İstanbul Üniversitesi
Açık ve Uzaktan Eğitim Fakültesi

Veri Tabanı Tasarımı

DOÇ. DR. EMRE AKADAL

İÇİNDEKİLER

1. VERİ

Bölümle İlgili Özlü Söz

Eğitimdir ki, bir milleti ya özgür, bağımsız, şanlı, yüksek bir topluluk halinde yaşatır; ya da esaret ve sefalete terk eder.

Mustafa Kemal Atatürk

Kazanımlar

1. Verinin önemini açıklayabilir.

2. Verinin dönüştürülmesi aşamalarına hakim olur.

3. Örnek verisetleri elde etmeyi öğrenir.

4. Veriyi üzerinde çalışmaya uygun hale getirir.

5. Verinin çeşitli biçimlerdeki hallerini tanıyabilir.

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 değişle 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 oluşturmasıdır. 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 önemsenmeyecek 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 bilgeliğe 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 parçacıklarıdır. Daha kapsamlı karar almak için enformasyonun bilgiye dönüşmesi gerekir. Örneğin; bir işletmede personelin işe geliş 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 edeceğiniz üzere enformasyonların bir araya gelmesi sonucunda elde edilebilen bir olgudur. Bu akşama 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 bilge erişilebilir. Yine fark edeceğiniz 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. Açıdan 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 incelemeden 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 diski 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 yazılımın 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, işaretsiz (unsigned) 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 kullanılabilmektedir. 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 süreklidir 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.

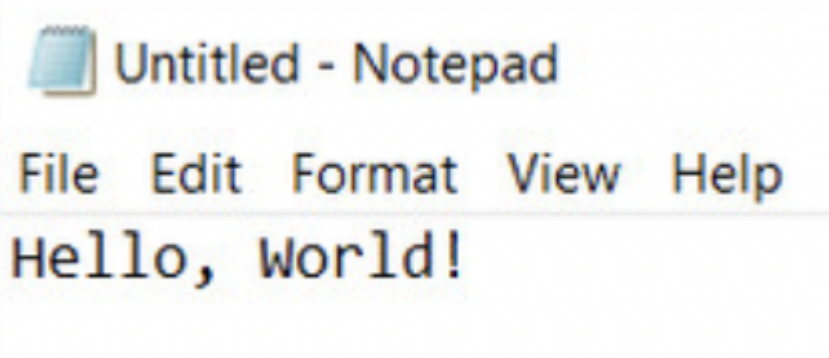
Bu noktaya kadar öğrendiğimiz şeyleri gözden geçirmekte fayda var. Bilgisayarın elektriğin olup olmaması durumuna göre sıfır ve birleri kullanarak verileri sakladığını ve işlediğini öğrendik. Her bir sıfır ve bire, bit adını verdik ve sekiz bit ile bir baytlık kapasiteyi elde ettiğimizi gördük. Ayrıca 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-çıktı işlemi ile uğraşılırken bu ayrıntılar performansa büyük etkilerde bulunacaktır.

1.4. Veri Dosyaları

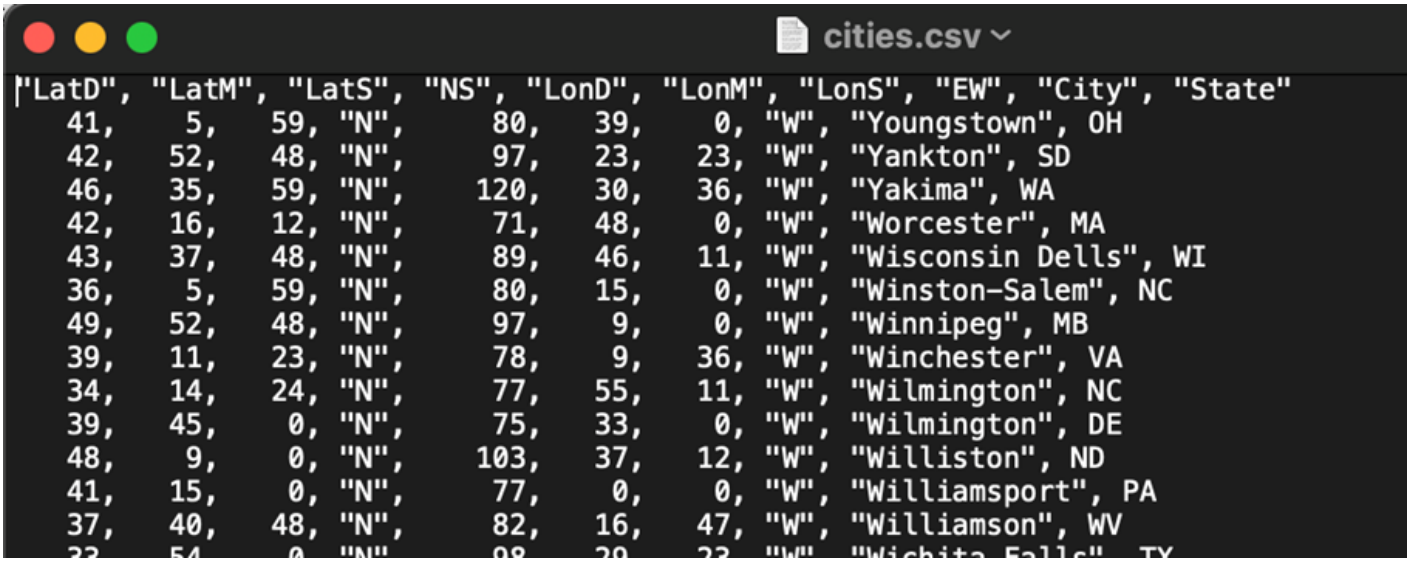
Microsoft Word yazılımı ile Windows işletim sistemine sahip bilgisayarlarda bulunan Not Defteri yazılımını hiç karşılaştırdınız mı? Okumaya devam etmeden önce bunun üzerine biraz düşünmeniz ve hatta bir göz atıp geri dönmenizi öneririm. Not Defteri, metni en sade haliyle gösteren; herhangi bir biçim 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 bulunduğunun, yazı ve arka plan renklerinin ne olduğunun kaydedilmesi söz konusudur.



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 virgülü ayraç kabul eder ve her bir virgülde 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. İnternetten 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.


cities.csv

```
"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
```

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.

Veriyi saklamanın çeşitli yol ve yöntemleri mevcut. Programlama dillerinden hangisinin en iyi olduğu tartışılamayacağı 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-çıktı 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 saklanılması 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.

Ünite Soruları

Soru-1 :

Rowley Bilgelik Hiyerarşisi'ndeki sıralamayı aşağıdan yukarıya doğru belirtiniz.

(Çoktan Seçmeli)

(A) Veri – Enformasyon – Bilgi – Bilgelik

(B) Enformasyon – Veri – Bilgi – Bilgelik

(C) Bilgi – Veri – Enformasyon – Bilgelik

(D) Bilgi – Enformasyon – Bilgelik – Veri

(E) Bilgelik – Bilgi – Enformasyon – Veri

Cevap-1 :

Veri – Enformasyon – Bilgi – Bilgelik

Soru-2 :

Termometre tarafından ölçülen sıcaklık miktarı nedir?

(Çoktan Seçmeli)

(A) Veri

(B) Enformasyon

(C) Bilgi

(D) Bilgelik

Cevap-2 :

Veri

Soru-3 :

Enformasyonları elde ederek ulaştığımız olgu nedir?

(Çoktan Seçmeli)

(A) Veri

(B) Bilgi

(C) Bilgelik

Cevap-3 :

Bilgi

Soru-4 :

Bilgisayarlar için her bir 1 ya da 0 değerinin boyutu nasıl ifade edilir?

(Çoktan Seçmeli)

(A) Bit

(B) Bayt

(C) Kilobayt

(D) Megabayt

(E) Gigabayt

Cevap-4 :

Bit

Soru-5 :

Hangisi temel veri tiplerinden değildir?

(Çoktan Seçmeli)

(A) Ondalıklı sayı

(B) Tam sayı

(C) Mantıksal

(D) Metin

(E) Görsel

Cevap-5 :

Görsel

Soru-6 :

Biçimsiz metinleri görüntülemek için kullandığımız en temel Windows yazılımı nedir?

(Çoktan Seçmeli)

(A) MS Word

(B) MS Excel

(C) Not Defteri

(D) Powerpoint

(E) Paint

Cevap-6 :

Not Defteri

Soru-7 :

Veriyi virgüllerle ayrılmış şekilde saklayan, Not Defteri ya da MS Excel tarafından görüntülenebilen dosya türü nedir?

(Çoktan Seçmeli)

(A) XML

(B) JSON

(C) TEXT

(D) CSV

(E) Binary

Cevap-7 :

CSV

Soru-8 :

Platformlar arası veri aktarımı için hazırlanmış yazılım türü nedir?

(Çoktan Seçmeli)

(A) Web sitesi

(B) Web servis

(C) Mobil uygulama

(D) JSON

(E) PDF

Cevap-8 :

Web servis

Soru-9 :

Hafif ve hızlı veri aktarımı için kullanılan biçim hangisidir?

(Çoktan Seçmeli)

(A) Web servis

(B) XML

(C) JSON

(D) CSV

(E) MS Excel

Cevap-9 :

JSON

Soru-10 :

Veriyi “tag” yapıları kullanarak saklayan ve sunan veri biçimi nedir?

(Çoktan Seçmeli)

(A) XML

(B) JSON

(C) Mantıksal

(D) Metin

(E) CSV

Cevap-10 :

XML

---

2. VERİ SAKLAMA YÖNTEMLERİ

Bölümle İlgili Özlü Söz

Web'i henüz görmedik. Gelecekte, geçmişten çok daha büyük olacak.

Tim Berners-Lee

World Wide Web (WWW)'in Mucidi

Kazanımlar

1. Verinin elektronik ortamda saklanmasıyla ilgili bilgi sahibi olur.

2. Hiyerarşik, ağ ve ilişkisel veri tabanı kavramlarını bilir.

3. İlişkisel veri modeli ve veri tabanı kavramlarına hakim olur.

4. Veri tabanı tasarlama sürecinin gerekliliklerini bilir.

5. Veri tabanı yönetim sistemleri hakkında bilgi sahibi olur.

6. Veri tabanı yönetim sistemleri arasındaki farkları bilir.

7. SQL dili hakkında bilgi sahibi olur.

8. SQL dili ile temel komutlar yazabilir.

Birlikte Düşünelim

Çok miktarda veriye sahip organizasyonlar, aradıkları veriyi kolayca nasıl bulabiliyorlar?

İlişkisel veri modeli konsepti ilişkisel veri tabanı yönetim sistemlerine dönüşürken ne tür süreçlerden geçmişlerdir?

Neden çok sayıda ilişkisel veri tabanı yönetim sistemi ihtiyacı oluştu?

İlişkisel veri tabanı yönetiminde kullanılan dil olan SQL, neden bazı veri tabanı yönetim sistemlerinde ek özellikler kazanmaktadır? Bunun avantaj ve dezavantajları nelerdir?

Birden fazla tablo üzerinde aynı anda sorgu yapmak ve sonuçları birleşik olarak göstermek için SQL yeterli midir?

Hangi durumlarda SQL, programlama dilinin gücünden faydalanmalıdır, hangi durumlarda tek başına SQL sorgusu ile sonuca ulaşılabilir?

Yalnızca SQL kullanılarak bir bilgisayar programı yazılabilir mi?

Başlamadan Önce

Veri tabanı tasarlamak önemli ve zor bir iştir. Veri tabanıyla ve veri tabanı tasarlamakla ilgili tüm kavram ve kurallara hakim olabilirsiniz. Ancak bu bilgi, risksiz ve yüksek performansla çalışan bir veri tabanı tasarlamayı garanti etmez. İyi çalışan bir veri tabanı tasarlamak, temel terimleri bilmekten daha fazlasını gerektirmektedir.

Bu bölümde öncelikle veriyi saklamak için kullanılan teknolojilerin kronolojisinden bahsedeceğiz. Ardından 1970'te ortaya çıkan ve günümüzde hala en yoğun kullanılan veri tabanı türü olan ilişkisel veri

tabanlarından bahsedeceğiz. İlişkisel veri tabanları, önceki alternatiflerine göre birçok avantajı içerisinde barındıran ve evrensel bir kullanıcı deneyimi yaratmak için çeşitli motivasyonlarla ortaya çıkarılmış bir yapıdır. Tüm avantajlarını bölüm içerisinde teker teker inceleyeceğiz.

İlişkisel veri tabanlarını platformlarla buluşturmak, programlama dilleriyle harmanlamak ve yazılımlar üretmek için veri tabanı yönetim sistemlerine ihtiyaç duymaktayız. Önde gelen veri tabanı yönetim sistemleri ve bunların öne çıkan özelliklerini ele alacağız.

Son olarak veri tabanının iletişim şekli olan SQL'den bahsedeceğiz. SQL'in temel komutlarını öğrenecek, bir veri tabanıyla temel düzeyde iletişim kurabilecek kadar bu dili öğreneceğiz.

2.1. Veriyi Saklamak

Bir önceki bölüm içerisinde Bilgelik Hiyerarşisi yoğun şekilde ele alınmıştı. Bu hiyerarşi bize veriden bilgeliğe giden yolculuğu hem sıralama hem de hacim açısından sunmaktadır. Veri, enformasyon, bilgi ve bilgelik aşamalarında her bir aşamaya erişim için bir önceki aşamanın elde büyük miktarda olması gerekmektedir. Bilgisayar programcılığı, genellikle veriden enformasyon elde edilmesi, enformasyonun biriktirilerek yöneticilere gösterilebilmesi, farklı enformasyonların bir araya getirerek yöneticinin bilgiye erişebilmesi aşamalarında fayda sağlamaktadır. Bu tarz sistemler; enformasyon teknolojileri ve enformasyon sistemleri olarak karşımıza çıkmaktadır. Enformasyon teknolojileri, verinin enformasyona dönüştürülmesi ve bu sürecin etrafındaki diğer faaliyetlerin gerçekleştirilebilmesi için gerekli altyapı ve teknolojik olguları ifade ederken enformasyon sistemleri bu süreci gerçekleştirebilecek yazılımı ifade etmektedir.

Enformasyon ve bilgi elde edebilmek için veriye ihtiyaç duyulduğu bir gerçektir. Bununla birlikte verinin yolculuk süreci iki şekilde gerçekleşebilir. Bu iki durumu "veriden enformasyon" ve "enformasyon için veri" olarak ikiye ayırabiliriz.

Veriden enformasyon olarak anılan birinci durumda; erişilebilen tüm kaynaklardaki verilerin toplanmaktadır. Bu aşamada veriden elde edilecek enformasyona henüz karar verilmemiş olabilir. Öncelikle ulaşılabilen en fazla veri elde edilir. Enformasyona dönüştürme süreci veri toplandıktan sonra, elde edilen verinin niteliğine göre karar verilir. Toplanan veri genellikle ikincil veridir.

ÖNEMLİ

Doğrudan kendimizin topladığı veya ürettiği veriye birincil; başka kaynaklardan edinilmiş veriye ikincil veri adı verilmektedir (Çakıcı & Yılmaz, 2021).

İkincil veriyi toplamak, bir dizi uğraşıyı da beraberinde getirmektedir. Bunlardan kısaca bahsedelim:

Saklama: Her veri kümesi, düzenli şekilde kayıt altına alınmak istenildiğinde özel bir veri tabanı tasarımı gerektirir. Elde edilen veri kümeleri beraberinde veri tabanı tasarımını getirmeyebilirler. Bu durumda kişi, elde ettiği ikincil veri kümesini ayrıntılı analiz etmeli ve uygun bir veri tabanı tasarımı hazırlayarak elde ettiği veri kümesini bu tasarımı kullanarak saklamalıdır.

Entegrasyon: Farklı veri kaynaklarından gelen benzer verilerin eşleştirilerek saklanılması gerekmektedir. Aksi durumda ver kümelerini ayrı ayrı analiz etmek gerekir ki bu da istenilen bir şey değildir. Genellikle birden fazla kaynaktan veri elde edildiğinde tüm veri kümelerinin bir bütün olarak analiz edilmesi istenilmektedir. Bulunduğunuz ilin ilçeleriyle ilgili bir çalışma yapmak istediğinizi düşünün. Farklı kaynaklardan; hava durumu, trafik bilgisi, demografik veriler, kamu binalarıyla ilgili istatistikler, vergi bilgileri ve benzeri çeşitli bilgileri edinebilirsiniz. Tüm bu verileri birlikte analiz etmek, oldukça sağlıklı enformasyonlara ulaşmayı sağlayabilir. Ancak bu iyi bir entegrasyon çalışmasıyla yapılabilir. Bazı veri kaynaklarında ilçe adları, bazılarında ilçelerin kodları, bazılarında mahalle bilgileri yer alıyor olabilir. Hatta Türkçe bir karakterin İngilizce olarak kullanılması ya da gereksiz bir boşluk karakteri bile ifadenin bilgisayar programı tarafından farklı olarak yorumlanmasına sebep olabilir. Entegrasyon aşamasında tüm veri kümelerinin ihtiyaç duyulan değişkenler odağında tekilleştirilmesi gerekmektedir.

Ön işleme: Elde edilen ikincil veri kümeleri bir amaçla hazırlanmış, ortaya çıkmış ve dağıtılmıştır. Bu veri kümelerinin ortaya çıkmalarına sebep olan olgular artık mevcut olmayabilir. Ancak izleri muhtemelen veri

kümesinde hala bulunuyordur. Örneğin veri kümesi başka bir veri kümesiyle entegre edildiyse, bazı işaret değerler içeren sütunlar içeriyor olabilir. Bu sütunlar genelde farklı veri kümelerinde bulunan ID değerleri (kaydı tekilleştirmeye yarayan, benzersiz değerler) olabilir. Bu tarz bilgiler, siz bu veri kümesini ikincil olarak kullanmak istediğinizde işinize yaramayacaktır. Bir başka örnek olarak adres bilgisini düşünelim. Elde ettiğiniz veri kümesi, Türkiye'deki PTT şubelerinin iletişim bilgileri olsun. Her bir şubenin adres bilgileri il, ilçe, mahalle, cadde, sokak ve numara sütunları şeklinde saklanıyor olabilir. Dahası siz adres verisine tek bir hücre içerisinde erişmek istiyor olabilirsiniz. Bu durumda veri kümesindeki tüm kayıtlar için ilgili sütunların birleştirilmesi gerekecektir. Elde edilen ikincil veri kümelerine, istenilen biçime getirilene kadar uygulanan işlemlere ön işleme denilmektedir. Bu aşama, veri üzerinde çok miktarda manipülasyon gerektirse de gerçekte yapılan istenilen enformasyona dönüşüm sürecinin dışındadır. Ön işleme adını alma sebebi de budur. Bu aşama henüz enformasyon elde etme süreci başlamadığı halde yoğun bir analiz sürecinin uygulanmasını gerektirmektedir.

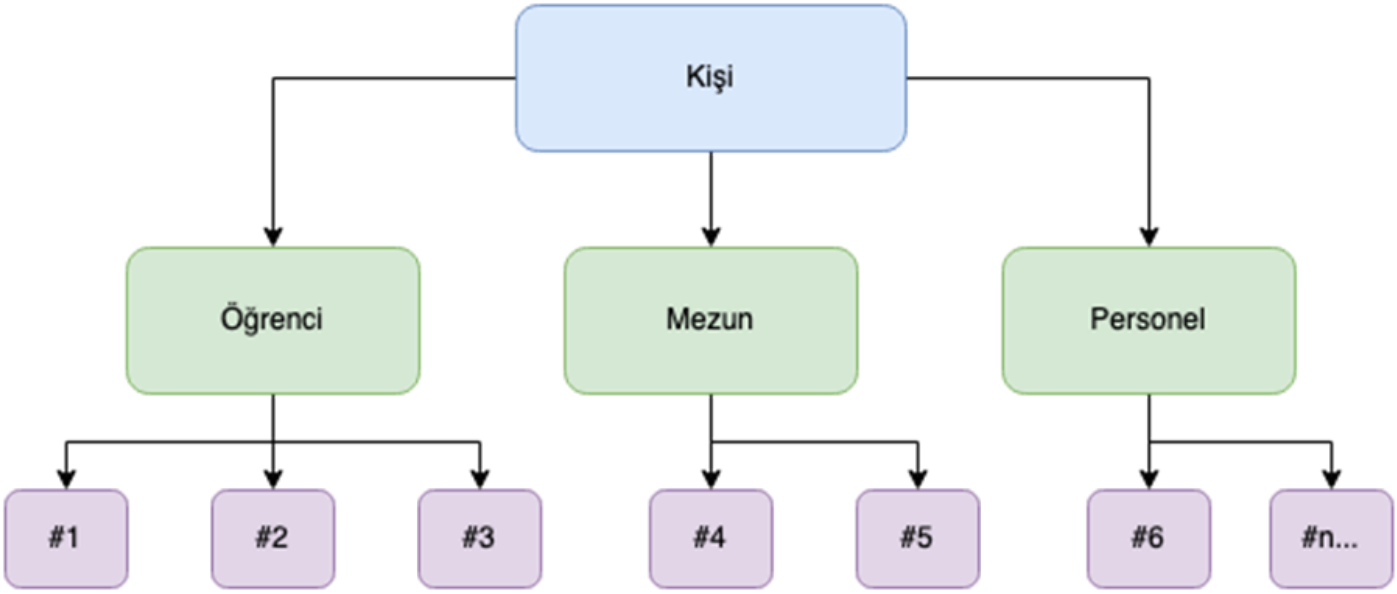
Enformasyon için veri olarak adlandırdığımız ikinci durum ise elde edilmek istenilen enformasyonun belirlenmesi ve bu enformasyona ulaşabilmek için gerekli veri kümelerinin elde edilmesidir. Bu durumda hem birincil hem de ikincil veri kaynaklarından istifade edilmektedir. İkincil veri kaynakları hali hazırda toplanmış ve dağıtılmış veri kaynaklarıdır ve elde etme süreci tamamlandığı için daha düşük maliyetle yeniden elde edilir ve kullanılır. İkincil veri kümelerinde elde etmeden öte ön işleme maliyeti daha yüksek olacaktır. Yine de bir enformasyona ulaşmada öncelikle ikincil veri kaynaklarından faydalanılır. Eğer enformasyona erişmede bu kaynaklar yetersiz ise birincil veri kaynaklarına başvurulur. Ölçüm, gözlem ve anket yöntemleri sıklıkla kullanılan birincil veri elde etme yöntemleridir. Birincil veri elde etmek oldukça maliyetlidir. Toplanacak verinin özelliklerinin belirlenmesi, veri toplama düzeneğinin hazırlanması, veri toplama ve kaydetme aşamaları zaman, insan kaynağı ve operasyonel açıdan maliyetli süreçlerdir. Ayrıca süreç tamamlandığında bir eksikliğin fark edilmesi de geri dönülmez bir riski içerisinde barındırmaktadır.

Enformasyona ulaşmak için kullanılan çok sayıda yöntem ve araç bulunmaktadır. Bu yöntem ve araçların neredeyse tamamı bir önceki bölümde anlattığımız veri yapılarını içe aktararak sürece alabilmektedir. Asenkron ve çevrimdışı süreçlerde genellikle CSV ve XLSX (MS Excel) dosya tipleri kullanılırken, senkron ve/veya çevrimiçi süreçlerde JSON veya XML gibi dosya türlerini kullanan web servisler kullanılabildiği gibi; doğrudan veri tabanı bağlantısı sağlanarak sorgulamalarla veriye erişim de mümkündür. Elde edilen veri kümesinin, kullanılacak araç ve yöntemlere bağlı olarak uyumlu bir biçime dönüştürülmesi ve sürece dahil edilmesi gerekecektir. Ön işleme süreci içerisinde gerçekleşen bu işlem, genellikle arayüz üzerinden gerçekleştirilebilirken bazı biçim dönüştürme işlemleri için özel bilgisayar programı hazırlanması; bu sayede veri kümesinin tamamının dönüştürülmesi gerekebilir.

Kullanılacak veri kümesi ya da biçimi ne olursa olsun, mevcut ya da sonraki uygulamalarda verinin farklı biçimlerine ihtiyaç duyulabilir. Dolayısıyla verinin, diğer biçimlere kolay şekilde dönüştürülebileceği bir yöntem seçilmelidir. Bu noktada veri tabanları ile karşılaşmaktayız. Veri tabanı ile saklanan veri, ihtiyaç anında, ihtiyaç duyulan şekilde sorgulanarak uygun formata anında denilebilecek kadar kısa bir sürede ulaşılabilir.

2.2. Veri Tabanı

Verinin farklı ihtiyaçlar için tekrar ve kolayca kullanılabilmesi; onun nasıl saklanıldığıyla ilgilidir. Ham veri kümesi olarak saklanan bir veri yığını, bütünselliği açısından bir kayba uğramasa da ihtiyaç duyulduğu anda hızlıca erişilemeyebilir. Bu durum veri kümelerinin yapılandırılmış şekilde saklanması gerekliliğini beraberinde getirir. Veri tabanı kavramı bu şekilde ortaya çıkmıştır. Geleneksel dosya temelli (verinin ham olarak saklandığı) sistemlere kıyasla daha kullanışlı olarak hiyerarşik ve ağ veri tabanları önerilmiştir (Akadal, 2017). Hiyerarşik veri tabanları, veri yapısında her bir elemanın birbirine göre bir seviyeye sahip olduğu, ters ağaç şeklinde, yukarıdan aşağıya genişleyen bir yapıdan oluşmaktadır.



Hiyerarşik veri tabanları bazı sorunlar için çözüm üretiyor olsalar da hem tüm veri kümeleri için uygun olmamalı; hem de güncelleme sürecinde bazı problemler yaşanması sebebiyle yeterli olmamıştır.

Bir veri kümesi içerisinde yer alan verinin tamamı her zaman tek yönlü bir hiyerarşiye sahip olmayabilir. Verinin çeşitli kısımları kendi içerisinde farklı hiyerarşiler barındırıyor olabilir. Ayrıca hiyerarşiye hiç katılmayan parçaları da bulunabilir. Tüm veri kümelerini saklayabilme ve gerekli şekilde kullanabilmeyi sağlaması beklenen veri tabanları için sıkça karşılaşılabilecek bu tarz problemler; hiyerarşik veri tabanlarının kullanımının önünde büyük engel oluşturmaktadır. İkinci büyük problem ise güncellemekle ilgilidir. Şekil ile verilen örneği incelerseniz bir eğitim kurumunun kişileri hiyerarşik düzende saklamasının bir temsilini görebilirsiniz. Bu düzende hazırlanmış bir veri tabanı içerisine yeni bir kayıt eklemek istediğimizde öncelikle eklenecek kişinin öğrenci, mezun ya da personel olma bilgisini edinmemiz gerekmektedir. Bu bilgiye göre hiyerarşik yapıda ilgili ekleme gerçekleştirilebilir. Buradaki birinci risk, eklenecek kişinin verilen 3 türden birine dahil olmaması durumunda bu veri tabanına eklenememesidir. Bu durumda hiyerarşide, kategorilerin belirlendiği seviyede yeni bir kategori tanımlamasının yapılması gerekebilir. Elbette kullanıcının bu eklemeyi yapmak için gerekli yetkisi varsa… Aynı örnekte yer alan bir diğer problem ise veride değişiklik durumunda ne yapılması gerektiğidir. Bir kişinin yer aldığı kategori değişebilir. Daha önce aynı kurumda öğrenci olan kişi önce mezun daha sonra da personel kategorisine dahil olabilir. Hatta bir adım daha öteye götürelim, bir kişi personel kategorisinde yer alırken bu eğitim kurumunda eğitim görmeye başlayabilir. Bu durumda hem personel hem öğrenci olmalı; bir süre sonra da hem personel hem de mezun kategorilerinde yer almalıdır. Mevcut şemada bunu yapmak ancak veri tekrarı (duplication) ile mümkündür ki bu da veri tabanları için asla karşılaşılmaması gereken bir durumdur.

Hiyerarşik veri tabanlarının sayılan dezavantajlarına yönelik ağ veri tabanlarının ortaya çıktığı görülmüştür. Farklı hiyerarşi seviyelerindeki çeşitli veri ve veri gruplarının birbirleriyle ilişkili hale getirilebilmesi odağındaki ağ veri tabanları da çeşitli sebeplerle yoğun kullanılabilen bir veri tabanı haline gelememiştir.

Bu süreçte Codd (1969) tarafından önerilen ilişkisel veri modeli ve sonrasında yine Codd (1970) tarafından önerilen ilişkisel veri tabanları literatüre eklenmiştir. Veri tabanı kavramı, verilere kolaylıkla erişilmesini sağlayan yapılar olarak tanımlanmaktadır (Çağıltay ve Tokdemir, 2010). Temel işleyişi; ilişkili değişkenlerin (veri kümesindeki sütunlar) gruplanarak bir araya getirildiği varlık (ya da tablo) yapıları ve bu yapıların da birbirleriyle çeşitli ilişki türleri ile bağlanması şeklindedir. İlişkisel veri tabanları; kayıtlar, veri kümeleri, kayıtlar arasındaki ilişkiler ve tüm bu kayıtların sahip oldukları veri tiplerinden meydana gelmektedir (Read, Fussell ve Silberschatz, 1992; Nizam, 2011).

Codd (1982), ilişkisel veri tabanlarının ortaya çıkış sürecindeki motivasyonunu 3 temel amaca bağlamıştır.

Birincisi; veri tabanı yönetiminde fiziksel ve mantıksal beklentilerin kesin ve net olarak birbirinden ayrılmasıdır. Veri bütünlüğü, verinin miktarından bağımsız olarak tartışılması ve sonuçlandırılması gereken bir konudur. Bunu iki açıdan ele alabiliriz. Küçük bir işletme için tüm kayıtların tek bir bilgisayarda toplanabilmesi ancak ölçek büyüdüğünde farklı özellikli verilerin farklı kaynaklara bölüştürülmesi eski zamanlarda performans yönetimi için gerekli bir davranış türü olsa da ilişkisel veri tabanlarıyla birlikte terk

edilmiştir. Veri kümesinin ele alınan iki parçası her ne kadar farklı özellikler barındırıyor olsa ve işletmenin farklı birimleri tarafından ihtiyaç duyuluyor olsa da verinin tamamı bir arada ve ilişkisel olarak saklanmalıdır. Bu yaklaşım, veri kümesi içerisinde henüz keşfedilmemiş enformasyonun ortaya çıkarılması ve verinin tamamının görselleştirilebilmesi gibi süreçlerde fayda sağlayacaktır. Bunun yanında donanımsal altyapının sınırlı olması sebebiyle veri tabanı tasarımında güncellemeye gitmek de gerçekleştirilmemesi gereken davranışlardan biridir. Normalde 2 farklı varlık (veri tabanı tablosu) içerisinde yer alması gereken bir verinin çeşitli altyapı kaynaklı sebeplerle tek bir varlık içerisinde toplamak, veri yapısına zarar verecek ve veri tabanının işlevlerinin eksilmesine sebep olabilecektir. Nihayetinde bir veri kümesinin ilişkisel veri tabanı yapısına dönüştürülmesi sürecinde veri tabanı tasarımı ve bu veri tabanının barındırılacağı fiziksel altyapının birbirini etkilemeyecek şekilde kurgulanması gerekmektedir.

İkinci motivasyon, tüm kullanıcı ve programcıların anlayabilecekleri; basit bir yapı ortaya koymaktır. Bilişim sistemleri zaman zaman karmaşık hale gelebilmektedirler. Hatta bazı durumlarda elimizde bir dokümantasyon olsa dahil bir sistemi kullanmayı öğrenebilmek büyük çaba gerektirebilir. Ancak elbette ki bu durum, programcıların bu süreci özellikle zorlaştırmasından kaynaklanmamaktadır. Genel konuşmak gerekirse bilgisayar programlarının karmaşıklığı genellikle zaman içerisinde ilk ortaya çıkma amaçlarının dışına çıkılması ve kontrolsüz bir büyümenin sonucu olmaktadır. Bu da sistem analizi ve tasarımı aşamasının yeterince sağlıklı gerçekleştirilmemesi ya da yönetim vizyonunun sistem analizi ve tasarımı aşamasında doğru istekleri ortaya koyamamasıdır. İlişkisel veri tabanları, temel bileşenleri ve bu bileşenlerin kullanılarak yüksek performanslı yapılar ortaya çıkarılması açısından son derece basitleştirilmiş bir temele sahiptir. Kullanıcılar ve programcılar, bir veri tabanını hayata geçirmek için çok fazla şey öğrenmek zorunda değillerdir. Ancak burada bir parantez açmak gerekir. Yüksek performanslı, veri bütünlüğünün zarar görebileceği bir risk barındırmayan, iyi bir veri tabanı tasarımı oluşturmak; veri tabanı tasarlama ve yönetmeyle ilgili ayrıntılara hakim olmaktan ve bunu çok kez deneyimlemekten geçmektedir. Yine de bir yeni başlayan için yeni bir veri tabanı oluşturmak ve onu ayağa kaldırmak zorlayıcı bir süreç değildir.

Üçüncü motivasyon ise kullanıcıların kayıt üzerinde çeşitli eylemleri gerçekleştirebilecekleri bir üst seviye dil oluşturmaktır.

Burada konumuz dışında olsa da dilin seviyesinin ne demek olduğundan biraz bahsedebiliriz. Programlama dilleri kullanıcılar tarafından anlaşılabilir ve kolay yazılabilir olmalarına göre seviyeleri belirlenir. Alt, orta ve üst seviye programlama dilleri bulunmaktadır. Alt seviye diller kullanıcılar tarafından zor öğrenilen, yazılan ve hızlıca bakıldığında sonucu kolay anlaşılamayan dillerdir. Üst seviye diller ise bakıldığında kolaylıkla anlaşılabilen, çok az satır kodla çok iş yapılabilen dillerdir. Bu iki seviyenin ortasında bir zorlukta olan dillere ise orta seviye dil adı verilmektedir. Aynı zamanda alt seviye diller makine diline daha yakın oldukları için bilgisayar tarafından daha kolay yorumlanırlar. Bu da hızlı çalışmalarını sağlar. Üst seviye diller ise makine diline dönüştürülebilmeleri için çok fazla işlem gerektiren dillerdir. Bu da nispeten daha yavaş olmalarına sebep olur. C alt seviye dillere bir örnek iken Python üst seviye bir dildir. Burada Java ve C# gibi dilleri ise orta seviye dillere örnek gösterebiliriz.

İlişkisel veri tabanlarının önerilmesindeki üçüncü motivasyona geri dönersek; gerçekleştirilmek istenilenin biraz daha netleştiğini görebiliriz. Bir veri kümesinin, büyüklüğünden bağımsız olarak bir arada saklanabilmesi ve ihtiyaç halinde hızlı bir şekilde sorgulanabilmesi; ayrıca bu becerileri kullanıcıya kolay şekilde sunması ilişkisel veri tabanlarını önemli kılan özelliklerden biridir. İlişkisel veri tabanlarının yönetiminde Yapılandırılmış Sorgu Dili (Structured Query Language – SQL) kullanılmaktadır. Sonraki başlıklarda SQL'in ne olduğuna daha detaylı değinilecek olsa da bu ders kapsamında detaylı bir SQL kullanımı yer almamaktadır. SQL, veri tabanı tasarımından öte veri tabanının yönetilmesiyle ilgili bir konudur.

İlişkisel veri tabanı yaklaşımı çeşitli avantajlara sahiptir (Sumathi ve Esakkirajan, 2007; Zeng & diğ, 2010). Şimdi bu avantajları kısaca inceleyelim.

Yapılandırılmış veri: İlişkisel veri tabanları, veri kümesinin tamamını yapılandırılmış olarak saklarlar. Buradaki kasıt, veri kümesindeki her bir kaydın atomik ölçekte belirli bir düzende kayıt altına alınmış olmasıdır. Veri kümesi üzerinde bir sorgulama yaparken, mevcut verinin belirli bir düzende kayıt altına alınmış olması; ilgili algoritmanın daha az kaynak harcayarak daha kolay sonuç bulabilmesini sağlayacaktır. İlişkisel veri tabanı, veri kümelerini yapılandırılmış olarak saklayarak bu durumu bir avantajlarından biri haline getirmektedir.

Veri tekrarı olmaması: Ele alınacak herhangi bir veri kümesinde aynı verinin tekrar tekrar yer aldığı görülebilir. Örnek vermek gerekirse; satış kayıtlarını içeren bir veri kümesinde eğer bir müşteri birden fazla kez satın alım yapmışsa her bir alımı ayrı bir kayıt olacaktır ve her kayıt satırında müşteri bilgilerinin yer alması gerekecektir. Eğer müşteriye ait ad, soyad, telefon numarası, adres, kimlik numarası ve benzeri birkaç kayıt bulundurulması gerekiyorsa her bir satış kaydı için bu bilgilerin tekrar girilmesi gerekmektedir. Bu da tüm veri kümesi göz önünde bulundurulduğunda aynı verinin çok fazla kez tekrar etmesine sebep olacaktır. Veri tekrarının çok olması, veri kümesinin güncellenmesinde çeşitli zorluk ve veri bütünlüğünün bozulması gibi riskler oluşturmaktadır. İlişkisel veri tabanı yapısında kayıtlar farklı varlıklar olarak kümelendirilir. Bu kümelendirilme neticesinde veri tekrarı da en aza inmektedir. Benzer bir örnekte, veri tabanı içerisinde Müşteri adında bir varlık oluşturulabilir ve her bir müşteri kaydı bu varlık içerisinde bulunabilir. Daha sonra bu müşteri bilgisinin tekrar etmesi gereken yerlerde müşteriye ait kaydı temsil eden bir işaret ile kayıtlar eşleştirilebilir. Bu temsil değeri günlük hayatta genelde karşımıza ID adıyla çıkmaktadır. Bu konuya ayrıca değineceğiz.

Ölçeklenebilirlik: Veri kümeleri, ele alınan projeye göre çeşitli büyüklükte olabilir. Çoğu proje, genellikle ortalama kaynaklara sahip bir bilgisayarla rahatça çalıştırılabilecek olsa da bazı projeler için yüksek kaynaklara ihtiyaç duyulmaktadır. İlişkisel veri tabanları, farklı ölçekteki projeler için hizmet verebilir niteliktedir. Veri bütünlüğü ile fiziksel altyapı birbirlerinden ayrı değerlendirilmektedir. Veri bütünlüğüne bir zarar vermeden fiziksel altyapıyı ihtiyaca yönelik olarak güncellemek mümkündür.

İçerikten ve yöntemden bağımsız: İlişkisel veri tabanları, herhangi bir amaçla toplanmış veri kümeleri için kullanılabilir. Veri kümesinin içeriği ilişkisel veri tabanları için önemsizdir. Önemli olan, verilerin sahip olduğu tiplerdir ki tüm veri çeşitleri ilkel veri tipleri ve bunların özelleştirilmiş halleri ile saklanabilmektedir. Bu durum ilişkisel veri tabanını içerikten bağımsız hale getirir. Ayrıca kullanılacak yöntem ve araçlar da önemli değildir. İlişkisel veri tabanı, tüm programlama dilleri ve birçok analiz aracıyla entegre olabilmek üzere sürücülere sahiptir. Bir sürücüye sahip olmadığı durumlarda ise veri, CSV ve benzeri bir biçimde dışarı aktarılabilir ve ilgili araç içerisinde kullanılabilir. Dolayısıyla ilişkisel veri tabanı kullanmaya başlarken sürecin devamında kullanılacak programlama dili ve analiz araçlarıyla ilgili bir kısıtlamaya gitme ihtiyacı duyulmamaktadır.

Veri yönetimi: Geleneksel veri yönetiminde veri kümesi bir bilgisayar üzerinde bulunur ve bu bilgisayarı kullanan kişi veri üzerinde çeşitli haklara sahip olabilir. Ancak daha büyük projeler için aynı veri kümesine aynı anda çok sayıda kişi ve/veya sistemin erişmesi gerekebilir. Bu ihtiyaç da güncel ağ teknolojileri ile aşılabiliyor olsa da aynı anda (yaklaşık olarak) veri kümesinde güncelleme yapan iki istemci olması durumunda veri bütünlüğünde bozulma ya da veri kaybı yaşanması söz konusu olabilir. İlişkisel veri tabanı yönetim sistemleri, veri tabanına erişimin kontrollü şekilde gerçekleştirilmesi sağlarlar. Gelen istekleri sıraya koyar, bir istek sebebiyle başka bir isteğin zarar görmesine engel olurlar. Bu sayede çok sayıda kullanıcı aynı veri tabanına bağlanarak sorunsuzca işlem gerçekleştirebilir. Gündelik hayatta sıkça kullandığımız sosyal medya uygulamalarını düşünelim. Aynı anda bu uygulamayı kullanan çok sayıda kişi ilgili sosyal medyanın veri tabanına erişmekte ve çeşitli işlemler gerçekleştirmektedir. Veri tabanı, bu süreci yöneterek tüm bağlanan kullanıcıların problemsiz şekilde isteklerine yanıt alabilmelerini sağlamaktadır.

Entegrasyon: İlişkisel veri tabanı, iletilen sorguya göre veriyi oluşturabildiği ve diğer biçimlere kolayca adapte olabildiği için diğer sistemlerle kolaylıkla entegre olabilir. Sistemlerin birbiriyle entegre olması günümüz için oldukça önemlidir. Aynı sistemin bile web, masaüstü yazılım, mobil yazılım gibi farklı platformlarda çalışması ve eşlenik (senkronize) olması zorunluluğu entegrasyonu gerekli kılmaktadır. İlişkisel veri tabanına yapılan bir sorgu neticesinde elde edilen sonuç veri kümesi, daha önce de vurgulandığı gibi neredeyse tüm programlama dilleri ve analiz araçlarıyla uyumludur. Bu da farklı platformlardaki sistemin aynı veri tabanına bağlanabilmesini; dolayısıyla birbirleriyle entegre olabilmelerini sağlamaktadır.

Tutarsızlıktan kaçınma: İlişkisel veri tabanları, istemciler tarafından gelen istekleri bir sıraya koyar ve veri kaybı yaşanmadan işlenmesini sağlar. Büyük sistemlerde aynı veri tabanına çok sayıda istek gelebilir. Yine de veri tabanları bu isteği veri bütünlüğü önceliğinde işleyebilir. Tüm bunlar veri bütünlüğünün korunması ve veri içerisinde tutarsızlıklar görülmesinin önüne geçecektir. Bunun yanından aynı verinin tekrar edilmek yerine bir kez kullanılarak sayısal değerlerle temsil edilmesi; veri üzerinde güncelleme yapılması durumunda sorgulamalarla elde edilen tüm raporların da güncellenmesi anlamına gelmektedir. Daha önce yapılan satış ve müşterilerle ilgili kayıtlarda müşteriye ait bir verinin güncellenmesi durumunda; eğer geleneksel kayıt yöntemi kullanılarak tekrarlı verinin bulunduğu bir veri kümesinde işlem yapılıyorsa, aynı müşteri kaydının

bulunduğu tüm kayıtların teker teker güncellenmesi gerekecektir. Bir kaydın atlanması, aynı müşterinin farklı satışlarında farklı kayıtlara sahip olmasına sebep olabilecektir. Veri tabanı yapısında müşteriye ait kayıtlar tek bir yerde tutulacağı için bu kaydın güncellenmesi neticesinde müşteriyi temsil eden değerin kullanıldığı her yerde artık güncellenmiş kayıt ile karşılaşılacaktır. Tüm bu örneklerin de gösterdiği üzere ilişkisel veri tabanları tutarsızlıktan kaçınma konusunda fayda sağlamaktadır.

