

1. VERİ

Birlikte Düşünelim

Veri kolayca saklanabiliyorsa hacmi arttığında; bir diğer deyişle adı “büyük veri” olduğunda neden problemlerle karşılaşmaya başlıyoruz? Büyük verinin sebep olduğu problem nedir?

Sürekli veri üretmek sınırsız saklama kapasitesine sahip olmamızı gerektirmez mi?

Büyük firmalar neden sistemlerini çalışır durumda tutabilmek için yüzlerce bilgisayar programcısına ihtiyaç duyuyorlar?

Veri eskiyebilir mi?

Yapay zekanın veri ile ilişkisi nedir?

Saniyede binlerce istek alan internet sunucuları, bu kadar isteğe yanıt verecek veriyi nasıl saklıyorlar?

Başlamadan Önce

Bir bilgisayar programının veri ihtiyacını karşılamaya yönelik bir veri tabanı tasarımı gerçekleştirmeden önce veri kavramını daha yakından incelememiz gerekiyor. Nasıl edinebiliriz? Nasıl saklayabilir ve aktarabiliriz? Veri hangi biçimlerde bulunur? Performansı arttırmak için verinin biçimini dönüştürebilir miyiz? Veriyi tanımak, iyi bir veri tabanı tasarımı gerçekleştirmek için bize büyük fayda sağlayacaktır.

1.1. Veri Nedir?

Elinize bir kağıt ve kalem alıp 35 sayısını yazdığınızı düşünün. Bu anlamlı bir içerik midir? Burada 35 sayısı kendi içinde anlamlı gibi gözükse de bize ne ifade ettiği önemlidir. Burada 35, devamı gelecek bir kayıtlar dizisinin tek bir elemanı olarak karşımıza çıksa da tek başına değerlendirdiğimizde ancak veri olarak nitelendirebiliriz. Böylece veri kavramıyla karşılaşmış oluyoruz. Kaydedilebilir bilginin yapı taşı diğer bir deyişle atomik haline veri adını veriyoruz. Veriyi kaydedebilmek için illaki dijital ortama ihtiyaç duymasak da bu dersin kapsamı içerisinde elektronik ortamda veri ve verinin kaydedilmesi ile ilgileneceğiz.

Önemli

Veri kelimesi karşımıza zaman zaman “data” olarak çıkmaktadır. “Data” kelimesi İngilizce’dir ve veri kelimesinin çoğul halidir. Aynı kelimenin tekil hali yine İngilizce’de “datum” olarak geçmektedir.

Veri; ölçüm, sayım, deney, gözlem ya da araştırma yöntemiyle elde edilen kayıtlara denilmektedir. Yapılandırılmış veri; sayısal, kategorik, mantıksal, tarih, görsel ve benzeri içeriklerdir. Yapılandırılmamış veri ise herhangi bir ön tanım olmadan kayıt altına alınan metin, ses, görsel ve video gibi içeriklerdir (Aksoy, Çelik ve Gülseçen, 2020). Kaydettiğimiz en küçük birim olan veri (datum: tekil, data: çoğul), dijital dünyadaki gelişmelerle birlikte her alanda çok daha fazla adı anılan bir kavram olmaya başladı. Veriyi birçok şekilde elde edebiliyoruz. Bir değeri ölçmek ya da saymak ile elde edebileceğimiz gibi deney, gözlem ya da araştırma sonucunda da veriye ulaşmaktayız. Bunun yanında veriyi bulunduğu biçim sebebiyle nicel ya da nitel olarak ikiye ayırmaktayız. Ölçüm ile elde ettiğimiz veri türüne nicel, araştırma vb. faaliyetler ile elde ettiğimiz ifade, durum, özellik vb. sayısal olmayan veri türüne ise nitel adı verilmektedir.

Bu kavramları örnekleyerek açıklamakta fayda var. Bir etkinlik düzenlediğimizi ve katılımcıların yaş bilgisini bir deftere not aldığımızı düşünelim. Herhangi tek bir katılımcıya ait yaş bilgisi bizim için veri niteliği taşımaktadır. Burada dikkat edilecek nokta tek bir katılımcının yaşından bahsediyor olmamızdır. Örneği 35 yanıtını aldıysak burada elimizde yalnızca “35” değeri mevcuttur. Bu değer bizim için ilgili katılımcının 35 yaşında olması durumu dışında hiçbir bilgi sunmamaktadır. Veri için söylenen, tek başına bir

anlam ifade etmeme durumu bu sebeple karşımıza çıkmaktadır. Eğer elde 100 katılımcıya ait yaş bilgisi bulunursa etkinliğe katılanların yaş ortalamaları konusunda bilgi sahibi olabiliriz. Bu da eldeki verilerin bir araya gelerek yeni bir enformasyonu oluşturmaktadır. Enformasyon konusunu ayrıca irdeleyeceğiz.

Gelen katılımcının yaşının 35 olması, sayısal ve ölçülebilir bir değer olması sebebiyle nicel olarak karşımıza çıkmaktadır. Bunun yanında bu katılımcının etkinliğe katılma durumu “katıldı” ya da “katılmadı” şeklinde olduğu için nitel olarak değerlendirilebilir. Bazı durumlarda nitel ve nicel verilerin birbirine dönüştürülmesi de mümkün olabilmektedir. Örneğin katıldı bilgisini 1 katılmadığı bilgisini ise 0 olarak kodlamamız mümkündür. Böylece nitel türdeki bir veriyi nicel hale dönüştürmüş olacağız. Veriyi bu şekilde kaydettiğimizde istersek 1 ve 0 değerlerini katıldı ya da katılmadı olarak anlamlandırabileceğimiz gibi 1 değerlerini topladığımızda toplam katılımcı sayısını da elde edebiliriz. Ayrıca veri kaydında yalnızca tek haneli veri kaydetmemiz gerektiği için hacim açısından da oldukça tasarruf etmiş oluruz.

