikleri ile tanımlanır. Her varlığın niteliğinin bir değeri olur. Örneğin öğrenci varlığı olan o1 varlığının ad, öğrenci numarası, adres, telefon niteliklerinin değeri sırasıyla “Halil İbrahim Onay”, “08020618” “Gülbahar Mh., Rize”, “5128688888” verilebilir.

6.2. Varlık kümesi

Veritabanında benzer varlıklar ve nitelik değerlerinden oluşan kümeye varlık kümesi denir. Örneğin öğrenci varlıklarından o1 varlığı olan “Halil İbrahim Onay”, “080206018”, “Gülbahar Mh., Rize”, “5128688888”, ile birlikte o2 varlığı olan “Ferhat Kavcı”, “08020641”, “Avcılar, İstanbul”, “2125098888” ve o3 varlığı olan “Işıl Mavier”, “08020609”, “Manisa, Merkez”,

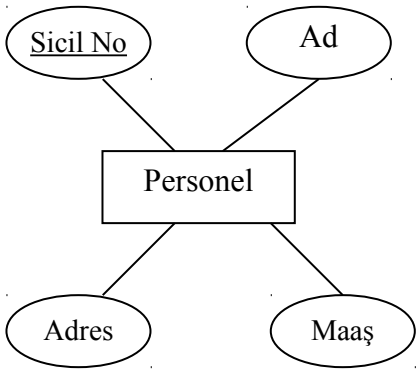
“02362348888” tümü öğrenci varlık kümesini oluştururlar. ER şemasında varlık kümeleri dikdörtgen içinde belirtilir. Nitelikler ise oval bir daire içinde belirtilerek ilgili varlık kümesine çizgi ile bağlanır. Şekil 6.1’de öğrenci varlık kümesi ve ad, numara, adres ve telefon niteliklerinin şeması görülmektedir.



Şekil 6.1. Öğrenci varlık kümesi ve nitelileri

6.3. Anahtar Nitelik

Varlık kümesindeki bir veya daha fazla niteliğin değeri, herbir varlık için farklı ise bu nitelik anhahtar niteliktir. Örneğin öğrenci varlık kümesinde öğrenciNo anahtar niteliktir. Çünkü bir üniversitede, öğrenci varlık kümesinde hiçbir varlığın öğrenciNo niteliği aynı olamaz. Benzer şekilde personel varlık kümesinde, sicilNo niteliği anahtar niteliktir. Hiçbir personelin sicilNo’su aynı olamaz. Anahtar nitelik, ER şemasında niteliğin altı çizilerek gösterilir. Şekil 6.2’de Personel varlık kümesinin anahtar niteliği ve diğer nitelikleri görülmektedir.

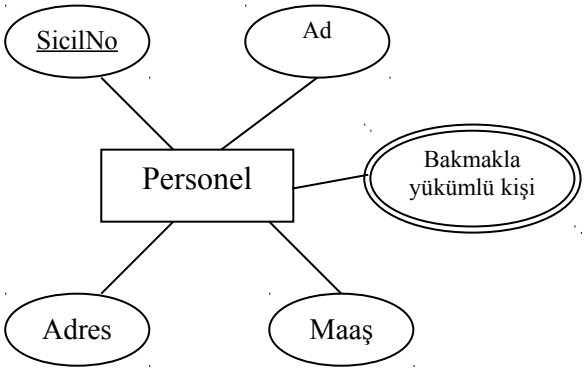


Şekil 6.2. Personel varlık kümesi ve sicilNo anahtar niteliği

6.4. Tek değerli nitelikler - çok değerli nitelikler

Genellikle bir varlığın tek bir değeri vardır. Örneğin personel varlığının sicil numarası niteleiği tek değeri olur. Bir varlığın bir niteliğinin aldığı değer tek ise bu niteliğe tek değerli nitelik denir. Ancak bazı niteliklerin birden çok değeri olabilir. Örneğin personel varlığının yabancı dil

niteliği, birden çok değer alabilir. Bu durumda personel varlığının yabancı dil niteliği çok değerli nitelik denir. Başka bir örnek olarak, bir personelin bakmakla yükümlü olduğu kişi niteliği bir kişi de olabileceği gibi birden çok kişi de olabilir. Bu durumda personel varlığının bakmakla yükümlü olduğu kişi niteliği çok değerli niteliktir. ER şemasında çok değerli nitelikler çift çizgili oval olarak gösterilir (Şekil 6.3).



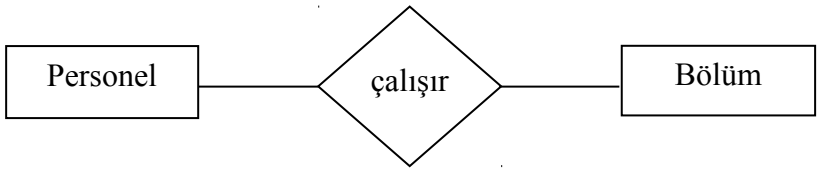
Şekil 6.3. Personel varlık kümesi ve çok değerli niteliği

6.5. Domain

Varlık kümesinde niteliklerin alabileceği değerler aralığını belirler. Örneğin öğrenci varlık kümesinde, dogumTarih niteliği 01.01.1980 ile 01.01.1990 arasında olabileceğini belirlemek için domain kullanılır. Domain aralığı ER şemasında gösterilmez.