Çok kullanıcı: Bir veri tabanına birden fazla kullanıcının bağlanması gerekebilir. Kullanıcılar, bu veri tabanı üzerinde işlem yapması gereken yetkili kişiler olabilecekleri gibi, eğer elektronik ortamda hizmet veren bir sistemden bahsediyorsak bu sisteme erişen herhangi bir kullanıcı da olabilir. Bu durumda veri tabanına erişimin yanında yetkilendirme de önem arz eder. Yönetici yetkisine sahip kişi tüm verileri görüntüleyebilir, veri girişi yapan kişinin yeni kayıt ekleme yetkisine ihtiyacı vardır, ziyaretçi kullanıcılar ise yalnızca kendileri için izin verilen veriyi izin verilen miktarda ve sıklıkta görüntüleyebilirler. Tüm bu erişim rollerinin tanımlanması ve ilişkisel veri tabanı üzerinde uygulanması gerekir. İlişkisel veri tabanları, veri tabanı üzerindeki operasyonları, bağlanan kişinin yetkisini kontrol ederek buna göre gerçekleştirebilir. Dolayısıyla her seviye kullanıcı aslında birebir aynı sunucu ve veri tabanına bağlanırken her biri kendi limitlerinde işlemler gerçekleştirebilirler.

2.3. Veri Tabanı Tasarlamak

Tablo biçimindeki bir verini, herhangi bir elektronik tablo (MS Excel gibi) yazılımıyla kayıt altına almak oldukça kolaydır. Veriyi kopyalar, yapıştırır ya da elle girişini yapar ve kaydedersiniz. Bir veri kümesinin veri tabanı yapısına dönüştürülerek kaydedilmesi için ise ilişkili veri kümeleri oluşturmaktan ve bu şekilde kayıt altına almaktan bahsetmiştik. Peki elimizde n tane sütuna, m tane satıra sahip bir veri kümesi varken bu parçalama işlemini nasıl yapacağız? Rastgele mi? Elbette hayır. Kaç tane küme oluşturmamız gerekli? Her birinde kaç tane ve hangi sütunlar yer almalı? Böldüğümüz kayıtların gerektiğinde yeniden birleştirilebilmesi için ne yapmak gerekir? Bu soruların yanıtı, bu dersin amacını içerisinde barındırmaktadır. Bu bir veri tabanı tasarımı sürecidir.

Veri tabanı tasarlamak, sistem analizi ve tasarımı sürecinin önemli bir parçasıdır. Yeni bir sistem geliştirilmeden önce uygulanan birçok adım ve verilen karar içerisinde sistemin sahip olması gereken bileşen ve yetenekler belirlenir. Buna göre de sistemin teknik bileşenlerine karar verilir. Yazılımla ilgili mimari tasarımının yanı sıra veri tabanı tasarımının da tüm ihtiyaç ve gerekliliklere paralel hazırlanması gerekmektedir.

Veri tabanı tasarımı süreci özneldir (sübjektif), zordur ve risklidir. Ayrıca deneyim gerektirir. Şimdi bu olguların sebeplerini tartışalım.

Veri tabanı tasarım süreci içerisinde risk barındırmaktadır. Bunun sebebi, kurgulanan bir veri tabanının faaliyete başladıktan sonra tasarımdaki bazı hatalardan dolayı veri üzerinde işlemlerin tam performansla gerçekleştirilememesi ve işleyen bir veri tabanı tasarımında güncelleme yapmanın farklı riskler barındırıyor olmasıdır. Yeni bir sistem için sistem analizi ve tasarımı süreçleri görece daha az streslidir. Çünkü bu süreçte yapılan hatalar, süreç sonlanmadan giderildiği sürece bir tehlike içermezler. Hatalı ya da eksik algoritmalar, unutulan bir kullanıcı rolü, testi tamamlanmamış bileşenler, kayıt altına alınması gereken bir özelliğin göz ardı edilmesi gibi konular sistem analizi ve tasarımı sürecinin kaçınılmaz parçasıdır. Zaten bu süreç kendi içerisinde bu hataları yok etme odaklı çok miktarda iş yükü ve sinerji gerektirmektedir. Sistem analizi ve tasarımı süreci tamamlandığında gerçekleştirilecek sistem, bütün ayrıntısıyla dokümante edilmiş durumda olur. Bu durum yeni yapılacak bir binanın bütün planlarının hazırlanması gibidir. Yapım süreci başladığında bilgisayar programcıları, tasarımcılar ve mühendisler eldeki dokümanların yönlendirmelerine bağlı olarak sistemi inşa ederler. Sistem tasarımı sürecinde tüm ayrıntılar ele alındığı için genellikle gerçekleştirme sürecinde karar verme ihtiyacı bulunmamaktadır. Gerçekleştirmenin tamamlanmasının ardından yine ilgili dokümana göre gerekli testler yapılır ve sistem hazırdır. Sistem, seçilen stratejiye göre alfa, beta ve benzeri test süreçleriyle kademeli ya da doğrudan herkesin erişimine açılır. Veri tabanıyla ilgili sözünü ettiğimiz risk bu noktada karşımıza çıkar. Bir sistem canlıya alındığında (genel kullanıma açıldığında) daha önce test edilmemiş bir durum ya da sistem analizi aşamasında akla gelmeyen bir ihtiyaç fark edilebilir. Bu ihtiyaç, mevcut süreçleri zora sokan, derhal gerçekleştirilmesi gereken, aksi halde mevcut süreçlerin devam edememesine sebep olabilecek bir ihtiyaç olabilir. Ya da daha az riskli olan bir yan süreç; geliştirilmesi için bir süre beklemenin göze alınabileceği bir ihtiyaç da olabilir. Her iki durumda da sistem analizi ve tasarımı

süreçlerinin ilgili adımlarına dönülür, eksik kalan ihtiyaç analiz edilir ve mevcut sisteme entegresi için gerekli planlama yapılarak sistem üzerinde güncelleme süreci başlar. Ancak bu süreç yeni bir sistem gerçekleştirmekten daha stresli olabilir. Kimsenin kullanmadığı bir sistemin (yeni) geliştirilmesi sürecinde kullanıcılar bu sürecin bir parçası değildir. Ancak kullanımda olan bir sistemde değişiklik yapmak; her şey yolunda gitse bile memnuniyetsizliklere yol açabilir. Sisteme erişim kesintileri yaşanabilir, güncellemeler mevcut işleyişte hatalara sebep olabilir, geliştirilen özellik tam olarak istenildiği şekilde hazırlanmamış olabilir. Bu süreç içerisinde veri tabanı tasarımında yapılması gereken güncellemeler, programlama tarafında olanlara göre daha riskli olabilir. Bunun sebeplerinden bazıları;

· Veri tabanı tasarımı güncellendikten sonra mevcut veri tabanındaki verilerin yeni tasarıma uygun şekilde yerleştirilmesi gerekliliği,

· Yazılım içerisindeki veri tabanı bağlantılarının veri tabanına göre güncellenmesi gerekliliği,

· Yeni tasarımın genellikle yeterince test edilememesinden kaynaklı veri yönetiminde hatalar görülmesi

olarak sayılabilir.

Yazılımda görülen hatalar, yazılım üzerindeki güncellemelerle giderilebilirken; veri tabanı tasarımında görülen hatalar hem yazılımda hem de eldeki veri bütünlüğünde de güncellemeler ve çeşitli kontroller yapılmasını gerektirebilir. Bu sebeplerden dolayı sistem analizi ve tasarımı süreçlerinde veri tabanı tasarımlarının daha fazla zaman ve iş yükü maliyetini göze alarak dahi doğru şekilde gerçekleştirilmesi, geliştirilen sistemin ihtiyaç duyulan duruma daha uygun olmasını sağlayacaktır.

Veri tabanı tasarımını, yalnızca ihtiyaçların ve hedeflerin doğrultusunda gerçekleştirmeye çalışmak; hangi kararların alınacağı konusunda her zaman yeteri kadar yol gösterici olmayabilir. Geliştirilen sistem, analiz ve tasarım aşamasında elde edilen hedeflere göre yazılım geliştirme aşamalarından geçecektir. Ancak veri tabanı tasarımı süreci, aynı zamanda verinin en iyi ne şekilde yönetileceği, hangi sorgulara ihtiyaç duyulacağı, elde ne tür veriler olduğu, gelecekte ne tür veriler ekleneceğini tahmin etmeye dayalı olarak farklı seviyelerde karmaşıklıklara sahip olabilir. Veri tabanı tasarımı gerçekleştirmek, geleceğe yönelik bazı tahminleri yapmayı da gerektirebilir. Bir hastaneye ait bilgi sisteminin geliştirilmesi sürecinde hastanenin mevcut süreçlerinin yazılıma aktarılması elbette beklenen bir istek olacaktır. Ancak bunun yanında hali hazırda devam eden bazı süreçlerin bilgi sisteminin desteğiyle daha az iş gücü ve zaman gerektiren farklı bir yolla yapılması istenebilir, idari çeşitli değişikliklerin sisteme yansıtılması gerekebilir, çeşitli yeni süreçlerin de bilgi sistemleriyle birlikte hayata geçirilmesi hemen ya da bir süre sonra istenebilir. Mevcut bir kurum için bir bilgi sistemi tasarlanması süreci bile çok miktarda belirsizlik içerirken yeni bir sistemin tasarlanma süreci içerisinde çok daha fazla belirsizlik bulunması beklenebilecek bir gerçektir.

İyi bir veri tabanı tasarlamak için hakim olunması gereken iki konu vardır: Bunlardan birincisi iyi bir ilişkisel veri tabanının sahip olduğu özellikler, ikincisi ise normal formlardır. İyi bir veri tabanı tasarımının sahip olması gereken özellikleri bilmek, bir veri tabanı tasarımı yaparken sürekli bunu gözetmeyi de gerektirecektir. Örneğin veri ne şekilde çağrılacak, kaç istemci aynı anda erişecek, hangi veriye ne şekilde ihtiyaç duyulacak, hangi veri parçası hangi diğer parçalarla birlikte bir küme olmalı? Bu soruların yanıtını aramak daha kullanışlı ve yüksek performanslı veri tabanı tasarımı gerçekleştirmenin anahtarı olacaktır. Normal formlar, iyi bir veri tabanı tasarımı gerçekleştirilmek için uyulması gereken kuralları sunmaktadır (Fagin, 1981; Hoffer, 2016). Normal formlara uygun olarak tasarlanmış bir veri tabanına normalize, bu sürece ise normalizasyon adı verilmektedir (Lee, 1995; Hoffer, 2016). Bölüm 6 içerisinde normalizasyon konusunu ve normal formları ayrıntılı inceleyeceğiz. Burada değinmiş olmamızın sebebi, bir veri tabanı tasarımı yaparken sahip olunması gereken bilgi birikimini vurgulamak. Teknik olarak veri tabanlarının özelliklerini ve normal formları bilmek iyi bir veri tabanı tasarımı gerçekleştirmek için yeterli olmalı. Ancak veri tabanı tasarımı; tasarımcının deneyim, tecrübe ve bilgi birikimiyle ilgilidir. Bunun yanında tasarımcılar bu özelliklerin tamamına sahip olsalar da yine de aynı proje için farklı tasarımlar ortaya koyabilirler (Akadal, 2021). Bu da veri tabanı tasarımının nesnel (objektif) değil, öznel (sübjektif) olmasına yol açmaktadır.

Veri tabanı tasarımı sürecinin öznellik içeriyor olması çok da şaşırtıcı bir olay değildir. Tasarım sürecinin başında elde bir veri kümesi varsa, nispeten elde edilmesi beklenen veri tabanı tasarımı yapısı daha net olacaktır. Veri kümesinde bulunması gereken tüm alanlar, içerisinde örnek veriyle birlikte sunuldukları zaman, oluşturulacak veri tabanı tasarımı için yol gösterici olabilecektir. Ancak bu durumda bile mükemmel veri tabanı tasarımına ulaşmak mümkün olmayabilir. Daha önce geleneksel yöntemlerle saklanılan bir veri

kümesinin, bir bilgi sistemi hazırlandıktan sonra aynı şekilde kullanılacağı garanti değildir. Veri kümesinde, veri yapısında, bileşenlerde, kullanıcı ve paydaşlarda çeşitli güncellemeler ve genişlemeler gerçekleştirilebilir. Dahası, bu potansiyel değişiklikler planlanmamış ve hatta henüz düşünülmemiş bile olabilir. Bu durumda veri tabanı tasarımcısının rolünün önemi öne çıkmaktadır. Veri tabanı tasarımı gerçekleştiren kişi karşılaşılması muhtemelen durumları önceden belirlemek, bunlarla ilgili hem geliştiriciler hem de sistemi yönetecek ve kullanacak kişilerle işbirliği yapmak mecburiyetindedir. Elde bir veri kümesi varken bile bu risklerle karşılaşma ihtimali varken, elde yalnızca istek ve öngörülerin olduğu bir sistem analizi ve tasarımı sürecinde çok daha fazla belirsizlik olması beklenecek bir şey olacaktır.

Ulaştığımız bu noktada, iyi bir veri tabanı tasarlamak için “deneyim”in sahip olunması gereken bir özellik olduğunu savunabiliriz. Bunun sebebi, bahsettiğimiz riskleri öngörme olasılığının deneyimle arttırılmasıdır. Biraz daha açalım. İyi bir veri tabanı tasarımı yapabilmek için sahip olunması gereken bilgi ve beceriye her zaman yeterli gelmeyecektir. Eldeki tüm bilgi ve beceri kullanıldığında dahi veri tabanı tasarımı oluşturulduktan ve bilgi sistemi faaliyete alındıktan sonra süreçte hatalar ya da geliştirilmesi gereken durumlar keşfedilebilir. Bir kez bu süreci deneyimlemiş olan uzman, gelecekte benzer projeler içerisinde yer aldığında daha önce yaşanmış problemleri hatırlayarak sistem analizi ve tasarımı sürecinde tanımlanmamış kullanım durumlarını da hesaba katabilir ve tasarımını buna göre güçlendirebilir. Bazı yazılım projeleri hayata geçtikten sonra krizlere sebep olabilir, geliştirme sürecinden daha yoğun bir iyileştirme süreci geçirilmesine sebep olabilir. Bu gibi durumlar için tasarım aşamalarında deneyimli geliştiricilerin yönlendirmeleri riskleri ve bunlara bağlı krizleri en aza indirebilir.

Veri tabanı tasarlama süreci normal formlar sayesinde kurallarla gerçekleştirilen bir süreç gibi görülebilir. Benzer şekilde karşılaşılan durumlar karşısında benzer faaliyetleri göstermek, tasarım gerçekleştirmenin uzmanlık ve deneyim gerektirmeyeceğini düşünmeye sebep olabilir. Ele alınan girdiye uygulanan bazı işlemlerin sonucunda çıktı elde etmek, algoritması yazılabilen ve dolayısıyla bilgisayar programı hazırlanabilecek bir süreci işaret etmektedir. Veri tabanı tasarımı gerçekleştirme sürecini bilgisayar programıyla otomatikleştirmek, uzun yıllar boyunca farklı algoritmalarla denenmiş; ancak kullanıcıdan alınan ek girdiler sayesinde kısmen gerçekleştirilebilmiştir. Akadal (2017), yalnızca ham veri kümesini girdi alarak ilişkisel veri tabanı tasarımı önerisi sunan bir algoritma önermiştir. Bu konu hala gelişmekte olan, üzerinde çalışılan bir alandır. Otomatik veri tabanı tasarımı gerçekleştirmenin mümkün olmadığı gerçeği, veri tabanı tasarımının normal formların kurallarına göre kolayca yapılamayacağını da göstermektedir. Tüm bilginin elde olmasına rağmen yine de iyi bir veri tabanı tasarımı oluşturmayı garanti edememenin sebebi deneyim ve buna bağlı olarak özgün/öznel yaklaşım gerekliliğidir. Dahası, yeterli bilgi birikimi ve tecrübeye sahip iki uzmanın önereceği veri tabanı tasarımlarında da farklılıklar görülmesi olasıdır. Bu iki uzman, farklı durumları deneyimlemiş, farklı riskleri öngörerek karar almış olabilirler. Bu noktada siz değerli öğrencilerin yapması gereken şey en fazla sayıda örnek uygulama görerek mümkün tüm riskleri tahmin etmeniz ve kullanım sırasında probleme yol açmayacak bir veri tabanı tasarımı gerçekleştirebilmenizdir.

Tüm bu sebeplerin, neden size bir veri tabanı tasarımı hazırlama görev listesi (checklist ve to-do list kavramları sıklıkla kullanılır) vermek yerine bunu yapmayı öğreten bir ders hazırladığımızı gösterdiğini umuyorum.

2.4. Veri Tabanı Yönetim Sistemleri

İlişkisel veri modeli ve ilişkisel veri tabanı kavramları, ilgili kaynaklarda bir olguyu ifade etmektedirler. Yani aslında doğrudan bir yazılım ya da veri tabanını içerisine yerleştirip kullanabileceğimiz bir yapı değil, bunun nasıl gerçekleştirilebileceğini gösteren bir izahname olarak düşünebilirsiniz. Bu önerinin ardından elbette bu yapıyı hayata geçirecek yazılımların oluşturulması gerekmektedir. Veri tabanı yönetim sistemleri, çok miktarda verinin saklanmasına ve üzerinde sorgulama işlemleri gerçekleştirilebilmesine olanak sağlayan yazılımlardır (Raghu ve Johannes, 2000). Veri tabanı tasarlamak için bir veri tabanı yönetim sistemine ihtiyaç yoktur. Hatta bir bilgisayara bile ihtiyaç yoktur. Veri tabanı tasarımı elde sadece kağıt ve kalemle gerçekleştirilebilir. Bu yöntem, tasarım aşamasında kullanılmasını önerdiğimiz bir yöntemdir. Bir işlemi ya da süreci gerçekleştirirken bilgisayardan faydalanmak gerekir. Ancak bir tasarım hazırlamak zihni bir süreçtir. Bu sebeple veri tabanı tasarımı hazırlama sürecini bilgisayardan uzak gerçekleştirmek daha yüksek performans elde etmeyi sağlayabilir.

Bir veri tabanı tasarımı hazırladıktan sonra bu tasarımı bilgisayar ortamında gerçekleştirmek gerekir. Bu da bir veri tabanı yönetim sistemi kullanmayı gerektirir. Veri tabanı yönetim sistemleri sayesinde;

· Yeni bir veri tabanı tasarımını gerçekleştirebilir,

· Veri tabanı içerisine kayıtlar ekleyebilir,

· Kayıtlar üzerinde güncelleme ve silme işlemleri yapabilir,

· Özelleştirilmiş sorgular çalıştırabilir,

· Kullanıcılara çeşitli sorgulama ve operasyon yetkileri verebilir,

· Bir bilgisayar programının geliştirilen veri tabanına bağlanarak verileri işlemesine olanak sağlayabilirsiniz.

Tüm veri tabanı yönetim sistemleri, ilişkisel veri tabanlarının sahip olması gereken temel özellikleri sunarlar. Kağıt üzerinde tasarladığınız veri tabanını bilgisayar ortamında gerçekleştirebilir ve kayıtlar ekledikten sonra çeşitli sorgularla soyut elektronik tablolar üretebilirsiniz. Eğer bir ilişkisel veri tabanını en temel özellikleri için kullanıyorsanız hangi veri tabanı yönetim sistemini seçeceğinizi belirlemek için çok düşünmenize gerek yoktur. Donanımsal ve yazılımsal altyapınıza uygun bir sistem seçerek devam edebilirsiniz. Ancak tasarlanan veri tabanı üzerinde bazı özel operasyonlara ihtiyaç duyuyorsanız ya da geliştirdiğiniz bilgisayar yazılımı özellikle bir veri tabanı yönetim sistemini kullanmanızı gerektiriyorsa size uygun seçeneklerden birini seçerek devam edebilirsiniz.

Sık karşılaşılan veri tabanı yönetim sistemlerinden bazılarını ele alalım:

Microsoft Access: Microsoft Office yazılım paketi içerisinde bulunan Microsoft Access, genellikle yerel çalışmalarda ve eğitim amaçlı kullanılan, kullanımı kolay bir veri tabanı yönetim sistemidir. Kolayca kurulabilir ve kullanıma hazır hale getirilebilir. Detaylı, kolay kullanılabilen, kullanıcı dostu bir arayüze sahiptir. Genellikle küçük ölçekli, tek bir istemci (sunucu performansına sahip olmayan) bilgisayar üzerinde çalıştırılabilen bir yazılımdır. Hem görsel hem de komutlarla veri tabanı yönetimi mümkündür. Ayrıca yine yazılım içerisinde veri tabanı ile etkileşime geçen arayüzler tasarlamak mümkündür. Bu sebeplerden dolayı veri tabanı eğitimlerinde sıklıkla kullanılmaktadır. Microsoft Access sayesinde yeni bir veri tabanını arayüz sayesinde ya da komutlarla oluşturabilir, tablolar arasındaki bağlantıları kurabilir, bu tablolara birçok yöntemle veri girişi sağlayabilirsiniz. Access içerisinde arayüz oluşturarak veri girebilme olanağı, veri tabanlarıyla etkileşime geçerken bir ya da araç daha öğrenme zorunluluğunu ortadan kaldırmaktadır. Diğer birçok veri tabanı yönetim sistemini hayata geçirdikten sonra ilgili veri tabanıyla etkileşime geçecek kullanıcı arayüzleri oluşturmak için harici bir programlama dili ya da yazılım kullanmak gerekir. Access ile buna gerek bulunmamakta. Örneğin bir bakkal dükkanının günlük kayıtlarını veri tabanında saklamak istediğini düşünelim. Bunu Access ile kolaylıkla gerçekleştirebilir. Hazırlanan veri tabanına, yine Access içerisinde hazırlanmış arayüzlerle giriş yapabilir; dilediğinde belirlediği koşullara göre sorgular yaparak geçmiş kayıtları ekrana liste olarak yazdırabilir ve çeşitli hesaplamalar yaptırabilir ya da veriyi dışarı aktararak farklı yazılımlarla işleyebilir. İlk kez bir veri tabanıyla karşılaşmak isteyenler için Access tercih edilebilecek bir başlangıç noktası olacaktır.

Microsoft SQL Server: İnternet tabanlı uygulamalar için sıklıkla kullanılan iki platform bulunmaktadır. Bunlardan biri Microsoft'a ait olan .NET (dotnet) platformudur. C# programlama dili kullanılan bu platform üzerinde veri tabanı yönetim sistemi olarak Microsoft SQL Server sıklıkla tercih edilmektedir. Her ne kadar platformlar arası çapraz kullanım mümkün olsa da alışılageldik olan kullanım şekilleri bulunmaktadır. Ayrıca bu kullanım tercihleri bilgisayar programcılığı sektöründe kabul görmüş olduğu için istihdam sürecinde de beklenti bu yönde olmaktadır. SQL Server; güçlü ve geniş ölçekteki çok kullanıcılı internet tabanlı sistemlerin inşasında sık kullanılan bir veri tabanı yönetim sistemidir. SQL Server, yüksek uyumluluk arayışında genellikle diğer Microsoft platformlarıyla birlikte kullanılır. Orta ve büyük ölçekli projelerde kullanılmaktadır. Sunucu maliyeti sebebiyle küçük ölçekli projeler için kullanılabilir olsa da tercih sebebi olmayacaktır.

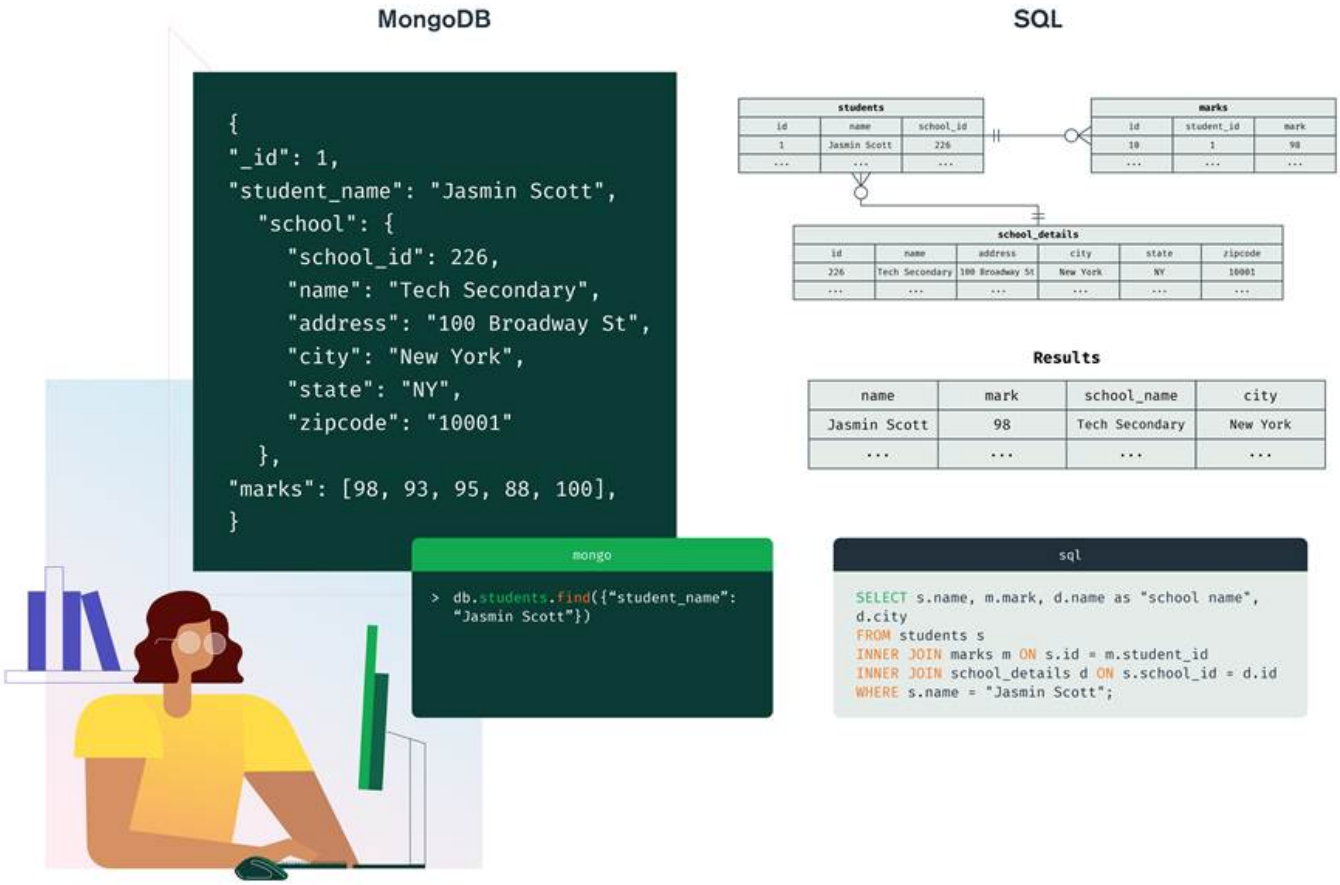
MySQL: İnternet tabanlı uygulamalar için sıklıkla kullanılan ikinci platform Linux'tur. Linux üzerinde çok sayıda farklı teknoloji çalıştırılabilir olsa da PHP dili ve MySQL veri tabanı sıklıkla kullanılmaktadır. Linux platformu, özgür yazılım ve açık kaynak kod felsefesi desteğiyle geliştirilmiş; dünya çapında çok sayıda

geliştirici ve kullanıcısı olan bir işletim sistemidir. Bu işletim sistemi üzerinde geliştirilen yazılımlar da aynı felsefeyi benimsemişlerdir. MySQL veri tabanı da SQL Server gibi güçlü, çok sayıda kullanıcıya sahip internet tabanlı sistemler için hizmet sunabilen bir veri tabanı yönetim sistemidir. Yönetimi için PhpMyAdmin adlı web tabanlı uygulama sıklıkla tercih edilse de MySQL, birçok yazılımla erişilebilen ve kullanılabilir, hemen hemen her platform için sürücülere sahip bir veri tabanıdır. Çeşitli ücretli ve ücretsiz yazılımlar ile MySQL veri tabanlarına bağlanılabilir ve yönetilebilir. Tüm ölçekteki projeler için kullanılabilir. Özellikle paylaşımlı sunucularda da kolaylıkla çalıştırılabilir olması sebebiyle küçük ölçekteki projelerde de tüm özellikleriyle faydalanılabilmesi mümkündür.

Oracle Database: Oracle firması tarafından ortaya çıkarılan bir veri tabanı yönetim sistemidir. Oracle, yoğun olarak işletmeden işletmeye (B2B: Business to Business) hizmet vermekte olan global bir firmadır. Bu sebeple Oracle veri tabanları genellikle büyük ölçekli projelerde kullanılmak üzere global firmalar tarafından tercih edilmektedir. Oracle veri tabanları PL/SQL (Procedural Language / SQL) adı verilen, özelleştirilmiş bir yazılımı içerisinde barındırmaktadır. PL/SQL, özetle içerisinde prosedürler yani diğer bir deyişle küçük programlar yazılabilen veri tabanı komutlarıdır. Bu sayede bilgisayar programlarına bağımlılık bir miktar daha azalmaktadır. PL/SQL sayesinde prosedür içeren bazı aşamaların veri tabanı üzerinde çalıştırılabilmesi mümkün olacaktır. Böylece hazırlanan bilgisayar programı, veri tabanından bir seviyeye kadar işlenmiş, prosedür uygulanmış çıktı olan veri kümesini elde edebilecektir.

Postgre SQL: Postgre SQL, en gelişmiş olduğu iddia edilen, açık kaynak kodlu, 30 yıldan uzun süredir geliştirilmeye devam edilen, birçok programlama dili ve araçla uyumlu veri tabanı yönetim sistemidir.

MongoDB: MongoDB, bir ilişkisel veri tabanı değildir, doküman tabanlı veri tabanı kategorisinde sayılabilir. Günümüzde bilgi sistemleri ve yeni medyanın yapılandırılmamış veri konusunda da oldukça zengin olması ve bu verinin de saklanılması sonrasında da işlenmesi ihtiyacı neticesinde popülerliği artmıştır. Bu veri tabanında veriler nesne ve hiyerarşik yapıya uygun olarak kayıt altına alınırlar. JSON veri formatına da çok benzeyen veri saklama yapısı, MongoDB içerisinde de karşımıza çıkmaktadır. Daha iyi anlatabilmek için bir örnek verelim. Tek bir veri kümesinden oluşan bir veri yapısı düşünelim. Bu veri yapısı içerisinde çok sayıda değişken (sütun) olsun. Her bir kayıtta (satırda) bu değişkenlerin tümü için bir değer atanmamış olabilir. Örneğin kişilerle ilgili kayıtların toplandığı bir yapıda lisans, yüksek lisans ve doktora mezuniyet bilgileriyle ilgili sütun için tüm kişilerde bir kayıt olmayabilir. Ya da bir işletmenin yıllara dayalı kazanıcını gösteren tabloda, yeni bir işletme için bazı veriler eksik kalabilir. Bilgi sistemlerinde çok daha fazla ayrıntıyı kayıt altına almaktayız ve bunların hepsi her zaman bir değer almayabilir. Ayrıca kayıtlar hiyerarşik olarak birbirine bağlı da olabilir. Ayrıca bu yapı sayesinde her bir kayıt birbirinden farklı sayıda değişkene sahip olabilir. Örneğin kişiler varlığında yetişkinler için farklı, çocuklar için farklı değişkenler kullanılarak veri kaydedilebilir. MongoDB ve JSON veri biçimi bu türdeki verileri saklamak için oldukça kullanışlıdır.



SQLite: Günümüz bilgisayar ve mobil cihazları genellikle oldukça iyi kaynaklara sahipler. Bu sebeple bir yazılım çalıştıracağımız zaman, eğer yazılım çok fazla kaynak tüketmiyorsa cihazın gücünü dert etmeden yazılımı çalıştırabiliyoruz. Veri tabanı tercihinde de kullandığımız platform ve entegre edilecek programlama diline yönelik önemli bir etken. Genellikle bu doğrultuda karar alınması en uygunu. Ancak bazı durumlarda teknik kapasite oldukça sınırlı olabilir. Örneğin nesnelerin interneti (IoT – Internet of Things) kapsamı altında Raspberry Pi gibi becerikli araçlar, oldukça düşük kapasiteleriyle işlem yapmaktadırlar. Buzdolabı, çamaşır makinesi, ampul, televizyon ve benzeri akıllı ev araçlarının bazıları da bir veri tabanına ihtiyaç duyabilir ama bunun için ayırabileceği kapasite oldukça sınırlıdır. SQLite bu ihtiyaca karşılık verebilen bir veri tabanı yönetim sistemidir. SQLite hem bellek hem de iş gücü anlamında çok düşük kaynak tüketen, bunun yanında büyük ölçekte hizmet veren veri tabanlarının sağladığı birçok avantajı sağlayabilen bir yapıya sahiptir. Elbette çok istemciye yanıt verme konusunda iyi bir seçenek olmayacaktır ancak kıt kaynaklara sahip bir cihaz içerisinde, bu cihazın ihtiyaç duyduğu kaydetme ve sorgulama işlemlerini kolaylıkla sağlayabilir. Veri tabanını öğrenme konusunda Access'ten bir sonra gelebilir. Kurulumu ve kullanımı oldukça kolay olmakla birlikte bir grafik arayüzü bulunmadığı için komut satırı ya da harici bir veri tabanı yönetim aracı ile kontrol edilmesi gerekmektedir. Eğer veri tabanını öğrenme konusunda MS Access gibi grafik arayüzü olan bir veri tabanı yönetim sisteminin bir adım ötesine geçmek istiyorsanız, SQLite iyi bir seçenek olacaktır.

Bulut Çözümler: Sunucu taraflı ihtiyaçların neredeyse tamamının bulut odaklı bir çözümü artık mevcut. Dilediğimiz veri tabanı yönetim sistemini kendi sunucumuza kurabileceğimiz gibi bir bulut servis sağlayıcıdan da ilgili hizmeti alabilir, kendi yazılımımıza webservis, soket ve benzeri teknolojilerle bağlayabiliriz. Adını saydığımı ve sayamadığımız birçok veri tabanı yönetim sistemi de yine bulut hizmet olarak karşımıza çıkmaktadır. Bulut hizmetlerden faydalanmanın çeşitli avantaj ve dezavantajları vardır. Öncelikle sağlayacağı çok sayıda faydaya göre elbette daha maliyetlidir. Ancak kurulum ve bakım süreçlerinin olmaması, anında hazır olması ve kolay entegre edilebilir olması gibi avantajlar oldukça cezbedici. Eğer sistemimizi tek bir sunucu üzerinde çalıştırıyorsak, bir süre sonra bu sunucunun kapasitesi kullanım oranına bağlı olarak zorlanmaya başlayabilir. Bu durumda da bulut hizmetler kullanarak bazı hizmetleri sunucu dışına taşımak maliyet yönetimi açısından faydalı olabilir. Küçük ve orta ölçekli işletmeler tek bir sunucu ile sistemlerini çalıştırabilseler de büyük ölçekte olanlar genellikle sistemin bileşenlerini farklı cihazlara bölmek mecburiyetinde kalabilirler. Dakikada binlerce istek alan bir internet sitesinin tek bir

sunucu ile hizmet vermesini beklemek gerçekçi olmayacaktır. Her halükarda bulut hizmetlerin hemen ya da ihtiyaç olduğunda kullanılabilecek güzel bir alternatif olduğu unutulmamalıdır.

2.5. Yapısal Sorgu Dili (SQL)

Yapısal Sorgu Dili (Structured Query Language, SQL), veri tabanı üzerinde sorgu ve komutlar çalıştırmaya yarayan özel bir dildir. Hiç karşılaşmamış olanlar için bir örnek sunalım ve sonrasında üzerine konuşmaya devam edelim:

SELECT ad, soyad, telefonNumarasi **FROM** kisiler **WHERE** yas > 30

Yukarıda yer alan SQL ifadesi, kisiler tablosu içerisinde yas sütunu değeri 30'dan büyük olan satırlardaki ad, soyad ve telefonNumarasi sütunlarının içeriğini getiren bir sorgudur. Daha önce bahsettiğimiz gibi, veri tabanları birden fazla tablodan meydana gelmektedir. Bu sebeple sorgulama yaparken hangi tablo ya da tabloları kullandığımızı belirtmemiz zorunludur. Where kelimesi koşulları belirlemek içindir ve kullanımı zorunlu değildir. Select altında ise hangi alanları istediğimizi belirleyebiliriz. Bu alan da isteğe bağlıdır. * işareti koyarak tüm alanları istediğimizi belirtebilir, verinin tümünün getirilmesini sağlayabiliriz. Benzer şekilde kisiler tablosundaki tüm kayıtları almak için şu sorgu yeterli olacaktır:

SELECT * FROM kisiler

Bu sorgu neticesinde hangi veri tabanı yönetim sistemini kullandığımızdan bağımsız olarak kişiler tablosunun içeriğinin tamamı ekrana bastırılır.

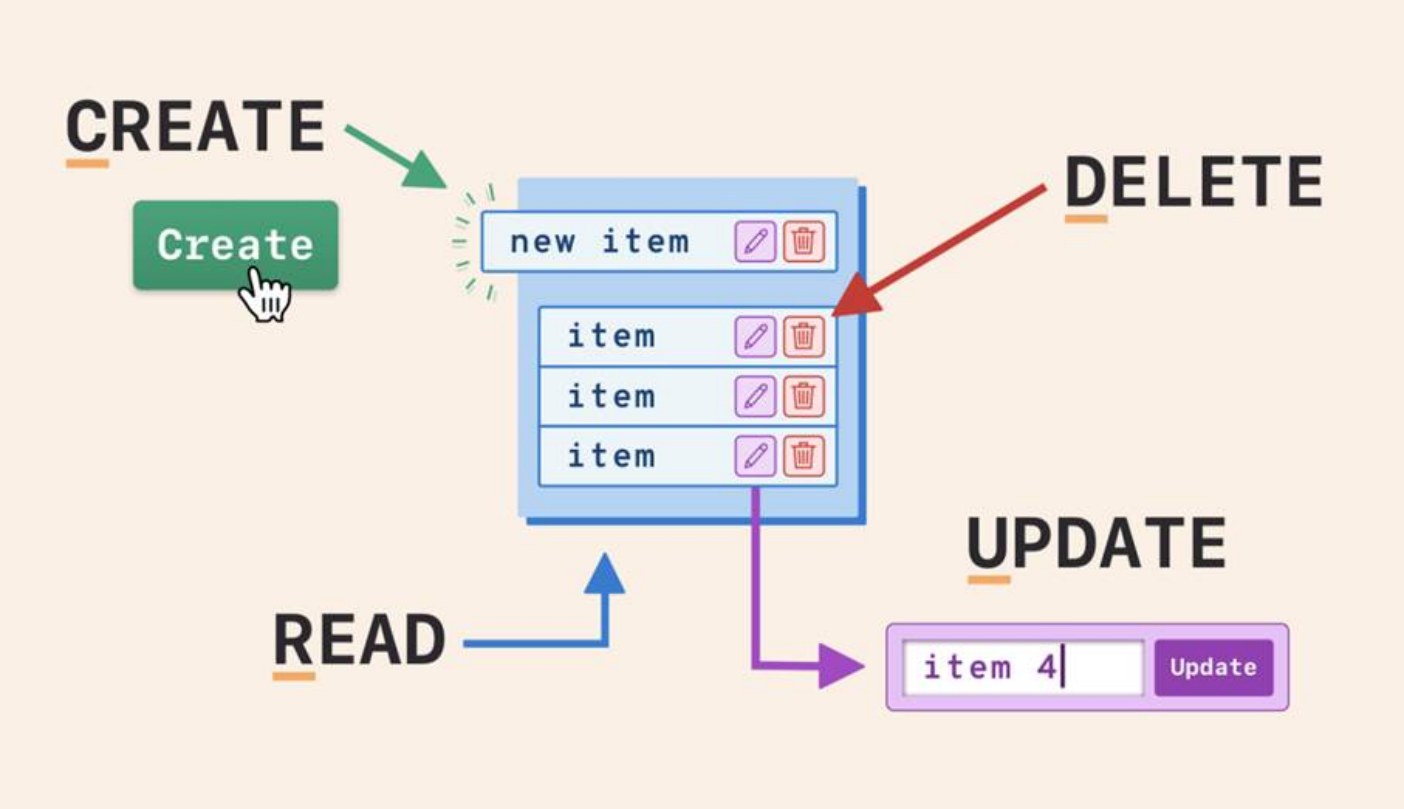
SQL evrensel bir dildir. Hangi veri tabanıyla çalıştığınızda bağımsız olarak sorgulama ve komutlarınızı SQL formatında vermeniz mümkündür. Bazı veri tabanı yönetim sistemleri, SQL üzerinde kendi özelleştirmelerini yapmaktadır. Ancak bu özelleştirmeler, standart SQL komutları üzerindeki değişiklikler değil, SQL'e yeni yetenekler kazandırmak üzeredir. Örneği Oracle, PL/SQL ile diğer veri tabanı yönetim sistemlerinde olmayan SQL komutlarını da yorumlayabilmektedir. Böylece SQL daha becerikli hale gelmektedir. MySQL içerisinde SQL ile zamanlanmış görevler tanımlamak mümkündür. Bu sayede istenilen zamanda ve istenilen şekilde tekrar eden SQL komutları oluşturmak ve bu görevi makineye bırakmak mümkündür.

SQL'in evrensel olduğundan bahsetmiştik. Bu evrensellik, SQL'in veri tabanı yönetim sistemlerinin dışına çıkmasını mümkün hale getirmiştir. Örnek vermek gerekirse veri bilimi çalışmalarında sıklıkla kullanılan R dili için geliştirilmiş olan sqldf paketi sayesinde, R dili ile kullanılan veri kümeleri (DataFrame) üzerinde SQL komutları çalıştırılabilmesi mümkündür. Herhangi bir veri tabanı olmadan doğrudan veri kümeleri üzerinde SQL komutu çalıştırmak, çok daha ayrıntılı sorgular ve sonuçlar elde etmeyi sağlamaktadır.

SQL bir programlama dili değildir. Ancak programlama dilleri ile veri tabanı arasında bir köprü görevi görebilir. Programlama dillerinin veri tabanları ile bağlantı kurmalarını sağlayan kütüphaneleri bulunmaktadır. Bu kütüphaneler programcının SQL komutları çalıştırmasını sağlarlar. Programın yazıldığı dil ne olursa olsun bir SQL sorgusu, kendi biçiminde yazılarak ilgili programlama dili içerisinde kullanılabilir. Kütüphane, SQL sorgusunu yetkilendirme için sağlanan bağlantı ayarlarını kullanarak veri tabanına iletir ve veri tabanının yanıtını bir fonksiyon yanıtı biçiminde döndürür. İletilen SQL sorgusu bir komut ise yanıt olumlu ya da olumsuz olabilir. Eğer iletilen SQL sorgusu veri üzerinde gerçekleştirilmesi istenilen bir sorgulamaysa yanıt olarak oluşturulan veri kümesi programlama dili içerisinde, kütüphane tarafından ilgili programlama diliyle işlenebilecek bir biçime getirilerek sunulur. Böylece programlama dili içerisinde bir fonksiyon çağırma ve bunun yanıtını alma kolaylığında veri tabanı bağlantısı sağlanabilir.

Bu ders kapsamında veri tabanı tasarımı daha çok odağımızda olacak. Bu sebeple SQL ile sadece tanışma seviyesinde çalışacağız. Gelecek dönem alacağınız veri tabanı yönetimi dersinde ise SQL kullanımı daha ön planda olacaktır. Şimdi temel SQL komutlarını ele alalım.