Veri ile ilgilenmeye başladığımızda genellikle tek bir sayı ya da sadece katıldı veya katılmadığı bilgisini saklamaktan çok daha büyük veri kümeleriyle uğraşacağımızı biliyoruz. Ancak ilgileneceğimiz veri kümelerinin atomik boyutta hangi biçimde tutulduğu ya da ne şekilde saklanması gerektiğini bilmemiz oldukça önemlidir. Bir verinin kağıda yazılmasıyla bilgisayar ortamında saklanması çok ayrı şeylerdir. Örneğin 35 sayısının kağıt üzerinde sayıyla ya da yazıyla bulunması yalnızca efor farkı yaratırken; bilgisayar ortamında metin, tam sayı ya da ondalıklı sayı hallerinden birinde olması, bilgisayarın bu veriye yaklaşımını tamamen değiştirecektir. Hatta 35 sayı biçiminde gösterilse de bilgisayar belleğinde sayı ya da metin olarak tutulabilir. Tüm bu durumlar bilgisayar gücü için önemsizmeyecek boyutlu olsa da ilgilendiğimiz veri miktarı çok arttığında performansı büyük etki eden olgulardır.

1.2. Verinin Yolculuğu

Verinin tek başına bir anlam ifade etmediğinden bahsetmiştik. Ancak aynı zamanda bilginin yapıtaşı olduğundan da bahsettik. Peki bu dönüşüm nasıl gerçekleşiyor? Bunun için Rowley’in bilgelik hiyerarşisine göz atmak gerekiyor (Rowley, 2007).



Hiyerarşiyi iki açıdan ele almak gerekiyor. Birincisi verinin, hiyerarşide en alt basamak olarak gösterilmiş olması. Bu durum hiyerarşideki diye adımlara ulaşmak için veriyi elde etmenin zorunlu bir adım olduğunu göstermektedir. Bir konuda enformasyona, bilgiye ya da çok daha önemlisi olan bilgelığe yani karar verici olabilmeye veri sayesinde ulaşılabilir. Elde yeterli veri bulunmadığında doğru ya enformasyon ya da bilgiye ulaşmanın ötesinde, tahminde bulunmak dahi mümkün olmayacaktır. İkinci açı ise genişlik ile görselleştirilmiş olan miktar bilgisidir. Piramidin her bir aşaması bir üst aşamadaki olgudan daha az miktarda elde edilmesini sağlar. Bu da her halükarda çok fazla miktarda veriye sahip olmamız gerektiğini gösterir. Çok fazla veriye sahip olmak çok fazla enformasyon elde edeceğimizi garanti etmez ancak az miktarda veri ile çok az enformasyona sahip olabileceğimiz kesindir (Akadal, 2020). Bunu biraz açalım. Herhangi bir şekilde çok fazla veriye sahip olmuş olabiliriz. Ancak çok fazla veriye sahip olmak her zaman enformasyonu elde edebileceğimizi göstermez. Bu sebeple verinin miktarının yanında niteliğinin de önemi bulunmaktadır. Hiyerarşiye enformasyon açısından baktığımızda ise şunu söyleyebiliriz ki belirli bir miktarda enformasyon elde edebilmek için veriye mutlaka ihtiyaç duyulmaktadır. Burada karşılaştığımız problem şudur: bir veri

kümesini toplarken ya da edinirken, bu veri kümesini kullanarak enformasyon elde edebileceğimiz hiçbir zaman kesin değildir. Veri madenciliği kavramı tam bu noktada karşımıza çıkmaktadır. Veri madenciliği araştırmalarıyla araştırmacılar, çok miktarda veri üzerinde çeşitli yöntemler kullanarak enformasyon elde etmeye odaklanmış durumdadırlar.

Dikkatle incelendiğinde ve üzerine düşünüldüğünde bu piramit günümüzde verinin neden bu kadar önemli olduğunu göstermektedir. Özellikle işletmeler mevcut ya da potansiyel müşterileri hakkında çok fazla veri toplamak isterler. Çünkü veri miktarı arttıkça müşteriler hakkında edindikleri enformasyon ve dolayısıyla bilgi miktarı artar. Bu da satışlarını olumlu yönde arttırmak için doğru kararlar almalarını sağlayabilir.

Şimdi verinin yolculuğunun ikinci adımı olan enformasyon kavramını inceleyelim. Enformasyon veri kullanılarak elde edilebilen anlamlı yargılardan oluşmaktadır. Yine aynı örnekten yola çıkarsak, düzenlediğimiz etkinliğe katılan tüm kişilerin yaş bilgisini kaydettiğimizde ve bu verinin ortalamasını aldığımızda etkinliğe katılımın ortalama yaşını, hangi yaş gruplarının bu etkinliğe ilgi gösterdiğini ve odaklanılması gereken hedef grubu hangi grup olduğunun belirlenmesi sağlanabilir. Enformasyon kelimesi gündelik hayatta karşımıza pek çıkmamaktadır. Bunun en büyük sebebi “Knowledge” ve “Information” kelimelerinden Türkçe’ye “bilgi” olarak geçmesidir. Biz, “Knowledge” kelimesini bilgi, “Information” kelimesini ise enformasyon karşılığı ile kullanacağız.

Önemli

IT kısaltmasıyla sıklıkla karşımıza çıkan Bilgi Teknolojileri ifadesi gerçekte Information Technologies ifadesinin kısaltmasıdır ve doğru çevirisi Enformasyon Teknolojileri’dir.

Uluslararası arenada sıklıkla kullanılan ancak ülkemizde pek bilinmeyen bir diğer çalışma alanı ise Enformatik’tir (Informatics). Bu alan verinin enformasyona dönüşümündeki tüm süreçleri konu alır. Verinin elde edilmesi, saklanması, işlenmesi, enformasyona dönüştürülmesi için hazırlanması ve bu süreçte kullanılacak tüm araç ve yöntemleri içermektedir.