7. İlişkiler

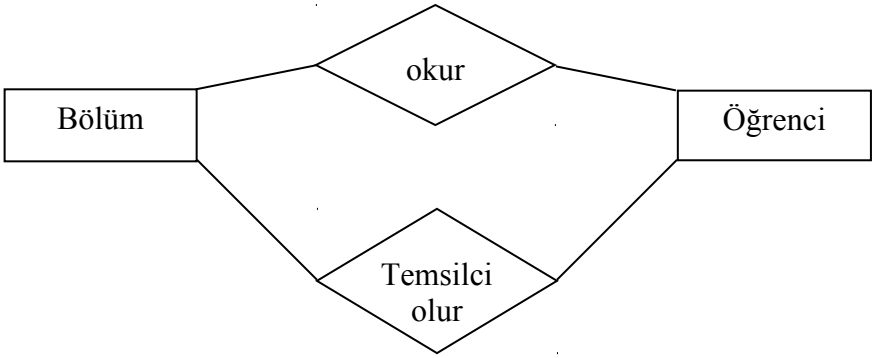
İki veya daha fazla varlık kümesi arasında kurulan anlamlı bağıntılara ilişki denir. İlişkiler ER şemasında dörtgen ile gösterilir. Dörtgen içine ilişkinin adı yazılır. Personel varlık kümesi ile bölüm varlık kümesi arasında çalışır ilişkisi (Şekil 7.1), müşteri varlık kümesi ile ürün varlık kümesi arasında satın alır ilişkisi örnekleri verilebilir (Şekil 7.2).



Şekil 7.1. Personel- Bölüm varlık kümesi çalışır ilişkisi

Şekil 7.2. Müşteri-Ürün varlık kümesi satın alır ilişkisi

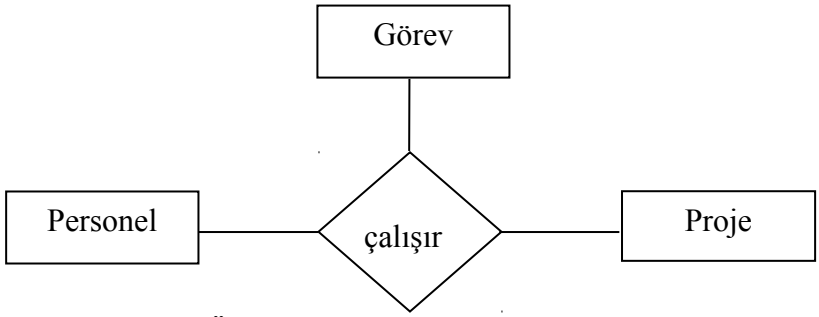
İki varlık kümesi arasında birden fazla ilişki de olabilir. Örneğin bölüm varlık kümesi ile öğrenci varlık kümesi arasında okur ilişkisi olabileceği gibi temsilcilik yapar ilişkisi de olabilir (Şekil 7.3).



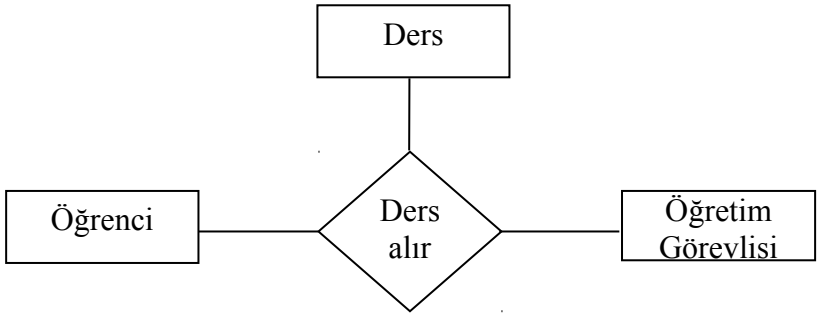
Şekil 7.3. Bölüm ile Öğrenci varlık kümeleri arasında iki farklı ilişki ER şeması

5.1. İlişki Derecesi

İlişkide bulunan varlık kümelerin sayısı ilişkinin derecesini oluşturur. Genelde ilişkiler iki varlık kümesi arasında yapılır. Ancak bazı durumlarda, ilişkide ikiden fazla kümesi yer alabilir. Örneğin bir personelin hangi projede hangi görevi yaptığı bilgisi gerekli ise; personel, proje ve görev varlık kümeleri arasında çalışır ilişkisi yapılır (Şekil 7.4). Benzer şekilde hangi öğrencinin hangi dersi hangi öğretim görevlisi bilgisi gerekli ise; öğrenci, ders ve öğretim görevlisi arasında ders alır ilişki yapılır (Şekil 7.5).