Veri yönetiminde temel 4 işlem bulunmaktadır. CRUD olarak terim haline getirilmiş bu işlemler grubu oluşturma (create), okuma (read), güncelleme (update) ve silme (delete) kelimelerinin ilk harflerinden meydana gelmektedir.



https://rohitchouhan.com/how-to-convert-your-mysql-database-into-crud-rest-api/

CRUD yapısı, kayıt işlemleri içeren bilgi sistemleri içerisinde sıklıkla karşılaşılan bir yapıdır. Herhangi bir kategoride tüm kayıtların listelenmesi (read), bu kayıtlarının her birinin yanında güncelleme (update) ve silme (delete) ikonlarının yer alması, tablonun sağ üst köşesinde yeni tanımla (create) butonunun bulunması, CRUD yapısının en sık karşımıza çıkma biçimlerinden biridir.

SQL veri tabanının oluşturulmasından başlayarak, tüm tabloların, değişkenlerin tipleriyle birlikte belirlenebildiği; veri yönetimi sürecinde birçok işlemi gerçekleştirebilmektedir. Bu ders kapsamında SQL'in yalnızca CRUD yapısı için gerekli komutlarını öğreneceğiz. En temel SQL komutu, SQL'in gerçek amacını da temsil eden ve zaten bir örnek sunmuş olduğumuz SELECT komutudur.

ÖNEMLİ

SQL komutları büyük ya da küçük harf fark etmeksizin çalışırlar. Ancak kod yazma etiği gereğince SQL komutları büyük harfler kullanılarak yazılmaktadır. Bu, hem programcılar arasındaki yazılı olmayan kurallardan biridir, hem de SQL sorgusunu okumaya çalışırken rahatlık ve hız sağlar. Bu sebeple SELECT komutunu bir yerde select şeklinde görürseniz lütfen şaşırmayın ancak siz SELECT biçimiyle kullanın.

SELECT komutunun kullanım şekli aşağıdaki gibidir:

SELECT [seçilen değişkenler]

FROM [tablo adı]

WHERE [koşullar]

… ;

Öncelikle burada alt satıra inmekle ilgili ayrıntıyı da verelim. SQL komutu içerisinde alt satıra inerek SQL komutunuzu çok satırlı yazabilirsiniz. Uzun SQL komutları için bu yöntem okunurluğu da arttıracağından sıklıkla kullanılır. Ancak tek satırda yazmanızda da bir mani yoktur. SQL komutlarının sonunda ";" işareti kullanılmalıdır ancak SQL ifadeniz tek bir komuttan oluşuyorsa kullanmamanız durumunda da hata almayacaksınız. Birden fazla SQL komutunu aynı anda veri tabanına göndermek isterseniz SQL komutunuzun bittiğini göstermek için ";" işaretiyle her bir SQL ifadesini birbirinden ayırmalısınız.

Verilen SQL şablonunda SELECT ve FROM bölümlerini kullanmak zorunludur. FROM yanında tablo adı, SELECT yanında ise seçilen değişkenler virgülle ayrılarak verilir. Tüm değişkenler için "*" işareti kullanılabilir. WHERE, zorunlu olmasa da sık kullanılan, sorgu içerisinde koşulların tanımlandığı alandır. Yine sık kullanılan ORDER BY komutu da sonuçların belirlenen kritere göre sıralanmasını sağlamaktadır.

CRUD yapısında oluşturma (create) bileşeni, SQL komutları arasında INSERT komutu bulunmaktadır. INSERT, veri tabanında ilgili tabloya kayıt girişini sağlar. Bir INSERT komutunu şablon biçiminde inceleyelim:

```
INSERT INTO [tablo adı] ( [veri girilecek değişken adları] )

VALUES ( [değişkenler için belirlenen değerler] ) ;
```

INSERT komutu INSERT INTO ile başlar, sonrasında hangi tabloya veri girişi yapılacağı bilgisini alır. Ardından parantez içerisinde, bu tabloda hangi değişkenlere değer ataması yapılacağı virgüllerle ayrılarak sunulur. Sonrasında VALUES komutu yanında yine parantez içerisinde virgüllerle ayrılmış şekilde, bir önceki parantezdeki değişkenlere denk gelen değerler verilir. SQL ifadesi çalıştırıldığında girilen değerlerle ilgili tabloya bir satır kayıt eklenir.

```
INSERT INTO kisiler (id, ad, soyad, telefonNumarasi)

VALUES ( 1, "Emre", "Akadal", "1234567" ) ;
```

Yukarıda bir INSERT sorgusu örneği bulunmaktadır. Değerlerin girildiği alanda id değeri için tırnak kullanılmadığı, diğer değerler için kullanıldığını fark etmişsinizdir. Bunun sebebi metinlerin programlama dili içerisinde tırnak içerisinde sunulması ancak sayısal değerlerin doğrudan kullanılabilmesi. SQL komutları içerisinde de sayısal değerler doğrudan kullanılabilir, metin değerler ise tırnak içerisinde verilebilir. Verilen örnekteki komut, kisiler tablosu içerisine id'si 1, ad değeri Emre, soyad değeri Akadal, telefonNumarasi değeri ise 1234567 olan bir kayıt eklemektedir.

CRUD yapısı içerisinde güncelleme için UPDATE komutu kullanılmaktadır. Komut, veri tabanı içerisinde bir tabloda bulunan bir kaydın istenilen değişkenlerinin güncellenmesini sağlamaktadır. UPDATE için bir şablon komut inceleyelim:

```
UPDATE [tablo adı]

SET [değişken = değer]

WHERE [koşullar] ;
```

Bu SQL komutunda UPDATE ve SET alanları zorunlu, WHERE alanı ise isteğe bağlıdır. UPDATE komutu yanında güncelleme yapılacak tablonun adı verilmektedir. SET komutu yanında ilgili tabloda hangi değişken için yeni değerin ne olacağı tanımlaması yapılmaktadır. WHERE komutu isteğe bağlı olsa da pratikte kullanımı gereklidir. Bunun sebebi genellikle bir tablonun sadece bir kısmının güncellenecek olmasıdır. Örnekle inceleyelim:

```
UPDATE kisiler

SET telefonNumarasi = "7654321"
```

Örnekteki SQL ifadesi kisiler tablosunda telefonNumarasi alanının 7654321 değeriyle güncellenmesini sağlamaktadır. Buradaki ayrıntı şudur: kisiler tablosu içerisinde kaç kayıt olursa olsun, bu SQL ifadesi çalıştırıldığında tüm kayıtlar için telefonNumarasi değeri güncellenir. Bu da genellikle istenilen bir durum değildir. Güncellemeler genellikle belirli bir koşula uyan kayıtların güncellenmesi üzerinedir. Dolayısıyla örneğimizi şu şekilde güncelleyebiliriz:

```
UPDATE kisiler
```

```
SET telefonNumarasi = “7654321”

WHERE id = 1
```

Güncellenmiş SQL komutumuzda, bir önceki SQL komutundaki güncellemenin aynısı yalnızca id değişkeni 1 değerine eşit kayıtlar için gerçekleştirilecektir. Bu sayede kişiler tablosu içerisinde id değeri 1 olan satırlarda telefonNumarasi 7654321 olarak güncellenmektedir.

CRUD yapısında yer alan son komut silme için kullanılan DELETE komutudur. Bu komut, verilen koşullara uygun kayıtları ilgili tablodan kaldırır. Şablonu inceleyelim:

```
DELETE

FROM [ tablo adı ]

WHERE [ koşullar ] ;
```

Bu SQL ifadesinde de DELETE ve FROM komutları zorunlu, WHERE ise isteğe bağlıdır. Ancak UPDATE komutuna benzer şekilde WHERE ifadesi kullanılmazsa tüm tablo etkileneceği için genellikle kullanılması gerekmektedir. SQL ifadesi; ilgili tabloda, ilgili koşullara uygun kayıtların tablodan kaldırılması gerekir. Örnek verelim.

```
DELETE

FROM kisiler

WHERE id = 1
```

Verilen örnekte kisiler tablosunda yer alan, id değeri 1 olan kayıtlar tablodan kaldırılmaktadır.

Bölüm Özeti

Veri tabanları, geleneksel dosya saklama yönteminin hiyerarşik ve ağ veri tabanlarıyla geliştirilmesi; sonrasında da ilişkisel veri modeli ve ilişkisel veri tabanının ortaya çıkması ile yaygın kullanılmaya başlanmıştır. İlişkisel veri tabanları üç motivasyon altında ortaya çıkmıştır. Bunlar;

· Fiziksel ve mantıksal beklentilerin kesin olarak birbirinden ayrılması,

· Herkesin anlayabileceği basit bir yapıya ayrılması,

· Üst seviye bir dil ile kontrol edilmesi

olarak sayılabilir.

İlişkisel veri tabanları önceki teknolojilere göre birçok avantajı bünyesinde barındırmaktadır. Bu avantajları şu şekilde özetleyebiliriz:

· Yapılandırılmış veri

· Veri tekrarından arınma

· Ölçeklenebilirlik

· İçerikten ve yöntemden bağımsızlık

· Veri yönetimi

· Entegrasyon

· Tutarsızlıktan kaçınma

· Çok kullanıcı ve farklı yetki seviyelerine uygun

Veri tabanı tasarımı, veri tabanı ilkelerine ve normal formlar adı verilen kurallar bütününe uygun olarak gerçekleştirilebilir. Teorik olarak veri tabanı tasarlama işi belirli adımlardan oluşan mekanik ve nesnel bir süreç gibi gözükse de öznel ve deneyime dayalı kararların oldukça etkili olduğu bir süreçtir. Bu sebeple iyi bir veri tabanı tasarımı gerçekleştirmek için tüm kural ve adımları çok iyi şekilde öğrenmek yeterli olmayabilir. Veri tabanı tasarlama işi daha önce bu süreci yaşamış ve hazırladığı tasarımın hangi durumlarda ne gibi değişiklikler gerektirdiğini deneyimlemiş kişilerin de ekip içerisine dahil edilmesiyle daha sağlıklı hale getirilebilir.

Veri tabanı yönetim sistemleri, bir ilişkisel veri tabanı oluşturmayı ve bunun üzerinde işlemler yapmayı sağlayan yazılım ve platformlardır. Çok sayıda veri tabanı yönetim sistemi mevcuttur. En çok bilinenleri; Microsoft Access, Microsoft SQL Server, MySQL, Oracle Database, Postgre SQL, MongoDB ve SQLite olarak sıralayabiliriz. Her veri tabanı yönetim sistemi farklı konularda avantajlar sağlayabilmektedir. Ancak temelde her biri SQL dili ile yönetilebilir. Bunun yanında bazıları arayüzlerle de kullanım imkanı sunmaktadır. Bazı veri tabanı yönetim sistemleri SQL diline ek özellikler de sağlamaktadır. Örneğin Oracle Database, PL/SQL ile prosedürler yazılması ve veri tabanı içerisinde çalıştırılabilmesini sağlamaktadır.

Yapısal Sorgu Dili (Structured Query Language, SQL) veri tabanları üzerinde sorgulama yapılabilmesini sağlayan ortak bir dildir. Bir programlama dili değildir. Veri tabanı üzerinde komutların çalıştırılabilmesini sağlar. Neredeyse tüm programlama dilleri ile birlikte kullanılabilir. Bu sebeple programlama dilleri kolaylıkla veri tabanı yönetim sistemleri ile bağlantı kurabilir. Bazı diller bazı veri tabanı yönetim sistemleriyle daha sık kullanılsa ve daha uyumlu olsa da gerekli ayarlamalarla çapraz bağlantılar mümkündür.

CRUD yapısı oluşturma (create), okuma (read), güncelleme (update) ve silme (delete) fonksiyonlarının birlikte anıldığı bir olgudur. SQL'de de bu dört özellik sık ve öncelikli kullanılmaktadır. Bu ders kapsamında SQL'in yalnızca bu özelliklerine değinmekteyiz.

SQL'de oluşturma için INSERT, okuma için SELECT, güncelleme için UPDATE ve silme için ise DELETE komutlarından faydalanılır. Her bir SQL ifadesi içerisinde FROM komutu da mevcuttur. FROM komutu hangi tablo ile işlem yapılacağını seçmemizi sağlar. İsteğe bağlı olarak WHERE komutu da sıkça kullanılmaktadır. Hazırladığımız sorgu içerisinde koşulları belirlediğimiz alan WHERE alanıdır.

Kaynakça

Akadal, E. 2017. Ham verilerin genetik algoritmalarla ilişkisel veritabanlarına dönüştürülmesi ve bir uygulama. İstanbul Üniversitesi Fen Bilimleri Enstitüsü. Doktora Tezi.

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

Codd, E. F., 1969. Redundancy and consistency of relations stored in large data banks. SIGMOD Rec., 17-36.

Codd, E. F., 1970. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387.

Codd, E. F., 1982. Relational database: a practical foundation for productivity. Communications of the ACM 25.2, 109-117.

Demirağ Çakıcı, E., & Yılmaz, K. G. 2021. Uluslararası Pazarlarda Hedef Pazar Seçimi Üzerine Bir Araştırma. Sosyal, Beşeri Ve İdari Bilimler Dergisi, 4(9), 833–849. https://doi.org/10.26677/TR1010.2021.801

Nizam, A., 2011. Veritabanı Tasarımı: İlişkisel Veri Modeli ve Uygulamalar, Papatya Yayıncılık Eğitim.

Raghu, R. & Johannes, G., 2000. Database management systems. McGraw-Hill.

Read, R., Fussell, D. & Silbertschatz, A., 1992. A multi-resolution relational data model. Austin, Texas: Department of Computer Science, University of Texas at Austin.

Sumathi, S. & Esakkirajan, S. 2007. Fundamentals of relational database management systems, volume 47.

Zeng, Q., Cao, Q., Zhu, X., & Author, C. 2010. A Complex XML Schema to Map the XML Documents of Distance Education Technical Specifications into Relational Database Xin-hua A Complex XML Schema to Map the XML Documents of Distance Education Technical Specifications into Relational Database. Citeseer. https://doi.org/10.4156/jdcta.vol4

Ünite Soruları

Soru-1 :

Elde edilen veri kümesinin uygulanacak işlemlere uygun hale getirilmesi süreci hangisidir?

(Çoktan Seçmeli)

(A) Oluşturma

(B) Veri ambarı

(C) Ön işleme

(D) Kırpma

(E) Normalizasyon

Cevap-1 :

Ön işleme

Soru-2 :

Hangisi bir veri tabanı türü değildir?

(Çoktan Seçmeli)

(A) Geleneksel dosya temelli saklama

(B) Hiyerarşik

(C) Ağ

(D) İlişkisel

(E) Karşılıklı

Cevap-2 :

Karşılıklı

Soru-3 :

İlişkisel veri tabanının alt yapısı olan ilişkisel veri modeli kim tarafından önerilmiştir?

(Çoktan Seçmeli)

(A) Codd, 1969

(B) Goldberg, 1970

(C) Holland, 1920

(D) Satman, 1999

(E) Norman, 1970

Cevap-3 :

Codd, 1969

Soru-4 :

Hangisi ilişkisel veri tabanının ortaya çıkış motivasyonlarındandır?

(Çoktan Seçmeli)

(A) Bir üst seviye dil elde edilmesi

(B) Hızlı çalışması

(C) Verinin birleştirilmesi

(D) İnternet ortamında çalışması

(E) Daha az işlemci gücü harcaması

Cevap-4 :

Bir üst seviye dil elde edilmesi

Soru-5 :

İlişkisel veri tabanları için önerilmiş sorgu dili hangisidir?

(Çoktan Seçmeli)

(A) Python

(B) PHP

(C) SQL

(D) Java

(E) SQLite

Cevap-5 :

SQL

Soru-6 :

Hangisi ilişkisel veri tabanı avantajlarından değildir?

(Çoktan Seçmeli)

(A) Hızlı olması

(B) Tutarsızlıktan kaçınması

(C) Az yer kaplaması

(D) Az işlem gücü gerektirmesi

(E) Çok az veri türünü desteklemesi

Cevap-6 :

Çok az veri türünü desteklemesi

Soru-7 :

Hangisi veri tabanı yönetim sistemlerinden biri değildir?

(Çoktan Seçmeli)

(A) Microsoft SQL Server

(B) SQLite

(C) Oracle Database

(D) Microsoft Word

(E) Postgre SQL

Cevap-7 :

Microsoft Word

Soru-8 :

Oluşturma, okuma, güncelleme ve silme fonksiyonlarını ifade eden terim nedir?

(Çoktan Seçmeli)

(A) CREATE

(B) UPDATE

(C) DELETE

(D) CRUD

(E) READ

Cevap-8 :

CRUD

Soru-9 :

SQL’de yeni kayıt ekleme komutu hangisidir?

(Çoktan Seçmeli)

(A) SELECT

(B) UPDATE

(C) INSERT INTO

(D) DELETE

(E) CRUD

Cevap-9 :

INSERT INTO

Soru-10 :

SQL’de koşulların verildiği komut hangisidir?

(Çoktan Seçmeli)

(A) SELECT

(B) CRUD

(C) WHERE

(D) UPDATE

(E) DELETE

Cevap-10 :

WHERE

---

3. GÖSTERİM ŞEKİLLERİ VE TERİMLER

Bölümle İlgili Özlü Söz

Kelimeler (ya da işaretler), onları doğru kullanırsanız X ışınları gibi olabilirler. Her şeyin ötesine geçebilirler.

Aldous Huxley

Brave New World

Kazanımlar

1. Veri tabanıyla ilgili temel kavramlara hakim olur.

2. Veri tabanıyla ilgili gösterim yöntemlerini bilir.

3. Kaz Ayağı gösterim yöntemine hakim olur.

4. Fonksiyonel bağımlılık gösterimine hakim olur.

5. Veri kümesi, veri ambarı, tablo, nitelik, kayıt ve benzeri kavramlarını bilir.

Birlikte Düşünelim

Bilgisayar programlama sürecinin tamamı bilgisayar ortamında mı gerçekleştirilir?

Veri tabanı tasarımı dersi için “tasarım” kelimesi özellikle seçilmiştir. Tasarım kelimesini kullanınca ne hayal ediyorsunuz? Tasarım süreci doğası gereği hangi faaliyetleri barındırır?

Bir veri tabanı tasarımı gerçekleştirmekle sanat eseri tasarlamak arasında nasıl benzerlikler bulabiliriz?

Veri tabanı tasarımında “bilgelik” yeterli midir?

Bir veri tabanının performansı nasıl ölçülebilir?

Başlamadan Önce

Yazılım geliştirme -eğer hobi projesi değilse- bir ekip işidir. Aynı zamanda genellikle kısa bir süreç de değildir. Yazılım geliştirme süreci boyunca ekibin hangi amaç ve motivasyonla yazılım geliştirdiği ve önceliklerin neler olduğunu; yol haritasının ne şekilde tanımlandığının bilinmesi oldukça önemlidir. Bunun için de yazılım projelerinde sistem analizi ve tasarımı ile buna bağlı olarak dokümantasyon süreçleri oldukça önemlidir. Bir yazılımcı için dokümantasyon oluşturma ve inceleme zaman zaman sıkıcı olabilse de projenin sağlıklı yürütülmesi açısından gereklidir.

Veri tabanı tasarımı da sistem analizi ve tasarımı sürecinin bir parçasıdır. Bu sebeple henüz bilgisayar ortamında geliştirmeye başlanmadan önce yapılan çeşitli faaliyetler içerir. Bu faaliyetler neticesinde de bir dokümantasyon elde edilir. Elde edilen çıktının tüm proje ekibi ve potansiyel diğer ekip üyeleri tarafından rahatlıkla anlaşılabilir ve uygulanabilir olması gerekmektedir. Bunu sağlamak için ortak kullanılan çeşitli kavram ve diyagramlar bulunmaktadır. Bu bölüm içerisinde temel kavramların anlamları ve gösterim şekilleriyle ilgili temel bilgileri edinebilecek; bu tür teknik dokümanları oluşturmak ve incelemek için gerekli altyapıyı edinmiş olacaksınız.

3.1. Terimler

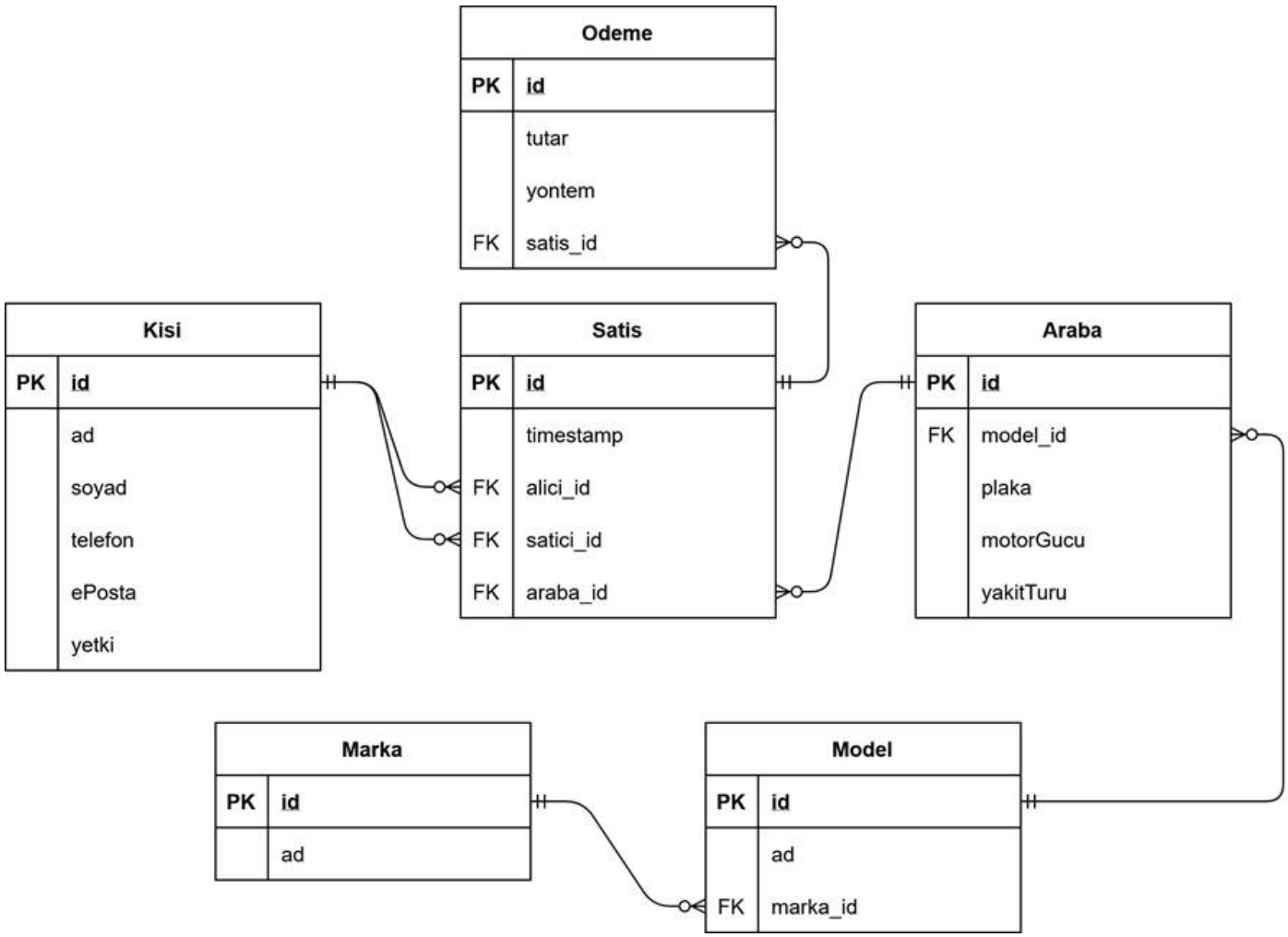
Bir bilişim sistemini hayata geçirme süreci baştan aşağıya sinerji içermektedir. Her bir mikro sürecin çıktısı diğer süreçlere girdi olmaktadır. Tüm süreçler birbirlerini beslerler ve süreçler tamamlandığında ihtiyaç olunan sistem hayata geçmiş olur. İhtiyaç duyulan sistemin geliştirilmeye başlanılmasından önceki geniş hazırlık sürecini sistem analiz ve tasarım süreci olarak ele alıyoruz. Veri tabanı tasarlama süreci de sistem analizi ve tasarımı sürecine dahil bir mikro süreç olarak karşımıza çıkıyor. Bilişim teknolojilerin deyince tüm sürecin bilgisayar ortamında gerçekleştirildiği düşünülüyor olabilir. Ancak yazılımın geliştirme adımına gelene kadar gerçekleştirilen faaliyetler, temelde bir tasarım sürecinin parçaları olduğu için bu süreç daha çok iletişime ve dokümantasyona dayalıdır. Bu aşamada süreç teknik anlamda bilgisayarda uzakta gerçekleştirilir. Elbette iletişim ve dokümantasyon için de bilgisayar kullanılır ancak burada teknik anlamda diyerek vurguladığımız konu, programlama ve gelişmiş araçların kullanımıdır.

Tasarım sürecinin çıktısı, analiz ve tasarım süreci boyunca elde edilen bulguların belgelendirilmesidir. Bu belgeler, tüm geliştirme, test, iyileştirme süreçlerinde kullanılacak; sistemle ilgili gelecekte yapılacak iyileştirmeler için de yol gösterici olacaktır. Bu sebeple her ne kadar sistem analizi ve tasarımı yapan ekiple geliştirme yapan ekip doğrudan iletişimde olsa da elde edilen dokümante çıktılar evrensel geçerlilikte olmalıdır. Elde edilen bulgular, ekibin değişmesi durumunda, yeni ekip tarafından hızlıca anlaşılabilir nitelikte olmalıdır. Bu da aynı dili kullanmakla mümkündür. Bulguları dokümante edenler ile geliştiriciler aynı teknik terimleri kullanmalıdırlar. Bu sayede hazırlanan belgeler genel bir geçerliliğe sahip olur ve ekiplerdeki değişiklik belgelerin içeriğinin kalitesinde bir düşüşe sebep olmaz.

Bu dersin sonunda siz de bir veri tabanı tasarımcısı olacağınız için bu dokümanları oluşturabilme ve oluşturulmuş olanları gerektiği şekilde inceleyebilme becerisine sahip olmanız gerekmektedir. Bu başlık altında bir veri tabanı tasarım dokümantasyonu yapmak için gerekli terimler ve gösterim şekillerini ele alacağız.

Yeni öğrenilen bir olguyla ilgili terimlerin akılda tutulması biraz zorlayıcı olabilir. Bu sebeple her bir tanımı, şekli olarak da göstererek sunmak; öğrenme aşamasını kolaylaştıracaktır.

Dersin başından beri bahsediyor olsak da veri tabanı kavramıyla başlayalım. Veri tabanları, veri kümelerini yapılandırılmış biçimde, birbirleriyle ilgili sütunların gruplanarak farklı tablolara bölünmesi ve çeşitli ilişki türleriyle birbirlerine bağlanarak ihtiyaç halinde veriyi oluşturulan sorgular çerçevesinde hazırlayarak geri sunan yazılımsal altyapıdır. Bu tanım içerisinde yer alan olguları da teker teker inceleyeceğiz. Eğer bu tanım size yeterince anlamlı gelmediyse lütfen diğer tanımları inceledikten sonra tekrar bu tanımı inceleyiniz.



Veri kümesi ifadesi günlük hayatta veri seti (ya da veriseti) şeklinde de karşımıza çıkan, “dataset” kelimesinin Türkçe karşılığıdır. Bunu bir Microsoft Excel dosyası gibi hayal edebiliriz. Bir elektronik tablo içerisinde, ilk satırda sütunların hangi veriyi taşıdıklarını belirttikleri bir etiket, diğer satırlarda ise her bir satır ayrık bir kayıt olmak üzere, sütunlarda ilgili etikete denk gelen veriler bulunmaktadır. Veriyi saklamanın ve iletmenin geleneksel ve en basit yöntemlerinden biri veri kümesi kullanmaktır. En yüksek performansa sahip olmasa da kolaylık sağladığı için sık tercih edilen veri saklama ve aktarma türüdür.

Name	*Age*	*Role*
Sarah	19	Student
Janine	43	Professor
William	27	Associate Professor

Zaman zaman veri ambarı kavramıyla karşılaşmanız da mümkündür. “Data warehouse” teriminin Türkçe karşılığı olan veri ambarı, kısa tanım olarak merkezi bir veri tabanıdır. Biraz daha açıklamak gerekirse;

birden fazla veri kaynağına erişimimiz varsa ve tüm veri kaynaklarını birleştirerek tek bir noktada enformasyon süreçlerini işletmek istiyorsak; tüm bu verilerin entegre olabilecekleri bir yapıda bir araya getirilmeleri gerekmektedir. Veri kaynaklarını bir araya getirdiğimiz yer veri ambarıdır.

Bir veri kümesini veri tabanına dönüştürmek için birbirleriyle ilgili sütunları gruplayarak birbirleriyle ilişkili birden fazla grup oluşturma sürecinden bahsetmiştik. Elde ettiğimiz her bir sütun grubuna tablo ya da varlık adı (table/entity) verilmektedir. Her bir veri tabanı tablosunu kendi içerisinde bir veri kümesi ya da elektronik tablo gibi değerlendirebilirsiniz. Tablolar basit anlamda satır ve sütunlardan oluşan ve veri içeren yapılardır. Veri tabanları, tablolara ek bazı özellikler katarlar. Bu da hem tabloların veri kümelerden farklılaşmasını hem de diğer tablolarla entegre olup veri bütünlüğünü sağlayabilmesine olanak sunarlar.


tablo

Şekilde Kullanici adında bir veri tabanı tablosunun gösterimini incelemektesiniz. Şekle göre şu bilgileri elde edebiliriz:

· Tablonun adı Kullanici'dir.

· Tablo 4 sütun içermektedir. Bunlar: id, ad, soyad, dogumYili

· Tabloda ayırt edici sütun id sütunudur. (PK işareti sayesinde bu yorumu yapabildik)

Veri tabanı tablosunun bu şekilde gösterilmesi durumunda içerisinde yer alan kayıtlarla ilgili bilgi sahibi olmak mümkün değildir. Ancak gerçekte, örnekteki gibi tarif edilen bir veri tabanı tablosunu göz önüne aldığınızda; id, ad, soyad ve dogumYili sütunlarından oluşan, içerisinde kayıtlar içeren bir veri tablosunu düşünebilirsiniz.

Şimdiye kadar "sütun" kavramı, elektronik tablolara aşina olmamız sebebiyle tercih edilmiştir. Ancak veri tabanları söz konusu olduğunda sütun olarak adlandırdığımız yapıya farklı adlar verildiğini de görmekteyiz. Bunlar sütun (column), nitelik (attribute) ve değişken (variable)'dir. Farklı kaynaklarda farklı şekillerde ele alınan bu terimlerin tümü, her bir tabloda yer alan dikey içeriği yani sütunu ifade etmektedir. Bu kitap ve ders kapsamında nitelik kelimesine öncelik verilecektir. Ancak yine de zaman zaman diğer karşılıkları da görmeniz mümkün. Bir ham veri kümesi basit bir şekilde sütunlardan ve bu sütunlara kayıtlar eklemek üzere satırlardan oluşmaktadır. Tüm verinin tek bir tablo şeklinde kaydedilmesi durumunda sütunlarla ilgili bir organizasyona ihtiyaç kalmaz. En fazla sıralamalarıyla ilgili değişiklikler yapılabilir ancak bu da verinin bütünlüğüne zarar verme konusunda bir risk barındırmaz. Ancak veri tabanı söz konusu olduğunda niteliklerin organizasyonu, veri tabanı tasarımının önemli bir parçası haline geliyor. Hangi niteliklerin birbirleriyle ilişkili olduğunu belirlemek, ilişkili nitelikleri gruplamak ve her bir grubu bir veri tabanı tablosu olarak tanımlamak sürecin önemli bileşenlerindendir.

Her bir tablo içerisinde yer alan satırı şimdiye kadar kayıt adıyla andık. Türkçe olarak kayıt adını kullanmaya devam edecek olsak da İngilizce dilindeki teknik karşılığının "tuple" olarak kullanıldığını bilmekte fayda var. Böylece her bir tablo/varlık içerisinde sütun/nitelik/değişkenlere karşılık kayıtlar bulunduğunu söyleyerek tüm terimleri bir arada tekrar kullanabiliriz.

Tablo ve nitelik adlarıyla ilgili, zorunlu olmayan ancak tüm geliştiriciler tarafından uyulan bazı kurallar bulunmaktadır. Öncelikle tablo adı, nitelik adı ve programlama içerisinde değişken adı, fonksiyon gibi sistematik bileşenlerin adlarında Türkçe karakter kullanılmaktan kaçınılmaktadır. Türkçe karakter için belirli karakter setlerinin kullanılması gerekliliği zaman zaman problemlere yol açtığı için artık bazı dillerde problem yaşanmasa da Türkçe karakter kullanmamak klasik bir yaklaşım haline geldi. Türkçe kelimelerden Türkçe karakterler kullanmadan faydalanılabileceği gibi doğrudan İngilizce karşılıklarını kullanmak da sık başvurulan bir yöntemdir.

Veri tabanlarında tabloların, programlamada ise sınıfların adları büyük harfle başlar ve eğer birden fazla kelimeden oluşuyorsa her kelimenin ilk harfi büyük seçilir.

Kullanicilar

SatisKayitlari

GecisBilgileri

uygun tablo adı seçimine birer örnek teşkil etmektedir.

ÖNEMLİ

Türkçe dilinin kullanımı konusunda hepimiz hassasiyetlere sahibiz. Bu sebeple burada yer alan yaklaşımlar eleştiriye sebep olabilir. Lütfen burada olayı teknik terim boyutuyla ele aldığımızı ve kabul gören yaklaşımları sunduğumuzu göz ardı etmeyelim. Bununla birlikte her ne kadar Türkçe ad kullanma yöntemini sunuyor olsak da gelecekte uluslararası projelerde de yer alabileceğinizi ve projenizi açık kaynak kodlu hale getirirseniz dünyanın her tarafından farklı insanların kodunuzu incelemek isteyebileceğini unutmayınız. Bu sebeple tüm anahtar kelime seçimlerinde programlamada ortak dil olan İngilizce kelimeleri seçmenizi tavsiye ederim.

Veri tabanları içerisindeki nitelik adları ve programlama dillerinde değişkenler ve fonksiyon adları küçük harfle başlar, birden fazla kelime içermesi durumunda her bir kelime büyük harfle başlar.

kullanici

erisimHakkiVarMi

sonGorulme

Eğer bir niteliği oluşturan kelimeler bir bütün değil de bir beraberliği ifade ediyorsa “_” işareti ile bağlanabilir. Örneğin bir kullanıcı ile o kullanıcıya ait ders bilgisi “kullanici_ders” şeklinde gösterilebilir. Bazı geliştiriciler birden fazla kelime içeren ifadeleri de “_” kullanarak belirtmektedirler. Örneğin “son_gorulme” de uygun bir kullanımdır.

Ham veri kümelerinin tek bir elektronik tablo yapısında bulunduğundan bahsetmiştik. Bu durumda satırlar ve sütunlar doğallıkla birbirleriyle bağlantılı oldukları için verinin bütünlüğünü tehdit eden herhangi bir unsur bulunmamaktadır. Ancak veri tabanı söz konusu olduğunda nitelikler gruplanmakta ve birden fazla tablo elde edilmektedir. Verinin birden fazla tabloya bölünmesi, normal şartlar altında veri bütünlüğünü tehlikeye sokmaktadır. Ancak veri tabanı yapısında tüm tablolar ve kayıtlar anahtarlar sayesinde veri bütünlüğünü korumaya devam etmektedirler. Anahtarlar, kayıtları temsil eden benzersiz değerler alan niteliklerdir. Birincil ve ikincil adı verilen iki anahtar türü bulunmaktadır.

Birincil anahtarlar (Primary Key, PK), bir tabloda her bir kaydı eşsiz olarak temsil eden değerleri içeren niteliğin özelliğidir. İkincil anahtarlar (Secondary/Foreign Key, FK) ise bir birincil anahtarın başka bir tablo içerisinde verileri birleştirme amacıyla yer alması durumunda sahip oldukları özelliktir. İlgili bölüm altında anahtarları ayrıntılı inceleyeceğiz ancak daha iyi kavramak adına bir örnek verebiliriz. Günlük hayatta sıkça kullandığımı kimlik numaralarımız birincil anahtardır. Kimlik numarası bir kişi hakkındaki tüm kayıtları temsil eden eşsiz bir değerdir (PK). Kimlik numarasının farklı bir tablo içerisinde bir kişiyi işaret etmek için kullanılması da mümkündür (FK).

3.2. Gösterim Şekilleri

Önceki başlıkta genel olarak kavramlardan bahsettik. Şimdi sistem tasarımı sürecinin ayrılmaz bir diğer bileşeni olan görsel gösterimleri ele alacağız. Elbette görsel gösterimler de tasarlanan sistemin belgelendirilmesi kadar önem taşımaktadır. Aynı zamanda yine teknik terimlerde olduğu gibi görsellerin de global geçerliliğe sahip olması önemlidir. Bir dokümandaki bir şema üzerinde yer alan en ufak bir işaretin, hem dokümanı hazırlayan, hem geliştiren, hem de gelecekte bu dokümanı inceleme potansiyeli olan ekiplerin aynı anlamı verecekleri bir yapı elde etmek gerekmektedir.

Sözü edilen motivasyon altında zaman içerisinde çeşitli gösterim şekilleri oluşturulmuştur. Bu gösterim şekillerinden sık bilinenleri; Chen, Kaz Ayağı (Crow’s Foot) ya da Martin, Merise, IDEF1X ve EXPRESS’tir (Bachman, 1969; Gogolla, 1991; Menzel & Mayer, 1998, Purchase & diğ., 2004; Nizam, 2011). Hangi

yöntem tercih edilirse edilsin, global bilinirliğe sahip bir yöntem tercih edildiğinde hazırlanan tasarım dokümanının geçerliliği ilgili yöntemin geçerliliğiyle orantılı olacaktır.

Bir tablo içerisinde yer alan nitelikler arasındaki fonksiyonel bağımlılıkları göstermek için de bir yöntemimiz bulunmaktadır. Fonksiyonel bağımlılık konusunu bir sonraki bölümde ayrıntılı inceleyeceğiz. Şimdilik hangi niteliklerin diğerlerini temsil edici özellikte olduğunu göstermemiz şeklinde özetleyebiliriz bu terimi. Fonksiyonel bağımlılıkta bir nitelik kümesinin başka bir nitelik kümesi tarafından temsili dokümante edilecektir. Notasyonun şablonu şu şekildedir:

Anahtarlardan bahsederken kullandığımız kimlik numarası örneğini burada tekrar ele alalım:

Burada kimlik numarasının ad ve soyad için temsil niteliği taşıdığı bilgisini edinmekteyiz. Gösterimin sağında ya da solunda tek bir nitelik olması durumunda { … gösterimini kullanma zorunluluğumuz bulunmamaktadır. Örnekte sol tarafta sadece kimlik numarası olduğu için doğrudan yazdık ancak sağ tarafta iki niteliğimiz olduğu için parantezli notasyonu kullandık. Bir varlığı metin olarak göstermek istersek aşağıdaki şablonu uygulayabiliriz:

Kullanici: kimlikNumarasi, ad, soyad

Bu notasyon kimlikNumarasi, ad ve soyad niteliklerinden oluşan Kullanici varlığını işaret etmektedir. Ayrıca kimlikNumarasi niteliğinin altı çizili olması, bu niteliğin temsil edici özelliğe sahip olduğunu vurgulamaktadır.

Bu ders kapsamında veri tabanı ile ilgili gösterimlerimizde Kaz Ayağı yönetimini kullanacağız. Bir sonraki alt başlıkta kaz ayağı yöntemini kullanarak çeşitli veri tabanı olgularını nasıl göstereceğimizi öğreneceğiz.

3.3. Kaz Ayağı

Dersin önceki bölümlerinde bir veri tabanı tablosunun nasıl gözüktüğüyle ilgili örnek vermiş ancak Kaz Ayağı gösteriminden bahsetmemiştik. Ders içerisinde daha önce kullandığımız gösterimler de Kaz Ayağı gösterimine uygundur.


tablo

Şekilde, Kullanici adlı veri tabanı tablosunun gösterimi mevcuttur. Kutunun üst kısmında tablo adı, altında tabloda bulunan niteliklerin bir listesi, niteliklerin solunda ise varsa niteliklere ait anahtar özellikleri belirtilmektedir. Anahtar özellikleri birincil anahtar için PK, ikincil anahtar için FK ifadeleriyle gösterilmektedir.

Birden fazla veri tabanı tablosunun aynı anda gösterimi için aşağıdaki örneği inceleyebiliriz.

Burada Kullanici ve Malzeme adında iki tablonun yer aldığını görmekteyiz. İki tablo için de nitelikler listelenmiştir. Kullanici tablosu için id niteliği birincil anahtarken, Malzeme tablosu için id birincil, kullanici_id ise ikincil anahtar olarak tanımlanmıştır. Ayrıca iki tabloyu birbirine bağlayan bir eğri olduğunu görmekteyiz. Bu işaret, iki tablonun birbirleriyle hangi nitelik üzerinden ve hangi türde ilişkili olduğunu göstermektedir. Görüldüğü üzere Kullanici ve Malzeme tabloları, Kullanici tablosu üzerinde id, Malzeme tablosu üzerinde ise kullanici_id nitelikleri üzerinden birbirleriyle bağlantılıdır. Bağlantının tablolara temas ettiği noktalarda bazı işaretlerin yer aldığını görmektesiniz. Şimdi bu işaretlerin anlamlarına odaklanalım.

Şekilde (a) gösterimi tek, (b) gösterimi ise çok bağlantı anlamındadır. Gösterim, kurulan bağlantıda, bağlanılan tabloda en fazla kaç kayıt ile bağlanılabildiğini göstermektedir. Örneğin Kisi ve Satis tablolarını

düşünelim. Bu iki tablo arasındaki bağlantı, Kisi tablosuna tek, Satis tablosuna ise çok işaretiyle bağlanacaktır. Bunun sebebi, bir kişinin birden fazla satışta yer alma ihtimali olması ancak bir satışın yalnızca ve en fazla bir kişiyle bağlantılı olmasıdır. Bu sebeple Kisi'den Satis yönüne çok, Satis'tan Kisi yönüne tek bağlantı kurulur.

Bu örnekte ise (a) ve (b) ilişki gösterimlerinin ikisinin de çok türünde olduğunu görmektesiniz. Ancak iki gösterimin yanında da 1 ve 0'a benzeyen işaretler görüyorsunuz. Bu işaretler, bağlantı kurulan tabloda en az kaç kayıt bulunması gerektiğini göstermektedir. Yine Kisi ve Satis tablolarını ele alalım. Bir kişi hiçbir satış kaydıyla eşleşmeyebilir. Ancak bir satış kaydı mutlaka bir kişiyle eşleşmelidir. Bu sebeple bağlantının Kisi tablosuna bağlanan kısmı zorunluluğu ifade eden 1 gösterimiyle, Satis tablosuna bağlanan kısmı ise zorunluluk olmadığını ifade eden 0 gösterimiyle kullanılacaktır. Burada odaklandığımız konu gösterim şeklidir. İlişki türleri ve nasıl tanımlanması gerektiğiyle ilgili konular gelecek bölümlerde tekrar ele alınacaktır.

Bölüm Özeti

Yeni bir sistem tasarlanması aşamasında tasarlanan yapının dokümante edilmesi oldukça önemlidir. Hem analiz ve geliştirme ekiplerinin "aynı dili" konuşabilmesi, bilgi aktarabilmesi, zaman içerisinde bilginin kaybolma riskinin ortadan kaldırılması gibi avantajlar hem de projenin gelecekte farklı ekipler tarafından devralınması durumunda herhangi bir bilgi kaybı yaşanmadan faaliyetlerin sürdürülebilmesi açısından dokümantasyon değerli bir süreçtir.