Havanın sıcaklık değerine bakıp sıcak olup olmadığına karar vermek, trafik yoğunluğuna bakarak tahmini yolculuk süresi hakkında bilgi sahibi olmak, evde bulunan su miktarına bakarak bir sonraki su siparişinin zamanlamasını tahmin etmek enformasyona birer örnektir. Enformasyon genellikle sadece veriden elde edilmiş küçük anlamlı parçacıklardır. Daha kapsamlı karar almak için enformasyonun bilgiye dönüşmesi gerekir. Örneğin; bir işletmede personelin işe gelişi ve işten ayrılma saatlerinin kayıtları personelin performans göstergesi olarak kullanılabilecek olsa da sadece bir enformasyondur ve tek başına karar vermek için yeterli değildir. Benzer şekilde sipariş sıklığının zamana göre değişim grafiği bir enformasyon sunsa da sipariş sıklığının değişimi konusunda kesin bir yargı içermeyecektir. Verilerden fayda elde edebilmemiz için onları anlamlı hale getirmemiz yani enformasyona dönüştürmemiz gereklidir ancak enformasyon bu yolculuğun yalnızca ikinci adımı olabilecektir. Herhangi bir konudaki istatistikleri incelediğimizde doğrudan net bir yargıya ulaşamamamızın sebebi budur. İstatistikler verilerin anlamlı hale getirilmesi olarak karşımıza çıkarlar ancak bu anlamlı parçaları yorumlamak için de ek bir efor harcamamız gerekmektedir. Verinin yolculuğunun üçüncü adımı burada karşımıza çıkmaktadır: Bilgi.

Verinin yolculuğunun üçüncü adımı olan bilgi, tahmin edeceğimiz üzere enformasyonların bir araya gelmesi sonucunda elde edilebilen bir olgudur. Bu aşamaya veriye dayalı kanıya sahip olabileceğimiz en sağlıklı aşamadır. Bir işletme üzerinden örnek vermek gerekirse; satış grafikleri, piyasa dinamikleri, reklam faaliyetleri, kullanıcı memnuniyeti, ekonomik olgular ve benzeri birçok enformasyonun bir adım arada değerlendirilmesi neticesinde bilgi erişilebilir. Yine fark edeceğimiz üzere enformasyon ve bilgi arasında da bir hacim ilişkisi bulunmaktadır. Bilgi için çok miktarda enformasyona ihtiyaç duyulur. Ancak çok fazla enformasyonun bilgiye dönüşebileceği her zaman kesin değildir. Bilgiye ulaşmak genellikle zor ve maliyetlidir. Bilgiyi oluşturacak miktarda enformasyon elde etmek, bu enformasyonları oluşturmak için gerekli veri toplamak ve doğru şekilde analiz etmek, tüm bunları doğru şekilde ve yeterince hızlı gerçekleştirmek bilgiyi elde etmek için ihtiyaç duyulan ana unsurlardandır. Açından baktığımızda doğru bilginin değerini daha net görebiliriz. Aynı zamanda doğru bilgiye ulaşmak için veri kaynaklarını iyi seçmenin önemi de oldukça açıktır.

Bilgiyi elde eden kişi bu gücü kullanmak isteyecektir. Bu da hiyerarşideki en üst seviye olan bilgelik aşamasını göstermektedir. Bilgelik, eldeki bilgilerin harmanlanarak karar verme aşamalarında kullanılmasını

içermektedir. Genellikle özel sektör ya da kamu yöneticilerinin bulundukları seviye bu aşamadır. Yöneticiler çeşitli kanallardan elde ettikleri bilgileri kullanarak yönetme sorumluluğunun taşıdıkları birimler için çeşitli kararlar alırlar. Eğer bir yönetici yeterli ya da doğru bilgiye sahip değilse doğru kararlar alamayabilir. Bilgelik hiyerarşisi, verinin yolculuğunun yanı sıra her bir aşama için ihtiyaç duyulanı da göstermektedir. Herhangi bir aşamada eksiklik ya da ihtiyaç duyulan kaliteye ulaşamama durumu olması halinde sonraki aşamalar risk altında bulunacaktır. Bu da günümüzde verilen neden bu kadar önemli olduğunu bize göstermektedir.

Verinin yolculuğunda, bilgisayar programcılarına çok fazla iş düşmektedir. Verinin toplanması, analiz edilmesi, saklanması ve enformasyon ve bilgi üretme süreçlerinin sağlıklı şekilde gerçekleştirilmesi bilgisayar programcılarının başarısı ile ilişkilidir. Bu ders kapsamında verinin saklanması özelinde bazı çalışmalar yapacak olsak da bilgisayar programcılarının genelinde verinin yolculuğuna hizmet ettiğimizi akıldan çıkartmamak gerekir.

Bir sonraki alt başlık içerisinde veri tipleri ve bunların öne çıkan özelliklerinden bahsedeceğiz.

1.3. Veri Tipleri

Bir Microsoft Word dosyası açıp içerisine herhangi bir içerik eklerken bu içeriğin türü hakkında uzun uzadıya düşünmeyiz. Çünkü bu bir dokümandır. Nihayetinde bunun bir pdf dosyası olmasını ya da kağıda basılarak doküman olarak kullanılmasını bekleriz. Ancak bilgisayar programları söz konusu olduğunda veri, bilgisayar için çok daha farklı anlamlar taşır. Bir veri tabanı ile çok fazla miktarda veriyi saklanabilir ancak aynı zamanda saklanan verinin ihtiyaç duyulduğu anda yüksek performansla geri getirilmesinden de veri tabanı sorumludur. Bu sebeple veriyi kaydetmeden önce verinin yapısı hakkında bilgi sahibi olmak gerekir. Bu noktada da karşımıza veri tipleri çıkmaktadır.

Çeşitli veri tabanı yönetim sistemlerinde ve bilgisayar programlarında benzer ancak farklı isimlerle anılsa da temelde 4 ilkel veri tipi bulunmaktadır. Bu 4 veri tipini incelemmeden önce bilgisayarın nasıl çalıştığına biraz daha yakından bakmalıyız. Bilgisayarların 0 ve 1'leri kullanarak çalıştığı gerçeği yaygın olarak bilinmektedir. Ancak bu gerçekte ne anlama geliyor? 0 ve birden kasıtlı ne nedir ve bu sayılar nasıl veri ve işlemlere dönüştürülebiliyorlar? Bilgisayarların elektronik birer cihaz olduklarını düşünürsek, veri ve işlemlerin elektronik temsillerinin elde edilmesi gerektiğini de kolaylıkla görebiliriz. Bu noktada çok temel bakarsak birin elektrik olması durumunu, sıfırın ise elektrik olmama durumunu temsil ettiğini söyleyebiliriz. Yalnızca varlık durumunu inceleyerek 0 ve 1 rakamına elde etmek kolay bir yaklaşım ancak yalnızca bu iki sayıyı kullanarak geri kalan tüm veri ve işlemleri temsil etmek kuvvetli dönüşüm algoritmalarını gerektirmekte. Bilişim dünyasında 0 ya da 1 kullanılarak oluşturulmuş her bir atomik kayda bit büyüklüğünde kayıt adı verilmektedir. 8 tane bitin 1 araya gelmesiyle 1 bayt büyüklüğünde veri elde etmekteyiz. Daha büyük kapasite birimleri için 1 önceki birimin 1024 katını ele almamız gerekiyor. Sırasıyla bu kapasite birimlerini kilobayt, megabayt, gigabayt, terabayt vd. olarak bilmekteyiz (Akadal, 2021).