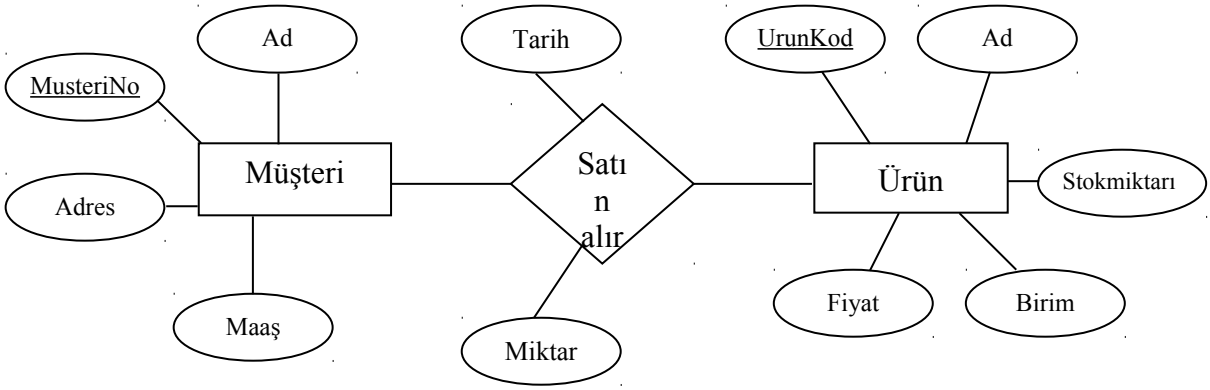
Şekil 7.4. Üç varlık kümesi arasındaki çalışır ilişkisi



Şekil 7.5. Üç varlık kümesi arasındaki ders alır ilişkisi

5.2. Tanımlayıcı Nitelik

Varlık kümeleri arasındaki ilişkide oluşan niteliklere tanımlayıcı nitelik denir. Tanımlayıcı nitelikler ilişkinin olması ile var olabilir. İlişki oluşmaz ise bu nitelikler de var olamaz. Müşteri varlık kümesi ile ürün varlık kümesi arasında satın alır ilişkisinde, müşterinin ürünü aldığı tarih ile satın aldığı ürünün miktarı nitelikleri tanımlayıcı niteliklerdir. Tanımlayıcı nitelikler ilişki ile bağlanır (Şekil 7.6).



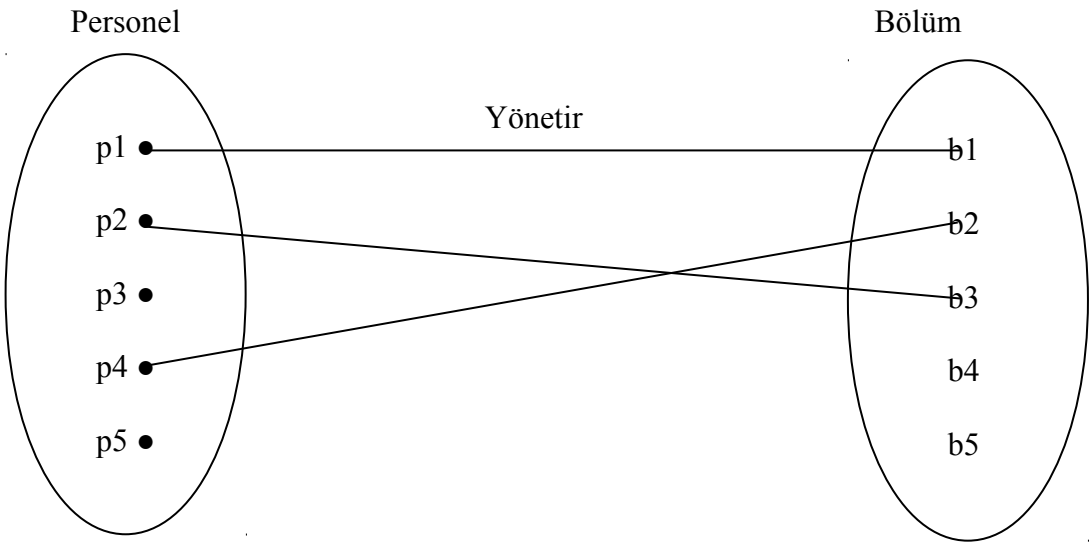
Şekil 7.6. Müşteri – ürün satın alır ilişkisinde tarih ve miktar tanımlayıcı niteliktir

5.3. İlişki Türleri

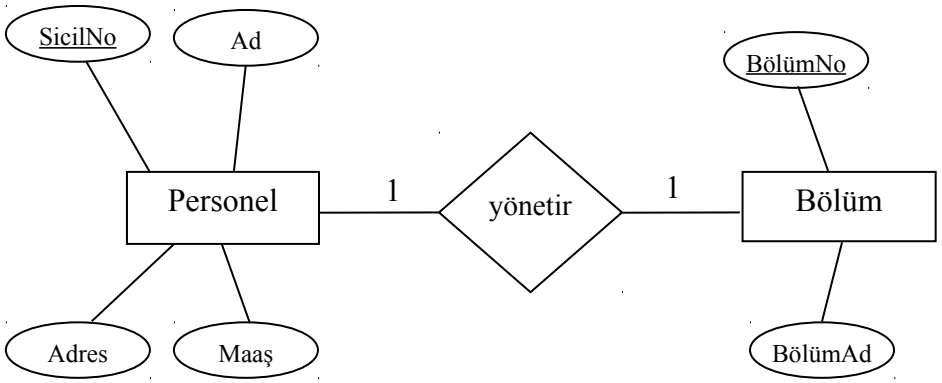
Varlık kümeleri aralarında 3 türde ilişki kurulabilir.