Veri tabanı tasarımında da bilgisayar ortamında uygulamaya geçmeden önce gerçekleştirilmesi gereken eylemler ve bunların dokümante edilmesi ihtiyacı bulunmaktadır. Oluşturulacak dokümanın global geçerliliğe sahip olması için bilinmesi ve uyulması gereken çeşitli kurallar bulunmaktadır. Bu bölüm içerisinde çeşitli genel terimler ve gösterim şekilleri ifade edilmiştir. Bu terimleri kısaca gözden geçirelim:

Veri kümesi: Bir veri parçasının elektronik tablo biçiminde satırlar ve sütunlar şeklinde gösterilmesidir. Genellikle MS Excel ve CSV formatlarında karşımıza çıkar.

Veri ambarı: Birden fazla veri kaynağından elde edilen verinin tek bir merkezde toplanarak, enformasyona dönüştürülmek üzere birlikte işlenmesidir.

Tablo: Veri tabanı içerisinde nitelikler birbirleriyle gruplandırılırlar. Bu sayede tek bir elektronik tablo biçimindeki veri kümesi, çok sayıda tabloya dönüşür. Veri tabanı için bu yapılara tablo (table) ya da varlık (entity) adı verilmektedir.

Sütun: Her bir tablo içerisinde yer alan dikey veri yapılarına sütun (column), nitelik (attribute) ya da değişken (variable) adlarının verildiği görülmektedir.

Kayıt: Tablolarda yer alan her bir satır için kayıt (tuple) kelimesi kullanılmaktadır.

Bilgisayar sistemleri açısından bir hataya sebep olmasa da tablo ve nitelik adlarının yazımında uyulan çeşitli kurallar bulunmaktadır. Aynı zamanda bu kurallar programlama da geçerlidir. Bir veri tabanı tablosu ya da programlamada bir sınıfın adı büyük harfle başlar. Eğer birden fazla kelime içeriyorsa her kelime büyük harfle başlar (BuBirTabloAdidir). Bir nitelik, programlama değişkeni ya da fonksiyon adı ise küçük harfle başlar ve birden fazla kelime için her kelime büyük harfle başlatılır (buBirNitelikAdidir).

Eğer bu adlarda farklı olgular ele alınıyorsa (kişi ve ders gibi) kelimeler küçük harfle başlayabilir ve "_" işaretiyle birbirlerinden ayrılabilirler (kisi_ders). Ayrıca seçilen adlarda Türkçe karakterlere yer verilmez.

Bir tabloyu metin olarak tarif etmek için aşağıdaki şablon kullanılabilir.

TabloAdi: anahtarNitelik, nitelik1, nitelik2, …

Bir tablo içerisinde yer alan niteliklerin birbirleriyle olan ilişkisini göstermek için fonksiyonel bağımlılık gösterimine ihtiyaç bulunmaktadır. Fonksiyonel bağımlılık, nitelikleri bir küme grubu olarak

düşündüğümüzde, hangi kümenin diğerini temsil ettiğinin gösterilmesidir diye özetleyebiliriz. Gösterimi şu şekildedir:

Bu gösterimde 1 ve 2 indisli nitelikler, 3, 4 ve 5 indisli nitelikler için temsil edici niteliktedirler.

Ders boyunca veri tabanı tasarımlarımızın gösteriminde Kaz Ayağı gösterim yönteminden faydalanacağız. Literatürde Kaz Ayağı (Crow's Foot / Martin)'nın yanında IDEF1X, Bachman, Barker, Merise ve EXPRESS yöntemlerinin de yer aldığını bilmekte fayda bulunmaktadır.

Kaynakça

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

Bachman, C. W. 1969. Data structure diagrams. ACM SIGMIS Database, 1(2), 4–10. https://doi.org/10.1145/1017466.1017467

Gogolla, M., & Hohenstein, U. 1991. Towards a semantic view of an extended entity-relationship model. ACM Transactions on Database Systems (TODS), 16(3), 369-416.

Menzel, C., & Mayer, R. J. 1998. The IDEF family of languages. In Handbook on architectures of information systems (pp. 209-241). Springer, Berlin, Heidelberg.

Nizam, A., 2011. Veritabanı Tasarımı: İlişkisel Veri Modeli ve Uygulamalar, Papatya Yayıncılık Eğitim.

Purchase, H. C., Welland, R., McGill, M., & Colpoys, L. 2004. Comprehension of diagram syntax: an empirical study of entity relationship notations. International Journal of Human-Computer Studies, 61(2), 187-203.

Ünite Soruları

Soru-1 :

Veriyi satır ve sütunlar şeklinde tutan, genelde MS Excel ya da CSV biçiminde görülen veri türüne ne ad verilir?

(Çoktan Seçmeli)

(A) Enformasyon

(B) Bilgi

(C) Veri kümesi

(D) Veri tabanı

(E) SQL

Cevap-1 :

Veri kümesi

Soru-2 :

Farklı kaynaklardan gelen veriyi enformasyona dönüştürmek üzere bir araya getiren yapıya ne ad verilir?

(Çoktan Seçmeli)

(A) Veri tabanı

(B) Veri ambarı

(C) SQL

(D) MS Excel

(E) CSV

Cevap-2 :

Veri ambarı

Soru-3 :

Veri tabanı içerisinde verinin saklandığı, nitelikler kümesinden oluşan yapıya ne ad verilir?

(Çoktan Seçmeli)

(A) Tablo

(B) Veri tabanı

(C) Veri ambarı

(D) MS Excel

(E) Nitelik

Cevap-3 :

Tablo

Soru-4 :

Hangisi veri tabanı tablosundaki sütunlara verilen adlardan biridir?

(Çoktan Seçmeli)

(A) Birincil anahtar

(B) İkincil anahtar

(C) Nitelik

(D) Tablo

(E) İlişki

Cevap-4 :

Nitelik

Soru-5 :

Hangisi uygun bir tablo adıdır?

(Çoktan Seçmeli)

(A) Satislar

(B) kullanicilar

(C) blog_yazilari

(D) KişiselBilgiler

(E) son_kullanici_hareketleri

Cevap-5 :

Satislar

Soru-6 :

Hangisi uygun bir nitelik adıdır?

(Çoktan Seçmeli)

(A) Adi

(B) Soyadi

(C) telefonNumarasi

(D) Kimlik Numarası

(E) Adres

Cevap-6 :

telefonNumarasi

Soru-7 :

Kaç çeşit anahtar vardır?

(Çoktan Seçmeli)

(A) 1

(B) 2

(C) 3

(D) 4

(E) 5

Cevap-7 :

2

Soru-8 :

Hangisi uygun bir fonksiyonel bağımlılık gösterimidir?

(Çoktan Seçmeli)

(A) $a \rightarrow \{b, c\}$

(B) $a, b, c \rightarrow \{d, e\}$

(C) $a = \{b, c\}$

(D) $a == b, c$

(E) $\{a\} \rightarrow c, d$

Cevap-8 :

$a \rightarrow \{b, c\}$

Soru-9 :

Hangisi şema gösterim şekillerinden biri değildir?

(Çoktan Seçmeli)

(A) Kaz Ayağı

(B) Martin

(C) EXPRESS

(D) CRUD

(E) Merise

Cevap-9 :

CRUD

Soru-10 :

Veri tabanında kaydın zorunluluk durumu nasıl gösterilir?

(Çoktan Seçmeli)

(A) Tek ve çok işaretiyle

(B) 1 ve 0 benzeri bir işaretle

(C) Çizgi bağlantısıyla

(D) Nitelik yanına PK yazarak

(E) Altını çizerek

Cevap-10 :

1 ve 0 benzeri bir işaretle

---

4. FONKSİYONEL BAĞIMLILIK

Bölümle İlgili Özlü Söz

Makineler ancak yanlış bir şey yaptıklarında size ne kadar güçlü olduklarını hatırlatırlar.

Clive James

Gazeteci

Kazanımlar

1. Nitelikler arasındaki ilişkiyi keşfedebilir

2. Nitelikleri gruplayabilir

3. Anahtar niteliği seçebilir

4. Veri tekrarı riskini yorumlayabilir

5. Veri setini, ilişkili nitelikler açısından analiz edebilir

Birlikte Düşünelim

Bir nitelikte yer alan bilgi, dolaylı olarak başka nitelik altında da bulunuyor olabilir mi?

Aynı veriyi farklı biçimlerde veri tabanı içerisinde tutmak bir hata mıdır?

Veri tabanlarında performans artışı için neler yapılabilir?

Hatalı nitelik gruplaması yapıldığında oluşabilecek riskler nelerdir?

Hatalı tasarım, çalışan bir sistem üzerinde nasıl düzeltilebilir?

Başlamadan Önce

İlişkisel veri tabanlarında tablolar arasındaki ilişkiden öte nitelikler arasındaki ilişkiler de oldukça önemlidir. Eldeki veri kümesinde yer alan nitelikleri kullanarak bir ilişkisel veri tabanı hazırlama sürecinde hangi niteliklerin ne şekilde gruplanacağı ve tablolar oluşturulacağı, nitelikler arasındaki ilişkilerin belirlenmesi ile mümkündür. Nitelikler arasındaki ilişkilerin tanımlanması ise fonksiyonel bağımlılıkların belirlenmesiyle mümkündür.

Fonksiyonel bağımlılık, nitelikler arasında hangi nitelik grubunun hangi nitelik grubu tarafından temsil edilebildiğini gösteren bir ifadedir. Fonksiyonel bağımlılığın tanımlanması, niteliklerin birbirleri arasındaki ilişkileri daha kolay keşfetmeyi ve bu sayede daha doğru veri tabanı tabloları oluşturmayı sağlamaktadır. Bu sebeple fonksiyonel bağımlılık tanımlama becerisi her veri tabanı tasarımcısı tarafından mutlaka içselleştirilmesi gereken bir süreçtir.

Bu bölümde fonksiyonel bağımlılığın ne olduğu, nasıl gösterildiği ve nasıl kurulduğu gösterilecek, bununla ilgili beceriler kazandırılmaya çalışılacaktır. Bu bölümün yoğunluklu amacı bu konuda bilgi sunmak değil, beceriyi kazandırmaktır. Bu sebeple bölüm boyunca kapsamlı ve karmaşık alıştırmalarla karşılaşacaksınız.

Dersin bu bölümünden yeterince faydalanabilmek için mutlaka alıştırmalara öncelikle siz yorum getirmeli, sonrasında kitap içerisinde sunulan yorum ve çözüm önerilerini incelemelisiniz. Daha uğraştıracağı kesin olan bu bölüm, sonucunda size bu beceriyi kazandırmış olacak ve dersin ilerleyen bölümlerinde gerçekleştirilmesi gereken faaliyetleri için hazır oluş durumunuzu yükseltecektir.

Giriş

İlişkisel veri tabanı denildiğinde akla öncelikle tablolar arasındaki ilişkiler gelmektedir. Hem veri tabanı tasarımında kullanılan varlık-ilişki (ER) diyagramları, hem de verinin tablolara bölünmüş yapısı "ilişki" kelimesinin doğrudan tablolar arasındaki bağlantıya atfedilmesine sebep olmaktadır. Ancak ilişkisel veri tabanlarında tablolar arasındaki ilişki, zaten niteliklerin gruplandırılmasından sonra ortaya çıkan bir süreçtir. Tablolar arasındaki ilişkiden çok daha önem arz eden, nitelikler arasındaki ilişkilerin ortaya çıkartılması ve tanımlanmasıdır. Hangi niteliklerin birbiriyle ilişkili olduğu, aynı zamanda hangilerinin birlikte gruplanması gerektiğini göstermekte; bu da veri tabanı tablolarının oluşturulmasına olanak sağlamaktadır.

Fonksiyonel bağımlılık, bir nitelik grubu içerisinde yer alan bir alt nitelik kümesinin başka bir nitelik kümesi için temsil edici nitelikte olduğunun belirlenmesidir. Diğer bir deyişle nitelikler arasındaki temsil özelliğinin ortaya çıkartılmasıdır. Günlük hayatta sıkça karşımıza çıkan ID kelimesi, kayıtların işaret edilmesi için kullanılan bir yapay niteliktir. Biz de gerçekleştirdiğimiz tabloların tümünde ID niteliği kullanarak anahtar seçimimizi kolaylaştıracak ve riskimizi en aza indireceğiz. Ancak fonksiyonel bağımlılıkların belirlenmesi, niteliklerin gruplanması ve veri kümesine daha hakim olmamızı sağlayarak daha iyi bir veri tabanı tasarımı gerçekleştirmeyi mümkün hale getirdiği için önemli bir yer tutmaktadır.

Fonksiyonel bağımlılığın ne olduğuna dair tartışmak, bir noktadan sonra ilerlemeyi zorlaştıracaktır. Bu sebeple dersin bu bölümünde fonksiyonel bağımlılığı örnekler üzerinden öğrenerek devam edeceğiz.

Veri tabanı tasarımcılığında deneyimin önemini vurgulamıştık. Fonksiyonel bağımlılığı öğrenmek ve ihtiyaç halinde uygulayabilmek bu sürecin temelinde yer alıyor. Bir nitelik kümesi gördüğümüzde bu niteliklere aralarındaki ilişkilerin tanımlanması ve gerekli şekilde gruplanması gözüyle bakmaya başladığınızda veri tabanı tasarımcılığında farklı bir aşamaya erişmiş olacaksınız.

Fonksiyonel bağımlılık konusunda bilgi ve deneyim sahibi olmak, sizi dersin ilerleyen başlıklarında veri kümesini daha iyi inceleyen, veri tabanı tasarımını gerçekleştirme konusunda sezgi sahibi, hem öznel hem de nesnel kararları alma konusunda hızlı ve isabetli veri tabanı tasarımcıları haline getirecektir. Lütfen bölüm boyunca yer alan alıştırmalar için, yapılan açıklamaları incelemeden önce kendiniz bir yorum getiriniz. Ardından yapılan yorumu okuyarak karşılaştırmalı değerlendirme yapınız.

Bir önceki başlıkta fonksiyonel bağımlılığın gösterim şekline değinmiştik. Hatırlatmak gerekirse;

$$X \rightarrow Y$$

gösteriminde veri kümesine ait Y nitelik alt kümesi, X alt kümesine fonksiyonel bağımlıdır denir (Vardi, 1985; Riordan, 2005). Ayrıca X alt kümesi, Y alt kümesi için temsil edici niteliktedir. Y'nin herhangi bir değeri, benzersiz bir X ile temsil edilir. Böylece ilgili X değeri kullanıldığında Y değerine denk geldiği kesin olarak bilinir.

Bu bölümde, fonksiyonel bağımlılığı keşfetme becerisini elde ederek, nitelikler arasındaki ilişkileri ön görebileceksiniz. Bunu yapabilmek için çok kez alıştırma yapmak ve farklı örnekler incelemek gerekiyor. Bu bölümün alt başlıkları farklı alıştırmalar içermektedir. Her bir alıştırma ile yeni ayrıntılar fark edeceksiniz. Her bir alıştırma için öncelikle kendinizin bir çalışma gerçekleştirmesi faydalı olacaktır. Basit bir alıştırma ile başlayalım.

Kitap: adi, yayinEvi, isbn, yil, baski, yazar

Yukarıda tarif edilmiş bir veri kümesine sahip olduğumuzu düşünelim. Bu gösterim bize Kitap adında bir veri kümemiz olduğunu, bu veri kümesi içerisinde adi, yayinEvi, isbn, yil, baski ve yazar adlarında nitelikler bulunmaktadır. Amacımız hangi niteliğin ya da nitelik grubunun diğerleri için temsil edici nitelikte olduğunu belirlemektir. Tüm nitelikleri kısaca ele alalım: adi niteliği kitabın adını göstermektedir. İki sebeple bu niteliği temsil edici olarak kullanamamaktayız. Birincisi tüm kayıtlı kitaplar göz önüne alındığında aynı ada sahip birden fazla kitap olma ihtimali oldukça yüksektir. Kitabın adı temsil edici kabul edildiğinde bir ad bilgisi yüksek ihtimalle birden fazla kaydı karşımıza getireceği için tercih etmemeliyiz. İkinci sebep ise bu niteliğin uzun metinler içeriyor olması. Uzun metin içeren ifadeler temsil edici kabul edilmezler. Bunu uç bir örnekle açıklayalım. Bir blok yazısı blog başlığı ve metinden oluşuyor olsun. Blog başlığı tekrarlanabilir.

Ancak içerik tekrarlanmaz. Her bir blog içerik metni tüm bloglar için mutlaka farklı olur. Ama ele alınan son derece uzun bir metin temsil edici kullanılabilir mi? Bu pratik bir çözüm mü? İlgili blog kaydına ulaşmak için tüm blog metninin kullanılması, bu metne sahip blog kaydına ulaşmak istenilmesi gerekiyor. Bu da pratik bir çözüm olmayacaktır. Kitap adı da benzer bir özelliğe sahip. Bir niteliğin içerdiği metin miktarı arttıkça pratik olarak temsil edici olması zorlaşmaktadır. Bu sebeplerle adi niteliğini geçiyoruz. yayinEvi niteliği hakkında karar vermesi oldukça kolay olan bir nitelik. Bir yayın evine ait çok sayıda kitap olacağı aşikardır. Dolayısıyla bir kitap kaydının işaret edilmesi için yayin evi bilgisi kullanılmamalıdır. isbn, kitapların kimlik koduna benzer bir değerdir. Her bir kitap farklı bir isbn bilgisine sahip olacaktır. Dolayısıyla temsil edici özelliği kuvvetlidir ve tek başına bile bir kitabın temsil edilmesi için kullanılabilir. yil değeri yine çok fazla kitabı temsil edecektir. Aynı yılda basılan birçok kitap olacağı için bu niteliği tercih etmek uygun değildir. baski niteliği kitabın kaçıncı baskı olduğu bilgisini içerir. Bu nitelik; 1, 2, 3, … gibi benzer değerler alacaktır ve temsil ediciliği düşüktür. yazar niteliği de yine bir yazarın birden fazla kitabı olması mümkün olacağı için temsil edici değildir. Tüm nitelikler arasında isbn niteliği bariz şekilde temsil edici özelliktedir. Bu sebeple ele alınan veri kümesi için fonksiyonel bağımlılığı şu şekilde tanımlayabiliriz:

$$isbn \rightarrow \{adi, yayinEvi, yil, baski, yazar\}$$

4.1. Alıştırma 1

Bir önceki örnek, yaklaşımımızı göstermek üzere oldukça kolay seçilmişti. Şimdi çok benzer ama üzerine daha fazla düşünmeyi gerektiren bir örnek üzerine çalışacağız.

Kitap: adi, yayinEvi, yil, baski, yazar

Bir önceki örnekten farklı olarak bu veri kümesi isbn gibi bariz bir temsil edici nitelik içermemektedir. Tüm nitelikler için açıklamamızı zaten yapmıştık ve hiçbirinin tatmin edici düzeyde bir temsil edici olmadığını biliyoruz. Bu durumda bir kesişim kümesi oluşturmak ve bundan faydalanmak önemlidir. Diğer kayıtların fonksiyonel bağımlı olduğu en az sayıdaki niteliği tercih ederek bağımlılığı ortaya çıkarabiliriz. Eldeki tüm nitelikler, birden fazla kayıtta karşımıza çıkabilecek niteliklerdir. Seçeceğimiz nitelik grubu yani kayıt kesişimi tek bir kaydı temsil edebilmelidir. Amaç, en az sayıda kayıtla eşleşen nitelikleri gruplayarak en nadir durumu elde etmektir. Bunu yapmanın yolu, bir niteliğe ait benzersiz kaç kaydolduğunu tahmin etmektir. Yine uç bir örnekle inceleyelim. Eğer isbn niteliği hala söz konusu olsaydı, 1000 kayıt içeren bir veri kümesi için 1000 farklı değer almasını beklerdik. Bu da her bir kayıt için farklı bir isbn değeri olduğunun kanıtı oldurdu. Şimdi adi niteliğini inceleyelim. Yine 1000 kayıt içeren bir veri kümesinde kaç tane aynı ada sahip kitap olabilir?

Bu noktada tahmine dayalı bir süreci işlediğini fark etmiş olmalısınız. Önceki bölümlerde veri tabanı tasarımının riskleri ön görmek ve öznel süreçler sebebiyle deneyimin başarıya etkisini ele almıştık. Bu aşamada gerçekte karşılaşılacak, belirsizlik durumlarında karar alma süreçlerinin bir benzerini elde etmekteyiz. Bu sebeple verilen alıştırmalar, çeşitli belirsizlikler içerecek şekilde tasarlanmıştır. Dolayısıyla bir nitelik hakkında görüş üretirken biraz hayali, riskleri öngören, her zaman geçerli olmayabilecek planlamalar yapabiliriz.

Şimdi soruya geri dönelim. Bin kayıt içeren bir veri kümesinde aynı ada sahip kaç kitap olabilir? Ya da şu şekilde de sorabiliriz: Bin kayıt içerisinde aynı adda birden fazla kez kaç kitap yer alabilir? Bu sorunun elbette net bir cevabı yok. Elimizde veri kümesi bulunuyor olsaydı kolaylıkla bu sayıyı belirleyebilirdik ancak bu durumda tahmini bir yaklaşımda bulunmamız gerekmektedir.

Ele aldığımız örnekte tek bir nitelikle fonksiyonel bağımlılık ifadesini kuramadığımız için mümkünse 2 nitelik kullanarak bunu yapmaya çalışacağız. adi niteliği metin içerdiği için kullanmayı tercih etmediğimizi belirtmiştik. Bir yayın evine ait çok fazla kitap olabileceği için yayinEvi niteliğinin temsil edici özelliği düşüktür. yil değişkeni aynı yılda çıkan kitapları kategorize etmek için iyi bir seçim olabilir. baski, baskı sayısını içeren bir nitelik olduğu için kitap kayıtlarının çoğu; 1, 2, 3… gibi kayıtlar içerecektir. Bu sebeple temsil edici özelliği en düşük niteliktir. Kitabın yazarı ise, aynı yazara ait kitap kayıtlarının olması durumunda ayırt edici olacaktır. Dersin eğitmeni olarak benim çözüm önerim yazar ve yil niteliklerinin birlikte kullanılmasıdır. Bir yazarın aynı yıl yayınlanan kitabının birden fazla olması durumunda bu iki nitelik birden fazla kayda denk gelebilir. Ancak bunun dışında yazar ve yil niteliklerinin birlikte kullanılması

durumunda temsil edici nitelik olmaları mümkündür. Aynı yazara ait, aynı yıl yayımlanmış birden fazla kitap olması durumu dışında bu tercih herhangi bir risk barındırmamaktadır. Bu en iyi çözüm olmayabilir. Ancak kişisel olarak benim, en düşük riskli olarak öngördüğüm çözüm budur. Yakın başka çözümler de önerilebilir ve elde gerçekten bir veri kümesi olmadığı müddetçe hangi çözümün daha iyi olduğunu ispat etmek oldukça zordur. Bu durumda öngörülere göre karar vererek çözümü ele almak gerekmektedir. Elde ettiğimiz çözüm şudur:

$$\{yazar, yil\} \rightarrow \{adi, yayinEvi, baski\}$$

En iyi çözümün ne olduğunu söylemek zor olsa da yanlış çözümün ne olduğunu söylemek genellikle daha basittir. Kitabın adı temsil edici nitelikte olmamalıdır. Yayın evi çok fazla kitap kaydıyla eşleşebilir. Ayrıca baskı niteliği genellikle aynı değerleri içerdiği için yine temsil edici nitelikte olmamalıdır. Alıştırmalar boyunca doğru çözümü bulmanın yanında neyin kesinlikle yanlış olacağını da tayin edebilmek oldukça mühimdir.

4.2. Alıştırma 2

Müşteri kayıtlarının tutulduğu Musteri adında bir veri kümesi için fonksiyonel bağımlılığı araştıralım.

Musteri: ad, soyad, telefon, ePosta, dogumTarihi, il, ilce, kayitTarihi, sonSiparisTarihi

Ele alınan tablo müşteriye ait kayıtları içeren bir veri kümesidir. Alıştırmada, verilen niteliklerden hangisinin müşteri kayıtları için temsil ediciliğe en uygun olduğunu belirlemek gerekmektedir. Bu örneği zihinde canlandırmak, diğer birçok örneğe göre çok daha kolay olacaktır. Bir müşteri kaydını temsil edecek niteliği elde etmek, aslında doğrudan bir kişiyi temsil eden niteliği belirlemekle benzerdir. Dolayısıyla bir kişiyi temsil etmek için en kullanışlı niteliği tercih etmek gerekmektedir. Eldeki nitelikleri bu bağlamda inceleyelim.

Bir veri kümesinde aynı ad değerine sahip birden fazla kayıt bulunması oldukça mümkündür. Benzer şekilde soyad, dogumTarihi, il ve ilce değerleri de aynı kayıtlarla eşleşecek değerler içerebilirler. telefon ve ePosta nitelikleri müşteri kayıtlarını ayırt etmek için kullanılabilecek iyi seçeneklerdendir. il ve ilce nitelikleri, alacağı değerler belli olan niteliklerdedir. Belki 1000 kayıtlık bir veri kümesinde daha seyrek olabilir ancak çok daha fazla kayıt içeren bir veri kümesi ile karşılaşıldığında il ve ilce niteliklerinin aldığı değerler benzer olacağı için veri tekrarı ile daha sık karşılaşılacaktır.

kayitTarihi ve sonSiparisTarihi nitelikleri tarih bilgisi içeren niteliklerdir. Tarih bilgisi içeren bir niteliğin temsil edici olması pratikte pek mümkün değildir. Kayıt sayısının az olması durumunda, her bir kaydın farklı tarih ile ilişkilendirilmesi bu niteliklerin seçilmesini mümkün hale getirse de kayıt sayısı artınca aynı tarih farklı kayıtlarla eşleşeceği için temsil edici özelliği kaybolacaktır.

Elimizde 2 çok iyi seçenek var: telefon ve ePosta. İki nitelik de bir kişiyi temsil etmek için kullanılabilecek telefonlarda. Her bir müşteri kaydının bir telefon numarası ve e-posta bilgisine sahip olduğunu söyleyebiliriz. Ayrıca bir telefon numarası bilgisi ve e-posta bilgisi ayrı ayrı bir kaydı temsil edebilecek özelliktedir. Bu niteliklerden herhangi birini tercih etmek bu alıştırma için yanlış yanıt olmayacaktır. Yine de mutlaka bir niteliği tercih etmek için zorlarsak şu yorumu yapabiliriz: Telefon numaraların sahipleri zamanla değişebilir. Bir telefon numarası çok uzun süre kullanılmadığında kapatılır ve başka biri için tekrar hizmete alınır. Ancak bir e-posta adresi için bu geçerli değildir. E-posta adresleri daima bir kişiye ait olur. Bu sebeple çok düşük bir olasılık olsa da telefon numarasının birden fazla kişiyi temsil etmesi mümkün olabilir. Bu sebeple benim tercihim ePosta niteliğini kullanmaktan yanadır. Buna göre fonksiyonel bağımlılık ifadesini yazalım.

$$ePosta \rightarrow \{ad, soyad, telefonNumarasi, dogumTarihi, il, ilce, kayitTarihi, sonSiparisTarihi\}$$

4.3. Alıştırma 3

Bir sonraki alıştırmayı ele alalım. Bir bilgisayarda çeşitli fotoğrafların saklanıldığını ve bu fotoğrafların bir kayıt listesinin tutulduğunu düşünelim. İlgili veri kümesini aşağıdaki gibi tarif edebiliriz.

Fotograflar: ad, boyut, dosyaTuru, w, h, olusturulmaTarihi, degistirilmeTarihi

Veri kümesindeki fonksiyonel bağımlılığı belirlemek için nitelikleri teker teker ele alalım. ad niteliği fotoğrafın adını belirten, metin değerler alan bir nitelik. Metin nitelikleri temsil edici olarak seçmekten kaçındığımızı belirtmiştik. Bu sebeple bu alıştırmada da ad niteliğini tercih etmeyeceğimiz niteliklerden biri olarak belirleyerek bir kenara bırakıyoruz.

boyut niteliği tartışmamız gereken, ilginç sayılabilecek bir nitelik. Öncelikle şunu söylemek gerekir ki sayısal nitelikler temsil edici olma konusunda en sık tercih edilenlerdir. Az bir bellek kullanarak benzersiz şekilde kayıt temsil etmek, çok fazla kombinasyona sahip olmak ve kolay yönetilebilir olması sayısal niteliklerin temsil edici olmasını kolaylaştıran etmenlerdendir. Daha önceki bölümlerde sayısal veri ve metin içeren niteliklerin bellekte saklanılmasıyla ilgili ayrıntıları vermiştik. Sayısal bir değeri saklamak ve işlemek, bilgisayar ortamında daha az kaynak gerektiren bir karar olacaktır. Benzersiz sayısal değerler üretmek, rastgele sayı üretmek ya da sıralı sayı verme gibi yöntemler sayesinde kolaylık sağlamaktadır. Bu sayede sayısal değerleri temsil edici kullanmak, bu değerleri daha kolay yönetebildiğimiz için daha uygundur. Sayısal değerlerin sıralı kullanımında, ulaşılan sayı kadar çok kombinasyon kullanılması mümkün olduğu için çeşitlilik anlamında da fayda sağlamaktadır. Hatta 10 sayı tabanından daha yüksek başka bir sayı tabanına ulaşmak daha az karakterle daha fazla kombinasyon elde etmemizi sağlayacaktır. İşin özü; sayısal değerler her zaman bizim için bir adım öndedir.

Şimdi boyut niteliğine geri dönelim. Dosya boyutunu hangi mertebede aldığımız oldukça önemli. Örneğin fotoğraflar için konuştuğumuzda megabayt mertebesinde her fotoğraf 5, 6, 7 gibi benzer değerler alacaktır. Dosya boyutundaki farklar, megabayt mertebesine ulaşırken yuvarlanacağı için tüm dosyalar aynı boyutta gibi görülecektir. Ancak kilobayt ya da bayt mertebesinde her dosya farklı değerlerde boyutlara sahip olacaktır. Bu sebeple boyut niteliğini kilobayt ya da bayt mertebesinde kabul ederek boyut niteliğini tercih edilebilecek olanlar listemize ekleyerek diğer nitelikleri incelemeye devam edebiliriz.

dosyaTuru niteliği, dosyanın hangi formatta kaydedildiği bilgisini içermektedir. Benzer dosyalar aynı dosya türüne sahip olacağı için temsil edici nitelikte olmayacaktır. Farklı kaynaklar birleştirilse bile görsel dosyaların olabilecekleri biçimlerin sayısı oldukça sınırlıdır (jpg, png, gif, vb). Bu sebeplerden dolayı bu niteliğe fonksiyonel bağımlılık da mümkün gözükmemektedir.

w ve h nitelikleri genişlik (weight) ve yükseklik (height) anlamındadır. Genellikle benzer kalitede fotoğraflar aynı çözünürlüğe sahip olur. Örneğin Full HD olarak bilinen görüntü türü için 1920 piksel genişlik, 1080 piksel yükseklik kullanıldığı görülmektedir. Her ne kadar w ve h nitelikleri de sayısal olsa da genellikle aynı değeri alan nitelikler olacağı için temsil edicilik özellikleri düşük olacaktır.

Son 2 niteliğimiz; olusturulmaTarihi ve degistirilmeTarihi. Tarih içeren niteliklerle ilgili ayrıntılardan daha önce bahsetmiştik. Tarih içeren nitelikler hem aynı tarihin kayıtlarda tekrar yer alması sebebiyle hem de metin içerikli nitelikler olduğu için anahtar olarak kullanılmaması gereken niteliklerdendir.

Tüm bu ayrıntıları incelediğimizde en iyi seçenek boyut niteliği gibi gözükmektedir. Peki boyut niteliği fonksiyonel bağımlılık tanımlaması için tek başına yeterli midir? Eğer dosya boyutu mertebesi, daha hassas sayı sunan bir ölçekteyse her kayıt için yeterince farklı değer elde etmek mümkün olacaktır. Birebir aynı boyuta sahip iki dosyaya sahip olma olasılığımız oldukça düşük olmakla birlikte bunu engelleyecek ikinci bir nitelik öneremiyoruz.

$$\text{boyut} \rightarrow \{\text{ad, dosyaTuru, w, h, olusturulmaTarihi, degistirilmeTarihi}\}$$

olarak yanıtımızı kayıt altına alabiliriz.

4.4. Alıştırma 4

Bir teknik servis tarafından tutulan kayıtlarla ilgili bir veri kümesini ele alalım. Bu veri kümesi aşağıdaki gibi tanımlanmış olsun:

TeknikServisKayitlari: urunTuru, marka, model, ariza, teknisyen, müşteri, iletişim

Yine tüm nitelikleri tek tek ele alıp değerlendirmemiz gereken bir problem. Bunu, hem doğru sonucu belirleyebilmek için hem de yanlış ¸yanıt verenlere gerekçe sunabilmek için yapıyorum. urunTuru niteliği birkaç değerden birini alan, dolayısıyla çok sayıda kayıtta kendini sürekli tekrar edebilecek bir nitelik. Aynı marka ya da modele sahip birçok cihazın servise gelmesi de muhtemel olduğu için tercih etmek mantıklı olmayacaktır. ariza çeşitli değerler alabilecek olsa da çok sayıda kayıtta yine kendini tekrar etmesi muhtemel bir nitelik. Serviste çalışan teknisyen sayısı sınırlı olacağı için teknisyen de kendini tekrar eden kayıtlara sahip olacak ve bu sebeple işaret edici olmayacaktır. Bir müşterinin aynı servise birden fazla kez uğraması mümkün olduğundan musteri ve buna bağlı bir nitelik olan iletisim de işaret edicilikten uzaktır. Görüldüğü üzere tüm nitelikler için çeşitli bahanelerimiz var. Çözüm için kafa yorarken sizin de problem yaşadığınızı düşünüyorum. Bu zor durumu kurtarmak için birden fazla nitelikten faydalanmamız gerektiği açık. Dikkat etmemiz gereken nokta riski en aza indirmektir.

Tüm niteliklere tekrar göz gezdirerek en az tekrar eden nitelikleri belirlememiz gerekiyor. En az tekrar eden iki niteliği seçmek, bu ikisinin kesişiminin tekrar etme olasılığını ciddi şekilde düşüreceği için bize güzel bir çözüm getirebilir. Burada tercih, problemi hayal etmenize ve tüm riskleri düşünmenize bağlı olarak değişebilir. Eğer zorlanıyorsanız şu yöntem faydalı olacaktır: Bu veri setinde 10.000 kayıt olduğunu düşünün ve her bir niteliğin bu kadar kayıtta kaç farklı değer alacağını tahmin etmeye çalışın. En az tekrar eden ikiliyi inceleyin.

Tamamen kendi hayal gücüme güvenerek, 10.000 kayıt için niteliklerin benzersiz kaç kayıt içerdikleri konusunda s ¸öyle bir tahmin yaptım: urun- Turu: 10, marka: 3, model: 50, ariza: 500, teknisyen: 20, musteri: 8.000, iletisim: 8.500. Bir müşterinin iletişim bilgisini güncelleyeceğini düşünerek iletisimi biraz daha fazla tuttum. Bu tahminlere göre musteri ve iletisimi birlikte kullanmak harika bir çözüm gibi görünüyor ancak bu iki değer çoğunlukla birlikte hareket ettiği için mantıklı da olmayacaktır. Dolayısıyla bunlardan biri ve bence daha sağlıklısı olan iletisim niteliği kullanılmalıdır.

Riski en aza indirmek için seçilebilecek ikinci nitelik ise sayılar gözetildiğinde arizadır. Bununla birlikte çözüm önerimiz şöyle şekillenmiştir:

$$\{\text{iletisim, ariza}\} \rightarrow \{\text{urunTuru, marka, model, teknisyen, musteri}\}$$

Buradan çıkan sonuç; bir iletişim bilgisi ve arızanın ne olduğu bilgileri birlikte kullanılarak en iyi yaklaşım elde edilebilir, diğer niteliklerin bu iki niteliğe fonksiyonel bağımlı olduğu söylenebilir. Bu çözümün de mükemmel olmadığını söylemeye gerek olmayacaktır ancak riski tahminlerimiz doğrultusunda en aza indirmeyi başardık. Sizin ürettiğiniz çözüm, yine hayal ettiğiniz durum ve tahminleriniz doğrultusunda farklılık gösterebilir. Bu sebeple farklı yanıtlara doğrudan hatalı demek mümkün değildir ancak yine de verdiğiniz yanıt farklıysa ikinci kez göz gezdirmek ve yorumlamak faydalı olacaktır.

4.5. Alıştırma 5

Sınavlardan alınan puanların kaydedildiği bir veri kümesini ele alalım. Veri kümesi tanımlamamız aşağıdaki gibi olsun:

Sinav: ogrenci, ders, not, yil, donem

Üzerine düşünülmesi kolay olmayan bir alıştırma ile karşı karşıyayız. Problem; tek bir nitelikle çözüme ulaşılamayacak, niteliklerin bir kombinasyonunu gerektiren yapıdadır. Hangi nitelikleri seçeceğimizi belirlemeden önce tüm nitelikleri kısaca yorumlayalım.

öğrenci niteliğini doğrudan metin içeren bir nitelik olarak da görebiliriz ancak bunun yanında her bir kaydı bir kişiyi işaret eden bir nitelik olarak da görebiliriz. Yani, bir kayıt dosyasında bir öğrenciye ait kayıt tekrarlanma durumu ne sıklıkla yaşanılır sorusuna yanıt aramak gerekiyor. Her öğrenci için her bir dönem, her bir ders ve dönem için sınav notu kaydı yapılacaktır. Bu sebeple öğrenci kaydının çok fazla kez tekrar ettiğini söyleyebiliriz.

ders niteliği, aynı dersi alan her öğrenci için ve farklı yıllarda tekrar tekrar anılacağı için çok kez tekrar edecektir ve temsil niteliği yoktur. not niteliği, öğrencinin dersin sınavından aldığı notu ifade etmektedir. 0

ile 100 aralığında bir değer alacaktır. 100'den fazla kayıtla ilgileniyor olduğumuz her durumda kendini tekrar edeceği kesindir. yil niteliği ilgili sınav notunun alındığı yılı gösterir. Bir yıl içerisinde çok sayıda kayıt tutulacağı için yine temsil edici bir niteliğe sahip olmayacaktır. donem niteliği de çoğunlukla yalnızca 2 farklı değer alan, sınav notunun alındığı dönemi gösteren niteliktir. Bu nitelik de temsil edici özellikte sayılamaz.

Tüm nitelikleri eledik. Peki en iyi çözüm ne olabilir? Hangi nitelik(ler) hangi nitelik(ler)e fonksiyonel bağımlıdır? Fonksiyonel bağımlılığı tanımlamak için bir nitelik grubu tanımlanması gerekmektedir. En belirleyici iki niteliği seçmeye çalışalım. ogrenci ve ders nitelikleri, kesişim kümeleri sebebiyle en belirleyici 2 nitelik olarak gözüküyor. Belli bir yılın bir dönemindeki not nitelikleri, bir öğrenci ve ders niteliklerine fonksiyonel olarak bağımlı sayılabilir. Bu durumda;

$$\{ogrenci, ders\} \rightarrow \{yil, donem, not\}$$

gösterimi uygundur. Burada bir risk ile karşı karşıyayız. Eğer bir öğrenci farklı yıllarda aynı dersi tekrar alır ve bunun için yeni kayıt oluşturulursa mevcut çözüm önerimiz hatalı kalabilir. Daha iyi bir çözüm ancak şu şekilde elde edilebilir:

$$\{ogrenci, ders, yil, donem\} \rightarrow \{not\}$$

Veri kümesindeki not niteliği; ogrenci, ders, yil ve donem niteliklerine fonksiyonel bağımlıdır. Diğer bir deyişle not verisinin alınabilmesi için ogrenci, ders, yil ve donem bilgilerine ihtiyaç duymaktayız.

4.6. Alıştırma 6

Bir ülkede yer alan radyo kanallarıyla ilgili bir veri kümesini inceliyor olalım. Veri kümesi şu şekilde tanımlanmış olsun:

RadyoKanallari: ad, frekans, sehir

Burada ad niteliğinin metin bir nitelik olması sebebiyle zorunlu olmadıkça temsil edici nitelik olarak seçmeyeceğimizi biliyoruz. frekans ise radyo kanalının yayın yaptığı frekans değeridir ve sayısal içeriği bulunmaktadır. Bu sebeple mantıklı bir seçim olarak karşımıza çıkar. sehir ise daha önce öğrenci niteliği için yaptığımız yorumu hak eder. sehir niteliği metin içeren bir niteliktir ancak aslında bir kategori gösterir. Yani hiçbir zaman uzunca bir metin içermeyecektir. Dahası, şehrin adı yerine plaka kodu ya da kısaltması gibi daha kısa belirteçler içeriyor da olabilir. Kişiyi, şehri, ülkeyi ve benzeri olguları ifade eden, kategori içeren değişkenler de temsil edici nitelikle karşımıza çıkabilirler. Peki bu örnekte sehir niteliğini kullanmaya ihtiyacımız var mı? frekans niteliğini fonksiyonel bağımlılık ifadesinin sol tarafında, ad niteliğinin ise sağ tarafında olduğunu belirledik. Peki sehir ve ad niteliklerinin frekans niteliğine fonksiyonel bağımlı olması mı daha doğrudur yoksa ad niteliğinin frekans ve sehir niteliklerine fonksiyonel bağımlı olması mı? Radyo kanallarının frekans bilgileri şehirden şehire farklılık göstermektedir. Bu sebeple bir radyo kanalını temsil edecek en iyi değer sadece bu radyo kanalının frekansı değildir. Hangi şehirde olduğunun bilgisinin de alınması gerekmektedir.

$$\{frekans, sehir\} \rightarrow ad$$

Yukarıda yer alan fonksiyonel bağımlılık ifadesi, belirlediğimiz çözüm için uygundur.

4.7. Alıştırma 7

Araç takibi cihazı üreten bir firmanın, cihazlardan topladığı bilgiyi kaydettikleri veri kümesini ele alalım. Bu veri kümesi aşağıdaki gibi tanımlanmış olsun:

AracTakip: plaka, kullanici, mesafe, ilkKm, sonKm, tarih

Bir aracı temsil edebilecek en net nitelik oldukça kolay göze çarpıyor. plaka niteliği, bir aracı kolayca tanımlayabilecek, benzersiz, fonksiyonel bağımlılık ifadesinin sol tarafında yer alabilecek bir niteliktir. Bunu kesinlikle kullanmamız gerektiğini bir kenara not edebiliriz.

Peki plaka niteliği bu veri kümesi için yeterli midir? Eğer veri kümemiz Arac adında, araçların listelendiği bir veri kümesi olsaydı muhtemelen yeterli olacaktı. Ancak araç takibinden bahsediyoruz. ilkKm ve sonKm değerleri aracın farklı km aralıkları için takip edildiğini göstermektedir. Bu sebeple plaka tek başına yeterli olmayabilir. Bununla birlikte hangi takip sürecinin kayda alındığını da belirtmek gerekecektir. Bunun için de en kullanışlı nitelik tarih niteliğidir. Böylece çözümümüzü aşağıdaki gibi gösterebiliriz.

$$\{plaka, tarih\} \rightarrow \{kullanici, mesafe, ilkKm, sonKm\}$$

Bölüm Özeti

Fonksiyonel bağımlılık, bir nitelik grubunda yer alan bir alt nitelik kümesinin bir diğer alt nitelik grubu tarafından temsil edilebilirliğini gösteren yapıdır. Fonksiyonel bağımlılık sayesinde bir nitelik grubunda hangi nitelik ya da niteliklerin temsil edici olduğu belirlenebilir. Temsil edilen nitelikler için temsil eden niteliklere fonksiyonel bağımlıdır denir. Bu ilişkinin tanımlanması niteliklerin gruplandırılması açısından önemlidir. Aralarında fonksiyonel bağımlılık olan nitelikler gruplandırılır ve bir veri tabanı tablosu olarak tanımlanır. Bu sebeple dersin ilerleyen bölümlerinde yer alan veri tabanı tasarlama süreçlerinde bu tanımlamaların yapılması, yanlış bir veri tabanı tasarımı yapılmaması açısından önemlidir.