Birim	Değeri
1 Bit	0 ya da 1
1 Bayt	8 bit
1 Kilobayt (KB)	1024 bayt
1 Megabayt (MB)	1024 Kilobayt
1 Gigabayt (GB)	1024 Megabayt
1 Terabayt (TB)	1024 Terabayt
1 Petabayt (PB)	1024 Terabayt
1 Eksabayt (EB)	1024 Petabayt
1 Zettabayt (ZB)	1024 Eksabayt
1 Yottabayt (YB)	1024 Zettabayt

Tabloyu incelediğinizde birçok kapasite biriminin birbirine dönüştürülmesi için kullanılabilecek bir cetvel ile karşılaşacaksınız.

ÖNEMLİ

500 gigabayt kapasiteli bir hard diskin gerçekte 500 gb kapasiteye sahip olmadığını fark ettiniz mi? Bunun sebebi disk kapasitesini 500 gigabayt yerine 500 milyar bayt büyüklüğüne sahip olmasıdır. Hesaplama esnasında her bir aşamada kapasite 1024'e bölündüğü için gigabayt mertebesine geldiğinde kayıpla karşılaşmaktayız. Lütfen bilgisayarınızda bu durumu inceleyiniz. 500 GB büyüklüğünde bir disk muhtemelen bilgisayarınız yaklaşık 465 GB kapasiteli olarak görecektir.

Sabit diskin yanı sıra RAM olarak da bilinen elektronik saklama birimi bellek içerisinde de aynı kapasite birimleri kullanılmaktadır. Bu saklama birimi elektronik ve hızlı olduğu için genellikle bilgisayar tarafından anlık olarak kullanılır. Bilgisayar programcılığı ve veri tabanı ile ilgili yaptığımız işlerde bellek üzerindeki kapasite harcamaları bizim için performansı etkileyen en önemli noktalardan biri olacaktır. Bu sebeple veri saklamak için seçtiğimiz veri tipinin hangisi olduğu bellek kapasitesini dolayısıyla bilgisayarın kapasitesini ve yine dolayısıyla oluşturduğumuz bilgisayar programının performansını doğrudan etkileyecektir. Bu sebeplerle veri tipleri hakkında bilgiye sahip olmamız ve saklayacağımız veri için en uygun veri tipini seçmemiz oldukça önemlidir.

Dört temel veri tipini; karakter (character, char), tam sayı (integer, int), ondalıklı sayı (float) ve ikili karar değişkeni (boolean) olarak ele alabiliriz (Öztürk, 2014). Tüm veri tipleri için altı üstü ümitler bulunmaktadır. Ancak zaman zaman bu limitler yeterli olmadığı için bu ilkel veri tiplerinin daha yüksek kapasiteli halleri de bulunmaktadır. Örneğin tam sayılar 32 bit kapasiteye ihtiyaç duyarken, daha büyük sayıları kaydedebilmek için 64 bit kapasiteye ihtiyaç duyan long adında bir tam sayı veri tipi de bulunmaktadır.

Karakter veri tipi metinleri saklamamızı sağlayan atomik boyuttaki veri tipidir. Sekiz bit yani 1 bayt kapasiteye sahiptir. En fazla 256 farklı değer alabilir. Bu değer 2^8 hesaplaması ile elde edilir. Bir metnin kaydedilebilmesi için birçok karakter içeren bir veri tipine ihtiyaç duyulmaktadır. Karakter katarı (string) veri tipi bu ihtiyacı karşılamaktadır ancak bu veri tipi ilkel veri tipleri arasında sayılmamaktadır. Dolayısıyla bu veri tipini ayrıca inceleyeceğiz.

Bir diğer temel veri tipi tam sayıdır. Integer olarak da bilinen bu veri tipi, bilgisayarın sahip olduğu işlemci kapasitesine bağlı olarak değişmekle birlikte günümüz teknolojisinde genellikle 32 bit kapasiteye sahiptir. Bu kapasite tam sayı veri tipine sahip bir değişkenin en fazla 4.294.967.296 (2^{32}) farklı değer alabilmesini mümkün hale getirmektedir. Bir değişkenin gerek bilgisayar programı içerisinde gerekse veri tabanı içerisinde tam sayı olarak tanımlanması kararı oldukça önemlidir. Çünkü verilen bu karar neticesinde ilgili değişken ya da veri tabanı alanına yalnızca tam sayıların kaydedilmesi mümkün olacaktır. Kullanılan programlama dili ya da veri tabanına bağlı olarak ondalıklı bir sayının tam sayı veri tipine sahip bir değişkene kaydedilmek istenmesi durumunda, ya hata alınacak ya da ondalıklı sayı tam sayı ya dönüştürülerek kaydedilecektir. Bodrum yazılımının çalışmasında bir hata oluşması ne ya da Veri bütünlüğünün bozulmasına sebep olabilir. Bu sebeple tam sayı veri tipi seçilirken tüm koşullarda değerlerin tam sayı olarak alınacağından emin olunmalıdır. Tam sayı veritipi bazı özel alt veri tiplerine sahiptir. Örneğin tam sayı veri tipine sahip bir değişken -2.147.483.648 ile +2.147.483.648 aralığında bir de yer alırken, işaretli (signed) tam sayı veri tipi 0 ile +4.294.967.296 aralığında bir değer alacaktır. Kullandığımız bilgisayar 32 bitlik bir işlemciye sahip olsa da 16 bitlik bir tam sayı değişkenine ihtiyaç duyabiliriz. Bu durumda kısa (short) tam sayı alt veri tipini kullanabiliriz. Bazı durumlarda 32 bitlik tam sayı veri tipine ulaşmak için uzun (long) tam sayı veri tipi kullanılması gerekebilir. Ayrıca bu alt veri tiplerinden bazılarının kombinasyonlarını kullanmak da mümkündür.