5.3.1. Bire-bir (1-1) ilişki

A varlık kümesinin bir elamanı, B varlık kümesinin sıfır ya da bir elemanı ile ilişki kurabiliyorsa bu ilişki türüne bire-bir ilişki denir. Örneğin personel varlık kümesi ile bölüm varlık kümesi arasında yönetir ilişkisi bire-bir ilişkidir (Şekil 7.7- Şekil 7.8)



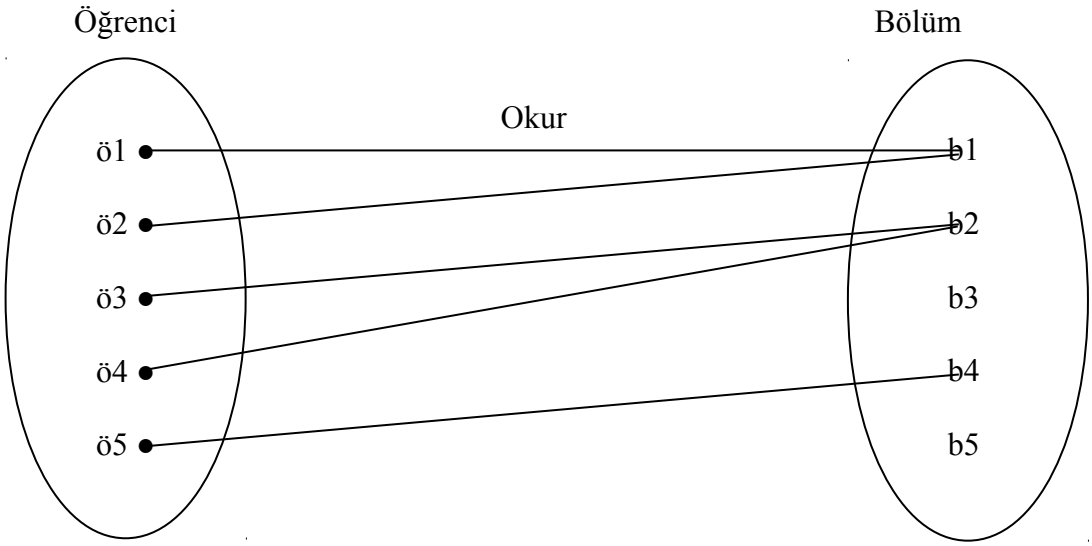
Şekil 7.7. Personel - Bölüm yönetir 1-1 ilişkisi



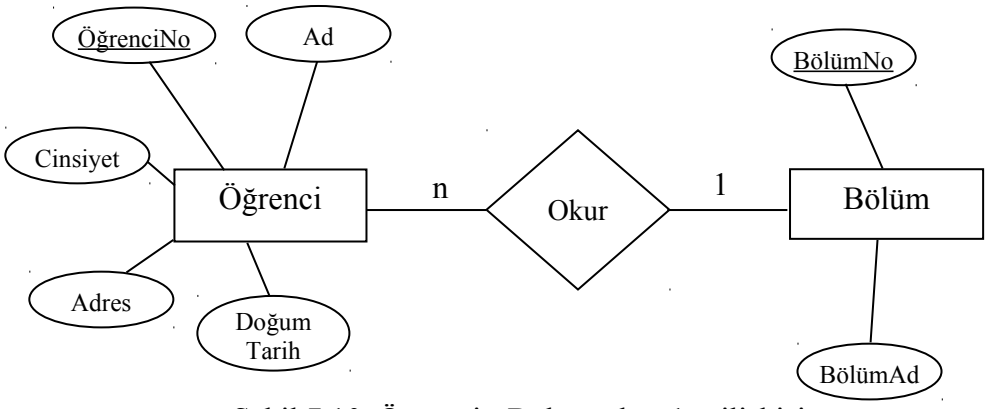
Şekil 7.8. Personel - Bölüm yönetir 1-1 ilişkisi

5.3.2. Bire-birçok (1-n) ilişki

A varlık kümesinin bir elemanı B varlık kümesinin 0 ya da birden çok elemanı ile ilişki kurabiliyorsa bu ilişki bire-bir ilişkidir. Örneğin öğrenci bölüm okur ilişkisinde bir öğrenci bire-birçok ilişkidir. Bir öğrenci en çok bir bölümde okur iken, bir bölümde birden çok öğrenci okuyabilir (Şekil 7.9-Şekil 7.10).