Fonksiyonel bağımlılık konusu çok fazla bilgi içeren bir konu değildir. Ancak bu ilişkileri tanımlamak deneyim gerektirmektedir. Bu sebeple bu bölümde bilgiden çok alıştırmalara ağırlık verilmiş, bu sayede ilgili becerinin kazandırılması hedeflenmiştir.

Fonksiyonel bağımlılık, hangi nitelik ya da nitelik grubunun temsil edici özellikte olduğunun belirlenmesi sürecidir. Bu süreç boyunca mümkün olan en az sayıda niteliği temsil edici olarak tercih etmek gerekmektedir. Ancak bazı durumlarda hiçbir nitelik tek başına iyi bir temsil edici olmayabilir. Bu durumda bir nitelik grubunu tercih etmek gerekir.

Tercih edilen niteliklerin ne tür veri içerdikleri de önemlidir. Sayısal veri içeren nitelikler öncelikli tercih sebebidir. Metin içeren nitelikler ise her seferinde benzersiz içeriğe sahip olsalar bile pratikte kullanışlı bir temsil edici olmayacakları için tercih edilmezler. Bir de bu iki durumun ortası var. Bir nitelik metin içerik alıyor ancak bu metin içerikler bir kategoriyi işaret ediyor olabilirler. Şehir bilgisi içeren nitelikler bu duruma bir örnektir. Şehir adları bir metin içeriği olsa da gerçekte bir kategoriyi ya da olguyu işaret ederler. Kişileri belirten nitelikler için de aynı durum geçerlidir. Kişi adıyla ilgili bir nitelik, bir kişinin adını ve soyadını içerdiği için metin içeriklidir. Ancak gerçekte bir kişiyi işaret etmektedir. Şöyle netleştirebiliriz; kişi ile ilgili nitelik içerisinde kişinin ad ve soyad bilgisini silip kimlik numarası bilgisini ekleyebiliriz. Böylece veri bütünlüğü bozulmaz. Biz hala ilgili kişiyi işaret ediyor oluruz. Bu sebeple bu tarz kategori belirten nitelikleri de temsil edici özellikte kullanmamız mümkündür.

Fonksiyonel bağımlılık belirlemenin önemli özelliklerinden biri zaman zaman herkes için geçerli doğru bir yanıt bulunmamasıdır. Elimizde yalnızca bir nitelik grubu varsa ve bu niteliklerin alacağı değerler konusunda varsayımlarda bulunmak zorundaysak, her veri tabanı tasarımcısı farklı varsayımlarla farklı kararlar alabilir. Bu sebeple bölüm içerisinde veriler çözümlere benzer ancak farklı çözümler üretmiş olabilirsiniz. Bu bir hata değildir. Ancak şuna dikkat etmek gerekiyor; her zaman kesin doğru olmayabilir ancak her zaman kesin yanlışlar vardır. Uzun metin içeriklerini temsil edici olarak kullanmak ya da bir nitelik yeterliyken ikinci bir niteliği de temsil edici tarafta kullanmak hatalı görülebilecek kararlardandır.

Fonksiyonel bağımlılıklar konusunda iyi çözüm önerileri sunmak, dersin ilerleyen kısımlarında da başarılı tasarımlar gerçekleştirmenizde fayda sağlayacaktır.

Kaynakça

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

Vardi, M. Y. 1985. Fundamentals of dependency theory. IBM Thomas J. Watson Research Division.

Riordan, R. M. 2005. Designing effective database systems. Addison- Wesley Professional.

Ünite Soruları

Soru-1 :

Hangisi geçerli bir fonksiyonel bağımlılık tanımlamasıdır?

(Çoktan Seçmeli)

(A) $a = b, c$

(B) $a \rightarrow b, c$

(C) $a \rightarrow \{b, c\}$

(D) $a \approx \{b, c\}$

(E) $a \rightarrow [b, c]$

Cevap-1 :

$a \rightarrow \{b, c\}$

Soru-2 :

Hangi niteliğin temsil ediciliği daha yüksektir?

(Çoktan Seçmeli)

(A) kitapAdi

(B) yazarAdi

(C) isbn

(D) basimYili

(E) kaynakca

Cevap-2 :

isbn

Soru-3 :

Hangi niteliğin temsil ediciliği daha yüksektir?

(Çoktan Seçmeli)

(A) sirketAdi

(B) vergiKimlikNumarasi

(C) vergiDairesi

(D) adres

(E) telefon

Cevap-3 :

vergiKimlikNumarasi

Soru-4 :

Hangi niteliğin temsil ediciliği daha yüksektir?

(Çoktan Seçmeli)

(A) aracMarkasi

(B) aracModeli

(C) uretimYili

(D) plaka

(E) uretimYeri

Cevap-4 :

plaka

Soru-5 :

Hangi niteliğin temsil ediciliği daha yüksektir?

(Çoktan Seçmeli)

(A) dosyaAdi

(B) dosyaTuru

(C) dosyaBoyutu

(D) dosyaOlusturulmaTarihi

(E) dosyaYolu

Cevap-5 :

dosyaBoyutu

Soru-6 :

Hangi niteliğin temsil ediciliği daha yüksektir?

(Çoktan Seçmeli)

(A) ad

(B) soyad

(C) telefon

(D) ePosta

(E) kimlikNumarasi

Cevap-6 :

kimlikNumarasi

Soru-7 :

Hangi niteliğin temsil ediciliği daha yüksektir?

(Çoktan Seçmeli)

(A) urunKodu

(B) urunAdi

(C) stok

(D) fiyat

(E) marka

Cevap-7 :

urunKodu

Soru-8 :

Hangi niteliğin temsil ediciliği daha yüksektir?

(Çoktan Seçmeli)

(A) yolculukBaslangicYeri

(B) yolculukBitisYeri

(C) tarih

(D) seferNumarasi

(E) aracPlakasi

Cevap-8 :

seferNumarasi

Soru-9 :

Hangi niteliğin temsil ediciliği daha yüksektir?

(Çoktan Seçmeli)

(A) gondericiHesapNo

(B) aliciHesapNo

(C) tarih

(D) tutar

(E) dekontNumarasi

Cevap-9 :

dekontNumarasi

Soru-10 :

Hangi niteliğin temsil ediciliği daha yüksektir?

(Çoktan Seçmeli)

(A) ders

(B) bolum

(C) ogrenciSayisi

(D) sinif

(E) dersKodu

Cevap-10 :

dersKodu

---

5. BAĞLANTI TÜRLERİ

Bölümle İlgili Özlü Söz

Asıl tehlike bilgisayarların insanlar gibi düşünmesi değil, insanların bilgisayar gibi düşünmesidir.

Sydney J. Harris

Gazeteci

Kazanımlar

1. Veri tabanı ilişki türlerini bilir.

2. İki varlık arasındaki ilişki türünü tanımlayabilir.

3. İhtiyaca yönelik anahtar nitelik seçimini gerçekleştirebilir.

4. İlişkileri içeren veri tabanı tasarımını çizebilir.

5. Birincil ve ikincil anahtar eşleştirmelerini sağlayabilir.

Birlikte Düşünelim

Bilişim sistemlerinde her olgu bir ID niteliğiyle ifade edilmek zorunda mıdır? Birincil anahtar olarak seçilebilecek daha iyi bir nitelik bulunamaz mı?

İki veri tabanı tablosu hiçbir ilişki ile birbirine bağlı değilse bu yapı için hala veri bütünlüğünden bahsedilebilir mi?

Birden fazla ilişkisel veri tabanı, çapraz bir bağlantı ile anahtarları kullanarak veri bütünlüğünü sağlayabilirler mi?

Bir veri tabanı tablosu birden fazla birincil ve/veya ikincil anahtar bulundurabilir mi?

Hangi ilişki tipi daha önemlidir? Tablolar arasındaki mi yoksa nitelikler arasındaki mi?

Başlamadan Önce

Bir önceki bölümde ilişkisel veri tabanlarının iki tür ilişki barındırdığını vurgulamıştık. Bununla birlikte önceki bölüm, nitelikler arasındaki ilişkilerin belirlenmesi üzerineydi. Bu bölümde ise tablolar arasındaki ilişkilere yoğunlaşacağız.

Ayrık iki konu gibi gözükse de aslında birbirini tamamlayan iki süreçten bahsediyoruz. Önceki bölümde; eldeki bir ham veri kümesinin sahip olduğu nitelikleri, aralarındaki fonksiyonel bağımlılıklara göre gruplayarak tabloları elde ediyoruz. Şimdi de tablolar arasındaki ilişkileri tanımlayarak veri kümesi biçimindeki veri yapımızı veri tabanı biçimine dönüştürmüş oluyoruz. Böylece verimizi, veri bütünlüğünü kaybetmeden küçük parçalara bölerek ilişkisel veri tabanları sayesinde elde etmek istediğimiz avantajları yakalamış oluyoruz.

Bu bölümde öncelikle bağlantı türlerinin neler olduğunu inceleyeceğiz. İlişkisel veri tabanı tasarımında 3 çeşit ilişki türü bulunmaktadır. Bu ilişki türleri, iki veri tabanı tablosunun bağlanabileceği her türlü kombinasyonu içermektedir. Ayrıca bölüm içerisinde birincil anahtar seçimine yönelik çeşitli yaklaşımları da ele alacağız. Böylece ihtiyaca yönelik olarak hangi birincil anahtarın kullanılabileceğini belirleme becerisi kazandırılmış olacaktır.

İki veri tabanı tablosu, aralarında çoğa çok ilişki türü ile bağlantı sağladıysa bu durumda yeni bir varlık oluşturulması gerekebilir. Pivot ya da bağlantı tablosu adı verilen bu tablonun nasıl oluşturulacağıyla ilgili bilgileri de sağlayarak bu bölümde anlatılacakları tamamlayacağız.

5.1. Bağlantı Türleri

İlişkisel veri tabanlarının içerdiği ikinci tür ilişki olan; tablolar arasındaki ilişkinin sağlanması konusunu ele almaya başlıyoruz. Bu bölüm, bir önceki bölümle farklı gibi gözükse de birlikte ele alınması gerekir. Bir ilişkisel veri tabanı tasarlamadan önce elimizde en iyi ihtimalle örnek verinin ham veri kümesi hali bulunmaktadır. En kötü ihtimalle ise sadece gerçekleştirilecek yapının bir tarifi bulunur. Her durumda, eldeki veri kümesinin yapısı ya da ihtiyaçlar listesi gözetilerek hangi niteliklerin ya da veri alanlarının kullanılması gerektiği veri tabanı tasarımcısı tarafından belirlenebilir (Hoffer, Ramesh & Topi, 2016; Akadal, 2021).

Tabloların oluşturulması aşaması, fonksiyonel bağımlılıkların belirlenmesi aşamasında yaptığımız gibi hangi niteliklerin birbirleriyle ilişkili olduklarını belirleyerek ve birbirlerini temsil etme durumlarını irdeleyerek onları gruplandırmak üzerine kuruludur. Bu aşamada, ilk başta elde tüm nitelikleri içeren tek bir grup varken; ilişkili niteliklerden oluşan çeşitli gruplar elde edilmiş olmaktadır. Bu sayede tasarlamaya çalıştığımız veri tabanı yapısı için tablolar elde edilmiş olmaktadır. Daha önce vurguladığımız üzere; veri tabanları, büyük veri kümelerini küçük tablolar halinde sunmakta ve bu sayede çeşitli avantajlar sağlamaktadır.

Görece büyük bir veri kümesinin küçük tablolar haline dönüştürülmesiyle birlikte veri yapısı ayrık ve bağımsız olacağı için veri bütünlüğünün kaybedilmesi söz konusudur. Örneği kişiler ve bu kişilerin yaptıkları satın alımları düşünelim. Kişi ve satın alımla ilgili iki tablo elde etmemiz ve bu iki tablonun bağımsız olması durumunda, hangi satın alımın kim tarafından gerçekleştirildiği bilgisini de kaybetmiş olacağız. İlişkisel veri tabanlarının veri bütünlüğünü kaybetmeden bu süreci gerçekleştirebildiğini biliyoruz. Bu da demek oluyor ki veri kümesi tablolara bölündükten sonra veri bütünlüğünü sağlamaya devam etmenin bir yolu mevcut. Bu da tabloların birbirleriyle ilişkiler üzerinden bağlanmasıdır.

Veri tabanı tabloları, belirlenen nitelikler üzerinden en uygun ilişki türü seçilerek bağlanırlar. Alt başlıklarda bu bağlantı türlerinin neler olduğu, hangi durumda hangi bağlantı türünün seçilmesi gerektiği konusuna değineceğiz.

Tablolar arasındaki ilişkiler, tablolarda yer alan nitelikler üzerinden sağlanırlar. Bu nitelikler de tablolar için birincil ve ikincil anahtar özelliği kazanırlar. Birincil ve ikincil anahtarları ayrıntılı olarak ele alacağız. Birbirleriyle bağlanmış iki tablo, uygulama anında iki tablo içerisinde yer alan birer satır kaydın aynı anda aynı satırda birleştirilmesi ihtiyacıyla gerçekleştirilmektedir. Örneklerle açıklayalım.

Ogrenci ve Ders tablolarını ele alalım. Bu iki tablo, hangi öğrencinin hangi dersi aldığının belirlendiği bir veri kümesinin ilişkisel veri tabanına dönüştürülmesiyle elde edilmiş olsun. Dolayısıyla eldeki kayıt gerçekte her bir satırda bir öğrencinin bir dersi alıyor olması durumudur. Veri tabanını elde etme süreci içerisinde biz bu veri kümesini Ogrenci ve Ders adında iki tabloya ayırabiliriz. Böylece veri kümesinde yer alan öğrenciyle ilgili nitelikler bir grup, ders ile ilgili nitelikler de başka bir grup oluşturacaktır. Fonksiyonel bağımlılık açısından ele aldığımızda niteliklerin 2 grup oluşturduğu sonucuna varacağız. Sonuç olarak elimizde bir Ogrenci tablosu yani öğrencilerin bir listesi ve bir Ders tablosu yani alınabilecek derslerin bir listesi oluşacaktır. Ancak burada bir veri kaybı var, fark ettiniz mi? Hangi öğrencinin hangi dersi aldığının bilgisini içeren bir veri kümemiz varken artık sadece öğrenci ve dersler listesine sahibiz. Halbuki bizim her bir satırda bir ders alma durumunu ifade eden, öğrenci ve ders kayıtlarının eşleşmesine ihtiyacımız var. Tablolar arasındaki ilişki bizim için bunu sağlamaktadır. Kayıtları, iki farklı tablo içerisinde yer alacak şekilde bölmüş olsak da ihtiyaç halinde veri bütünlüğünü sağlayacak yapıyı kurmamız gerekmektedir. Bu noktada ilişkiler ve anahtarlar bizim aradığımız veri bütünlüğünü sağlamamız konusunda yardımcı olacaktır. İlişkiler, bir tablonun diğer tabloya bağlantı kurma aşamasında o tablo üzerinde kaç kayıt ile eşleşebileceğini gösteren türlere sahiptir. Bu özelliklere göre ilişkileri birbirlerinden ayırmaktayız.

Tablolar arasındaki ilişkiyi belirlerken, bu ilişkileri yönlü kabul ederek inceleme yaparsak ilişki türünü belirlemek çok daha kolay olacaktır. Örneğin A tablosunun B tablosuyla olan bağlantısının özelliklerini incelediğimizde çıkan sonuç ile B tablosunun A tablosuyla olan bağlantısının özelliklerini incelediğimizde

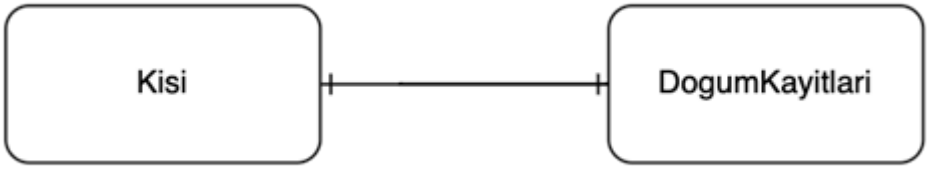
çıkan sonuç ilişki türünü iki adımda elde etmemiz sağlayacaktır. Bu durum örneklerle çok daha iyi anlaşılacaktır.

Şimdi sırasıyla bu ilişki türlerini inceleyelim.

5.1.1. Bire Bir (1-1)

İlk inceleyeceğimiz ilişki türü bire bir olarak adlandırılan ilişki türüdür. Ele alınan iki tablo, karşılıklı olarak yalnızca bir kaydı işaret edecek şekilde bir bağlantı sağladıysa bire bir ilişki türü ele alınır. Anlatımı kolaylaştırmak için A ve B tablolarını ele aldığımızı düşünelim. A tablosunda yer alan herhangi bir kayda karşılık olarak B tablosunda yalnızca tek bir kayıt elde ediyorsak ilişkinin A'dan B'ye yönü "tek" özelliğine sahip olacaktır. B tablosu içerisindeki herhangi bir kayıt A tablosunda yalnızca tek bir kayıt ile eşleşiyorsa bu durumda ilişkinin B'den A'ya olan yönü de "tek" özelliğine sahip olur. Bu durumda A ve B tablolarının arasındaki ilişki türünü bire bir olarak belirtebiliriz.

Örnek vererek bu ilişki türünü açıklayalım. Kisi ve DogumKayitlari tabloları arasındaki ilişkiyi ele alalım. İlişki türünü iki yön için de incelemek en faydalı yaklaşım olacaktır. Kisi tablosunda yer alan bir kayıt DogumKayitlari tablosunda yalnızca tek bir kayda denk gelebilecektir. Çünkü bir kişinin ancak tek bir doğum kaydı bulunabilir. Bu durumda Kisi ve DogumKayitlari tabloları arasındaki ilişkinin Kisi tablosundan DogumKayitlari tablosu yönüne; yani ilişki gösteriminin DogumKayitlari tablosuyla birleştiği noktaya tek işareti koyulmalıdır. Diğer ilişki yönünü de incelersek eğer; DogumKayitlari tablosunda yer alan herhangi bir kayıt, Kisi tablosunda yalnızca tek bir kayıt ile eşleşebilecektir. Çünkü bir doğum kaydı ancak tek bir kişiyle ilgili olabilir. Birden fazla kayıtla eşleşmesi mümkün değildir. Bu durumda ilişkinin diğer yönü yani Kisi tablosuyla birleştiği nokta da tek olarak işaretlenmelidir. İlişkinin iki yönünün de tek türünde belirlenmesi, bu iki tablo arasındaki ilişki türünün bire bir olmasına sebep olmaktadır. Şekildeki özet gösterimi inceleyiniz.

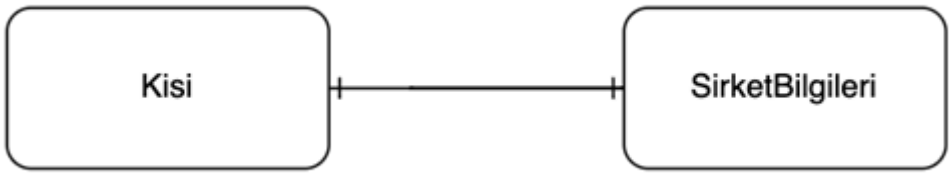


Hatırlayacağınız üzere tablo gösterimlerinde nitelikleri ve anahtarları da ifade ediyorduk ancak şu anda niteliklerle ve anahtarlarla ilgilenmiyoruz. Bu sebeple özet bir gösterime başvurduk. Soldaki kutu Kisi tablosunu, sağdaki kutu ise DogumKayitlari tablosunu ifade etmektedir. Aradaki ilişkinin iki tabloyla birleştiği yerde de tek işaretine sahip olduğuna dikkat etmelisiniz.

Bir şirket veri tabanı içerisinde tüm çalışanlara ait hem kişisel hem de şirketle ilgili bilgilerin kayıt altında olduğunu varsayalım. Bu kayıtlar içerisinde kişinin adı, doğum tarihi ve yeri bilgileri, adresi, vergi kimlik numarası gibi kişisel bilgilerin yanında şirketin hangi bölümünde çalıştığı, ofisinin hangi odada olduğu, hangi pozisyonda görev yaptığı gibi şirket dahilinde geçerli bilgileri de bulunabilir. Tüm bu sayılan özellikler temelde tek bir kişiye ait bilgiler gibi gözükebilir. Ancak biri kişinin kişisel bilgileri, diğerleri ise çalıştığı şirket dahilinde geçerli olan bilgileridir. Bu sebeple bu nitelikler iki gruba ayrılır ve iki tablo ile karşı karşıya kalırız.

Burada tartışmalı bir noktayla karşılaşabiliriz. Sayılan niteliklerin tümü tek bir tablo içerisinde yer alabilir. Doğrudur, bire bir ilişki türü olan tüm tablolar için bu durum geçerlidir. Bu ilişki türüne sahip iki tablo birleştirilerek tek tablo olarak da sunulabilir. Burada tek risk, bazı niteliklerin çok sayıda boş değere sahip olacak olmasıdır. Aynı örnekte şirketin eski çalışanların kayıtlarının da bulunduğunu düşünelim. Kişiyle ilgili kayıtlar varlığını sürdürmeye devam edecektir. Ancak bu kişi şirketten ayrıldığı için şirket içindeki pozisyonu, ofisi ve benzer nitelikler herhangi bir değere sahip olmayacaktır. İki farklı tablo olması durumunda şirket bilgilerinin bulunduğu tabloda eğer kişi hala çalışmıyorsa kayıt eklenmeyebilir. Ancak iki tablonun birleştirilmesi durumunda bunu yapmak mümkün olmayacak, kaydın şirketle ilgili niteliklere denk gelen hücreleri boş değer alacaktır. Bu sebeple bire bir ilişkili tabloları da ikiye ayırarak kullanmak fayda sağlayacaktır.

İlişki türünü araştırmaya geçelim. Kişisel bilgilerin bulunduğu tabloyu Kisi, şirket bilgilerini içeren tabloyu ise SirketBilgileri olarak adlandıralım. Kisi tablosunda yer alan bir kayıt, SirketBilgileri tablosunda ancak tek bir kayda denk gelebilir. Bu sebeple ilişkinin bu yönünde tek işareti kullanılmalıdır. SirketBilgileri tablosunda yer alan herhangi bir kayıt ise yalnızca tek bir kişi kaydıyla eşleşmelidir. Bu durumda ilişkinin bu yönde de tek işaretini alması beklenir. Sonuç olarak bire bir ilişki türünü elde ederiz.

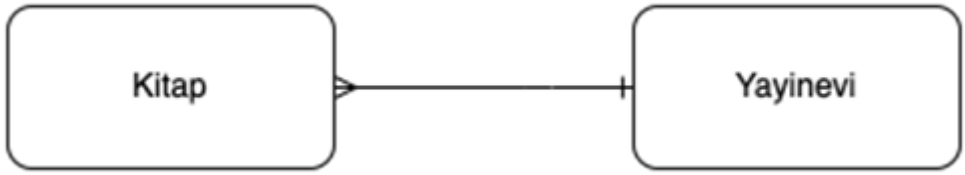


Şekil incelendiğinde bu örnekte de bire bir ilişki türüyle karşılaştığımızı görmüş oluyoruz.

5.1.2. Bire Çok (1-m) / Çoğa Bir (m-1)

İkinci ilişki türümüz bire çok ya da çoğa bir olarak adlandırılan ilişki türüdür. Bu ilişki türü ile şu durumda karşılaşmaktayız: A tablosunda yer alan herhangi bir kayıt, B tablosunda yer alan birden fazla kayıt ile eşleşiyorsa bu ilişki türüne çok adı verilmektedir. Burada dikkat edilmesi gereken, bu durumun her zaman gerçekleşme zorunluluğu olmamasıdır. Yalnızca bir kez bile A tablosundaki bir kayıt B tablosundaki çok kayıtla eşleşse; hatta eşleşmese ancak eşleşme olasılığı olsa çok türünü kullanmak zorundayız. Bir tablodan diğerine çok, ancak tersi yönde tek türünü belirlememiz halinde bu ilişki türü bire çok ya da çoğa bir adını almaktadır. Bunlardan hangisinin seçileceği ise sadece tabloların dizilişiyle ilgilidir. A tablosundan B tablosu yönüne çok, B tablosundan A tablosu yönüne tek türünde ilişki bire çok; iki tablo sadece yer değiştirdiğinde ilişki türü çoğa bir olarak anılır. Örnek ile devam edelim.

Kitap ve kitabın yayınevine ait bilgilerin yer aldığı iki tabloyu düşünelim. Her bir kitap yalnızca tek bir baskı yapmış olsun. Dolayısıyla bir kitap ancak bir yayınevinden dağıtılabilir. İlişki türünü inceleyelim. Kitap tablosunda yer alan bir kayıt, Yayinevi tablosunda birden fazla kayıtla eşleşebilir mi? Hayır. Bir kitap yalnızca bir yayın evi ile eşleşecektir. Peki Yayinevi tablosunda yer alan bir kayıt, Kitap tablosunda birden fazla kayıt ile eşleşebilir mi? Elbette. Yayınevleri çok sayıda kitaba ev sahipliği yapmaktadır. Dolayısıyla Yayinevi tablosundaki bir kayıt, Kitap tablosundaki çok sayıda kayıtla eşleşme olasılığına sahiptir. Bu incelemeye göre ilişkinin Kitap'tan Yayinevi'ne olan yönü tek, Yayinevi'nden Kitap'a olan yönü ise çok olarak belirlenir. Şemayı inceleyerek devam edelim.



Şema çizilirken önce Kitap, sonrasında Yayinevi tablosunun çizilmesi sebebiyle ilişkiler çok ve tek olarak sıralanmıştır. Bu durumda ilişki türünü çoğa bir olarak adlandırmaktayız. Hiçbir değişiklik yapmadan sadece şemada Kitap ve Yayinevi tablolarının yerlerini değiştirsek ilişki türüne bire çok dememiz gerekecekti. Yani bu iki tür arasındaki fark yalnızca tabloları hangi sırada ele aldığımızla alakalıdır.

Bir örnek daha ele alalım. Yapılacak işleri kaydetmeyle ilgili çeşitli uygulamalar var, gündelik hayatta denk gelmişsinizdir. Daha çok "to-do" adıyla geçen bu uygulama türünde yapılacak işleri unutmamak için maddeler halinde kayıt altına alabiliyoruz. Dilersek bunları kategoriler altında gruplayarak daha düzenli bir yapı elde edebiliyoruz. Bu yapı için elimizde iki veri tabanı tablosu olduğunu düşünelim: Gorev ve Kategori. Gorev tablosu eklenen yapılacak işler maddelerini, Kategori ise görevlerin gruplanacağı kategorileri kayıt altına almak için hazırlanan bir tablo olsun. Bu iki tablo arasındaki ilişkiyi inceleyelim.

Gorev tablosunda her alan herhangi bir kayıt, Kategori tablosunda birden fazla kayıt ile eşleşebilir mi? Diğer bir deyişle bir görev birden fazla kategori ile ilişkili olabilir mi? Aslında burada geliştirilen uygulamanın yapısına göre olabilir de olmayabilir de. Ancak biz klasik yaklaşımla yanıtlarsak bir görev yalnızca bir kategorinin altına eklenebileceğine göre Gorev tablosunda yer alan kayıtlar Kategori tablosunda yalnızca bir kayıt ile eşleşebilecektir. İki tablo arasındaki ilişkinin Kategori tablosuna bağlanan ucunda tek işaretinin yer alması gerekmektedir. Peki Kategori tablosunda yer alan herhangi bir kayıt, Gorev tablosunda birden fazla kayıtla eşleşebilir mi? Evet. Bir kategori altında birden fazla görev yer alabilir, dolayısıyla Kategori tablosunda yer alan bir kayıt, Gorev tablosunda birçok kayıtla eşleşebilir. Bu durumda çoğa bir ilişki türüyle karşı karşıyayız. Şema üzerinde inceleyelim.

Gorev ve Kategori tabloları çoğa bir ilişki türüyle bağlantılıdır. Bu sebeple ilişki Gorev tablosuna çok, Kategori tablosuna tek işaretiyle bağlanmıştır.

5.1.3. Çoğa çok (m-n)

Ele alacağımız üçüncü ve son bağlantı türü de çoğa çok bağlantı türüdür. Önceki başlık ve örnekleri inceledikten sonra bu ilişki türünün nasıl belirlendiğini kolaylıkla kavrayabilirsiniz. Özetle yine iki varlık alacağız ve bu iki varlık arasındaki ilişki iki yönde de çok bağlantı türüne sahip olacak. Şimdi açıklayalım. Bir A tablosu içerisinde yer alan herhangi bir kayıt, B tablosu içerisinde birden fazla kayıt ile eşleşebiliyorsa ve B tablosu içerisinde yer alan herhangi bir kayıt A tablosu içerisinde birden fazla kayıt ile eşleşebiliyorsa burada çoğa çok ilişki türünden bahsedebiliriz.

Bir örnek ile devam edelim. Bir video paylaşım sitesinde kullanıcıların kanallara abonelik durumlarını içeren bir veri kümesini ele alalım. Kullanıcılarla ilgili bilgiler Kullanici, sitede yer alan kanalların bilgileri Kanal tablosunda bulunuyor olsun. Kullanici ve Kanal tabloları arasında yer alan ilişki türünü belirleyeceğiz. Kullanici tablosunda yer alan herhangi bir kayıt, Kanal tablosunda birden fazla kayıt ile eşleşebilir mi? Burada eşleşmekten kastımız kanala abone olmak. Dolayısıyla soru diğer bir deyişle bir kullanıcı birden fazla kanala abone olabilir mi? Yanıtımız evet. Böylece iki tablo arasındaki ilişkinin Kanal tablosuyla birleşen ucunun çok türünde olduğunu biliyoruz. Peki Kanal tablosundaki herhangi bir kaydın Kullanici tablosunda birden fazla kayıtla eşleşmesi mümkün müdür? Bu soruyu da diğerine benzer şekilde sorarsak; bir kanala birden fazla kullanıcı abone olabilir mi? Evet. Bu durumda iki tablo arasındaki ilişkinin Kullanici tablosuyla birleştiği yerde de çok ilişki türünü seçmemiz gerekecektir. Sonucu şema ile görelim.

Burada ilişki işaretinin hem Kullanici hem de Kanal tablolarına çok işaretiyle bağlandığını görebiliyoruz.

Bir başka örneği ele alalım. Bir otel içerisinde gelen misafir kayıtları; konaklayan kişi ve bu kişinin hangi odada konakladığı bilgisini içeriyor olsun. Eldeki nitelikler incelendiğinde de basitçe iki tablo elde ettiğimizi varsayalım: Musteri ve Oda. Musteri tablosu, konaklama yapan kişiyle ilgili kişisel nitelikleri içerirken, Oda tablosu, müşterilerin konakladığı odaya dair özelliklerin bulunduğu nitelikleri barındırıyor olsun. Şimdi bu iki tablo arasındaki ilişkiyi inceleyelim. Musteri tablosunda yer alan bir kayıt, Oda tablosunda birden fazla kayıt ile eşleşebilir mi? Yanıt verebilmek için durumu biraz incelememiz gerekecektir. Elbette bir müşteri

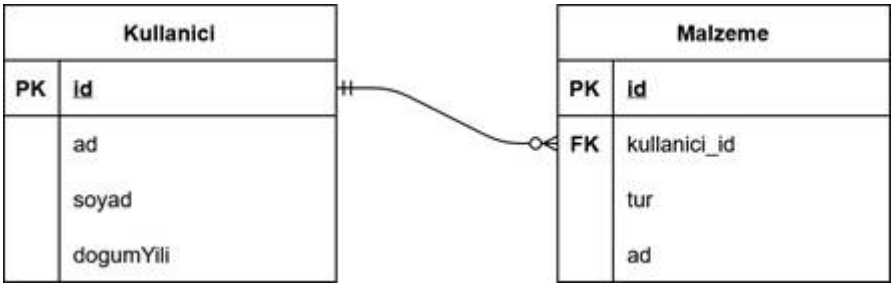
yalnızca bir odada konaklayabilir. Ancak ele alınan bu kayıtlar uzunca süredir tutulmaktaysa ve kişi bu oteli daha önce de ziyaret ederek farklı bir odada konakladıysa birden fazla oda kaydı ile eşleşmesi mümkün olacaktır. Biz veri tabanlarında geçmişe yönelik kayıtları da bulundurduğumuz için ikinci yanıtı kabul ederek devam edebiliriz: Evet, Musteri tablosunda yer alan bir kayıt, Oda tablosunda birden fazla kayıt ile eşleşebilir. Şimdi Oda tablosu açısından ele alalım. Oda tablosu içerisinde yer alan bir kayıt, Musteri tablosunda birden fazla kayıt ile eşleşebilir mi? Yine benzer bir inceleme yapmamız gerekmektedir. Burada, bir odada birden fazla müşteri kalabileceği gibi, zaman içerisinde ilgili oda birden fazla müşteri tarafından kullanılmış olabilir. Bu sebeple iki tablo arasındaki ilişkinin Oda tablosundan Musteri tablosuna olan yönünde de çok türünde bağlantı kurulması gerekecektir. Yanıtı şema üzerinde inceleyelim.

Şemada da görüldüğü üzere Musteri ve Oda tabloları iki yönde de çok işaretiyle birbirlerine bağlanmıştır. Bu sebeple bu örnekteki bağlantı türü de çoğa çok olarak ifade edilebilir.

5.1.4. Zorunlu Kayıt Durumu

İki tablo arasındaki ilişki türü gösterilirken her bir yönde iki işaret kullanılır. Bunlardan birincisinin ne olduğunu önceki başlık altında sunduk. İlişki türünün tek ya da çok olmasına bağlı olarak bir işaretleme gerçekleştirmekteyiz. Bu işaret, bir tabloda yer alan herhangi bir kaydın diğer tabloda birden fazla kayıtla eşleşmesine göre belirlenmektedir. Yani aslında olası en fazla durumu irdelemekteyiz. Bir de olası en az durumu irdelediğimiz bir durum mevcut. Tarif ederek ilerleyelim.

Ele aldığımız bir tablo içerisinde yer alan herhangi bir kayıt, diğer tabloda hiçbir kayıt ile eşleşmeyebilir mi? Bu bizim temel soru cümlemiz. Fark edeceğiniz üzere bir önceki ilişki türünde birden fazla kayıtla eşleşme olasılığını sorgularken, bu ilişki türünde hiçbir kayıtla eşleşme olmaması durumunu irdelemekteyiz. Bu sorunun yanıtına göre; eğer hiçbir kayıtla eşleşmeme ihtimali varsa 0, en az bir kayıtla eşleşme zorunluluğu varsa da 1 işareti koyarak ilişki türünü tamamlıyoruz. Şema ile inceleyelim.



Örnekte Kullanici ve Malzeme tablolarını ve bunlar arasındaki ilişkiyi görmekteyiz. Bu ilişki türünün bire çok olduğunu artık biliyoruz. Ancak bununla birlikte zorunluluk ifadesinin de eklenmiş olduğunu görüyoruz. İlişkiyi iki yönlü de ayrıntılı olarak ele alalım. İlişkinin Kullanici tablosundan Malzeme tablosuna olan yönünde, yani Malzeme tablosuna bitişik yerde çok ve 0 işaretlerini birlikte görüyoruz. Çok işareti; Kullanici tablosunda yer alan bir kaydın, Malzeme tablosunda birden fazla kayıt ile eşleşebileceğini göstermektedir. Yani bir kullanıcı birden fazla malzemeye sahip olabilecektir. Peki, Kullanici tablosunda yer alan bir kayıt, Malzeme tablosunda hiçbir kayıt ile eşleşmeyebilir mi? Yanıtımız evet. Çünkü bir kullanıcı henüz hiçbir malzemeye sahip olmayabilir. Bu durumda kullanıcı kaydı mevcuttur ancak bu kullanıcıya ait herhangi bir malzeme yoktur. Bu yorumumuz neticesinde ilgili ilişki türü için 0 gösterimini kullanmaktayız. Böylece ilişkinin bu yönündeki tür belirlemesi sürecini tamamlamış oluyoruz. Diğer yönü incelersek iki tane 1 işaretinin yer aldığını göreceksiniz. Bunu yorumlamaya çalışalım. Malzeme tablosunda yer alan bir kayıt, Kullanici tablosunda birden fazla kayıt ile eşleşebilir mi? Yanıtımız hayır. Bir malzeme ancak tek bir kişiye ait olabilir. Dolayısıyla tek işaretini kullanıyoruz. En çok durumunu inceledik, şimdi en az durumunu inceleyelim. Malzeme tablosunda yer alan bir kayıt, Kullanici tablosunda hiçbir kayıt ile eşleşmeyebilir mi? Bunun yanıtı da hayır. Bir malzeme sisteme kaydedildiyse bir kullanıcı ile eşleştirilmesi gerekmektedir. Bu

sebeple Malzeme tablosunda yer alan kayıtlar Kullanici tablosunda en az bir kayıt ile eşleşmek zorundadır. Hem en az bir kayıtla eşleşme hem de en fazla bir kayıtla eşleşme durumu birlikte gerçekleştiği için ilişki türü iki tane 1 kullanılarak ifade edilmiştir.

Böylece ilişki türlerinin tüm gösterimlerini ele almış olduk. Bir ilişki türünü tanımlamak için hem en az hem de en çok durumlarını inceleyerek bunu şema üzerinde gösterdik. Şimdi bu ilişkilerin veri bütünlüğünü nasıl sağlayacağı üzerine gideceğiz. Bunu yapmanın yolu da, anahtarlardır.

5.2. Anahtarlar

Önceki başlıkta ilişki türlerini ve zorunluluk ifadesini, nasıl tanımlanması gerektiğini ve şema üzerinde gösterim şeklini gördük. Tabloların birbirleriyle nasıl bağlantılı olduklarını artık biliyoruz. Ancak bir ayrıntı var. Daha önceki başlıklarda da ele aldığımız üzere, veri tabanı tasarımı sürecinde biz en başta elimizde tek bir elektronik tablo olarak bulunan veri kaydını birden fazla tabloya böldük. Böylece elimizde birden fazla bağımsız tablo oldu. Ancak bu tek başına etkin bir çözüm olamaz. Çünkü bağımsız olarak ele alınan veri tabanı tabloları içerisinde yer alan veri, anlam kaybına uğramıştır. Bunun da ötesinde tekrar birleştirmenin de bir yolu kalmayacaktır. Veri tabanının avantajlarından bahsederken veri bütünlüğünün korunuyor olduğunu söylemiştik. Dolayısıyla bu problemi aşacak bir şey yapmamız gerektiği açıktır. Bunun da yolu anahtarları kullanmaktır.

Anahtar kelimesi, bir tabloda yer alan niteliğin temsil edicilik özelliğini göstermektedir. Fonksiyonel bağımlılık konusunda bu durumu detaylı şekilde ele almıştık. Bir tabloda yer alan hangi niteliklerin temsil edici özellikte olduğunu, dolayısıyla hangi niteliklerin diğerlerine fonksiyonel bağımlı olduğunu tespit etmeye çalışıyorduk. Ancak fonksiyonel bağımlılıkların belirlenmesi, anahtar nitelik belirlemekten öte nitelikler arasındaki ilişkileri fark edebilme ve nitelikleri gruplayabilme açısından faydalıdır. “En düşük riskli” niteliğin seçilip anahtar olarak kullanılması, pratikte uygun olmayan bir çözümdür. Bir anahtar nitelik her durumda tekil olarak bir kaydı işaret etmek zorundadır. Bunun istisnası yoktur. Bu sebeple veri tabanı tasarımcıları genellikle anahtar olarak mevcut bir niteliği seçmek yerine yeni bir nitelik tanımlayarak ve bu niteliği yöneterek kendi anahtar niteliklerini oluşturma yolunu tercih ederler. Tablo içerisinde anahtar olmaya müsait bir nitelik olsa bile bu yöntem sıklıkla tercih edilir. Örneğin kişilerle ilgili kayıtlarda kimlik numarası bilgisini anahtar olarak kullanmak oldukça iyi bir çözümdür. Ancak yine de id adında bir nitelik tanımlayarak kullanıcıyı id niteliğine bağlı olarak yönetmek tercih edilmektedir.

İki ayrı tablo, bir ilişki türü ile birbirine bağlanırken tablolarda yer alan nitelikler üzerinden bu süreç gerçekleştirilir. Tablolar arasındaki ilişkiyi gösteren şema, iki tablo üzerinde de bir niteliği işaret eder. Böylece iki tablonun hangi nitelikler üzerinden bağlandığı da tanımlanmış olur. Niteliğin tablo içerisindeki görevi, hangi tür anahtar olduğunu gösterir. Az önce örneğini verdiğimiz, temsil edici nitelikteki anahtarlar birincil anahtar olarak adlandırılırlar. Veri bütünlüğünü korumak için bir birincil anahtarın başka bir tablo içerisinde sunulması durumunda elde edilen niteliğe ise ikincil anahtar adı verilmektedir. Şimdi anahtarlara dair ayrıntıları daha detaylı inceleyelim.

5.2.1. Birincil Anahtar

Birincil anahtar, varlık içerisinde her bir kayıt için benzersiz olan, dolayısıyla her bir değeri tek bir kaydı işaret eden niteliktir (Chen, 1976). Bir diğer deyişle tüm niteliklerin fonksiyonel bağımlı olduğu nitelik denebilir. Sıkça karşılaştığımız ID'ler birincil anahtarların en yaygın örneğidir. Mevcut bir niteliği birincil anahtar olarak kullanmak her zaman bir riske sahiptir. Ancak veri tabanlarında birincil anahtarların asla tekrar etmemesi gerekmektedir. Bu sebeple genellikle yeni bir nitelik birincil anahtar görevini üstlenmek üzere oluşturulur.

Birincil anahtarlar her ne kadar varlıklar arası bağlantıların oluşturulması için kullanılıyor olsa da genellikle tüm varlıklara (bir bağlantıya sahip olup olmadığına bakmaksızın) eklenirler. Dolayısıyla birincil anahtarın ne ve nasıl seçileceği varlığın sahip olduğu bağlantı türlerinden bağımsızdır. Bir kitap için ISBN, bir kişi için kimlikNumarasi ve bir film için IMDB ID'si, veri tabanı yapısı ve varlığın bağlantılarından bağımsız olarak tanımlanabilir.

Birincil anahtar olarak kullanılmak üzere oluşturulacak niteliklerin çeşitli biçimleriyle karşılaşıyoruz. Alt başlıklarda bu türler ele alınacaktır.

Sıralı Sayısal Birincil Anahtar

En sık karşılaşılan bu anahtar türünde kayıtlar 1'den başlayarak ardışık değerler alırlar. Varlığa eklenen her yeni kayıt için en son atanan sayının bir fazlası atanır. Silinen kayıtlar için kullanılan sayılar tekrar kullanılmazlar. Böylece her kaydın benzersiz bir birincil anahtara sahip olması garanti altına alınmış olur. Bunu sağlamak genellikle veri tabanı yönetim sistemi içerisinde ilgili nitelik tanımlanırken birincil anahtar ve AUTO_INCREMENT özelliği verilerek gerçekleştirilir.