ÖNEMLİ

Burada bazı kelimelerin İngilizce karşılıklarını veriyor olmamız yalnızca bu kelimelerin İngilizcelerini de öğrenmeniz için değil; aynı zamanda komut olarak kullanmanız gerektiğinde doğru kelimeyi bulabilmeniz içindir. Lütfen her bir anahtar kelimeyi, İngilizce karşılığı ile birlikte öğreniniz.

Ondalıklı sayı (float) veri tipi $3,4 \times 10^{-38}$ ile $3,4 \times 10^{38}$ aralığında değer alabilir. Bu veritipi bellekte 32 bit yer kaplayacaktır. Bu kapasitenin yeterli olmaması durumunda daha yüksek kapasiteye sahip olan double veri tipi kullanılabilir. Bu veri tipi bellekte 64 bit yer kaplamaktadır. Böylece kapasite $1,7 \times 10^{-308}$ ile $1,7 \times 10^{308}$ aralığında ulaşır. İşaretsiz (unsigned) etiketi kullanılması durumunda kapasiteden 0'dan

başlayacak şekilde faydalanılabilir. Matematikten bilindiği üzere ondalıklı sayılar sürekli ve her iki sayı arasında sonsuz ondalıklı sayı bulunmaktadır. Ancak bilgisayar belleği bitleri kullanarak çalıştığı için sonsuz tane sayıyı temsil edemez. Bunun yerine mümkün en yüksek kesinlikle ondalıklı sayıyı taklit eder. Bu durum çok hassas işlem yapılmadığı sürece genellikle herhangi bir hataya sebep olmaz.

Sonuncu veri tipimiz mantıksal (boolean) olarak karşımıza çıkmaktadır. Mantıksal veri tipi yalnızca sıfır ya da bir değerini alabilir. Aynı zamanda bu değişkenin aldığı yer doğru (true) ya da yanlış (false) olarak da adlandırılır. Mantıksal veri tipi bellekte bir bayt yer kaplar. Her ne kadar bir mantıksal değişkenin değerini saklamak için bir bitlik yere ihtiyacımız olsa da, bellek üzerinde adreslenebilir en küçük kapasite bayt olduğu için, her bir mantıksal değişken için bellekte 1 bayt yer ayrılır.

Bu noktaya kadar öğrendiğimiz şeyleri gözden geçirmekte fayda var. Bilgisayarın elektriğin olup olmaması durumuna göre sıfır ve birleri kullanarak verileri sakladığını ve işlediğini öğrendik. Her bir sıfır ve bir, bit adını verdik ve sekiz bit ile bir baytlık kapasiteyi elde ettiğimizi gördük. Ayrıca baytları kullanarak da temel veri tiplerini elde etmiş olduk. Bilgisayar dünyasında da verinin yolculuğundakine benzer şekilde her adımın bir önceki adımın üzerine inşa edildiğini görmekteyiz. Daha becerikli veri tipleri için, yine ilkel veri tiplerinden faydalanacağız. Sonrasında da bu veri tiplerini kullanarak kendi bilgisayar programlarımızı elde edeceğiz. Bu ders kapsamında veri tabanları dahilinde kullanılan veri tipleri ile ilgileniyoruz. Burada ilkel veri tiplerine ek olarak verilmesi gereken en önemli ve sık kullanılan veri tipi metinlerin saklanması için kullanılan veri tipleridir. Veri tabanı yönetim sistemleri, performansı artırmak ve kullanıcı işlerini kolaylaştırmak için çok sayıda, becerikli veri tiplerine sahip olabilir. Ancak tüm veri tabanı sistemlerinde karşılaşacağımız ve metin saklamaya yarayan iki veri tipi bulunmaktadır. Bunlardan birincisi Varchar olarak karşımıza çıkar. Varchar, kısa metinlerin saklanması için kullanılan, en fazla 255 karakter uzunluğunda metin saklayabilen bir veri tipidir. Ad, soyad, adres ve benzeri verilerin saklanması için genellikle bu veri tipi kullanılır. Metin saklamak için kullanılan diğer bir veri tipi ise Text'tir. Text veri tipi, herhangi bir kısıtlamaya bağlı olmaksızın istenilen uzunlukta metinleri saklayabilir. Ancak bu veri tipi neredeyse her bir hücre için büyük miktarda yük taşıma potansiyeline sahip olduğundan dolayı hantal olarak görülebilir. Dolayısıyla çok büyük metin saklama ihtiyacı bulunmadı sürece Text veri tipi tercih edilmez.

Bilgisayar programlamada ve veri tabanı tasarımında eldeki verinin hangi veri tipine sahip olduğunu bilmek önemlidir. Bunun yanında, aynı veri tipi kullanılacak değerler arasında farklı özelliklerde veri bulunuyorsa, kapsayıcı olan veri tipinin seçilmesi önemlidir. Örneğin bir sütun hem tam sayı hem de ondalıklı sayıları içeriyorsa, bu sütun için ondalıklı veri tipinin seçilmesi daha uygun olacaktır. Eğer bu sütun için tam sayı veri tipi seçilirse, ondalıklı sayılar için bir veri kaybı yaşanacaktır. Ancak ondalıklı sayı veri tipi seçildiğinde de tam sayılar bu veri tipinde saklanabilir olacaklardır. Mantıksal veri tipi ile saklanabilecek bir değişken, aynı zamanda tam sayı veri tipi ile de saklanabilir. Ancak bu durumda bir bayt kapasite harcanacakken, 64 bit yani 8 baytlık bir veri kapasitesinin harcanması söz konusu olacaktır. Bahsi geçen kapasite büyüklükleri oldukça küçük görünse de çok yüksek miktarda veri ve girdi-çıkış işlemi ile uğraşılırken bu ayrıntılar performansa büyük etkilerde bulunacaktır.