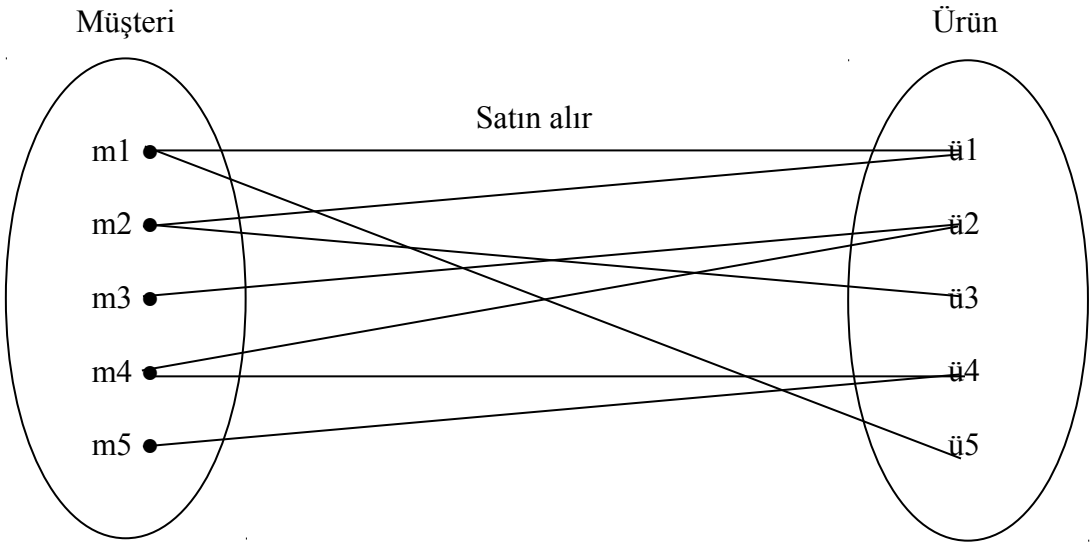
Şekil 7.9. Öğrenci - Bölüm okur 1-n ilişkisi



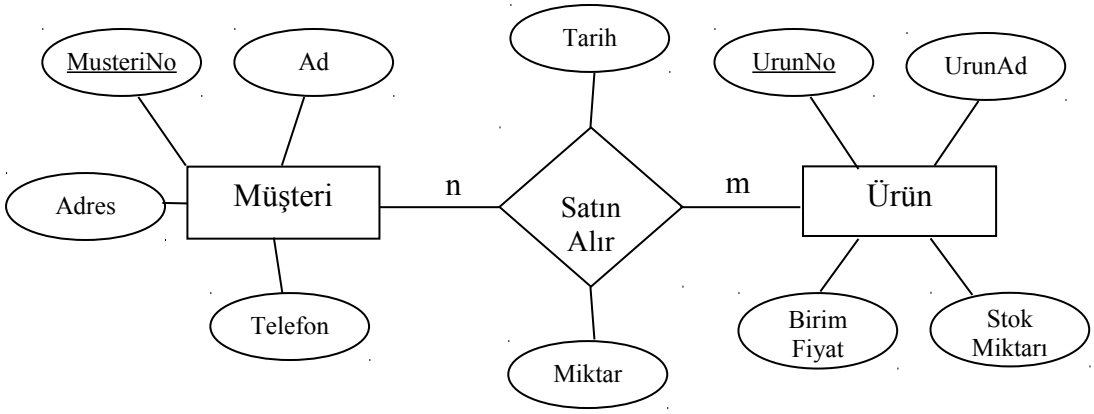
Şekil 7.10. Öğrenci - Bölüm okur 1-n ilişkisi

5.3.3. Birçoğa-birçok (n-n ya da n-m) ilişki

A varlık kümesinin bir elemanı, B varlık kümesinin sıfır ya da birçok eleman ile ilişki kurabiliyor ve B varlık kümesinin bir elemanı, A varlık kümesinin sıfır ya da birçok elemanı ile ilişki kurabiliyorsa bu ilişki türü birçoğa birçok ilişki türüdür. Birçoğa birçok ilişki kısaca n-m ilişki olarak da adlandırılabilir. Örneğin müşteri ile ürün varlık kümeleri arasında satın alır ilişkisi birçoğa birçok ilişkidir. Çünkü bir müşteri birden çok ürün satın alabilirken, bir ürünü birden çok müşteri satın alabilir (Şekil 7.11- Şekil 7.12).



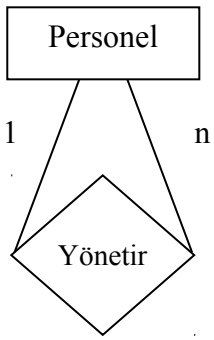
Şekil 7.11. Müşteri-ürün satın alır n-m ilişkisi



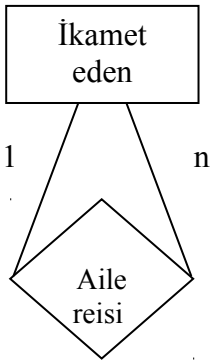
Şekil 7.12. Müşteri-ürün satın alır n-m ilişkisi