Birçok sistem içerisindeki birçok varlık için bu yaklaşım, uygulanmasının da oldukça kolay olması sebebiyle sıklıkla tercih edilir. Ancak hassas veri içeren varlıklar için bu yöntem bir güvenlik açığı oluşturabilir. Örneğin laboratuvar sonuçlarını aldığımız kodlar sıralı sayılar ile kodlanmış olsaydı, dileyen herkes sırayla tüm sayıları deneyerek diğer hastaların kayıtlarına da ulaşabilirdi. Bir diğer örnek olarak; çok fazla içeriğe sahip bir websitesinin her bir içeriğini doğrudan sıralı sayılar ile erişime açması, kötü niyetli birinin kolayca tüm içeriği kolayca ele geçirmesine sebep olabilir. Yazılımsal ve donanımsal yükün çok az olması sebebiyle akla gelecek ilk yöntem olması gereken sıralı sayısal birincil anahtarların bir güvenlik açığı yaratıp yaratmayacağı göz önünde bulundurulmalıdır.

Rastgele Sayısal Birincil Anahtar

Bu yöntem ile birincil anahtar, belirlenen aralıktaki rastgele bir sayı seçilerek oluşturulur. Sıralı olmaması, sıralı olması durumunda karşılaşılacak riski ortadan kaldıracaktır. Bu yöntemin dezavantajı ise her yeni kayıt için yeni bir sayı üretme, bu sayının varlık içerisinde olup olmadığını kontrol etme ve eğer varsa yeni bir tane üretme gibi süreçler sebebiyle zaman maliyetli olmasıdır. Ayrıca belirli bir aralık tanımlanacağı için bu aralığın gelecekte de yeterli olacağından emin olunmalıdır.

Alfanumerik Birincil Anahtar

Sayılar çoğu durumda anahtar görevini çok iyi üstlenmektedirler. Ancak 10 sayı tabanını kullanmamız sebebiyle her bir rakam elde edebileceğimiz farklı durum sayısı 10'dur. Bu da durum sayısı arttıkça sayıyı oluşturan rakamların sayısının da artması anlamına gelmektedir.

Alfanumerik anahtarlar hem sayı hem de karakterleri birlikte kullanmaktadırlar. Burada karakterleri de kullanma amacı kullandığımız sayı tabanını artırmak, bu sayede tek bir karakter ile çok daha fazla durum ifade edebilmektir. Sadece sayıları kullanarak tek bir karakter ile 10 durum ifade edebilirken, sayılar ve İngiliz alfabesini kullanarak 36 durum ifade edebiliriz. Bir diğer deyişle 10 sayı tabanından 36 tabanına geçmiş oluruz.

Örneğin; 8 haneli sayısal bir içerik en fazla $10^8 = 100.000.000$ durum ifade edebilirken, alfanumerik içerik $36^8 = 2.821.109.907.456$ durum ifade edebilir. Bu da yaklaşık 28.000 kat daha fazla seçenek demek. İki durumda da aynı sayıda (8) karakter kullandığımızı hatırlatmak isterim.

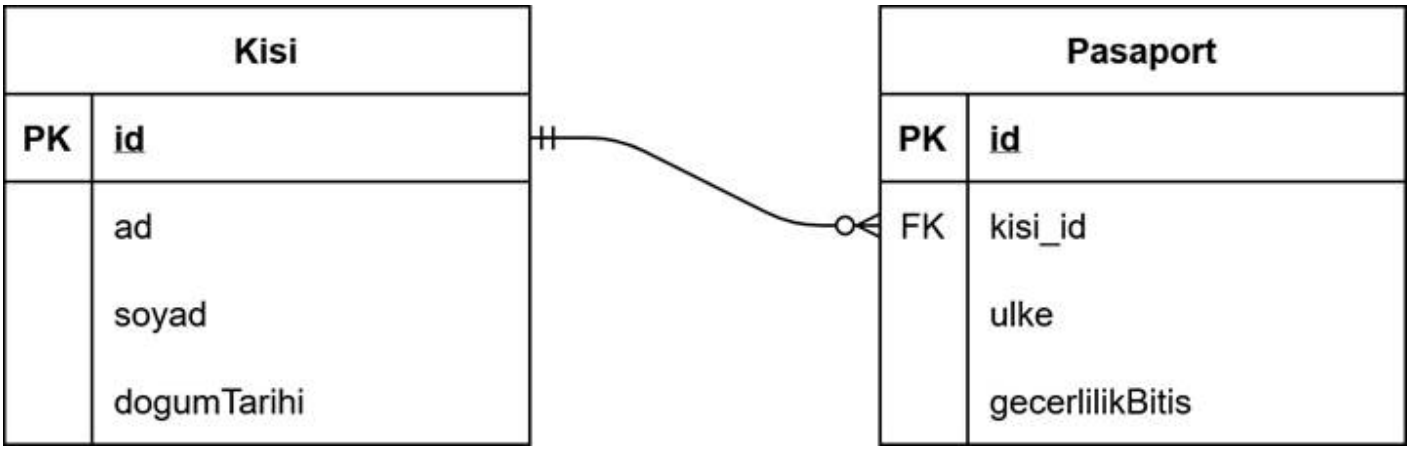
Bu sebeplerle eğer kayıtlarımızın sayısı çok fazlaysa ve biz bu kayıtları daha az karakterle ifade etmek istiyorsak, alfanumerik anahtarlar doğru bir tercih olabilir. Gerçek hayatta doktordan alınan reçete kodları, elektronik imza ispatları ya da renk kodları (16 sayı tabanı kullanırlar) alfanumerik anahtar kullanımına örnek teşkil ederler.

Kontrollü Birincil Anahtar

Kontrollü anahtarlar, anahtarın belirlenen bir kural ya da daha fazla kurala uymasıyla oluşturulurlar. Böylece anahtar rastgele oluşturulamaz. Her bir anahtarın geçerliliği bu kurallara uygunluğu kontrol edilerek doğrulanabilir. Kimlik numaraları bu anahtarlara uygun bir örnektir. 11 haneli olan kimlik numarasının ilk 10 hanesinin toplamının 10'a bölümünden kalanı son rakamı vermelidir. Ayrıca kimlik numarası doğrulamak için daha karmaşık kontroller de vardır. Bu sayede beyan edilmiş bir kimlik numarasının rastgele ya da gerçek olup olmadığı kolayca tespit edilebilir.

5.2.2. İkincil anahtar

Tablolar arası bağlantılar, birbirleriyle ilişkili iki tablodan birinin birincil anahtarının bir başka tablo içerisinde nitelik olarak yer almasıyla sağlanır. Bir başka tablo içerisinde tekrar eden bu anahtar değerlere ikincil anahtar adı verilmektedir. Ayrıntılardan bir örnek üzerinde bahsetmekte fayda var.



Şekil, bire çok ilişkili Kisi ve Pasaport tablolarını içeren bir veri tabanı tasarımı sunmaktadır. Bu tasarımda iki tablo da id adlı birer birincil anahtara sahiptirler. Ayrıca Pasaport tablosu içerisinde kisi_id adlı nitelik, ikincil anahtar özelliğine sahiptir. Bu nitelik, eşleşen kayıtlar için Kisi tablosundaki kayıtların birincil anahtar değerini taşır. Bu sayede Kisi ve Pasaport tablolarındaki kayıtlar olması gerektiği şekilde eşleşebilirler.

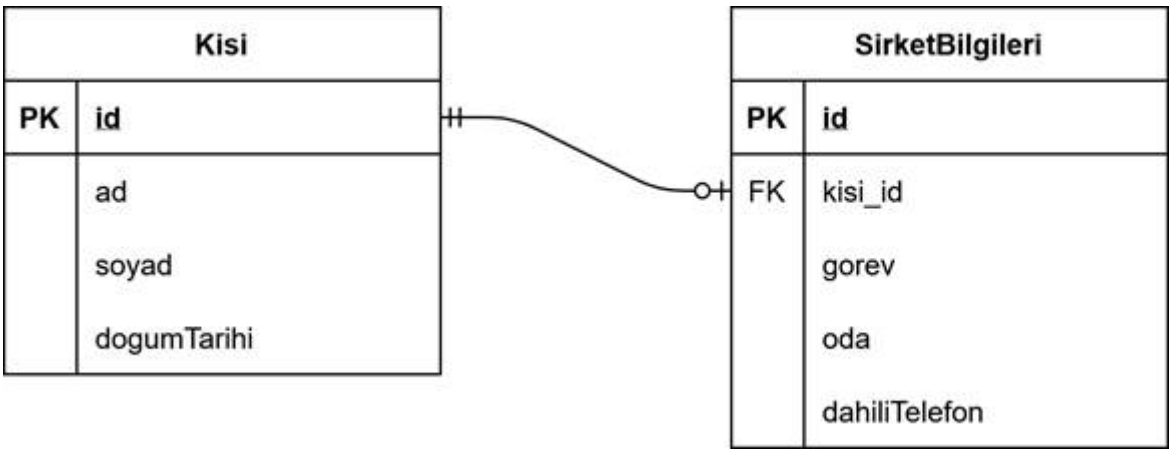
Eğer tablolar özet olarak (yalnızca tablo adını içeren şekilde) gösterilmiyorsa, ilişkiler tablolar arası değil nitelikler arasında tanımlanırlar. Bu sayede hem tablolar arasındaki ilişki tespit edilebilir, hem de hangi birincil ve ikincil anahtarlar kullanılarak verinin birleştirilebileceği rahatça anlaşılabilir.

İkincil anahtarlar bire çok ilişkili tablolar söz konusu iken çok bağlantının sağlandığı tabloya eklenirler. Bunun sebebini anlamak ve ezberlemek zorunda kalmamak için aykırı durumu inceleyelim. Şekil ile verilen örnekte ikincil anahtarın Pasaport değil de Kisi tablosunda olduğunu ve pasaport_id adlı bir nitelik olduğunu varsayalım. Bu durumda Kisi tablosunu incelemek istediğimizde, birkaç örnek kayıt sayesinde Tablodaki gibi bir görüntü elde edebiliriz.

id	pasaport_id	ad	soyad	dogumTarihi
1	1,5,9	Evangeline	Taylor	04.08.2001
2	2,3	Herbert	Todd	01.09.1987
3	6	Alvin	Miles	24.02.1995

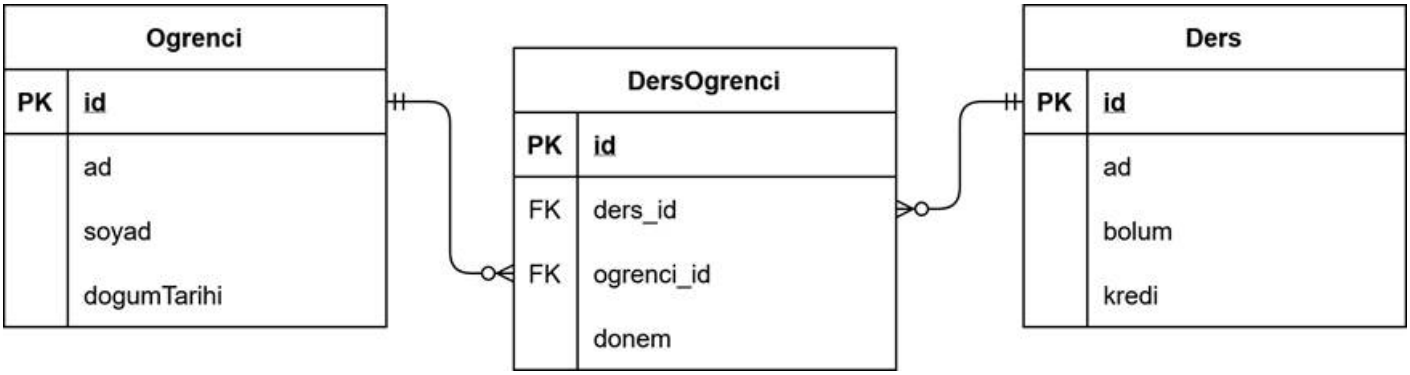
Bire çok ilişkinin; bir tablo içerisindeki tek bir kaydın birden fazla kayda denk gelme durumunda kullanıldığından bahsetmiştik. İkincil anahtar iki tablodan birinde yer almak zorundadır. Eğer Tablo'daki gibi Kisi tablosu içerisinde yer alırsa, her bir kişinin birden fazla pasaporta sahip olabilmesi sebebiyle pasaport_id niteliği birden fazla değer alabilir. Bu da iyi bir veri tabanı tasarımı oluşturmanın öncüllerinden biri olan, bir hücre yalnızca tek bir (atomik) kayda sahip olmalıdır kuralına aykırıdır. Eğer ikincil anahtar Pasaport tablosu içerisinde olursa, her bir pasaport yalnızca tek bir kişiye ait olacağı için her zaman tek bir değer alacaktır. Bu sebeple bire çok ilişkili tablolar için ikincil anahtar her zaman tek bir kayda eşlenen tablo içerisinde yer almalıdır.

Bire bir ilişkili tablolarda eğer ilişki türünde zorunluluk tanımı yoksa ikincil anahtarın hangi tablo içerisinde yer aldığı önemli değildir. Ancak zorunluluğun tanımlandığı durumlarda zorunlu olmayan taraftaki tabloda ikincil anahtarı bulundurmak, toplam veri hacmini azaltan; dolayısıyla en etkili çözüm olacaktır. Şekil buna bir örnek sunmaktadır.



Şekildeki örnekte kişiler ve bu kişilerin şirket bilgileri saklanmaktadır. İlişki türündeki zorunluluk tanımı göz önünde bulundurulduğunda; bir kişinin mutlaka şirket kaydının bulunması gerekmediği; ancak bir şirket bilgisinin mutlaka bir kişiye ait olması gerektiği görülmektedir. İkincil anahtarı Kisi tablosuna yerleştirmemiz durumunda, SirketBilgileri kaydı bulunmayan kişi kayıtları için bu nitelik boş değer alacaktır. Ancak anahtarı SirketBilgileri tablosuna yerleştirirsek mutlaka bir kişiye bağlı olduğu için her zaman bir değeri olacaktır. Ayrıca yine zorunluluk tanımı sebebiyle Kisi varlığındaki kayıt sayısının SirketBilgileri tablosundakinden fazla olması beklenmektedir. Bu durumda az kaydolan tabloya ekleyeceğimiz yeni nitelik, veri tabanının toplam hacminde diğer varlığa eklemeye kıyasla azalma sağlayacaktır.

Son olarak çoğa çok bağlantı türü için ikincil anahtarları ele alalım. Bu ilişki türünden bahsederken üçüncü bir tablo oluşturularak bunun üzerinden bağlantı kurulduğundan bahsetmiştik. Çoğa çok ilişkide her iki tablo da diğer tabloda birden fazla kayda denk geldiğinden ikincil anahtarı bu iki tablodan birine yerleştirerek elde edeceğimiz tasarım, veri tabanı ilkelerine aykırı olacaktır. Bu sebeple oluşturduğumuz üçüncü tablo içerisinde iki tabloyu da temsil etmek üzere iki tane ikincil anahtar tanımlanır. Şekil bunun bir örneğini sunmaktadır.



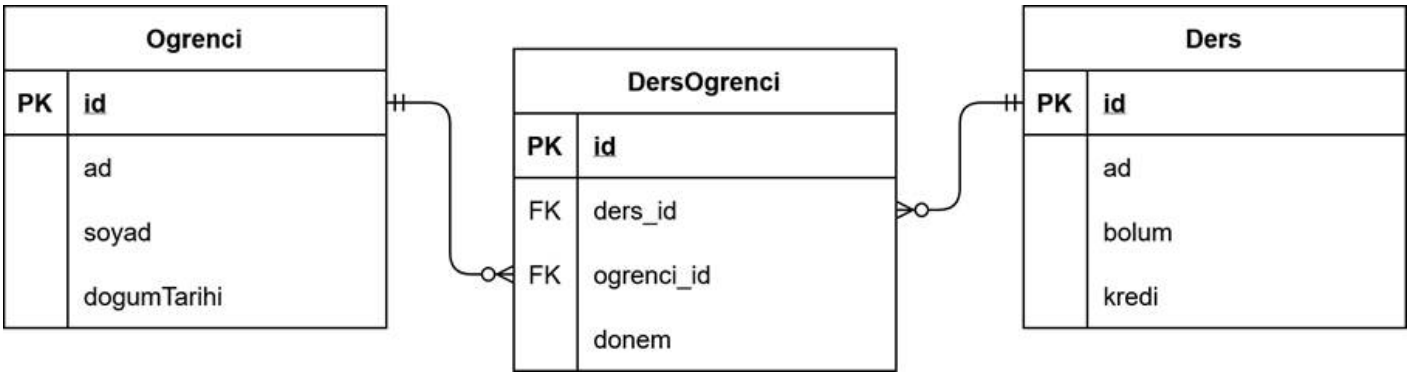
Örnekte verilen veri tabanı tasarımı; bir öğrenci birden fazla ders alabilir, bir ders de birden fazla öğrenci tarafından tercih edilebilir durumuna uygun olarak hazırlanmıştır. Bir öğrencinin aldığı birden fazla dersi Ogrenci tablosunda tutmak ya da bir dersi alan öğrencileri Ders tablosunda tutmak etkin bir yöntem olmayacaktır. DersOgrenci adlı bağlantı tablosu, bu iki tablonun tüm kesişim kayıtlarını tutmaktadır. Her bir ders-öğrenci ilişkisi tek bir öğrenci ve tek bir ders kaydına denk geleceği için ikincil anahtarların bu tabloya yerleştirilmesi gerekmektedir. Bunun yanında bağlantı tablosu, yapılan eşleştirmeyle ilgili farklı kayıtları da barındırabilir. Bu örnekte öğrencinin dersi aldığı dönem bilgisi bağlantı tablosu üzerindedir.

5.3. Bağlantı (Pivot) Tablosu

İki tablo arasında çoğa çok bağlantı kurulduğunda, bu bağlantıyı tablolardan birine ikincil anahtar koyarak gerçekleştiremeyiz. Bunun sebebi, iki tablo için de herhangi bir kaydın karşı tabloda birden fazla kayıtla eşleşme olasılığının olmasıdır. Bir kaydın karşıdaki tabloda çok kaydı işaret etmesinin tek yolu, karşıda yer alan kayıtların, gösterici kaydın birincil anahtarını ikincil anahtar olarak kullanmasıdır. Biraz daha açalım. Kullanıcı ve Kanal tablolarını düşünelim. Bir kullanıcı birden fazla kanalı takip edebilir. Bunu veri tabanına işlemek için Kanal tablosu içerisinde kullanici_id adında bir ikincil anahtar belirlenir ve bu anahtar, Kullanici tablosundaki birincil anahtarın değerini alır. Kanal tablosunda yer alan herhangi bir kayıt, hangi kullanici_id

değerini içeriyorsa aslında o kayıt ile eşleşmektedir. Ancak ilişkinin tersi yönünde de çok kayıt ile eşleşme ihtimali vardır. Bir kanal birden fazla kullanıcı tarafından takip edilebilir. Bunun gösterimi için de Kullanici tablosunda ikincil anahtar bulunması ve Kanal tablosunun birincil anahtarının kullanılması gerekir. Pratik olarak bunu gerçekleştirmek mümkün değildir. Bu sebeple çoğa çok ilişki olduğu durumlarda bir tablo daha oluşturulur ve ikincil anahtarlar bu tablo içerisine yerleştirilir. Böylece Kullanici ve Kanal tablolarının kayıtlarının bir araya geldiği her türlü kombinasyon, bu tablo içerisinde tutulmuş olur.

Elde edilen bu tabloya bağlantı tablosu ya da pivot tablo adı verilir. Bir örnek ile en basit halini inceleyelim ve sonrasında bu tablonun yetenekleri üzerine tartışalım.



Örnekte Ogrenci ve Ders tablolarını görmekteyiz. Bu tablolar arasındaki ilişki türü çoğa çoktur. Bir öğrencinin çok sayıda dersi olabilir ve bir ders çok sayıda öğrenci tarafından tercih edilebilir. Çoğa çok ilişki türüne sahip iki tablo gördüğümüzde hiç tereddüt etmeden bağlantı tablosunu ekleyebiliriz. Çünkü ikincil anahtarları yerleştirmenin tek yolu budur. Eğer bağlantı tablosuna bir ad vermekte zorlanıyorsak alfabetik olarak bağlı tabloların adını kullanmak yaygın bir yöntemdir. Bağlantı tablosunda yer alan her bir kayıt, iki ana tablonun kesiştiği kayıtları ifade etmektedir. Ancak burada bir ayrıntı daha var. Gördüğünüz üzere, bağlantı tablosu içerisinde bir de donem niteliği mevcut. Bunun sebebini incelemeden önce bir ayrıntıyı vurgulamak iyi olacaktır.

Bağlantı tabloları gerçek hayatta genellikle bir işlevi sembolize ederler. İki temel olgu arasında kayda alınması gereken bir işlev olduğunda bu bağlantı tablosu ile kayıt altına alınır. Örnek vermek gerekirse; Ogrenci ve Ders birer varlıktır. Ancak bu ikisinin ilişkisi bir faaliyettir. Öğrencinin dersi alma durumu, faaliyet olarak kayda geçmelidir. Bu sebeple örnekteki bağlantı tablosunun adını DersAlma olarak yazmamız da mümkündü. Şimdi donem niteliğine dönersek eğer; bir bağlantı tablosu, kayıt altına aldığı işlevle birlikte bu işleve dair ayrıntıları da barındırabilir. Eğer Ogrenci ve Ders tabloları DersAlma işlevi altında bir araya geliyorlarsa bu ders alma eylemiyle ilgili diğer ayrıntılar da bu tablo içerisinde yer alabilir.

Sosyal medya siteleri buna güzel bir örnektir. Video paylaşım sitesinden örnek verelim. Kullanici ve Video tablolarının ilişki türünü incelerseniz çoğa çok olduğunu göreceksiniz. Aynı zamanda bir kullanıcının bir video ile çok farklı şekillerde etkileşime geçmesi de mümkündür. Kullanıcı videoyu izleyebilir, beğenebilir, beğenmeyebilir, yorum yapabilir, şikayet edebilir ya da kendisi oluşturmuş olabilir. Tüm bu işlevler için iki tablo arasında bağlantı tabloları kurulabilir. Burada önemli olan ayrıntı şudur ki, her bir işlev için yeni bir bağlantı tablosu kurulması gerekmektedir. Yani çoğa çok bağlantılı iki tablo arasındaki bağlantı tablosu her zaman bir tane olmak zorunda değildir. Bununla birlikte bağlantı tablosu içerisindeki nitelikler de o eylemin ayrıntılarına bağlı olarak belirlenebilir.

Bağlantı tabloları fark edeceğiniz üzere yalnızca ikincil anahtarları yerleştirmek için değil, aynı zamanda faaliyetlerin kayda alınması için kullanılan önemli tablolardır. Birçok sistemde kayıtsal veri dediğimiz veri türleri bu tür tablolarda yer alırlar. Örneğin geçiş bilgileri, satış kayıtları, sürekli tekrarlanan kayıtsal işlemler bu tablolar üzerinden yaparlar. Bu sebeple bağlantı tabloları genellikle çok fazla kayda sahip olur. Yıldız şema başlığı altında bu konuya daha detaylı değineceğiz. Bu başlık itibariyle kesin olan konu şudur ki; çoğa çok bağlantılı iki tablo arasında mutlaka bir bağlantı tablosu tanımlanmalıdır. Bununla birlikte bu bağlantı tablosunun hangi faaliyeti kaydetmek için tasarlandığı da tespit edilmeye çalışılmalı, mümkünse bu faaliyete uygun bir tablo adı verilmelidir. Faaliyetin araştırılması, iki tablo arasında başka faaliyetler bulunmasının da mümkün olduğu durumlarda diğer olası bağlantı tablolarının da belirlenmesini ve uygulanmasını sağlayacaktır. Bu sayede olası bir hatanın önüne geçilebilir.

Bölüm Özeti

İlişkisel veri tabanları iki tür ilişki üzerine kuruludur. Birincisi nitelikler arasındaki ilişkidir. Bu tür ilişkiyi bir önceki başlık altında, fonksiyonel bağımlılıkla birlikte incelemiştik. İkinci ilişki türü ise tablolar arasındaki ilişki türüdür. Bu bölümde, tablolar arasındaki ilişkileri ayrıntılı şekilde ele aldık.

Tablolar arasında üç tür ilişki türü bulunması mümkündür. Bunlar: Bire bir, bire çok ve çoğa çok ilişki türleridir. Tablolar arasındaki ilişki türleri yönlü olarak kabul edilebilir. İlişki türü belirlenirken de bu yönden faydalanılmaktadır. Bir ilişki türü, ele alınan tabloda yer alan herhangi bir kaydın, diğer tabloda birden fazla kayıt ile eşleşip eşleşmeme durumuna göre belirlenir. Birden fazla kayıtla eşleşmesi çok, eşleşmemesi ise tek işaretiyle gösterilir. Aynı tayin işlemi ilişkinin ters yönünde de ele alınarak ilişkinin diğer ucunda yer alması gereken işaret de belirlenir. Nihayetinde ele alınan tablolara bağlanan ilişki türleri sırasıyla okunarak ilişki türü belirlenmiş olur. A ve B tablolarını düşünelim. İkisi arasındaki ilişkiyi belirlerken, bu ilişki türünün B tablosuna bağlanan ucu çok, A tablosuna bağlanan ucu tek işareti alsın. Bu durumda A ve B tabloları arasındaki ilişki türü bire çoktur. Tabloların yerlerini değiştirdiğinizde ilişki türleri de ters döneceği için ilişki türü çoğa bir olarak anılır.

İlişki türü belirlemesinde bir de zorunluluk ifadesi araştırılır. İlişki türü aranırken en çok eşleşme kontrol edilirken, bu araştırmada en az durum araştırılır. Bir tabloda yer alan kaydın, diğer tabloda herhangi bir kayıtla eşleşmemesinin mümkün olup olmadığı kontrol edilir. Eğer mümkün ise 0, değil ise 1 işareti kullanılır. Araştırma iki yönlü de gerçekleştirilerek 1 ve 0 işaretlerinden uygun olan kullanılır.

Bir veri kümesinin birden fazla alt parçaya bölünmesi, veri bütünlüğünün kaybolmasına sebep olabilir. Bu bütünlüğün sağlanması, yalnızca ilişki türlerini belirlemekle mümkün olmamaktadır. Belirlenen ilişki türleri, aynı zamanda anahtar özelliği verilmiş nitelikler üzerinden gerçekleştirilmektedir. İki tür anahtar bulunmaktadır: Birincil ve ikincil anahtarlar.

Birincil anahtar, bir tabloda yer alan temsil edici niteliktir. Bu niteliğe ait herhangi bir değer yalnızca tek bir kaydı işaret etmek zorundadır. Her tabloda mutlaka bir birincil anahtar bulunmaktadır. Mevcut niteliklerden birisi birincil anahtar olarak atanabilir ancak pratikte bu tercih edilen bir yöntem değildir. Herhangi bir risk ile karşı karşıya kalmamak için her tabloda bağımsız bir birincil anahtar niteliği tanımlanır. Çeşitli birincil anahtar belirleme yöntemleri bulunmaktadır. En sık kullanılanı sıralı sayısal birincil anahtardır. Bu anahtar aynı zamanda karşımıza genellikle “id” adıyla çıkar. “id” niteliği, 1’den başlayarak her yeni kayıt için bir arttırılarak değer alan sayısal bir niteliktir ve oldukça kullanışlıdır. Ancak toplam kayıt sayısı ve belirli bir sürede gerçekleştirilen yeni kayıt eklemeleri gibi örtük bilgileri içerisinde barındırdığı için güvenlik öncelikli bazı durumlarda tercih edilmezler.

İkincil anahtarlar ise bir birincil anahtarın bağlantılı olduğu tablo içerisinde veri bütünlüğünü sağlamak için tutulmasıyla elde edilen niteliktir. Bire çok bağlantılı tablolarda ikincil anahtarlar çok bağlantısı kurulan tabloda yer alırlar. Bunun sebebi, bu tabloda bulunan çok sayıda kaydın bağlı olduğu asıl kaydı işaret etmesidir. Bağlantı türü tek olan tabloda ikincil anahtar bulunması, çoklu bağlantı sebebiyle birden fazla değer almasını gerektirir. Bu sebeple tercih edilmez. İkincil anahtarlar her zaman çok türünde ilişkinin bağlı olduğu tabloda bulundurulurlar.

Çoğa çok bağlantıya sahip tablolar, çapraz olarak birçok birleşim kombinasyonuna sahip oldukları için ikincil anahtar ile doğrudan bağlanamazlar. Bu sebeple bu ilişki türünde iki tablo arasına bağlantı tablosu adı verilen, ikincil anahtarların ve dolayısıyla iki tablo arasındaki eşleşmelerin tutulduğu bir tablo oluşturulur. Bağlantı tabloları genelde bir eylemi ifade ederler. Bu sebeple çoğa çok bağlantılı iki tablo arasında mümkün olan her türlü eylem için birer bağlantı tablosu oluşturulabilir. Bir sosyal medya ağında kullanıcı ile içerik arasında beğenme, takip etme, yorum yapma ve benzeri eylemler gerçekleştirilebilmesi buna örnektir.

Kaynakça

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

Hoffer, A. J., Ramesh, V. & Topi, H. 2016. Modern database management. Pearson.

Chen, P. 1976, The entity-relationship model-toward a unified view of data. ACM transactions on database systems (TODS).

Ünite Soruları

Soru-1 :

Bir tabloda yer alan herhangi bir kayıt, diğer tabloda birden fazla kayıt ile eşleşebiliyorsa kullanılan işaret hangisidir?

(Çoktan Seçmeli)

(A) Tek

(B) Çok

(C) 0

(D) 1

(E) İşaret kullanılmaz

Cevap-1 :

Çok

Soru-2 :

Bir tabloda yer alan herhangi bir kayıt, diğer tabloda birden fazla kayıt ile eşleşemiyorsa kullanılan işaret hangisidir?

(Çoktan Seçmeli)

(A) Tek

(B) Çok

(C) 0

(D) 1

(E) İşaret kullanılmaz

Cevap-2 :

Tek

Soru-3 :

İki tablo arasındaki ilişki, iki yönde de çok ise bu ilişki türünün adı nedir?

(Çoktan Seçmeli)

(A) Bire çok

(B) Çoğa bir

(C) Çoğa çok

(D) Bire bir

(E) Geçersizdir

Cevap-3 :

Çoğa çok

Soru-4 :

Bir tabloda yer alan, temsil edici niteliğe verilen ad nedir?

(Çoktan Seçmeli)

(A) Birincil anahtar

(B) İkincil anahtar

(C) Nitelik

(D) Kayıt

(E) Bağlantı niteliği

Cevap-4 :

Birincil anahtar

Soru-5 :

Bir tabloda yer alan kayıt, diğer tabloda mutlaka bir kayıt ile eşleşiyorsa kullanılan işaret hangisidir?

(Çoktan Seçmeli)

(A) Tek

(B) Çok

(C) 1

(D) 0

(E) İşaret kullanılmaz

Cevap-5 :

1

Soru-6 :

Hangisi birincil anahtar türlerinden biri değildir?

(Çoktan Seçmeli)

(A) Sıralı sayısal

(B) Rastgele sayısal

(C) Rastgele ondalıklı

(D) Alfanumerik

(E) Kontrollü

Cevap-6 :

Rastgele ondalıklı

Soru-7 :

İlişkili olan, ancak birleştirilse bile veri tekrarına sebep olmayacak olan, yalnızca boş değerleri tutmamak için kullanılan ilişki türü aşağıdakilerden hangisidir?

(Çoktan Seçmeli)

(A) Bire bir

(B) Bire çok

(C) Çoğa bir

(D) Çoğa çok

(E) Çok

Cevap-7 :

Bire bir

Soru-8 :

Her kayıtta birer artan değer alan, en sık kullanılan anahtar türü hangisidir?

(Çoktan Seçmeli)

(A) Rastgele sayısal

(B) Alfanumerik

(C) Kontrollü

(D) Sıralı kontrollü

(E) Sıralı sayısal

Cevap-8 :

Sıralı sayısal

Soru-9 :

İki tablo arasındaki ilişkinin sağlanması için tablodaki bir niteliğin kazandırıldığı özelliğe ne ad verilir?

(Çoktan Seçmeli)

(A) Bir

(B) Çok

(C) Tek

(D) Anahtar

(E) Nitelik

Cevap-9 :

Anahtar

Soru-10 :

Çoğa çok bağlantı için oluşturulması gereken ek tabloya ne ad verilir?

(Çoktan Seçmeli)

(A) Bağlantı

(B) Nitelik

(C) Kayıt

(D) Anahtar

(E) İlişki

Cevap-10 :

Bağlantı

---

6. NORMALİZASYON

Bölümle İlgili Özlü Söz

Bir yazılımın yeni sürümünü yayınlamak bir botu suya indirmek gibidir. Deliklerin yüzde 80'ini kapatmak yeterli değildir.

Anonim

Kazanımlar

1. Normalizasyon kavramına hakim olur.

2. Normalizasyon süreçlerini bilir.

3. Normal formlar ve uygulama şekilleri hakkında bilgi sahibi olur.

4. Veri tabanı tasarlama sürecini öğrenir.

5. Veri tabanı tasarımı konusunda risk taşıyan durumları öğrenir ve gerekli önlemleri alabilir.

Birlikte Düşünelim

İlişkisel veri tabanı, yalnızca veriyi tablolara bölmek midir?

Tablolara bölünmüş veri bütünleri, nasıl hala aynı bilgiyi taşımaya devam eder?

Bir nitelik, yanlış bir veri tabanı tablosu içerisine yerleştirilirse ne gibi risklerle karşı karşıya kalınır?

Kaçıncı normal form etkin bir veri tabanı için yeterlidir?

Tüm normal formlar uygulanmış bir veri tabanı tasarımı, mükemmel bir veri tabanı tasarımı mıdır?

Normal formları izleyen veri tabanı tasarımcıları her zaman aynı sonuca mı ulaşırlar?

Başlamadan Önce

Veri tabanı tasarımcılığı uzmanlık ve deneyimle doğrudan ilişkilidir. Ancak elbette ki bu uzmanlığı ve deneyimi kazanmak için izlenmesi gereken bir yol vardır. Bunlardan birincisi kuralları öğrenmek, ikincisi ile hatalarla karşılaşarak iyileştirmeler yapmaktır.

Veri tabanı tasarımı hazırlarken, bir ver kümesinin uygun bir veri tabanı tasarımına dönüştürülmesi sürecine normalizasyon adı verilir. Bu süreç aynı zamanda normal form adı verilen kurallar bütününün uygulanmasını da içermektedir. Bir veri tabanı tasarımcısı, tasarım gerçekleştirirken bu kuralları göz önünde bulundurmak zorundadır. Gerçekleştirilen bir veri tabanının tasarımı tüm normal formlara uygun olduğunda etkin bir çözüm olacağı kabul edilmektedir. Ancak elbette kullanım alanına göre ek ihtiyaçlar da ortaya çıkabilir.

Bir veri tabanı tasarımı kullanıma alındığında daha önce düşünülmemiş bazı kullanım şekilleri sebebiyle çeşitli problemler oluşmasına sebep olabilir. Bu da tasarımcılığın deneyimle güçlenmiş tarafının önemini göstermektedir. Zaman içerisinde farklı kullanım şekilleri ve hatalarla karşılaşan tasarımcılar; veri tabanı tasarımcılığı sürecinde normal formları uygulamanın yanı sıra karşılaşılabilecek hataları tahmin eder ve buna göre iyileştirmeler yaparlar. Bazı hatalar henüz meydana gelmeden tahmin edilir ve düzeltilirse "daha iyi" bir veri tabanı tasarımı elde edilmiş olacaktır.

Bu bölümde normalizasyon kavramı ve normal formları inceleyeceğiz.

6.1. Normalizasyon Süreci

Veri tabanı tasarlama süreci kolayca formüle edilebilen, herkes tarafından adım adım ve nesnel olarak gerçekleştirilebilecek bir süreç değildir. Her ne kadar önerildiğinden beri geçen süre içerisinde iyi bir veri tabanı tasarımını tarif eden, tasarım sürecinde dikkat edilmesi gereken adımları sunan ve veri tabanı tasarlamayı öğretmeyi hedefleyen çeşitli çalışma ve yayınlar gerçekleştirilmiş olsa da mükemmel bir sonuç elde edilebilmiş değildir. Konunun uzmanları dahi aynı probleme yaklaşırken farklı çözümler önerebilirler ve işin ilginci bu çözümlerden birini en iyi olarak seçmek mümkün olmayabilir. Bu sebeple veri tabanı tasarlama sürecinde amaç en etkin tasarımı üretmekten öte, hatasız bir tasarım üretmek üzerine kuruludur. Bir veri tabanının hatasız olarak kurgulanmış olması gerçek bir projede kullanılabilmesi için genellikle yeterlidir. Bu durumda veri tabanının kullanımında gelecekte oluşması muhtemel problemlerin öngörülerek bunlar için önlem alınması iyi bir veri tabanı tasarımcısı olmanın gerekliliklerinden biridir.

Dersin bu aşamasına kadar veri tabanı tasarlamanın kesin yargılardan oluşmadığını; deneyim ve uzmanlıklara oldukça bağlı olduğunu vurguladık. Bununla birlikte uzman olsalar dahi farklı tasarımcıların farklı veri tabanı tasarımları sunabileceklerini de söyledik. Tüm bunlar beraberinde şu soruyu getirmekte: İyi bir veri tabanı tasarımcısı nasıl olunur? Deneyime bağlı bir süreçte “daha iyi” olmak ya da “yeterince iyi” olmak nasıl sağlanabilir? Bunun yanıtı; elde edilen tasarımın uygulama altında ne tür problemlerle karşılaştığı ve düzeltmek için hangi adımların atılması gerektiğini deneyimlemektir. Kağıt ortamında tasarlanan bir veri tabanının, kullanıma alınan bir sistem içerisinde çalıştırılması her zaman beklenildiği gibi olmayacaktır. Hangi noktalarda problem yaşanıldığına dair edinilen deneyim, gelecek veri tabanı tasarımlarında karşılaşılabilecek riskler listesine yeni bir madde eklenmesini ve bunu da gözeterek yeni veri tabanı tasarımının yapılmasını sağlayacaktır. Peki veri tabanı tasarımcılığını yeni öğrenirken ne yapacağız, hangi durumların riskli olduğunu, en azından temel risk ve hataları nasıl öğreneceğiz. İşte bunun yanıtı normalizasyon kavramının içerisindedir.

Veri tabanı tasarlama süreci aynı zamanda normalizasyon olarak da karşımıza çıkar. Normalizasyon, eldeki tüm niteliklerin birbirleriyle ilişkili ve veri tabanı ilkelerine aykırılık oluşturmayacak şekilde varlıkların içerisine yerleştirilme sürecidir (Hoffer, J.A., Ramesh, V & Heikki, T., 2016; Lee, H. 1995). Normalizasyon, iyi bir veri tabanı tasarımı elde edebilmek için uyulması gereken kurallar bütünü olarak da görülebilir. Bu kuralların her birine normal form adı verilmektedir. Literatürde 7 normal form ile karşılaşılsa da günlük hayatta genellikle ilk dördü dikkate alınmakta; ilk dört normal forma uyan tasarımlar normalize edilmiş kabul edilmektedir (Fagin, R., 1981; Hoffer, J.A., Ramesh, V & Heikki, T., 2016).

Normal formlar, veri tabanı tasarımı ve normalizasyonu sürecinde bize hazır bir reçete sunmazlar. Bunun yerine her biri veri tabanı tasarımına aykırılık teşkil eden durumları tanımlar ve ele aldığımız veri tabanı tasarımında bu aykırılığın yer almaması gerektiğini belirtirler. Veri tabanı tasarımını inceleyerek belirtilen riski taşıyıp taşımadığını analiz etmek ve tasarımı bu hatadan kurtarmak yine tasarımcının görevidir. Dolayısıyla kesin bir süreç olmamakla birlikte henüz daha performanslı bir yol da bulunamamıştır. Bu sebeple veri tabanı tasarımcılığı uzmanlık ve deneyim gerektiren bir süreç olarak görülmektedir.

Normal formlar, veri tabanının nasıl olması ve olmaması gerektiği konusunda çeşitli yargılar içeren kurallar bütünüdür. Örneğin birinci normal form, hücrelerin atomik yani tek bir değer almasıyla ilgilidir. Ancak bu normal formun uygulandığının bir testi mümkün değildir. Her bir hücre her seferinde gerçekten atomik kayıt mı alır? Bazı örneklerde veri tabanında adSoyad niteliği tanımlanabilir; ad ve soyad bilgisi bu alana kaydedilebilir. Ancak bu durumda, yalnızca ad niteliğine ihtiyaç duyulursa bu bilgi elde edilemeyecektir. Bu da aslında birinci normal formun uygulanmadığı anlamına gelebilir. Ya da bir telefon numarası kaydedilirken numara ve alan kodu birlikte kayıt altına alınmış olabilir ancak sadece alan koduna ihtiyaç duyulursa bu bilgi elde edilemeyebilir. Bu da birinci normal forma aykırılık içerir. Burada vurgulanan nokta, normal formların yargılarıyla birlikte ilgili veri tabanının kullanım amacının da birlikte değerlendirilmesi ve hatta gelecekte ihtiyaç duyulacak raporlama çeşitlerinin de tasarlama sürecinde ön görülmesi gerekmektedir. Tasarlanan bir veri tabanı tüm normal formlara uygun gözükürken ihtiyaç duyulan yeni bir rapor çeşidi sebebiyle bir anda kötü tasarlanmış bir veri tabanına dönüşebilir.

Bu başlık altında normalizasyon sürecinin bileşenleri olan normal formları inceleyeceğiz. Bu kısım dahilinde her bir normal form örneklerle açıklanacaktır.

6.2. Birinci Normal Form

Birinci normal form, veri yapısı içerisinde her bir içeriğin tek bir (atomik) değer almasını gerektirmektedir (Teorey, T. J., etc., 2011). Tablo ile 1NF'ye uymayan bir örnek sunulmuştur.

ad	soyad	iletisim
Lorem	Ipsum	05991234567
Dolor	Sit	05989876543,dolorsit@mail.com
Amet	Consectetur	consetetur@company.com
Adipiscing	Elit	05973456789

Kişilere ait ad, soyad ve iletisim bilgilerini içeren bir örnek görüyoruz. Burada iletisim niteliği hem telefon numarası hem de e-posta adresi kaydı için kullanılmış. Bu sebeple iki iletişim bilgisi birden bulunan kullanıcılar için hücre içerisinde çift kayıt sunulmuştur. Bu durum 1NF'ye aykırıdır. Çözüm için e-posta ve telefon numarası için ayrı niteliklerin tanımlanması ve iletişim yerine bu niteliğin kullanılması gerekmektedir.

Kisi: ad, soyad, ePosta, telefon

Bu sayede tasarım, 1NF'ye uygun hale getirilmiştir.

Bazı durumlar için 1NF hala sağlanmamış olabilir mi? Lütfen inceleyiniz.

6.3. İkinci Normal Form

2NF'ye uygunluk için öncelikle 1NF'ye uygunluk gerekli ve ön şarttır. Sonraki normal formlarda da aynı kural geçerlidir. Yani her bir normal form, bir önceki normal forma uyum sağlanması gerektiği kuralıyla başlar.