1.4. Veri Dosyaları

Microsoft Word yazılımı ile Windows işletim sistemine sahip bilgisayarlarda bulunan Not Defteri yazılımını hiç karşılaştırdınız mı? Okumaya devam etmeden önce bunun üzerine biraz düşünmeniz ve hatta bir göz atıp geri dönmenizi öneririm. Not Defteri, metni en sade haliyle gösteren; herhangi bir biçim özelliği barındırmayan ve metinleri bu şekilde içeren dosya türlerini rahatlıkla okuyabildiğimiz yazılımdır. MS Word ise baştan aşağı metni biçimlendirmekle ilgilidir. Yazdığınız bir yazıda harflerin ne olduğundan öte boyutunun, yazı fontunun, sayfanın neresinde bulunduğu, yazı ve arka plan renklerinin ne olduğunun kaydedilmesi söz konusudur.



Untitled - Notepad

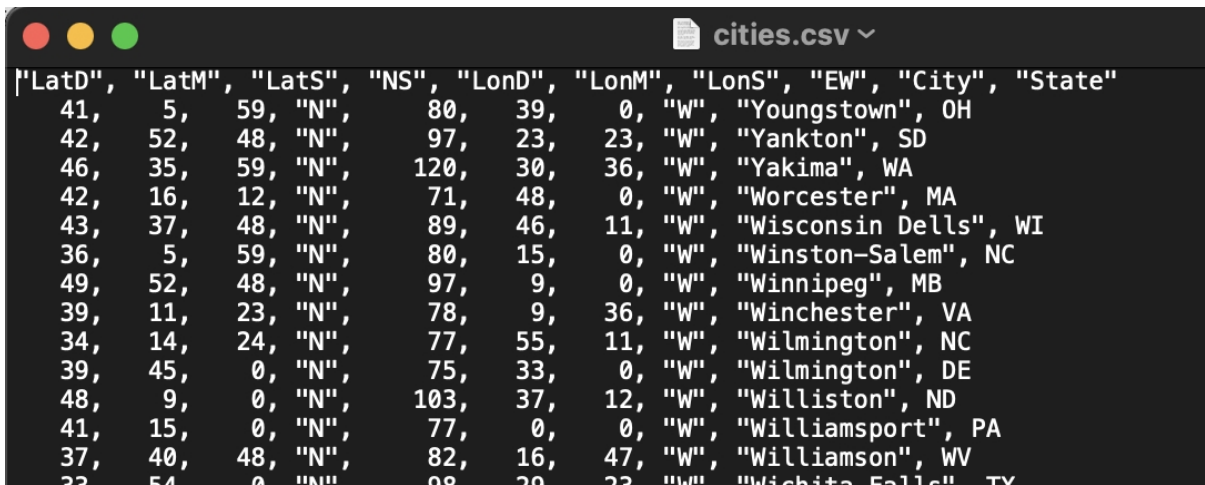
File Edit Format View Help

Hello, world!

Burada dikkatli arkadaşlarımız şunu savunacaktır: Not Defteri de bir metni belirli bir yazı karakterinde, büyüklükte, renkte ve biçimde göstermekte. Bu eleştiri haklı olmakla birlikte teknik yanıt şudur ki; Not Defteri metnin biçimini metin ile birlikte kaydetmez. Not Defteri yalnızca bir görüntüleyicidir ve bir metni görüntülemek için onu belirli bir biçimde göstermek gerekir. Daha net anlaşılabilmesi için bir örnek uygulamanızı rica edeceğim. Bilgisayarınızın masaüstünde a.txt ve b.txt adlı iki dosya oluşturun. Bu dosyalar eğer siz bir değişiklik yapmadıysanız varsayılan olarak Not Defteri ile açılacaktır. İki dosyanın içinde de farklı birer cümle yazarak kaydedip kapatın. Sonrasında a.txt dosyasını açıp Not Defteri'nin biçim (format) menüsünden yazının görünümünü değiştirin. Ardından kaydedip dosyayı kapatın. Sonrasında b.txt dosyasını açtığınızda bu dosyadaki metnin de öbür dosyadaki metinle aynı biçimde yani az önce sizin belirlediğiniz özelliklerde gözüktüğünü göreceksiniz. Bunun sebebi gerçekte metnin özelliklerini değil, Not Defteri'nin görüntüleme özelliklerini değiştirmiş olmanızdır. Not Defteri artık herhangi bir biçimsiz metni sizin tanımladığınız özelliklerle gösterecektir. Gerçekte metnin herhangi bir özelliği yoktur. Daha önce herhangi bir dilde kodlama yaptıysanız lütfen ilgili kod editörünü açarak inceleyiniz. Not Defteri gibi çalıştığını, yalnızca daha yetenekli bir görüntüleyici olduğunu göreceksiniz. Bir adım daha öteye götürürsek; yazdığınız herhangi bir dildeki kodu Not Defteri ile açmayı deneyin. Kodları düzgünce görüntüleyebildiğinizi fark edeceksiniz. Çünkü bilgisayarlar için komutların (metnin) biçimi değil ne olduğu önemlidir. Bu yüzden MS Word (docx uzantısı) ile kod yazamayız.

Veri dosyaları için de benzer durum geçerlidir. Bir veri kümesini bir dosyaya kaydettiğimizde bu dosya biçimsiz olarak kaydedilir. Böylece yine biçimden öte içerik ile ilgilenmiş oluruz. Burada MS Excel, SPSS gibi yazılımlar istisna oluşturuyorlar. Bu yazılımlar ile kullanılan veriler bir biçim ile kaydedilebilir. Bunun sebebi veriyi işlerken yine bu yazılımlardan faydalanabilmektir. Eğer platform ve dil bağımsız bir veri kaydı yapmak istiyorsak bu noktada çok sık kullanılan 3 temel yapı karşımıza çıkmaktadır: CSV, JSON ve XML (Han & diğ., 2011).