5.4. Tekrarlamalı (Recursive) İlişki

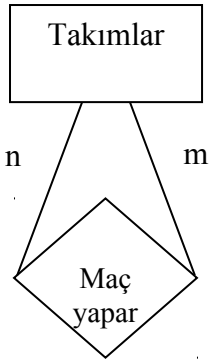
Genellikle, ilişkiler farklı varlık kümeleri arasında oluşur. Ancak bazı durumlarda tek varlık kümesi üzerinde ilişki kurulabilir. Örneğin personel varlık kümesinde yönetir ilişkisi tekrarlamalı ilişkidir. Bir personel birden çok personeli yönetebilir. Bu ilişki Şekil 7.13 de görülmektedir. Şekil 7.14'te ikamet eden varlık kümesinde aile reisi olur ilişkisi tekrarlamalı ilişkidir. Bir ikamet eden diğer ikamet edenlerin aile reisidir. Benzer şekilde futbol takımlarının kendi aralarında maç yapar ilişkisi de tekrarlamalı (recursive) ilişkidir Şekil (7.15)



Şekil 7.13. Personel varlık kümesinin yönetir 1-n tekrarlamalı ilişkisi



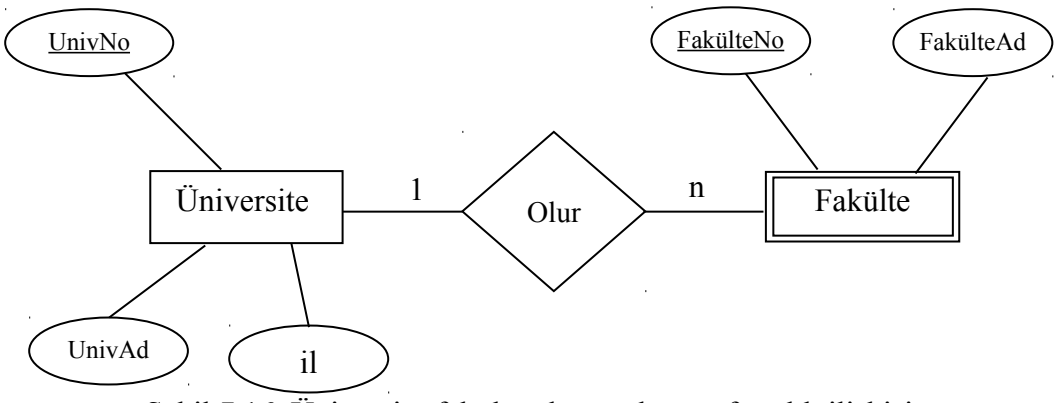
Şekil 7.14. İkamet eden varlık kümesinin aile reisi olur 1-n tekrarlamalı ilişkisi



Şekil 7.15. Takımlar varlık kümesinin maç yapar n-m tekrarlamalı ilişkisi

5.5. Güçlü-Zayıf Varlık Kümesi

Bir varlık kümesinin nitelikleri içerisinde anahtar nitelik (alan) oluşturulamıyorsa, bu varlık kümesine zayıf varlık kümesi denir. Bir varlık kümesinin nitelikleri içerisinde en az bir tanesi anahtar nitelik olabiliyorsa bu varlık kümesine güçlü varlık kümesi denir. Zayıf varlık kümeleri, güçlü varlık kümeleri ile 1-n ilişki oluştururlar. Ancak her 1-n ilişki güçlü-zayıf varlık kümesi ilişkisi değildir. Zayıf varlık kümeleri ER şemasında çift çizgili dikdörtgen ile gösterilir. Ayrıca zayıf varlık kümelerinin güçlü varlık kümelerine varolma bağımlılığı olması gerekir. Üniversite fakülte ilişkisinde, bir fakülte üniversite olmadan olamayacağı için ve aynı fakülte isminde başka üniversitelerde fakülte olabileceği için fakülte varlık kümesi zayıf varlık kümesidir Şekil 7.16.



Şekil 7.16. Üniversite-fakülte olur güçlü-zayıf varlık ilişkisi

SQL NEDİR?

SQL bir veri tabanıyla iletişim kurmak için kullanılır. ANSI standardına göre ilişkisel veri tabanı yönetim sistemlerinin standart dilidir. SQL cümleleri, bir veri tabanındaki verileri güncellemek, veri tabanından veri çıkarmak, veri silmek vb için kullanılır. Çok bilinen bazı ilişkilsel veri tabanı yönetim sistemleri şunlardır: Oracle, Sybase, MS SQL Server, Access, Ingres... Her ne kadar çoğu veri tabanı yönetim sistemi SQL'i kullansa da çoğunun kendilerine özgü ek özellikleri vardır.

Standart SQL cümleleri, SELECT, INSERT, UPDATE, DELETE, CREATE ve DROP'tur.

TABLO KAVRAMI

Bir ilişkisel veri tabanı sistemi (relational data base management system) bir veya daha fazla TABLO adı verilen nesnelerden meydana gelir. Veriler bu tablolarda saklanır. Tablolar, satır ("row") ve sütunlar ("column") dan meydana gelirler ve her tablonun benzersiz ("unique") bir ismi vardır. Aşağıda örnek bir tablo görüyorsunuz.