Tanım gereği 2NF, bir tabloda anahtar olmayan bir niteliğin birincil anahtarın bir alt kümesi olması durumunda ortaya çıkmaktadır (Kent, W., 1983). Bu durumu birden fazla şekilde açıklayabiliriz. Bunlardan birincisi, bir tablo içerisinde birden fazla tabloya ait niteliklerin toplanması 2NF'ye aykırıdır. Bir diğeri, ele alınan tablo içerisinde iki farklı fonksiyonel bağımlılık denklemi kurulabiliyor olmasıdır. Bu durum da yeni bir tablo oluşturulması gerektiğine işarettir. Üçüncü ve son olarak, ele alınan tabloda iki farklı olguyu işaret eden iki farklı birincil anahtar olması da 2NF'ye aykırılık göstergesidir. Anlaşılacağı üzere bu problemin çözümü varlığı gerekli sayıda yeni tabloya bölmektir. Böylece elde edilecek yeni tablolar 2NF'ye uygun olacaktır. Şimdi bu normal forma aykırılığı ve çözümü bir örnek ile ele alalım. Aşağıdaki tablo 2NF'ye uygun olmayan bir örnek içermektedir.

ad	soyad	not	ders	bolum	programTuru
Lorem	Ipsum	85	Veritabanı Tasarımı	YBS	lisans
Dolor	Sit	76	Programlamaya Giriş	YBS	lisans
Amet	Consecte	93	Veritabanı Tasarımı	YBS	lisans

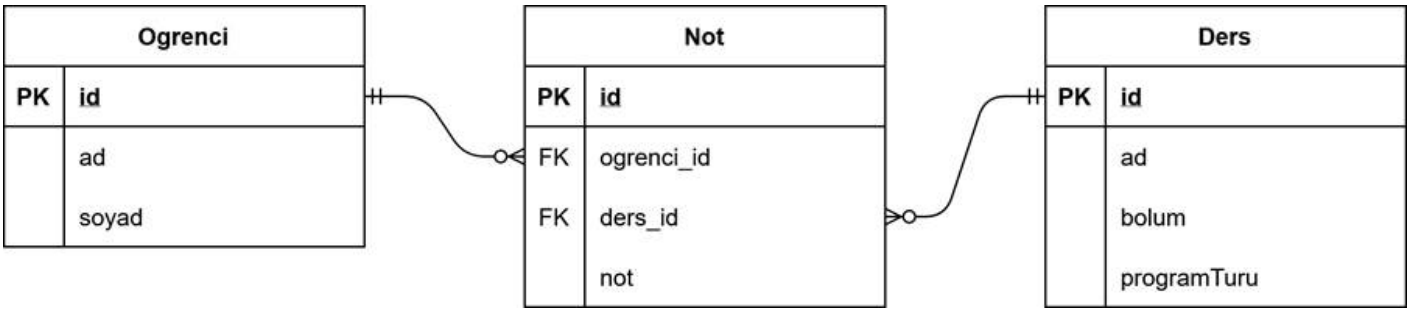
Örnekte öğrencilerin derslerden aldığı notlar yer alıyor. Ayrıca notların hangi derse ait olduğu ve bu dersin bulunduğu bölüm ile bölümün eğitim türü de belirtiliyor. Temelde iki grup bilgi sunduğundan bahsedilebilir. Birincisi öğrenci, ikincisi ise notun alındığı ders. Buradaki not öğrenci ile ilgili, bolum ve programTuru ise ders ile ilgilidir. Dolayısıyla iki bilgi grubundan bahsetmek mümkündür. Çözümlemek için Ogrenci ve Ders adlı iki varlık oluşturmak ve nitelikleri bu gruplara ayırmak gereklidir.

Burada bir alıştırma yapma fırsatı var, kaçırmayalım. Tablo ile verilen örneği bir kez de aşağıda görüyorsunuz.

SinavNotlari: ad, soyad, not, ders, bolum, programTuru

Bu örnek için iki varlık oluşturup nitelikleri bu varlıklara ekledikten sonra varlıkların arasındaki ilişki türünü belirleyiniz.

Bir dersin birçok öğrencisi olması muhtemel olduğu gibi, bir öğrencinin birçok derse ait sınava girmesi de muhtemeldir. Ancak bir öğrenci henüz hiç not almamış olabilir. Benzer şekilde bir dersin de henüz hiç sınavı yapılmamış olabilir. Bu açıklamalar doğrultusunda çözüm Şekil ile verilen gibidir.



Çözüm sayesinde Tablo ile verilen örneğin NF2'ye aykırılığı varlığın iki varlığa bölünmesi sayesinde giderilmiştir. Çözüm önerisindeki üçüncü (Not) varlık, Ogrenci ve Ders varlıklarının çoğa çok bağlantılı olması sebebiyle eklenmiştir. Temelde çözümün iki varlığa bölme işlemiyle elde edildiğini savunabiliriz.

Not: Dikkatinizi çekmiş olabileceği üzere; burada bolum ve programTuru ifadeleri de birer varlık olmayı hak edebilecek içeriğe sahip niteliklerdir. Ancak burada iki varlığa bölünme konu alınmıştır. Eğer örnek için çözümü irdelerken bu konuda takıldıysanız doğru yolda hızla ilerlediğinizi düşünebilirsiniz.

6.4. Üçüncü Normal Form

Bu normal form, bir varlık içerisindeki niteliklerin anahtar özelliğine sahip olanlar dışında hiçbir varlığa fonksiyonel bağımlı olmamasını gerektirmektedir (Bernstein, P. A., 1976). Bir diğer deyişle bir varlıkta anahtar özelliği başka bir nitelik tarafından taklit edilmemelidir. Tablo, 3NF'ye uyumlu olmayan bir örnek sunmaktadır.

id	no	ad	soyad	...
1	325896	Lorem	Ipsum	...
2	514896	Dolor	Sit	...
3	125870	Amet	Consecte	...

Tablo ile verilen örnek öğrenci kayıtlarını içeren bir varlığı temsil etmektedir. Bu varlıkta öğrenci bilgileri id niteliğine fonksiyonel bağımlı olmakla birlikte aynı zamanda no niteliğine de bağımlılık gösterdiği görülmektedir. Bu durum 3NF'ye aykırıdır. Çözümü için bu niteliklerden biri kaldırılabilir. Aşağıda bir çözüm önerisi sunulmaktadır.

Ogrenci: no, ad, soyad, ...

Böylece fonksiyonel bağımlılığı sağlayan tek bir nitelik kalmıştır. Bu haliyle sunulan varlık 3NF'ye uygundur.

Her ne kadar teoride 3NF uyulması gereken zorunlu kurallar arasında sayılsa da pratikte, projelerde bu kurala uyulmadığı ve bu konuda tasarımcının haklı çıktığı birçok uygulama ile karşılaşılabilmektedir. Bu sebeple 3NF'ye uyulmasının avantaj ve dezavantajlarını kısaca ele alalım.

3NF'ye uyulmaması durumu, aynı görevin birden fazla nitelik tarafından gerçekleştirilmesine sebep olabilir. Örnek üzerinden gidersek öğrenci işaret edici için sistemin bir kısmında Ogrenci.no bir kısmında ise Ogrenci.id kullanılırsa, geliştirme ve yönetimde, sonu hatalı veri bütünlüğüne gidebilecek birçok karmaşıklığa sebep olabilir. Bunun yanında ikinci bir nitelik veri hacminde de bir artışa sebep olacaktır.

Bunların yanı sıra; eğer geliştirme esnasında kurallar net olarak ortaya konulursa ve iyi bir dokümantasyon yapılırsa bu karmaşıklıklar ortadan kalkabilir. Bununla birlikte birincil anahtarın her zaman sistem tarafından

atanan ve hiçbir şekilde sistem yöneticileri tarafından müdahale edilemeyen bir yapıda olması veri bütünlüğünü garantiye alabilir. 3NF'ye uymamak uğruna yapılacak bu hareket ile bazı dezavantajlar göze alınacak olsa bile veri bütünlüğünü her daim sürdürmek daha öncelikli bir hedef olmalıdır.

6.5. Boyce-Codd Normal Form

Fonksiyonel bağımlılığı tanımlarken bir nitelik kümesinin başka bir nitelik kümesi tarafından temsil etmesinden bahsetmiştik. Fonksiyonel bağımlılığın tanımına ve başlarda yaptığımız atölyelere göre temsil eden tarafta niteliklerden oluşan bir kümeden bahsetmek oldukça normaldi. Ancak bu normal form ile bu konuda bir kısıtlamaya gidiyoruz. BCNF, varlık içerisinde tek bir birincil anahtar olmasını ve nitelikler arasında bir kısmi fonksiyonel bağımlılık olmamasını gerektirmektedir (Teorey, T. J., etc., 2011). Yani fonksiyonel bağımlılıkların tarifini, BCNF'ye uyan tasarımlar için bir nitelik kümesinin tek bir nitelik tarafından temsil edilebilmesi şeklinde güncellememiz gerekmektedir. Fonksiyonel bağımlılık başlığı altında sunduğumuz çözümlerde sol tarafta sıklıkla bir nitelik kümesi kullanmıştık. Ama aynı zamanda bunun en iyi çözümünün bir id kullanmak olduğundan da bahsetmiştik. Bir varlık içerisinde birçok niteliği kullanarak anahtar oluşturmak yerine id gibi bir nitelik ile bunu sağlamak neredeyse her zaman yeterli seviyede performans sağlayacaktır.

Bir varlık içerisinde, bir varlık kümesine fonksiyonel bağımlılık gerçekleşmesi durumunda bu çözüm id veya benzeri bir niteliğin eklenmesiyle kolayca çözülebiliyor ancak eğer veri kümesindeki mevcut fonksiyonel bağımlılık zaten tek bir niteliğe ise bu niteliği anahtar olarak kullanmak BCNF'ye uyum için yeterli oluyor. Teoride bu çözüm yeterli gözükse de pratikte veri kümesinde bulunan bir niteliğin anahtar olarak tanımlanması pek de tercih edilen bir yöntem değildir. Seçilecek, veri kümesinde var olan niteliğin anahtar olarak tercih edilmesi, gelecekte bu anahtarın içeriği, rolü ya da kullanım şeklinde bir güncelleme yapılması riski sebebiyle her zaman bir riski beraberinde getirmektedir. Öğrencilere ait verileri içeren bir veri kümesindeki öğrenci numarası niteliğini örnek alalım. Öğrenci bilgilerinin öğrenci numarasına fonksiyonel bağımlı olduğu söylenebilir. Böylece bir öğrenci varlığı için öğrenci numarası birincil anahtar olarak tercih edilebilecektir. Ancak zaman içerisinde kayıt olmuş tüm öğrencilerin sayısı çok artacağı için öğrenci numarasının sıfırlanması söz konusu olabilir. Sık karşılaşılan bir şey olup olmadığını bilmemekle birlikte bu örnek olayı yaşamış biri olarak söyleyebilirim ki bu durumda bir süre sonra aynı öğrenci numarasına sahip birden fazla öğrenci veri tabanında yer alabilir. Bu durum gerçekleştiği anda veri tabanı, veri bütünlüğünü sağlamak adına aynı numaralı yeni bir öğrenci kaydına izin vermez. Bu da sistemin bir arızası olarak karşılanır.

Tüm bu sebeplerden dolayı varlıklar içerisinde dışardan etki edilmeyecek, çalışma mekaniği değiştirilmeyecek ve birincil anahtar özelliklerini asla kaybetmeyecek niteliklerin seçilmesi oldukça önemlidir. Bunun da en kolay yolu herhangi bir sorgulama yapmadan her varlığa id niteliği eklemektir.

6.6. Dördüncü Normal Form

Verinin kendini tekrar etmesi veri tabanı tasarımları için önemli bir problemdir. Bu problem çok nadir koşullar dışında hoş görülmez ve giderilmesi gereklidir. 4NF, bir veri kümesinde bazı nitelikler farklı değerler alıyorken bir niteliğin aynı değeri tekrar alması durumunda ortaya çıkmaktadır (Fagin, R., 1977). Bir niteliğin aldığı değerler kendini tekrar ediyorsa bu niteliğin bulunduğu varlığın değiştirilmesi ya da yeni bir varlık tanımlanması gerekiyor olabilir. Bir önceki tablodan uyarlanan aşağıdaki Tablo, 4NF'ye uymayan bir örnek içermektedir.

ders	bolum	programTuru
Veri Tabanı Tasarımı	YBS	lisans
Programlamaya Giriş	YBS	lisans
Sistem Analizi ve Tasarımı	YBS	lisans

Tablo sırasıyla dersleri ve bu derslerin yer aldığı bölümler ile diploma programı türlerini sunmaktadır. Her bir satır, yeni bir ders için kayıt içerirken aynı bölümde ve aynı diploma programı seviyesinde birden fazla ders olması sebebiyle bu değerlerin tekrar ettiğini görmekteyiz. Bunun çözümü bölümler için yeni bir varlık

oluşturmak ve bire çok bağlantı sebebiyle derslerin olduğu varlığa ikincil anahtar olarak bolum_id eklemektir.

Fark ettiyseniz bu problem genellikle kategorik değer alan niteliklerle karşımıza çıkmaktadır. Eğer bir nitelik kategorik değer alacaksa henüz herhangi bir kayıt girmeden 4NF'yi düşünebilir ve gerekiyorsa önlem alabilirsiniz.

6.7. Beşinci Normal Form

Literatürde PJNF (Projection-Join Normal Form) olarak da anılan 5NF, eldeki çözümün daha küçük varlıklara bölünüp bölünemeyeceğini konu alır (Fagin, R., 1979). Bu normal forma göre; bir veri tabanı tasarımındaki tüm varlıklar, veri kaybı yaşanmadığı sürece bölünebilecekleri en küçük varlıklara bölünmelidir. Eğer bir varlık iki varlığa bölünebiliyorsa 5NF'ye uyumsuzluktan bahsedilebilir.

6.8. Alan/Anahtar Normal Form

Alan/Anahtar Normal Form (Domain/Key Normal Form - DKNF) olarak da adlandırılan 6NF, veri tekrarının bir başka formunun veri tabanı tasarımında bulunmasını konu almaktadır. Bu normal forma göre bir nitelik, başka bir niteliğin bir fonksiyonu ise; diğer bir deyişle bir nitelik başka bir niteliğe bağlı olarak hesaplanabiliyorsa bu bir nevi veri tekrarıdır (Fagin, R., 1981). Bu tür bir veri tekrarı 6NF'ye uyumsuzluk anlamına gelmektedir.

Bir varlık içerisinde hem doğum tarihi hem de yaş niteliklerinin bulunması 6NF'ye uyumsuzluğa güzel bir örnek teşkil eder. Yaş, doğum tarihine bağlı kolaylıkla ve nesnel olarak hesaplanabilen bir niteliktir. Bu sebeple bu iki niteliği taşıyan bir varlık 6NF'ye aykırıdır.

Bölüm Özeti

Normalizasyon süreci; bir veri tabanının, ilişkisel veri tabanı ilkelerine uygun, veri bütünlüğünü sağlamış ve yüksek performansla çalışan bir veri yapısı elde etme sürecidir. Dersin önceki bölümlerinde iyi bir veri tabanı tasarımı yapmanın deneyimle mümkün olduğunu vurgulamıştık. Ancak elbette öğrenme sürecindeki tasarımcıların da veri tabanı tasarımı gerçekleştirebilmeleri mümkün olmalıdır. Veri tabanı tasarımı sürecinin formüle edilmesini sağlayan çeşitli kurallar bulunmaktadır. Bu kurallar normal formlar olarak anılırlar. Normal formlara uygun hale getirilen veri yapılarının normalize edildiği yani iyi bir tasarıma sahip olduğunu savunulmaktadır.

Normal formlar; birinci, ikinci, üçüncü, Boyce-Codd, dördüncü, beşinci ve alan/anahtar normal form olarak literatürde yer almaktadır. Her bir normal form ilk öncül olarak veri yapısının bir önceki normal formda olması gerektiği kuralını içerir. Bununla birlikte her normal form, veri yapısı için uyulması gereken yapıyı tarif eder ve kurallaştırır.

Birinci normal form verinin atomik yapıda olması gerektiğini içerir. İkinci normal form, aynı tablo içerisinde alt fonksiyonel bağımlılıklar içeren nitelik kümeleri bulunmaması gerektiğini, yani yeni tablolar oluşturulması gereken durumları açıklar. Üçüncü normal formda birden fazla birincil anahtar olmaması gerektiği vurgulanır. Nitelikler, birden fazla niteliğe fonksiyonel bağımlı olmamalıdır. Boyce-Codd da yine tek bir birincil anahtar olması ve ona bağımlılıkla ilgili kuralları belirtmektedir. Dördüncü normal form veri tekrarının olmaması gerektiği bilgisini içerir. Veri tekrarı veri tabanı yapıları için ciddi bir problemdir. Beşinci normal form, bir tablo daha fazla tabloya bölünebiliyorsa bölünmelidir kuralını içerir. Veri yapısı, mümkün olan en küçük nitelik gruplarına bölünmeli ve veri bütünlüğü korunmalıdır. Alan/anahtar normal form, bir niteliğin başka bir niteliği kullanarak türetilmemesi gerektiği bilgisini içerir. Örneğin kişilerle ilgili bir tabloda hem doğum tarihi hem de yaş bilgisi saklanmamalıdır.

Kaynakça

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

Bernstein, P. A., 1976. Synthesizing third normal form relations from functional dependencies. ACM Transactions on Database Systems (TODS).

Fagin R. 1977. Multivalued dependencies and a new normal form for relational databases. ACM Transactions on Database Systems (TODS), 2(3):262–278.

Fagin, R. 1979. Normal forms and relational database operators. In Proceedings of the 1979 ACM SIGMOD international conference on Management of data, 153–160.

Fagin, R. 1981. A normal form for relational databases that is based on domains and keys. ACM Transactions on Database Systems (TODS), 6(3):387–415.

Hoffer, J. A., Ramesh, V. & Topi, H. 2016. Modern database management. Pearson.

Kent, W. 1983. A simple guide to five normal forms in relational data- base theory. Communications of the ACM, 26(2):120–125.

Lee, H. 1995. Justifying database normalization: a cost/benefit model. Information processing & management, 31(1):59–67.

Teorey, T. J., Lightstone, S. S., Nadeau, T & Jagadish, H. V. 2011. Database modeling and design: logical design. Elsevier.

Ünite Soruları

Soru-1 :

Veri tabanının iyi bir şekilde tasarlanması sonucunda ulaştığı duruma verilen ad nedir?

(Çoktan Seçmeli)

(A) Enformasyon

(B) Normalize

(C) Veri ambarı

(D) Normal form

(E) Bilgelik

Cevap-1 :

Normalize

Soru-2 :

Verinin atomik olarak saklanmasını gerektiren normal form hangisidir?

(Çoktan Seçmeli)

(A) Birinci NF

(B) İkinci NF

(C) Boyce-Codd NF

(D) Beşinci NF

(E) Alan/anahtar NF

Cevap-2 :

Birinci NF

Soru-3 :

Aynı tablo içerisinde alt fonksiyonel bağımlılıklar olmamasını gerektiren normal form hangisidir?

(Çoktan Seçmeli)

(A) Birinci NF

(B) İkinci NF

(C) Üçüncü NF

(D) Dördüncü NF

(E) Beşinci NF

Cevap-3 :

İkinci NF

Soru-4 :

Bir tablo içerisinde birden fazla birincil anahtar özellikli niteliğin olmaması gerekliliği hangi normal form ile belirtilmektedir?

(Çoktan Seçmeli)

(A) Birinci NF

(B) İkinci NF

(C) Üçüncü NF

(D) Dördüncü NF

(E) Beşinci NF

Cevap-4 :

Üçüncü NF

Soru-5 :

Tek bir birincil anahtar olması ve nitelikler arasında bağımlılık olmaması durumu hangi normal form ile belirlenmiştir?

(Çoktan Seçmeli)

(A) İkinci NF

(B) Üçüncü NF

(C) Boyce-Codd NF

(D) Dördüncü NF

(E) Alan/anahtar NF

Cevap-5 :

Boyce-Codd NF

Soru-6 :

Hangisi veri tekrarı olmaması gerektiğini belirten normal formdur?

(Çoktan Seçmeli)

(A) Üçüncü NF

(B) Dördüncü NF

(C) Beşinci NF

(D) Boyce-Codd NF

(E) Alan/anahtar NF

Cevap-6 :

Dördüncü NF

Soru-7 :

Mümkün olan en fazla/küçük tablolara bölünme gerekliliğini belirten normal form aşağıdakilerden hangisidir?

(Çoktan Seçmeli)

(A) Üçüncü NF

(B) Dördüncü NF

(C) Boyce-Codd NF

(D) Beşinci NF

(E) Alan/anahtar NF

Cevap-7 :

Beşinci NF

Soru-8 :

Bir niteliğin başka bir nitelik kullanılarak üretilememesi gerekliliğini belirten normal form hangisidir?

(Çoktan Seçmeli)

(A) Üçüncü NF

(B) Dördüncü NF

(C) Boyce-Codd NF

(D) Beşinci NF

(E) Alan/anahtar NF

Cevap-8 :

Alan/anahtar NF

Soru-9 :

Hangisi normal formlardan birisi değildir?

(Çoktan Seçmeli)

(A) Sıfırıncı NF

(B) Birinci NF

(C) İkinci NF

(D) Üçüncü NF

(E) Dördüncü NF

Cevap-9 :

Sıfırıncı NF

Soru-10 :

Veri yapısının ikinci normal forma uygun hale gelmesi için aynı zamanda hangi normal forma uygunluk zorunludur?

(Çoktan Seçmeli)

(A) Sıfırıncı NF

(B) Birinci NF

(C) İkinci NF

(D) Üçüncü NF

(E) Dördüncü NF

Cevap-10 :

Birinci NF

---

7. ÖZEL VERİ TABANI YAPILARI

Bölümle İlgili Özlü Söz

Bu, teknolojiye olan inanç değil; insana olan inançtır.

Steve Jobs

Kazanımlar

1. Özel durumlara uygun veri tabanı tasarımı gerçekleştirebilir.

2. Kayıtsal veri yapıları için uygun veri tabanı yapısını seçebilir.

3. Veri tabanı şeması gördüğünde veri akışı türünü tahmin edebilir.

4. İlişkili olmayan tablolar tanımlayabilir.

5. Klasik elektronik tablolar ile gerçekleştirilemeyecek, daha etkin veri yapıları oluşturabilir.

Birlikte Düşünelim

Her zaman yapmamız gereken eldeki bir veri kümesini bir veri tabanına dönüştürmek midir?

İki tablo arasında başka türde ilişkiler tanımlanabilir mi?

Veri tabanında ne tür veriler saklanabilir?

Etrafınızda yer alan, en sık kullandığınız bilişim sistemlerini ele alın. Sizce arka planda nasıl bir veri tabanı yapısına sahip olabilirler? Hangi tablolar ve hangi ilişkiler kullanılıyor olabilir?

Verinin değişip değişmediğini anlamak için neler yapılabilir?

Bir veriyi saklamadan saklamak (sadece doğrulayabilmek) mümkün müdür?

Başlamadan Önce

Veri tabanları her zaman aynı türde tablolar, nitelikler ve ilişkiler içermezler. Bazı durumlarda, probleme ya da elde edilmek istenilen duruma göre bir veri tabanı tasarımı tasarlamak gerekebilir.

Bu bölüm içerisinde klasik yaklaşımdan farklı ancak yine de sık karşılaşılan özel durumlar ve bu durumlarda kullanılacak veri tabanı tasarımlarına yer verilmiştir. Bu bölümde öğreneceğiniz tasarımların birçoğunu doğrudan kullanacak, bazılarını da ihtiyaç duydukça daha büyük veri tabanlarının bir bileşeni olarak sürece dahil edeceksiniz.

Yıldız şema, hiyerarşik veri, özetleme, bağlantısız varlık ya da varlık grupları ve zaman damgasıyla ilgili süreçler ve örnekler, ilgili alt başlıklar içerisinde sunulmuştur.

7.1. Yıldız Şema

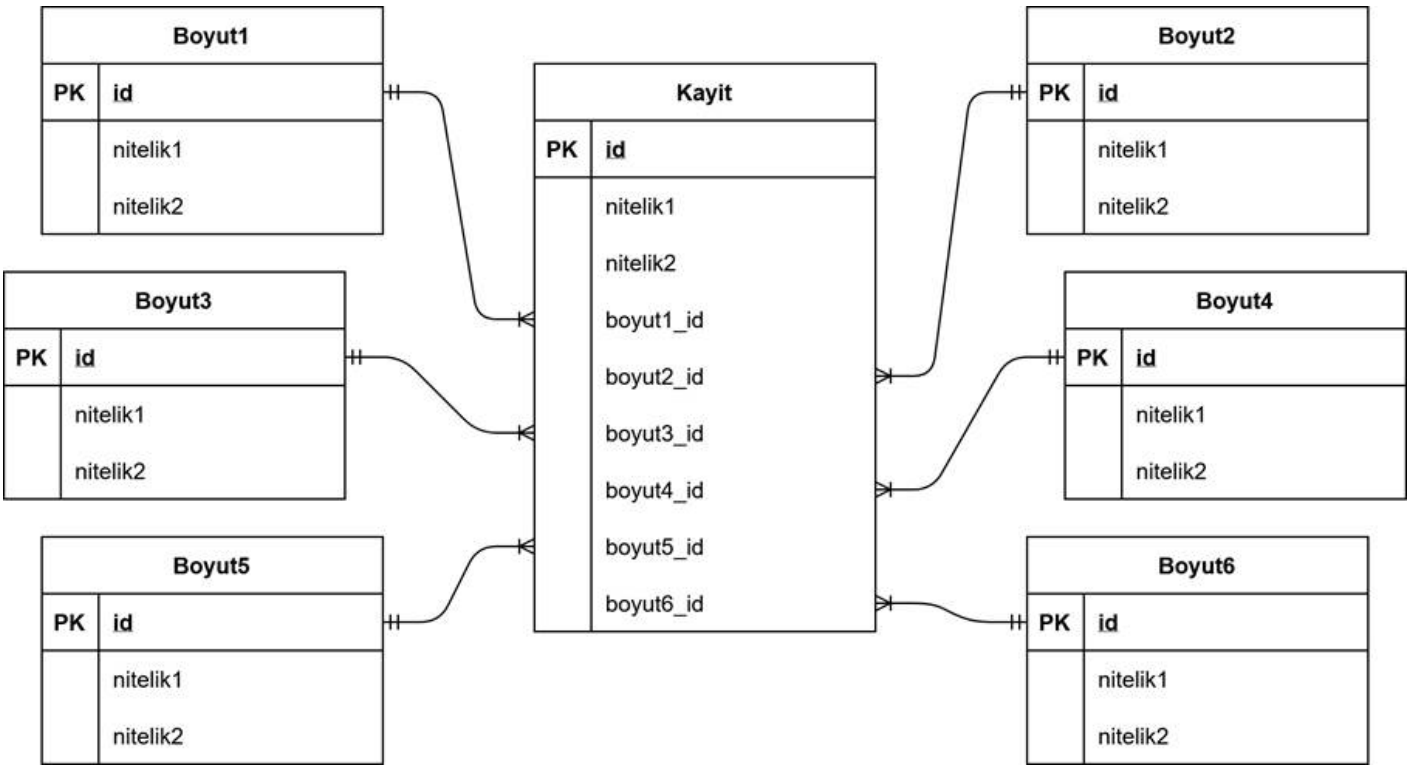
Sistemler, kullanım türlerine göre farklılıklar göstermektedirler. Veri tabanı içerisinde yer alan her bir tablo, genellikle bir olguyla eşleştirilir. Her bir bağlantı tablosu da bir eylemin kayda alınmasıdır. Dolayısıyla olguların yoğunlukla kayıt altına alındığı örneklerde çoğa çok bağlantı türüne sık rastlanmaz. Bir kütüphane örneğini alalım. Ancak bu örnekte kişilerin kütüphaneden kitap ödünç alma durumu yok. Yalnızca mevcut kitapların kayıt altına alınması isteniyor. Bu durumda kabaca bu veri tabanı yazar, kitap, dolap ve benzeri tablolar barındıracaktır. Her bir tablo, kütüphane içerisindeki olguları temsil edecek ve sınırlı sayıda kayıt

alacaktır. Tablo adları birer olguyu işaret etmekle birlikte tablo içerisinde yer alan kayıtlar bu olguların miktarını gösterici niteliktedir. Şimdi bu örnekte kişilerin kitapları ödünç alabilmesi durumunu da ele alalım. Bu durumda tüm ödünç alma durumlarının kayıt altına alınması gerekecektir. Odunc (ödünç) adında bir tablo tanımlayalım ve ödünç alınan kitapların kayıtlarını bu tabloya eklediğimizi düşünelim. Bu tablo hangi tablolarla ilişkili olacaktır? Başka bir şekilde sorarsak bir ödünç alma eylemi hangi olgularla ilişkilidir? Bir "kişi", bir "kitap" ödünç alıyor değil mi? Dolayısıyla Kisi ve Kitap tablolarıyla ilişkili olduğunu rahatlıkla görebiliyoruz. Bununla birlikte az önce bir ipucu daha sundum. Ödünç alma "eylemi" diye vurguladım. Bu bir eylem, yani muhtemelen bir bağlantı varlığı olacak. Dolayısıyla burada çoğa çok bağlantı olduğundan bahsedebiliriz. Gerçekten de Kisi ve Kitap tabloları arasındaki bağlantıyı incelediğimizde çoğa çok bağlantı olduğunu görebiliriz.

Ele aldığımız kütüphane örneğinde kitap ödünç alınabilmesi ve alınamaması arasında bir fark bulunmaktadır. Odunc tablosu olması en büyük farklılık elbette ancak daha geniş bakarsak; ödünç alma eylemi bulunduğunda bu veri tabanına yapılan giriş sayısı çok daha artacaktır. Sadece mevcut kitapların kayıtlarının tutulduğu durumda kütüphaneye yeni bir kitap geldiğinde, bir kayıt güncelleneceği zaman ya da silinmesi gerektiğinde işlem gerçekleştirilecektir. Ancak ödünç alma durumu işin içine girdiğinde sayılan eylemlere ek olarak her bir ödünç alma durumunun da kayıt altına alınması gerekecektir. Bu eylemin sıklığını da düşündüğümüzde belki de veri tabanındaki en hacimli ve en sık kullanılan tablonun bu olduğunu varsayabiliriz.

Bu türde, sıklıkla yapılan işlemlerin kayıtlarının saklandığı veri tabanı türüne "kayıtsal veri tabanı" (transactional database) adını vermekteyiz. Bu tür veri tabanlarında tüm işlemlerin yer aldığı, yüksek hacime sahip ve işlemlerin neredeyse tamamının üzerinde gerçekleştirildiği bir tablo bulunmaktadır. Otoban geçişleri, sınav sonuçları, etkinlik bileti satışı, banka hesapları gibi birçok durum bu kullanım şekline örnek olarak gösterilebilir.

Sıklıkla karşılaşılan bu veri saklama yöntemine bir ad verilmektedir. Yıldız şema (star schema) adı verilen bu tasarımda kayıtlar ve bu kayıtlarla ilgili olgular bir arada bulunmaktadır.



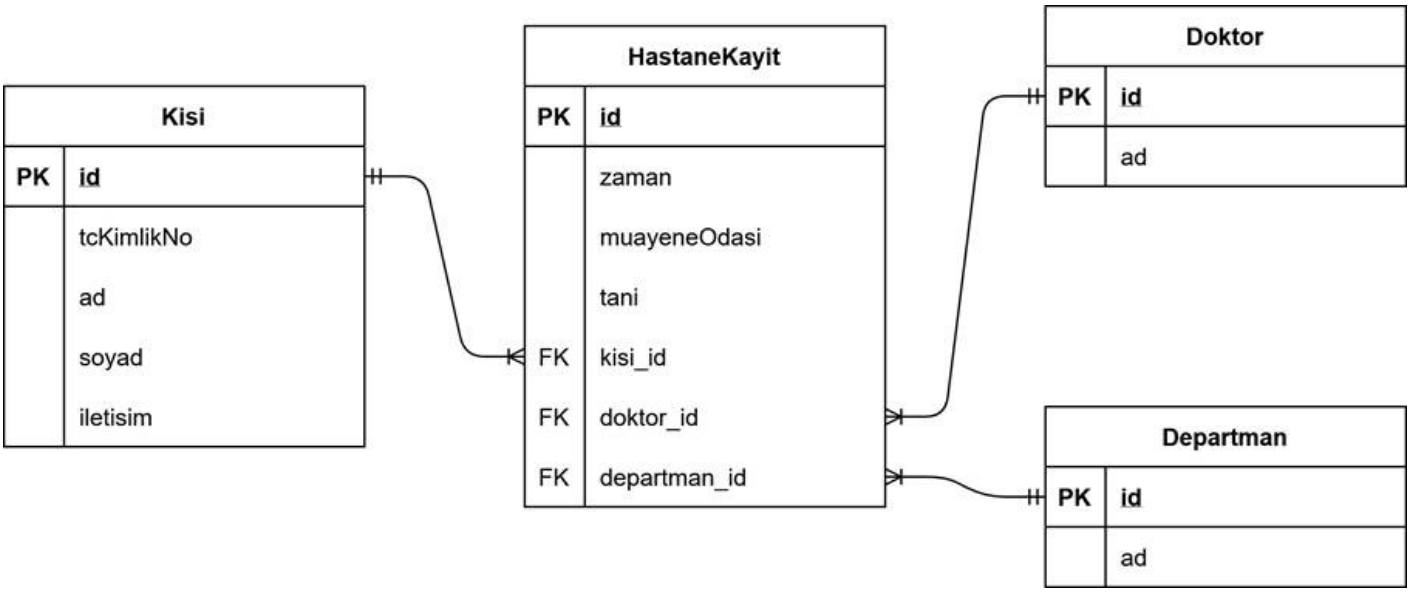
Şekilde bir yıldız şema örneği görmektesiniz. Ele alınan örnek her ne olursa olsun bu şema türü için tablolara verilen genel adlar bulunmaktadır. Elbette kendi veri tabanı tasarımınızda siz istediğiniz adı kullanabilirsiniz ancak bu adları da tabloların türü olarak aklınızda tutmanızı öneriyorum.

Yıldız şemada ortada yer alan tabloya kayıt (transaction) tablosu adı verilmektedir. Bu tablo az önce vurguladığımız tüm yoğun kayıtların tutulduğu, hacmi yüksek olan, veri tabanının kuruluş amacı olan kayıtları içeren tablodur. Bu tablo ve içerdiği kayıtlar elbette birçok olgu ile ilişkilidir. Örneğin bir otoban

geçiş kaydında geçiş işlemleri bir kayıt tablosuna yazılacaktır elbette ancak bu kayıt araç, kişi, gişe, il, tarife ve benzeri birçok olgu ile ilişkili olarak kayıt altına alınmalıdır. Kayıtların ilişkili olduğu her bir olgu bir veri tabanı tablosu olarak gösterilir.

Şimdi öğrendiklerimizi birleştirelim. Kayıt tabloları eylemlere ait kayıtları içeren tablolardır. Eylemler genellikle bağlantı tabloları olarak karşımıza çıkar. Bağlantı tabloları olgular arasındaki çoğa çok bağlantı sayesinde ortaya çıkmaktadır. Bu durumda şemadaki veri tabanı tasarımında kayıt tablosu etrafında yer alan tüm tablolar çoğa çok bağlantılıdır ve bağlantı tablosu kayıt tablosunun kendisidir. Evet, bu doğru. Kayıt tablosunu bir nevi bağlantı tablosu olarak görebiliriz. Her ne kadar çoğa çok bağlantıdan doğduğunu düşünsek de bu durum bağlantı tablolarını küçümsemek için yeterli değildir. Bağlantı tabloları genellikle veri tabanı içerisinde en çok kaydı içeren; veri bütünlüğünü sağlamak için en çok ihtiyaç duyulan tablo türüdür. Yıldız şema da bunu oldukça net olarak göstermektedir.

Yıldız şema tek başına kullanılabileceği gibi bir veri tabanı yapısının bir bileşeni de olabilir. Eldeki veri tabanı tasarımı içerisinde birçok tablo olabilir. Ayrıca kayıtsal eylemler için bağlantı tabloları da bulunabilir. Bu durumda boyut adı verilen, olguları bulunduran tablolar da olacaktır ancak veri tabanında kayıt tablosu dışında yer alan tüm tabloların bu özellikte olmasına gerek yoktur. Bir e-ticaret sistemini ele alalım. Satis (satış) tablosu kayıt tablosuna güzel bir örnektir. Gerçekleştirilen her bir satış işlemi bu tabloya kaydedilecektir. Bir eylemin kaydı için de kullanıldığı aşikardır. Olgulara göz gezdirdiğimizde, bir satış için kişi ve ürün olgularına ihtiyaç duyduğumuzu söyleyebiliriz. Genellikle bu tarz sistemler çok daha karmaşık olurlar. Dolayısıyla çok daha fazla boyut tablosuyla ilişki sağlanacaktır. Ama tüm tablolar değil! E-ticaret sistemi veri tabanında satışlarla doğrudan ilişkili olmayan çeşitli kayıtlar bulunması da mümkündür. Bu durumda bu tarz bir veri tabanı tasarımında yıldız şema yalnızca bu tasarımın bir bileşeni olabilir.



Şemadaki örnekte bir hastanedeki kayıtları görüyoruz. Ortada yer alan HastaneKayit tablosunu Muayene tablosu olarak da düşünebiliriz. Bu sefer boyut tablolarını inceleyerek başlayalım. Kisi tablosu hastaneye gelen herkesin bir kez kaydedildiği tabloyu ifade etmektedir. Hastaneyi ziyaret eden toplam kişi sayısı kadar kayda sahip olması beklenecektir. Bir olguyu ifade eder. Herhangi bir eylem barındırmamaktadır. Doktor tablosun da kişilerin kayıt altına alındığı bir tablodur. Ancak bu tabloya kaydedilen kişiler hastanede çalışan doktorlardır. Hastanede çalışmış olan toplam doktor sayısı kadar kayda sahip olması beklenir. Departman tablosu da hastanede yer alan bölümlerin bilgilerini barındıran tablodur. Gördüğünüz gibi tüm boyut tabloları olguları kayıt altına alan tablolardır. Bu tablolara yeni kayıt eklenmesi ya da değişiklik yapılması oldukça sınırlı olacaktır. HastaneKayit tablosu ise hastaneye yapılan tüm girişlerin kayıt altına alındığı tablodur. En çok kaydın ve en çok işlem yapılan tablonun bu olması beklenecektir.

Burada şu ayrıntı da dikkat çekebilir. Bu veri tabanı tasarımının işlevi bir MS Excel dosyası ile de sağlanabilir. Nihayetinde bir kişinin, bir departmanda görev alan doktorla, belirli bir zaman ve mekanda muayene eylemini gerçekleştirmesi kayıt altına alınmaktadır. Bu bilgi geleneksel elektronik tablo ile de kayıt altına alınabilir. Ancak şimdi kayıt tablosunun açık halini göz önüne getirin. Her bir satırda kişinin bilgileri, doktorun bilgileri, departman adı ve gerekli diğer bilgilerin tekrar etmesi gerekecektir. Eğer bu veri geleneksel elektronik tablolar ile yönetilirse aynı verinin çok fazla kez tekrar edilmesi kaçınılmazdır. Ancak

bu yapıyla birlikte her bir olgu farklı bir tablo ile kayıt altına alınmakta; kayıt tablosunda yalnızca bu olguların id'leri kullanılmaktadır. Böylece tekrar eden şey yalnızca bir tam sayı olacak, en performanslı şekilde tüm veri, bütünlüğü bozulmamış biçimde saklanabilecektir.

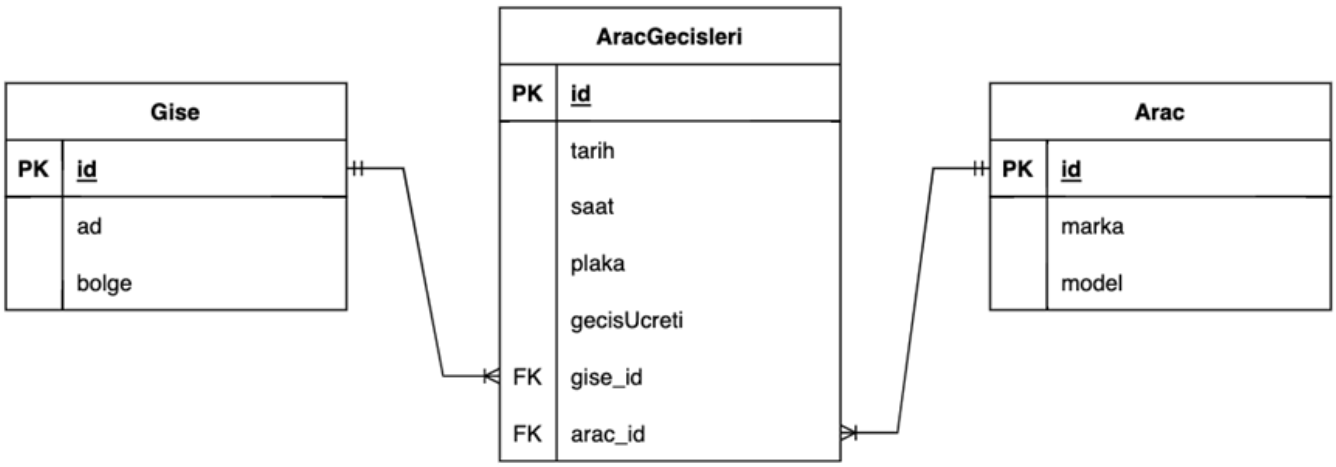
Bu ayrıntıyı şu şekilde de ele alabiliriz: Bir potansiyel kayıt tablosu ele alalım. Bu tablo içerisinde sıkça tekrar etmesi olası nitelikleri belirlemek zor olmayacaktır. Bu nitelikleri birer kategorik değişken olarak düşünebiliriz. Yani bir değer kümesi içerisinde yer alan değerleri yüksek frekansla tekrarlı şekilde alan nitelik kümeleri belirleyebiliriz. Bu durumda ortada yalnızca bir kayıt tablosu varken boyut tablolarını da belirlemek ve veriyi göz önünde bulundurarak veri tabanı tasarımını elde etmek zor olmayacaktır.

Aşağıdaki tabloyu ele alalım:

AracGecisleri: tarih, saat, gişe, bolge, plaka, marka, model, gecisUcreti

Burada AracGecisleri adında bir tablo ve bu tablo içerisinde yer alan nitelikleri görüyoruz. Sürekli yeni kayıtlar alması muhtemelen bir tabloyla karşı karşıyayız. Bunu bir yıldız şema olarak değerlendirme kararı vererek kayıtları içerecek tabloyu merkeze alarak üzerinde çalışmaya başlayabiliriz.

Tüm geçişleri kayıt altına alacak tablomuzun adı yine AracGecisleri olsun. Boyut tablolarını belirlemek için bu tablo içerisinde tekrar eden nitelikleri belirlemek yeterli olacaktır. tarih ve saat niteliklerini kayıt tablosunda bırakmak en mantıklısı olacaktır. Bir gişeden çok sayıda geçiş yapılabileceği için bu tablo içerisinde aynı gişe bilgisini çok kez görebiliriz. Bu sebeple gise niteliğini bir boyut tablosu olarak değerlendirebiliriz. Aynı zamanda bolge niteliği de sıkça tekrar edecektir. Bununla birlikte bolge ve gise niteliklerinin bağımlılık barındırdığını da söyleyebiliriz. Bir gişe bir bölgeye bağlı olmak zorundadır. Dolayısıyla bu iki niteliği tek bir boyut tablosu olarak birlikte değerlendirebiliriz. plaka niteliği zaten birincil anahtar olarak kullanılabilecek; aracı ifade eden bir nitelik olduğu için kayıt tablosunda kalabilir. Araçla ilgili birden fazla nitelik olsaydı bir boyut tablosu tanımlamak daha akıllıca olacaktı. Aynı olguyu ifade eden birden fazla nitelik olması durumunda bu nitelikleri gruplayarak bir boyut tablosu oluşturmak ve bu boyut tablosunu kayıt tablosunda temsil edebilecek bir nitelik belirlemek en iyi çözümdür. marka ve model nitelikleri aracın özelliklerini belirten iki niteliktir. Bu iki niteliği gruplayarak bir boyut daha elde edebiliriz. Bir arac_id niteliği ile kayıt tablosunda temsil edilmesi mümkündür. Burada marka ve model niteliklerinin neden plaka niteliğiyle gruplanmadığını düşünmüş olabilirsiniz. Bir aracın plakası zaman içerisinde özellikle satış işlemlerinde değişebilmektedir. Bu sebeple bir araç ile plaka her zaman birlikte anılmayabilir. Bu örnekte bu risk göz önünde bulundurulmuş ve ilgili nitelikler birlikte gruplandırılmamıştır. gecisUcreti niteliği sayısal bir niteliktir ve her geçiş için farklı değer alabilir. Ayrıca zaman içerisinde de güncellenen bir değer olacağı için bir boyut olarak ele alınması uygun olmayacaktır. Tüm bu yorumlar ışığında aşağıdaki çözümü doğru kabul edebiliriz.