CSV (Comma Separated Values – Virgülle Ayrılmış Değerler), sıklıkla kullanılan bir veri kümesi uzantısıdır. Genelde bilgisayarlar varsayılan olarak bu dosyaları MS Excel ile açarlar. Bunun sebebi, CSV dosyaları MS Excel tarafından kolayca işlenebileceği için MS Excel'in kurulum esnasında bu ayarı bilgisayara otomatik yapmasıdır. Bu dosya türünde her satırda yer alan birbirinden ayrı veriler virgül ya da noktalı virgül ile birbirinden ayrılır. MS Excel ya da CSV dosyasını okuduğumuz herhangi bir yazılım, her bir virgüllü ayraç kabul eder ve her bir virgülden sıradaki veri bloğuna geçtiğini bilir. Bu sebeple MS Excel yazılımı bir CSV dosyasını çalıştırdığında onu rahatlıkla hücrelere yerleştirebilir. Her satırı bir satır, her virgülle ayrılmış ifadeyi ise bir hücre içerisine koyar. Böylece dosya klasik bir MS Excel dosyasına benzer. Asıl farkı görmek, dosyanın ham haline bakmakla mümkündür. İnternette bir CSV dosyası indirin ve önce MS Excel, sonra da Not Defteri'yle açarak görüntüleyin. Sonra da herhangi bir MS Excel (xlsx uzantılı) dosyayı Not Defteri'yle görüntülemeyi deneyin. Lütfen farkı gözlemleyin ve yorumlayın. İpucu: Google'da "example filetype:xlsx" ve "example csv" (tırnaksız) aramaları yaparak örnek dosyalar edinebilirsiniz.



"LatD"	"LatM"	"LatS"	"NS"	"LonD"	"LonM"	"LonS"	"EW"	"City"	"State"
41,	5,	59,	"N",	80,	39,	0,	"W",	"Youngstown",	OH
42,	52,	48,	"N",	97,	23,	23,	"W",	"Yankton",	SD
46,	35,	59,	"N",	120,	30,	36,	"W",	"Yakima",	WA
42,	16,	12,	"N",	71,	48,	0,	"W",	"Worcester",	MA
43,	37,	48,	"N",	89,	46,	11,	"W",	"Wisconsin Dells",	WI
36,	5,	59,	"N",	80,	15,	0,	"W",	"Winston-Salem",	NC
49,	52,	48,	"N",	97,	9,	0,	"W",	"Winnipeg",	MB
39,	11,	23,	"N",	78,	9,	36,	"W",	"Winchester",	VA
34,	14,	24,	"N",	77,	55,	11,	"W",	"Wilmington",	NC
39,	45,	0,	"N",	75,	33,	0,	"W",	"Wilmington",	DE
48,	9,	0,	"N",	103,	37,	12,	"W",	"Williston",	ND
41,	15,	0,	"N",	77,	0,	0,	"W",	"Williamsport",	PA
37,	40,	48,	"N",	82,	16,	47,	"W",	"Williamson",	WV
22,	54,	0,	"N",	88,	20,	22,	"W",	"Wichita Falls",	TX

Bir diğer veri saklama biçimi, verinin yazılım ve platformdan bağımsız şekilde taşınabilmesini sağlayan, XML (The Extensible Markup Language) olarak bilinen dosya türüdür. XML içerisinde veri tag adı verilen yapılar içerisinde tutulur. Ağaç yapısına benzer şekilde hiyerarşik şekilde saklanan veri XML entegrasyonu bulunan tüm yazılımlar tarafından rahatlıkla okunarak yazılımın kendi veri saklama biçimine dönüştürülebilir. Ayrıca herhangi bir yazılım, mevcut veriyi XML formatında biçimsiz bir dosya içerisine saklayarak XML dosyasını üretmiş olur. Nihayetinde yazılımlar arası veri aktarımı için yakın geçmişe kadar

yoğun şekilde kullanılan bir dosya türüdür. Son zamanlarda JSON dosya türünün yaygınlığının artması sebebiyle kullanım oranı azalmaya başlamıştır.

```
<?xml version="1.0" encoding="iso-8859-8" standalone="yes" ?>
<CURRENCIES>
  <LAST_UPDATE>2004-07-29</LAST_UPDATE>
  <CURRENCY>
    <NAME>dollar</NAME>
    <UNIT>1</UNIT>
    <CURRENCYCODE>USD</CURRENCYCODE>
    <COUNTRY>USA</COUNTRY>
    <RATE>4.527</RATE>
    <CHANGE>0.044</CHANGE>
  </CURRENCY>
  <CURRENCY>
    <NAME>euro</NAME>
    <UNIT>1</UNIT>
    <CURRENCYCODE>EUR</CURRENCYCODE>
    <COUNTRY>European Monetary Union</COUNTRY>
    <RATE>5.4417</RATE>
    <CHANGE>-0.013</CHANGE>
  </CURRENCY>
</CURRENCIES>
```

JSON (Javascript Object Notation), insanlar için de okuma ve yazması oldukça kolay olan, Javascript dili içerisinde kullanılırken şimdi tamamen farklı diller için de kullanılabilen görece yeni bir veri saklama biçimidir. XML'e rakip olarak nitelendirebileceğimiz bu biçimin XML'e göre çeşitli avantajları bulunmaktadır. XML görseline bakarsanız her bir satır için her bir sütun adının yeniden yazıldığını görebilirsiniz. Bu durum, çok büyük veri kümeleri için gereksiz bir hafıza harcamasına sebep olmaktadır. Verilen XML örneğinde verinin hacminden daha fazla XML yapısından kaynaklanan bir hacim olduğu görülebilir. JSON için bu durum geçerli değildir. Dolayısıyla XML, JSON'a göre oldukça hantal kalmaktadır. İkinci büyük avantaj ise yazım kolaylığıdır. JSON kolaylıkla elle yazılabilir ancak XML için tüm sütun (değişken) adlarının akılda tutulması ve her satır için aynı ifadenin hatasız şekilde tekrar edilmesi gerekir. Bu gibi avantajlar sebebiyle son yıllarda JSON formatında veri aktarımının çok daha sık kullanıldığını söyleyebiliriz.

```
{
  [...]
  "timestamp": 1434199748,
  "source": "USD",
  "quotes": {
    "USDAUD": 1.293328,
    "USDCAD": 1.232041,
    "USDCHF": 0.928035,
    "USDEUR": 0.888104,
    "USDGBP": 0.642674,
  }
}
```