DERSLER tablosu

Adı	Ogr. Uyesi	Sınıf	Kredi
Eğitimde Bilgi tekn.	Filiz Eyüboğlu	1	4
Programlama Dilleri	Filiz Eyüboğlu	2	4
Ogretim Tasarımı	Feza Orhan	3	2
Yazarlık Dilleri	Betül Yılmaz	3	4

Tabloda 4 satır, 4 kolon var.

VERİNİN SEÇİLMESİ

SELECT cümlesi bir veri tabanından verileri seçmek ve çıkarmak ("retrieve") için kullanılır.
SELECT cümlesinin formatı:

```
SELECT "kolon1" [,kolon2", vb]
FROM "tablo adı"
WHERE "koşul"];
```

[ ] : seçimlik anlamına gelir.

WHERE clause'da kullanılabilecek operatörler şunlardır:

= eşittir
> büyüktür
< küçüktür
>= büyük veya eşittir
<= küçük veya eşittir
< > eşit değildir
LIKE ----------→ LIKE çok güçlü bir operatördür. Belirttiğiniz şeye benzeyenleri seçer.Like operatörü bize karakter içinde istediğimiz karakter dizisini aramımızı sağlar.
Örneğin

```
SELECT isim, soyadı
   FROM calisan
   WHERE soyadi LIKE 'Er%'
```

Bu sorgu, soyadı 'Er' ile başlayan kayıtları seçer ve getirir.

Diğer bir örnek:

```
SELECT isim, soyadı
   FROM calisan
   WHERE soyadi LIKE '%s';
```

Soyadı 's' ile bitenleri seçer ve getirir.

Örnek tablo
EMPINFO (Çalışan Bilgilerini tutan tablo)

First	Last	Id	Age	City	State
John	Jones	99980	45	Payson	Arizona
Mary	Jones	99982	25	Payson	Arizona
Eric	Edwards	88232	32	San Diego	California
MaryAnn	Edwards	882338	32	Phoenix	Arizona
Ginger	Howell	98002	42	Cottonwood	Arizona
Sebastian	Smith	92001	23	Gila Bend	Arizona
Gus	Gray	22322	35	Bagdad	Arizona
MaryAnn	May	32326	52	Tucson	Arizona
Erica	Williams	32327	60	Show Low	Arizona
Leroy	Brown	32380	22	Pinetop	Arizona
Elroy	Cleaver	32382	22	Globe	Arizona

SELECT cümlesiyle ilgili örnek sorgular:

1. Tablodaki herkesin ilk ismini ve yaşını görüntüleyiniz.

```
select ad,
```

```
        yas
    from tablo;
```

2. Payson'dan olmayan kişilerin ilk adını, soyadını ve şehrini görüntüleyiniz.

```
  select ad,
         soyad,
         sehir
    from tablo
  where sehir <>
    'Payson';
```

3. 40 yaşın üzerinde olan kişilerin tüm bilgilerini (tüm kolonlar) görüntüleyiniz.

```
       select * from tablo
        where yas > 40;
```

4. Soyadı 'ay' ile bitenlerin adını ve soyadını görüntüleyiniz.

```
  select ad, soyad from tablo
         where soyad LIKE '%ay';
```

5. Adı 'Mary' olanları tüm bilgilerini görüntüleyiniz.

```
  select * from tablo
         where ad = 'Mary';
```

6. Adında 'Mary' geçenlerin tüm bilgilerinin görüntüleyiniz.

```
  select * from tablo
         where ad LIKE '%Mary%';
```

TABLO YARATMA

`CREATE TABLE` cümlesi yeni bir tablo yaratmak için kullanılır. Formatı şöyledir:

```
CREATE TABLE "tablo adı"
   ("kolon1" "veri türü"[constraint],
    "kolon2", "veri türü"[constraint],
  ..... );
```

Örnek:

```
CREATE TABLE calisan
    (adi varchar(15),
    soyadi varchar(20),
    yas number(3),
    adres varchar(30),
    sehir varchar(20) );
```

adi,soyadi,yas,adres ve sehir sütunları olan calisan tablosunu yarat

Tablo ve kolon isimleri bir harf ile başlamalıdır. Devamında ise harfler, rakamlar ve underscore karakteri " _ " bulunabilir. Uzunluk 30 karakteri geçmemelidir. SQL özel sözcükleri (SELECT, INSERT, CREATE vb gibi) tablo ve kolon adı olarak kullanılamaz.

Bir kolona girilecek verilerle ilgili kurallar'a "**constraint**" denir. Örneğin **"unique"** constraint', tablodaki kayıtlarda bu kolona girilecek değerlerin benzersiz ("unique") olması gerektiğini yani her hangi iki kolonda aynı değerin olamayacağı kuralını koyar. **"primary key"** constraint'i bulunduğu kolon değerinin tablodaki kayıtlara erişilirken **birincil anahtar** olarak kullanılmasını söyler. Birincil anahtar bildirimi yapıldıysa kayıtlara – sistem sıralı okuma yapmadan - doğrudan erişir. Birincil anahtar değerleri benzersiz olmalıdır.

Tablo Yaratma Uygulaması

Bir şirkette çalışanların bilgilerini tutmak için bir tablo yaratılacak. Tablodaki bilgiler şunlar olacak: adi, soyadi, unvan, yas, maas.

ÖNEMLİ: Tablonuza isim seçerken **herkesin kullanmayacağı bir isim seçiniz. CALISAN_ogrenci numaranız** gibi. Çünkü bu tablolara aynı veri tabanında yer alacaktır ve bir veri tabanındaki tablo isimleri benzersiz olmalıdır.

```
  CREATE TABLE CALISAN_FILIZ
    (ADI VARCHAR(15),
     SOYADI VARCHAR(20),
     UNVAN VARCHAR(15),
     YAS NUMBER(2),
     MAAS NUMBER(8) );

CALISAN_FILIZ adlı tablo yaratılır
```

TABLOYA EKLEME YAPMA ("INSERT")

INSERT cümlesi tabloya veri eklemek için kullanılır.

```
INSERT INTO "tablo adı"
     (birinci kolon, ........, sonuncu kolon)
     VALUES(ilk değer,..........son değer);
```

Örnek:

```
INSERT INTO calisan
   (ad, soyad, yas, adres, sehir)
  VALUES ('Ayşe', 'Yılmaz', 30, 'Papatya sokak', 'Ankara');
```

Calisan adlı tabloya adı Ayşe, soyadı Yılmaz, yası 30 ,oturduğu şehir Ankara ve oturduğu sokak papatya sokak olan kişiyi ekle.

KAYITLARI GÜNCELLEME ("UPDATE")

Kayıt güncelleme için UPDATE cümlesi kullanılır. Formatı şu şekildedir:

```
UPDATE "tablo adı"
```

```
SET "kolon adı" = "yeni değer" [, bir sonraki "kolon adı" = yeni
                                 değer".....]
WHERE "kolon adı" OPERATOR "değer"
       [AND | OR "kolon" değer" OPERATOR "değer"];
```

Örnekler:

```
UPDATE calisan
SET yas = yas + 1
WHERE isim = 'Ayşe' AND soyadi = 'Yılmaz';
```

calisan tablosunda adı Ayşe,soyadı Yılmaz olan kişinin yaşını bir artır.

```
UPDATE telefon_defteri
SET alan_kodu = 212
WHERE posta_kodu = 34340;
```

Telefon_defteri tablosunda posta kodu 34340 olan kayıtların alan kodunu 212 yap.

```
UPDATE calisan
SET maas = 7000, unvan = 'veri tabani yöneticisi'
WHERE adi = 'ege' AND soyadi = 'erdem';
```

calisan tablosunda adı ege,soyadı erdem olan kişinin maasını 7000, unvanı veri tabani yöneticisi yap.

UPDATE uygulamaları:

Her UPDATE'den sonra güncellemenizi teyit edecek bir SELECT cümlesi yazıp çalıştırınız.
(daha önce yarattığınız **CALISAN_....tablosu** kullanılacak)

1- İpek Özgür, Ali Yıldırım ile evlendi; soyadını güncelleyiniz.

```
update
 tablo
set soyad=
  'Özgür-Yıldırım'
where ad=
      'İpek'
 and soyad =
     'Özgür';
```

2- Ege Erdem'in doğum günü oldu, yaşını 1 artırınız.

```
update tablo
  set yas=yas+1
  where ad='Ege' and soyad='Erdem';
```

3- Tüm sekretelerin ünvanı "Yönetici Asistanı" oldu; güncelleyiniz.

```
update tablo
  set title = 'Yönetici Asistanı'
  where title = 'Sekreter';
```

4- 4000 altında maaş alanlara 500 zam yapıldı; güncelleyiniz.

```
update tablo
  set maas = maas + 500
  where maas < 4000;
```

KAYIT SİLME (“DELETE”)

```
DELETE “tablo adı”
    WHERE “kolon adı” OPERATOR “değer”
     [ AND | OR “kolon adı” OPERATOR “değer” ];
```

Örnekler:

```
DELETE FROM calisan;
```
==➔ böyle yazıldığında tablodaki tüm satırlar silinir.

```
DELETE FROM calisan
   WHERE soyadi = ‘Yılmaz’;
DELETE FROM calisan
    WHERE adi = ‘İpek’ OR adi = ‘Canan’;
```

Delete uygulamaları

DELETE cümlelerinizin doğru çalıştığını görmek için SELECT cümlesi kullanınız.

1- Ege Erdem firmadan ayrıldı, kaydını siliniz.

```
delete
  from tablo
  where soyad =
    'Erdem';
```

2- 7000’in üzerinde maaş alanlar bütçe kısıtlaması nedeniyle işten çıkarıldı. Bu kişileri tablodan çıkarınız.

```
delete
  from tablo
  where maas >
    70000;
```

TABLO SİLME

DROP cümlesi tabloyu ve tüm kayıtları silmek için kullanılır.

```
DROP TABLE “tablo adı”;
```

Drop uygulaması:

Yaratılmış olan **calisan** tablonusu siliniz. => `DROP TABLE calisan;`