Veriyi saklamanın çeşitli yol ve yöntemleri mevcut. Programlama dillerinden hangisinin en iyi olduğu tartışılmayacağı gibi, bu yöntemlerden birinin en iyi olarak seçilmesi de uygun değildir. Çünkü her bir yöntem, farklı şartlar altında daha iyi olabilir. Eğer amaç, verinin ham olarak saklanmasıysa ve platform bağımsız olması isteniyorsa CSV dosya türü kullanılabilir. Eğer iki sistem arasında anlık veri aktarımı yapılacaksa tercih JSON'dan yana kullanılabilir. Platformlar arası veri aktarımının oldukça önemli olduğu bir dönemdeyiz. Diğer sistemlerle etkileşimli olanlar bir şekilde ellerindeki veriyi senkronize etmek ve yapılan işlemin bilgisini diğer paydaşlara hemen aktarmak isterler. Bir bankacılık mobil uygulamasını ele alalım. Bu mobil uygulamanın yazılımı, banka içerisinde kullanılan yazılımdan tamamen bağımsızdır. Ancak kullanılan verinin aynı olması ve hatta daha da önemlisi, eşlenik (senkronize) olması gerekmektedir. Bu sebeple bu tarz mobil uygulamalar Web servis adını verdiğimiz yapılar ile ana sisteme bağlanırlar. Web servisler, iki yazılımın birbiriyle iletişime geçmesini sağlayan, web üzerinden çalışan, yetkilendirmeli ya da herkese açık olarak kullanılabilen entegrasyon yöntemleridir. Veri aktarımı için genellikle JSON biçimini kullanırlar. Böylece web servisin çalıştırıldığı merkezi sistem ve bundan faydalanan diğer sistemlerin aynı ya da benzer

altyapıya sahip olmalarına gerek kalmamaktadır. Bir web sunucusu üzerinde hayata geçirilen web servis, android, ios ya da diğer mobil işletim sistemleri tarafından kolaylıkla işlenebilir. Benzer faydalar nesnelerin interneti (IOT – internet of things) için de geçerlidir. Çok sayıda internete bağlı cihazlar, merkezi bir sistemle veri alışverişinde bulunmalıdırlar. Bunun için genellikle merkezi sistem üzerinde web servis hayata geçirilir. Her bir cihaz ise bu web servis üzerinden ve JSON gibi basit ve hafif veri yapıları ile veri aktarımı gerçekleştirir. Günümüzde hem IOT hem de mobil uygulama geliştirme teknolojilerinde bu kavramlarla sıklıkla karşılaşılmaktadır.

Bölüm Özeti

Veri tabanı, büyük veri kümelerinin daha etkili işlenmesi için hazırlanmış yapılardır. Öncelik veriyi saklamaktan öte veriyle ilgili işlemlerde yüksek performans elde edebilmek olduğu için; performansı etkileyebilecek her türlü ayrıntıya dikkat edilmektedir. Veri tipleri de bilgisayar belleğinin en iyi şekilde kullanılabilmesi için bilinmesi gereken önemli konulardan biridir.

Dört temel veri tipini ele aldık: Tam sayı, ondalıklı sayı, metin ve mantıksal. Sayılar için tam sayı ve ondalıklı sayı olmak üzere iki veri tipini ihtiyaca göre seçerek kullanabiliriz. Bir tam sayı, ondalıklı sayı tipi kullanılarak kaydedilebilir. Bu çok uygun olmasa da veri kaybına sebep olmaz. Ancak ondalıklı bir sayıyı tam sayı tipinde kaydetmeye çalışmak, veride kayba sebep olacaktır. Benzer şekilde, mantıksal durum kaydedilebilen boolean veri tipi, tam sayı olarak da kaydedilebilir. Gereksiz bir kapasite kullanımı olsa da veri bütünlüğü korunacaktır. Ancak bir tam sayı mantıksal veri tipi olarak kaydedilemez. Metin veri tipi ise en kapsayıcı olarak karşımıza çıkmaktadır. Mantıksal, tam sayı, ondalıklı sayı da herhangi bir metin bu veri tipi ile kayıt altına alabilir. Ancak bu tip, aynı zamanda bilgisayar tarafından işlenmesi en zor olan, veri tabanı girdi-çıkı işlemleri sık yapılan durumlar için performans düşürücüdür. Bu sebeple zorunlu kalınmadığı sürece metin veri tipi yerine uygun olan başka bir veri tipi seçilir.

Her veri tipi bellekte farklı şekillerde saklandığı için eldeki verinin hangi veri tipi kullanılarak saklanacağını belirlemek oldukça önemlidir. Örneği sayısal bir içeriği metin veri tipiyle de saklamak mümkündür ancak uygun değildir. Diğer yandan ondalıklı bir sayı bir seviyeye kadar hassaslıkta saklanabilir. Mükemmel bir hassasiyet için metin tipi kullanmak gerekebilir. Bu kararlar ders içerisinde bol örnek ile ele alınarak işlenecek ve bununla birlikte deneyim oluşturulacaktır. Ders içerisindeki tüm konular birbirinin devamı niteliğinde olduğu için sırayla takip etmeniz oldukça faydalı olacaktır.

Verilerin saklanması kadar iletilmesi de önemlidir. Bu noktada karşımıza sıklıkla CSV, XML ve JSON formatları çıkmaktadır. CSV bir dosya formatıdır ve genellikle MS Excel dosyasıyla açılabilir. XML ve JSON ise daha çok sistemler arası veri aktarımı için hazırlanan web servisler tarafından kullanılan bir veri saklama biçimidir.

Kaynakça

Akadal, E. 2020. Veritabanı Tasarlama Atölyesi. Türkmen Kitabevi, İstanbul.

Akadal, E. 2021. Veri saklama yöntem ve uygulamaları. Tıp Bilişimi. İstanbul University Press, İstanbul. DOI: 10.26650/B/ET07.2021.003.03

Aksoy, T., Çelik, S. & Gülseçen, S. 2020. Data pre-processing in text mining. Who Runs the World: Data. Istanbul University Press, İstanbul. DOI: 10.26650/B/ET06.2020.011.07

Han, J., Haihong, E., Le, G., & Du, J. 2011. Survey on NoSQL database. Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011, 363–366. <https://doi.org/10.1109/ICPCA.2011.6106531>

Jennifer Rowley. 2007. The wisdom hierarchy: representations of the dikw hierarchy. Journal of information science, 33(2):163–180.

Öztürk, S. M., 2014. Veri Yapıları. Karabük Üniversitesi.