Şekilde AracGecisleri adında bir kayıt tablosu ile Gise ve Arac adlarında iki boyut tablosu görülmektedir. Böylece küçük çaplı da olsa bir yıldız şema veri tabanı tasarımı elde etmiş olmaktayız.

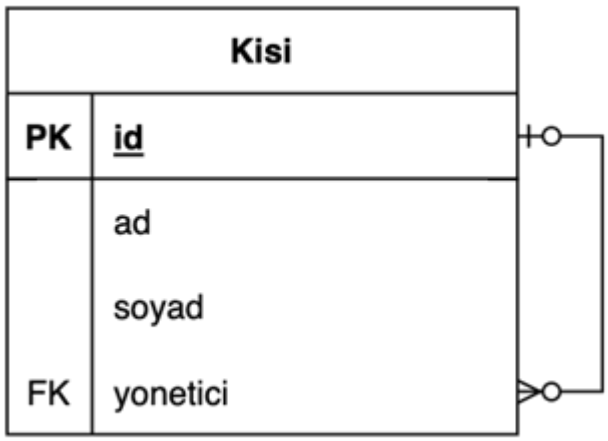
7.2. Hiyerarşik Veri

Veri tabanı yapısı içerisinde her bir tablonun genellikle bir olguyu, bağlantı tablolarının ise eylemleri kayıt altına aldığından bahsetmiştik. Olgular birbirleriyle çeşitli şekillerde bağlantı kurabiliyorlar. Bu bağlantı türü çoğa çok olduğunda bir bağlantı tablosunu da tasarımımızın bir parçası yapmaktayız.

Veri tabanı içerisinde kurulan bağlantılar genellikle iki olgu yani dolayısıyla iki tablo arasındaki kurulmaktadır. Ancak aynı olgu da kendisiyle bir ilişkiye sahip olabilir. Olguyu programlamadaki sınıf (class) kavramına, her bir kaydı da bir nesneye (object, instance) benzeterek bu durumu daha iyi açıklayabiliriz. Kisi (kişi) tablosunu ele alalım. Bu tablo kişilerin kayıt altına alındığı, genel olarak bir kişi kaydının hangi niteliklere sahip olması gerektiği bilgisini barındıran olgudur. Bu tablo içerisinde yer alan her kayıt, kişi olgusunun birer örneğine işaret eder. Kisi, genel olarak bir kişiyi ifade ederken bu tabloda yer alan kayıtlar gerçek kişilerdir. Tablolar arasındaki ilişkiler tanımlanırken olgular arasında kavramsal olarak sonuç elde edilir. Ancak kayıtlar da işin içine girdiğinde gerçek olaylar kullanılmış olacaktır.

Bir kişi, herhangi başka bir olguyla etkileşimde olabilir. Bir ürünü satın alabilir, öğrencilik kaydı oluşturabilir, otel rezervasyonu yaptırabilir, bir şirkette çalışabilir. Peki bir kişi başka bir kişiyle bir etkileşimde olabilir mi? Fiziksel dünyada bunu düşünmek oldukça kolay. Elbette bir insan başka bir insanla herhangi bir şekilde etkileşimde olabilir. Bir kişi başka bir kişinin yöneticisi olabilir, ebeveyni olabilir, vârisi olabilir. Şimdi veri tabanı açısından düşünelim. Bir Kisi tablosu içerisinde yer alan iki kişi kaydını ele alalım. Bu kayıtlardan biri diğerinin yöneticisi olsun. Bu bilgiyi bu tabloda nasıl saklayabiliriz?

Veri tabanımızı en atomik şekilde kurguladık. Elimizde yalnızca bir Kisi tablosu var ve içerisinde sadece 2 tane kayıt var. Bu iki kayıt birbiriyle ilişkili. Farklı tablolar arasındaki ilişkileri belirlemeye oldukça alışığız ancak aynı tablo içerisindeki ilişkileri tanımlamakla ilgili bir örnek görmedik. Yanıtı bulmak şimdiye kadar öğrendiklerinizle mümkün, ancak biraz zor olabilir. Şemayı inceleyelim.

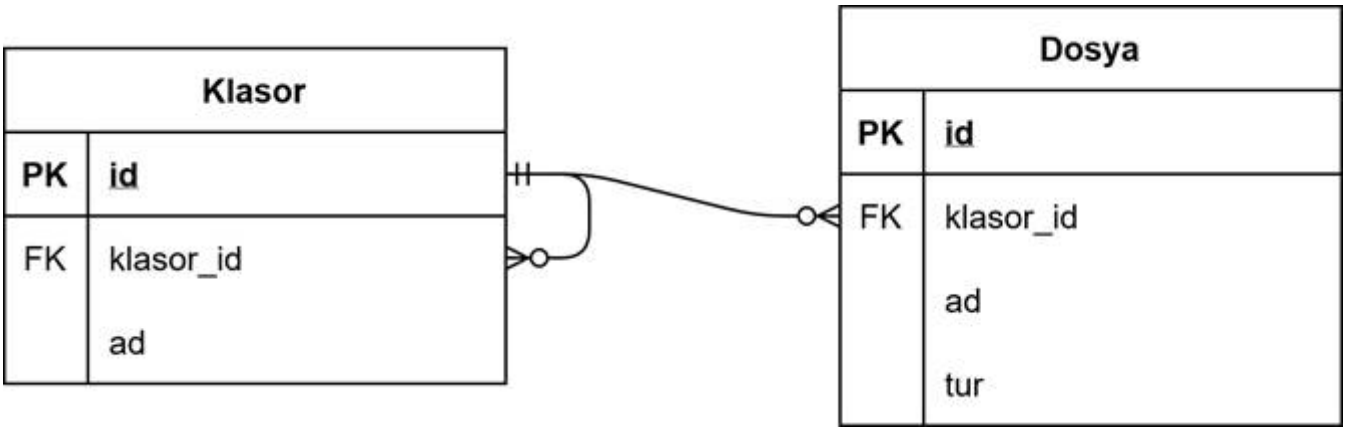


Lütfen okumaya devam etmeden önce tabloyu ayrıntılı olarak inceleyin. Klasik bir veri tabanı ilişkisine çok benzer bir yapıyı tek bir tabloda görüyoruz. Tablo, ilişki, birincil ve ikincil anahtarlar, nitelikler üzerinden kurulan bağlantı mevcut ancak tek bir tablo var. Standart yaklaşımımızı uygulayabiliriz. Kisi tablosunda yer alan bir kayıt, Kisi tablosunda birden fazla kayıt ile eşleşebilir mi? Yönetici olma durumunu göz önünde bulundurduğumuzu unutmayın lütfen. Yanıtımız evet. Bir kişi birden fazla kişinin yöneticisi olabilir, dolayısıyla birden fazla kayıt ile eşleşebilir. Şimdi ilişkinin diğer yönünü inceleyelim. Kisi tablosunda yer alan bir kayıt, Kisi tablosunda birden fazla kayıt ile eşleşebilir mi? Bu soru öncekiyle aynı gibi gözükse de ilişkinin tersini irdelediğimizi unutmayın. Bu sefer kişinin yöneticisini araştırıyoruz. Yanıtımız hayır. Bir kişi yalnızca tek bir yöneticiye sahip olabilir. Yani özet olarak; bir kişi birden fazla kişiye yöneticilik yapabilir ancak bir kişinin en fazla bir yöneticisi olabilir. Bu durumda çoğa bir (ya da bire çok) ilişki ile karşı karşıyayız. Bu örneği daha net anlamak için tablomuzu açık açık inceleyelim.

id	ad	soyad	yonetici
1	Emre	Akadal	
2	Lorem	Ipsum	1
3	Dolor Sit Amet		1

Tabloda üç kayıt yer alıyor. Birinci satır için yönetici niteliği değer almamış. İkinci ve üçüncü satırlar yonetici olarak 1 değerini almışlar. Bu durum; ikinci ve üçüncü kayıtların yöneticisinin 1 id'li kaydolduğunu göstermektedir. Bu durumda id'si 1 olan kayıt, 2 tane kaydın yöneticiliğini yapmaktadır. Ancak id'si 2 ve 3 olan kayıtlar ancak tek bir yöneticiye sahip olabilmektedirler. Tabloda yönetici olan kayıt, birden fazla kayıtla eşleşebilirken, diğer kayıtlar en fazla tek bir kayıtla eşleşebilmektedirler.

Farklı bir örnek ele alalım. Bir bulut dosya saklama sistemi için veri tabanı tasarımı gerçekleştiriyor olalım. Yine en temel bileşenlerini göz önünde bulunduracağız. Klasörler, dosyalar ve bunların veri tabanına yerleşimini anlamaya çalışalım. Klasör ve dosyalar farklı birer olgu kabul edilebilir ve iki ayrı tablo olarak tutulabilir. Adları Klasor ve Dosya olsun. Bir Klasör birden fazla dosya içerebilir ancak bir dosya ancak tek bir klasör içerisinde yer alabilir. İki tablo arasında bire çok ilişkiden söz edebiliriz. Ancak burada bir ilişki daha bulunmalı. Klasörler de kendileri arasında bir hiyerarşiye sahipler. Bir klasör başka bir klasörün içinde yer alabilir ya da bir klasör birçok klasör ya da dosyayı içeriyor olabilir.

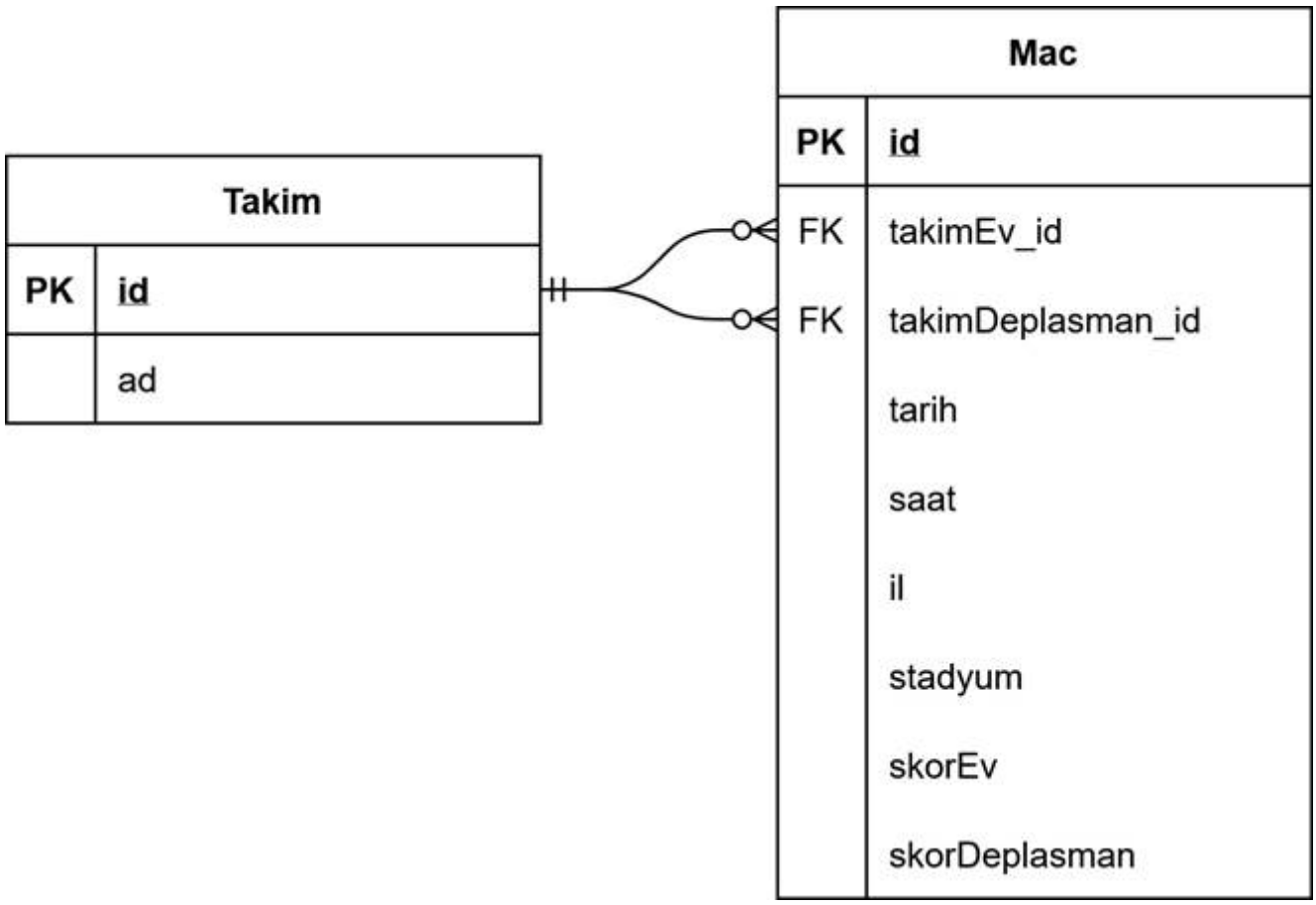


Buradaki ilişki türünü incelediğinizde, bir klasör birçok klasör içerebilir ancak bir klasör ancak tek bir klasörün içinde yer alabilir yorumu sayesinde bire çok ilişki ile devam etmemiz gerektiğini fark edeceksiniz. Böylece klasörler, bir klasor_id sayesinde hangi klasör altında yer aldıkları bilgisini saklamaktadırlar. Ele alınan bir klasör için, klasörün id'si kullanılarak klasor_id'si bu klasörün id'sine eşit kayıtlar aranılabilir, böylece seçilen klasör içerisindeki diğer klasörler listelenebilir. Aynı sorgulama Dosya tablosunda da yapıldığında klasör altındaki dosyalar listelenebilir.

Bu yaklaşım bir hiyerarşi içeren her türlü yapılanma için kullanılabilir. Şirketlerdeki insan kaynağı yapılanmaları, yönetim teşkilatları, kamu kurumları ve bölümleri, askeriye, üniversitelerde yer alan birimler bu tarz bir veri tabanı tasarımı kullanılarak ifade edilebilirler.

Şimdiye kadar ele aldığımız örnekler, bir tablonun kendisiyle bire çok ilişkili olduğu örneklerdi. Bildiğiniz üzere bire çok ilişkide iki tablo doğrudan bağlanır ve ilişki tipi çok olan yönde ikincil anahtar yerleştirilir. Ancak çoğa çok bağlantılı tablolarda bir bağlantı tablosu sağlanmalıdır. Bir tablo kendisiyle çoğa çok bağlantılı olduğunda tablo yapısı nasıl kurulmalı? Bir tablo hangi durumda kendisiyle çoğa çok bağlantılı olur? Yine örnek üzerinden gidelim.

Futbol takımları bir tablo içerisinde listelenebilirler. İki futbol takımının birbiriyle en bariz etkileşime geçtiği eylem nedir? Maç yapmak olsa gerek. İki futbol takımı maç yapma eylemi ile kayıt oluşturabilir. Ayrıca bu eylemi çok kez gerçekleştirebilir. Bir takım çok sayıda takımla maç yapabilirken, bir takım yine çok sayıda takımla maç yapabilir. Yanlış gibi gözükse de bu cümle ilişki tipini iki açıdan da incelememizi sağlıyor. Burada çoğa çok bağlantıdan bahsettiğimizi söyleyebiliriz.



İncelediğiniz örnekte Takim ve Mac adında iki tablo bulunmaktadır. İki tablo arasında iki bağlantı var gibi görülebilir ancak gerçekte Takim tablosu kendisiyle çoğa çok bağlantılıdır. Mac tablosu ise bu çoğa çok bağlantıdan kaynaklanan bağlantı tablosudur. İki takımın gerçekleştirdiği maç eylemi bu tablo içerisinde kayda alınır. Ve evet, bu bir yıldız şema örneğidir.

7.3. Özet Fonksiyonları

Bilgi sistemlerinin en önemli tarafı, herhangi bir güvenlik zafiyeti oluşturmadan bir bilgiyi saklayabilmektir. Günümüz şartlarında da bilgi zafiyeti oluşturulması en muhtemelen iki veri türü; kişisel veriler ve erişim bilgileridir. Kişisel verilerin korunmasıyla ilgili çeşitli yasal düzenlemeler bulunmaktadır. Bununla birlikte kişisel verinin türü değişken olduğu için teknik boyutta alınabilecek kesin kararlar bulunmamaktadır. Ancak erişim bilgileri konusunda atılabilecek, özellikle veri tabanı boyutunu ilgilendiren konular mevcuttur.

Erişim bilgilerinden kastımız bir sistem ya da özelliğe erişirken kullandığımız parola, anahtar kelime ve benzeri özel ifadelerdir. Her ne kadar veri tabanına erişimi kısıtlıyor olsak da internet ortamına açık sistemler, yazılımda, işletim sisteminde, kullanılan kütüphanelerde ve diğer bazı bileşenlerde meydana gelebilecek açıklar sebebiyle izinsiz kullanıcıların erişimine açılabiliyor. Bir veri tabanının tamamını görüntüleme yetkisi, eğer kullanıcılara ait erişim bilgilerini de içeriyorsa tüm kullanıcıların zarar görmesine sebep olabilmektedir. Bununla birlikte bazı kullanıcıların aynı şifreyi farklı sistemler için kullanması, şifrenin herhangi bir sistemde ele geçirilmesi sonrasında başka sistemlerde de zafiyet oluşmasına sebep olabilmektedir. Bu riski aşmak için sıklıkla kullanılan yöntem şifreyi veri tabanına doğrudan saklamamaktır. Şifre yerine şifrenin doğrulanmasını sağlayacak bir içeriği saklamak, şifrenin ele geçirilmesine engel olmaktadır. Bu da bizi özet fonksiyonları ile tanıştırıyor.

Özet (hash), bir girdi ifadesi sonucunda sabit uzunlukta bir çıktı üreten fonksiyona verilen addır (Ajao & diğ., 2019; Todorova & diğ., 2019). Birkaç örnek inceleyerek devam edelim.

Değer	Özet
1	c4ca4238a0b923820dcc509a6f75849b
Emre	34a1988c0a5878ddcea0cdb42d651e1e

Tabloda yer alan ilk sütun çeşitli değerler içerirken, ikinci sütun ise bu değerlerin MD5 fonksiyonuyla üretilmiş özet değerlerini içermektedir. İlk değer yalnızca 1 karakterlik bir ifadeyken özeti 32 karakterlik bir değerdir. İkinci satır 4 karakterlik, üçüncü satır çok daha uzun bir metni içermektedir. İkinci ve üçüncü satırdaki değerlerin özetleri de 32 karakterlik özet değerlere sahiptir. Özet fonksiyonları, aldıkları her bir girdi değer için bir çıktı üretirler. Girdi değer birebir aynı olduğunda ürettikleri değer de aynı olacaktır. Girdi değerlerde en ufak bir değişiklik bile özetin tamamen değişmesine sebep olmaktadır.

Veri tabanında saklanılması gereken, içeriği önemsiz ancak doğrulanma ihtiyacı duyulan içerikler özetlenerek saklanabilirler. Bir kullanıcının şifresinin içeriğinin ne olduğu önemsizdir. Ancak kullanıcı sisteme tekrar eriştiğinde şifresini doğru girip girmediği kontrol edilebilmelidir. Bu sebeple kaydolma sürecinde şifre özetlenerek veri tabanına kaydedilir. Kullanıcı sisteme tekrar eriştiğinde giriş yapmak için şifresini girer, şifre yazılım tarafından tekrar özetlenir ve elde edilen özetle veri tabanına kaydedilen özetin aynı olup olmadığı karşılaştırılır. Özetler aynı ise kullanıcı sisteme kabul edilir.

Özet fonksiyonları tek yönlü çalışan fonksiyonlardır. Bir girdi değer alarak bu girdi değere karşılık özet alabilirken; bir özet alarak bu özeti oluşturan değeri elde etmek teorik olarak mümkün değildir. Pratikte ise şöyle bir yaklaşım izlenebilir: Mümkün olan tüm değerler için özetleme yapılarak değer-özet ikilisinden oluşan bir kütüphane elde edilebilir. Böylece eldeki özet bu kütüphanede araştırılarak eğer kaydedildiyse hangi değer kullanılarak oluşturuldu öğrenilebilir. Buna “Brute Force” saldırısı adı verilmektedir. Aşmak için kombine de edilebilen iki yöntem kullanılır. Bunlardan birincisi şifrenin belirli uzunlukta, sayı, harf ve özel karakter içeren metinlerden oluşturulmasıdır. Bu kombinasyona sahip tüm olasılıkların bir kütüphanesinin hazırlanması neredeyse imkansız olduğu için erişim zorlaştırılacaktır. Bununla birlikte sistem için özel bir anahtar belirlenir ve tüm özetlemelerde bu anahtar da özetlenecek değerle birleştirilir. Böylece özetlenecek değerin çok basit olması durumunda dahi anahtar olarak kullanılan kısmın bilinmiyor olması sebebiyle özetin tersine döndürülmesi mümkün olmayacaktır.

Özellikle internet tabanlı sistemlerin saldırılara maruz kalma imkanının yüksek olması sebebiyle bu tür güvenlik önlemlerinin alınması oldukça önemli ve gereklidir. Özetleme sayesinde alınacak önlemler, bazı sızmalar olması durumunda dahi şifrelerin ele geçirilmesini engelleyecektir. Ancak bu yöntem kişisel verilerin korunması konusunda fayda sağlayamayacaktır.

7.4. Bağlantısız Varlıklar

Dersin bu bölümüne kadar ilişki niteliklerin tabloları oluşturmasından, ilişkili tabloların da veri tabanlarını oluşturmasından bahsettik. Peki veri tabanında yer alan tablolar hiçbir tablo ile ilişkili olmayabilir mi?

Zaman zaman veri tabanında yer alan ancak diğer tablolarla doğrudan bağlantılı olmayan tablolar ya da tablo grupları bulunmaktadır. Eğer tablolar, doğrudan bir ilişki tipi ile bağlı olmayabilirler. Bir tablo, doğrudan hiçbir tabloyla bağlantılı olmayabilir. Ya da iki bağlantılı tablo grubu, birbirleriyle bağlantılı olmayabilirler.

SistemAyarlari	
PK	id
	anahtar
	deger

Görselde, SistemAyarlari adında, sistem dahilinde sürekli kontrol edilmesi gereken ve bileşenlerin genel hareketlerini yönlendiren ayarların bulunduğu bir tablo gösterilmektedir. “id” adında bir birincil anahtar kullanılmıştır. Bununla birlikte tüm ayarlar “anahtar” ve “deger" nitelikleri kullanılarak tabloda sağlanmıştır.

Bu tablo, doğası gereği diğer tablolarla ilişkili değildir. Ancak yönetilen sistem için sistem ayarlarının saklanılması, ihtiyaç halinde güncellenmesi ve gerektiği anda hızlıca çağrılması ihtiyacı sebebiyle veri tabanı içerisinde bulundurulmaktadır. Dolayısıyla bu tablonun diğer tabloların hiçbiriyle bağlantılı olmaması bir hata değildir.

Veri tabanı tasarımı içerisinde bir tablo diğer tablolarla bağlantısız olabileceği gibi bir tablo grubu da bağlantısız olabilir. Birbirleriyle bağlantılı olan bir tablo grubu, yine birbirleriyle bağlantılı olan başka bir tablo grubuyla hiçbir tablo üzerinden bağlantı kurmuyor olabilir. Bu durum genellikle birden fazla bileşene sahip sistemlerde görülür. Bir firmaya ait bilgi sistemi içerisinde ürün, müşteri, nakliye, fiyatlandırma, bayi yönetimi, insan kaynakları ve benzeri çok sayıda bileşen yer alabilir. Bu bileşenler genellikle kendi içerisinde birkaç tablodan oluşur ve diğer bileşenlerle ilişkili olabilir ya da olmayabilir.

Eğer bir analiz gerçekleştirmek için ikincil veriye başvurduysak yine bağlantısız tablolardan oluşan bir veri tabanı elde edebiliriz. Gerçekleştirilecek bir analiz için toplu taşıma kullanım verilerini elde ettiğimizi düşünelim. Sonrasında bu veri ile hava durumu arasındaki ilişkiyi anlamak için hava durumu kayıtlarını da elde edelim. Tüm bu kayıtları tek bir veri tabanına yerleştirdiğimizde, toplu taşıma kullanımıyla ilgili verilerle hava durumu verileri arasında tablolar arası bir ilişki olmadığını fark edeceksiniz. Buradaki ilişki, analiz sürecinde araştırmacının tarih ve varsa bölge bilgilerini eşleştirmesiyle elde edilecektir. Bölge bilgileri üzerinden tablolar arası ilişki kurulması mümkün olsa da iki veri kaynağında da bölge belirtme şekilleri birebir aynı olmalıdır. Aksi halde bir ön işleme süreci de işletilmelidir.

7.5. Zaman Damgası

Zaman kontrolü veri tabanında sıklıkla yapılan işlemlerden birisidir. Bir kaydın ne zaman eklendiği, en son ne zaman güncellendiği, geçerlilik süresinin ne olduğu gibi birçok zaman bilgisi ihtiyacı bulunmaktadır. Bu ihtiyaçlar elbette ki bir nitelik içerisinde tarih ve saat bulundurmakla çözülebilir gibi gözüküyor. Ancak bu tür nitelikler metin içerikli olacağı için, zamanlar üzerinde bir işlem yapılmaya çalışıldığında birçok zorluk da beraberinde gelecektir. En basit uygulama olarak bir tarihin geçip geçmediğini anlamak, tarih ve saatlerin karşılaştırılabilir olmasını gerektirir. Ancak bunu doğrudan yapmak kolay olmayacaktır. Ya da bir zaman bilgisi üzerine 1 gün eklemek gibi bir işlem yapmak oldukça zor olacaktır.

Bu gibi eylemler sıklıkla ihtiyaç duyulan eylemler olduğu için elbette bir çözüm yöntemi de üretilmiş durumdadır. Zaman damgalarının veri tabanına kaydedilmesiyle ilgili iki tür zaman formatı bulunmaktadır. Bunlardan birincisi ISO 8601 ile tanımlanmış format (27), diğeri ise Unix Timestamp olarak bilinen zaman damgasıdır (11).

ISO 8601; tarih, saat ve zaman dilimi bilgilerini özel bir biçimde sunan bir yöntemdir. Yöntem, alfanümerik içeriğe sahip olsa da farklı sistemler tarafından aynı şekilde yorumlanması ve üzerinde kıyaslama ya da hesaplama yapabilecek yazılımlar olması sebebiyle tercih edilebilmektedir. Bir örneği şu şekildedir:

2022-09-16T11:45:30+03:00

Örnek şu bilgiyi içermektedir: 2022 yılı, 9. Ay, 16. gün, saat 11:45'in 30. saniyesi. Zaman dilimi ise +3 olarak belirlenmiş.

İkinci yöntem ise GMT zaman diliminde 1 Ocak 1970 saat 00:00'dan beri geçen toplam saniye sayısını veren Unix Timestamp'tir. Bir örneğini inceleyelim.

1663318419

Bu zaman damgası türünün dezavantajı, bakıldığında doğrudan anlamlandırılamamasıdır. Bu sayının hangi tarihe denk geldiğini anlamak için bile bir dizi matematiksel işlem yapmak gerekmektedir. Ancak bununla birlikte çeşitli avantajları bulunmaktadır. Birincisi sayısal veri olduğu için saklaması ve üzerinde işlem yapması kolaydır. Bu şekilde kaydedilmiş bir zaman damgasının 1 gün sonrasını hesaplamak için 86400 (1 günlük saniye sayısı) eklemek yeterlidir. Ya da zamanın geçip geçmediğini anlamak için doğrudan matematiksek büyüktür ve küçüktür işaretleri kullanılabilir.

İki zaman damgasının da kullanılıyor olmasının yanında Unix Timestamp türünün daha yaygın ve kullanışlı olduğunu söylemek gerekir. Özellikle oyunlarda bir eylemin gerçekleştirildiği ya da gerçekleştirileceği zaman üzerine yapılacak işlemler için bu türdeki zaman damgaları oldukça kullanışlıdır.

Zaman damgası hesaplaması manuel olarak gerçekleştirilmez. Neredeyse tüm programlama dillerinin zaman damgalarıyla ilgili fonksiyonları bulunmaktadır. Bu sebeple bir zaman damgasının üzerinde nasıl işlem yapılacağına dair kafa yormaya genellikle ihtiyaç duyulmamaktadır. Zaman damgası hesaplaması konusunda otomatik hesaplamalar yapan; tarihi zaman damgasına, zaman damgasını ilgili zamana çeviren web sayfaları da bulunmaktadır.

Bölüm Özeti

Veri tabanı tablolarını olgusal ve eylemsel olarak ikiye ayırabiliriz. Bir veri tabanı oluştururken ortaya çıkardığımız ilk tablo türü olgusal tablolardır. Kullanici, Arac, Kitap, Bolge ve Kategori bu türde tablolara örneklerdir. İki tablo birbiriyle çoğa çok bağlantılıysa bir bağlantı tablosu elde edilir. Bu tablo eylemsel olarak bahsedebileceğimiz tablolardır. Satis, Gecis ve TakipEtme eylemsel tablolara örnektir.

Yıldız şema, bir bağlantı tablosunun ortada, bu bağlantı tablosuna bağlantılı olgusal tabloların çevresinde olduğu yapıdır. Ortadaki tabloya kayıt, etrafındaki tablolara boyut tablosu adı verilmektedir. Ortada yer alan tablo genellikle çok sayıda kayıt alır. Boyut tabloları ise sabit içerikleri saklayarak kayıt tablosunun bunları id ile temsil etmesini mümkün kıldığı için veri hacminde büyük azalma sağlar. Aynı zamanda veri tekrarının da önüne geçilmiş olur. Bu veri tabanı türü başlı başına bir veri tabanı yapısı olabileceği gibi bir veri tabanı yapısının bir parçası da olabilir.

Tablolar her zaman birbirleriyle ilişkili olmak zorunda değildir. Bir tablo kendisi üzerinde bir ilişkiye de sahip olabilir. Bunun sebebi tabloların olgusal olması ve aynı türe ait iki olgunun bir eyleme sahip olabilmesidir. Örneğin Kullanici tablosunu ele alalım. Herhangi bir sosyal ağ üzerinde kullanıcıların birbirini takip etmesi sıklıkla görülen bir özelliktir. Bu özellik, Kullanici tablosunun kendisiyle ilişkili olması ile sağlanabilir. Burada ilişki tipi çoğa çok olacağı için bir bağlantı tablosu bulunması gerekecektir. Bu da TakipEtme tablosu olabilir. Böylece bir Kullanici, bir de TakipEtme tablosu elde edilecektir. TakipEtme tablosu üzerinde iki tane ikincil anahtar olması gerekmektedir. Hangi kullanıcının hangisini takip ettiği bilgisi bu şekilde sağlanacaktır. Zaten bağlantı tablolarında en az iki tane ikincil anahtar bulunduğunu daha önce öğrenmiştik. Bir tablo kendisiyle bire çok bağlantılı da olabilirdi. Bu durumda tablo üzerinde bir ikincil anahtar bulunması gerekir. Bu türde bağlantılar genelde hiyerarşiyi gösteren bağlantılardır. Örneğin bir dosya saklama sisteminde klasörler ve dosyaların hiyerarşisi bu türde bir veri tabanı yapısıyla saklanabilir. Her Klasor bir klasor_id sayesinde üst klasörünü işaret edecektir.

Özet fonksiyonları değişken uzunluktaki bir girdiye karşılık her zaman aynı uzunlukta ve aynı çıktıyı üreten tek yönlü fonksiyonlardır. Bir özet fonksiyonu için 1 karakterlik bir girdi de sağlanabilir; GB'lar boyutunda bir dosya da sağlanabilir. Her türlü girdi için eşit uzunlukta bir çıktı üretilecektir. Çıktının uzunluğu kullanılan özet fonksiyonuna göre değişir. Aynı girdi her zaman aynı çıktıyı üretirken girdideki en ufak değişiklik çıktıyı tamamen değiştirir. Özetleme tek yönlüdür. Bir girdi için özet üretilebilir ancak özet kullanılarak girdi elde edilemez. Bu fonksiyonlar genelde veri tabanı içerisinde şifre gibi içeriği önemsiz ancak doğrulanabilirliği önemli değerlerin saklanılmasında kullanılır. Bir şifre, özetlenerek veri tabanında saklanır. Tekrar kullanılması gerektiğinde sisteme girilen şifre tekrar özetlenir ve veri tabanındaki özet ile karşılaştırılır. Böylece şifrenin kendisi kaydedilmezken doğrulanabilmesi mümkün hale gelir.

Veri tabanı içerisinde bir tablo ya da tablo grubu, veri tabanının geri kalan kısmıyla bağlantılı olmayabilir. SistemAyarlari gibi yalnızca anahtar ve deger niteliklerinden oluşan ve ayarları saklayan bir tablo, veri tabanında yer alan hiçbir tabloyla bağlantılı olmayacaktır. Eğer eldeki sistem birçok bileşenden meydana geliyorsa bileşenler arasında bağlantı olmaması da mümkündür. İşletmeler genellikle çok sayıda bileşeni tek veri tabanında toplarlar. Tüm bileşenler firma için önemli olsa da veri tabanı açısından bağlantısız olabilirler. Bu bir hata değildir. Benzer şekilde bir analiz gerçekleştirilirken de farklı kaynaklardan alınan ikincil veriler doğrudan ilişkili olmayabilir ancak birlikte analiz edilmek üzere aynı veri tabanı içerisinde bir araya getirilebilirler.

Kaynakça

Ajao, L. A., Agajo, J., Adedokun, E. A., & Karngong, L. 2019. Crypto hash algorithm-based blockchain technology for managing decentralized ledger database in oil and gas in- dustry. J—Multidisciplinary Scientific Journal, 2(3):300–325.

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

Todorova, M., Stoyanov, B., Szczypiorski, K., Graniszewski, W., & Kor-dov, K. 2019. Bentsign: keyed hash algorithm based on bent boolean function and chaotic attractor. Bulletin of the Polish Academy of Sciences: Technical Sciences, pages 557–569.

Ünite Soruları

Soru-1 :

Hangisi yıldız şemada kayıt tablosu niteliğindedir?

(Çoktan Seçmeli)

(A) Kullanici

(B) Arac

(C) Gecis

(D) Plaka

(E) Gise

Cevap-1 :

Gecis

Soru-2 :

Hangisi yıldız şemada boyut tablosu niteliğindedir?

(Çoktan Seçmeli)

(A) Mac

(B) Odeme

(C) Satis

(D) Urun

(E) SatinAlim

Cevap-2 :

Urun

Soru-3 :

Hangisi tabloların kendi kendisiyle ilişkili olması konusunda doğrudur?

(Çoktan Seçmeli)

(A) Bire çok ilişkili ise ikincil anahtar içermelidir.

(B) Mümkün değildir.

(C) Yalnızca bire bir ilişki türü için geçerlidir.

(D) Çoğa çok ilişki türü için mümkün değildir.

(E) Aynı tablodan bir tane daha oluşturulması gerekir.

Cevap-3 :

Bire çok ilişkili ise ikincil anahtar içermelidir.

Soru-4 :

Hangisi tabloların kendi kendisiyle ilişkili olması konusunda doğrudur?

(Çoktan Seçmeli)

(A) Bir veri tabanında sadece bir tablo kendisiyle ilişkili olabilir.

(B) Kendisiyle ilişkili tablo başka tabloyla ilişkili olamaz.

(C) Kendisiyle ilişkili tablolar genellikle bağlantı tablolarıdır.

(D) Kendisiyle ilişkili tablolar birincil anahtar bulundurmazlar.

(E) Kendisiyle ilişkili tablo bağlantı tablosu gerektirebilir.

Cevap-4 :

Kendisiyle ilişkili tablo bağlantı tablosu gerektirebilir.

Soru-5 :

Hangisi veri özetlemeyle ilgili olarak doğru kabul edilebilir?

(Çoktan Seçmeli)

(A) Tek yönlüdür.

(B) Özetten girdi elde edilebilir.

(C) Girdiye göre özet uzunluğu değişir.

(D) Aynı girdi farklı özet üretebilir.

(E) Tüm özet fonksiyonları aynı uzunlukta çıktı üretir.

Cevap-5 :

Tek yönlüdür.

Soru-6 :

Hangisi veri özetlemeyle ilgili olarak doğru kabul edilebilir?

(Çoktan Seçmeli)

(A) Özetler anlamlı metinlerden oluşurlar.

(B) Aynı girdi için farklı zamanlarda bile aynı çıktıyı üretilir.

(C) Özetler sayısal veri türüyle saklanabilirler.

(D) Brute force yöntemiyle tüm özetler girdiye döndürülebilir.

(E) Özetlerin özeti üretilemez.

Cevap-6 :

Aynı girdi için farklı zamanlarda bile aynı çıktıyı üretilir.

Soru-7 :

Hangisi bağlantısız varlıklara örnek verilebilir?

(Çoktan Seçmeli)

(A) SistemAyarlari

(B) Kullanici

(C) Satis

(D) Sinif

(E) Ogrenci

Cevap-7 :

SistemAyarlari

Soru-8 :

Hangisi bağlantısız varlık grubuna örnek verilebilir?

(Çoktan Seçmeli)

(A) Kullanici

(B) HavaDurumu

(C) Yayinevi

(D) Firma

(E) Sayfa

Cevap-8 :

HavaDurumu

Soru-9 :

Hangisi zaman damgaları açısından doğrudur?

(Çoktan Seçmeli)

(A) Dönüşüm kodları için ayrıntılı programlar yazılmalıdır.

(B) Her zaman sayısal içeriklidirler.

(C) Veri tabanı içerisinde saklanamazlar.

(D) Programlama dilleri zaman damgalarıyla uyumludur.

(E) Zaman damgası yazılımı yavaşlatır.

Cevap-9 :

Programlama dilleri zaman damgalarıyla uyumludur.

Soru-10 :

Hangisi geçerli bir zaman damgasıdır?

(Çoktan Seçmeli)

(A) 16 Eylül 2022

(B) 16 Eylül 2022 11:45:30

(C) 2022-09-16-11-45-30

(D) 16-09-2022 11-45-30 03-00

(E) 2022-09-16T11:45:30+03:00

Cevap-10 :

2022-09-16T11:45:30+03:00

---

8. İHTİYACA ÖZEL VERİ TABANI TASARLAMAK

Bölümle İlgili Özlü Söz

Teknolojinin kendisi iyi ya da kötü değildir. İyi ya da kötü olan insandır.

Naveen Jain

İş insanı

Kazanımlar

1. Veri tabanı tasarımı yapabilir.

2. Veri tabanı tasarımlarını yorumlayabilir.

3. Veri tabanı tasarımını dokümante edebilir.

4. Sistem analizi ve tasarım süreçlerinde aktif rol alabilir.

5. Yazılım geliştirme ekiplerinde veri tabanı geliştiricisi olarak görev alabilir.

Birlikte Düşünelim

Yazılıma ihtiyaç duyan biri ve yazılımcı bir araya gelerek hazırlanacak sistemi birlikte tasarlayabilirler mi? Yoksa bu tasarlama süreci için farklı kişilere de ihtiyaç var mıdır?

Sistem analizi ve tasarımı süreçlerine hakim personel, bir yazılım firması için ne önemdedir?

Yazılım geliştirilerek çözülebilecek bir problem nasıl ele alınmalıdır?

Veri tabanı tasarımı sürecinde doğru soruları sormanın önemi nedir?

Başlamadan Önce

Dersin bu aşamasına kadar veri tabanı tasarımıyla ilgili çok şey öğrendiniz. Veri tabanlarına dair hiçbir bilgi ya da deneyimi olmayanların bile ilk bölümden itibaren veri tabanını tanıma, dokümante edebilme, kavramlara hakim olma, veri kategorizasyonu sağlama, tabloları ve ilişkileri oluşturma, özel durumlar için çözümler üretebilme gibi yeteneklere sahip olduğunu düşünüyorum. Şimdi her şeyi birleştirme zamanı. Bu dersin amacı, dersi alanların ihtiyaca yönelik veri tabanı tasarımını hazırlaması ve bunu dokümante edebilmesidir. Ayrıca tanımlanan veri tabanı tasarlama sürecinde gelecekte karşılaşılabilecek riskleri de öngörerek bu süreci gerçekleştirebilmek gerekmektedir.

Bu bölüm içerisinde doğrudan bir konu anlatılmayacak, bunun yerine gerçek dünya örnekleri üzerinde veri tabanı tasarımları gerçekleştirme alıştırmaları yapacağız. Böylece dersin bu aşamasına kadar eldeki tüm bilgileri birleştirecek ve bunları kullanarak asıl amacımıza ulaşacak şekilde kullanacağız.

Bölüm içerisinde 10 farklı alıştırma atölyesi bulunmaktadır. Her bir örnek için bir senaryo ile karşılaşacaksınız. Senaryolar içerisinde elde edilmek istenilen veri tabanıyla ilgili bazı ayrıntılar yer alacak olmakla birlikte bazı ayrıntılar kesin bilgiler içermeyecek. Böylece veri tabanı tasarımcısı olarak kesin olmayan durumlarda da gerçekleştirmeniz gereken yaklaşımı öğrenmiş olacaksınız.

Bu bölümde her bir problemi inceledikten sonra önce kendiniz çözüm üretmeniz, sonrasında çözüm yorumlarını incelemeniz oldukça önemlidir. Bu bölümle birlikte yazılım geliştirme projelerinde veri tabanı

tasarımıcısı olarak görev alma becerisine sahip olacaksınız.

8.1. Diyetisyenler İçin Platform

8.1.1. Problem

Bir diyet kliniği sahibi sizinle yeni kurgulanacak bir sistem için destek alma konusunda görüşmek istedi. Klinik sahibi, klinikte çalışan 8 diyetisyen olduğunu, destek almak isteyen kişilerin klinik sekreterlerini arayarak randevu aldığını ve görüşmelerin klinikte gerçekleştiğini belirtti. Yakın zamanda bu görüşmeleri internet ortamında da gerçekleştirmek istedikleri bir girişim içerisinde olduklarından bahsetti. Sizden bu girişim için gerçekleştirilecek bu sistemin veritabanının tasarlanması konusunda destek isteniyor. Yapılacak işi netleştirmek adına klinik sahibine çeşitli sorular yönelttiniz ve yanıtları not aldınız. Bu notlara istinaden bir veritabanı tasarımı gerçekleştiriniz. Aldığınız (aşağıda listelenen) önemli noktaların tamamına uyum sağlamalısınız. Sormayı ve not almayı unuttuğunuz durumlar için maalesef ki tekrar iletişime geçme şansınız yok. Dolayısıyla çeşitli varsayım ve öngörülerde bulunarak eksik gördüğünüz kısımları tamamlamalısınız.

Aldığınız notlar:

1) Sistem hem fiziksel hem de çevrimiçi ortamdaki rezervasyonların kayıtlarını tutmalı.

2) Birden fazla diyetisyen ile hizmet verebilmeli.

3) Sistemde yönetici, diyetisyen ve kullanıcı olarak üç yetki seviyesi yer almalı.

4) Gerçekleşmiş ya da gerçekleşecek tüm görüşmelerin başlangıç, bitiş saatleri ve diyetisyen ile kullanıcının katılım durumları kayıt altına alınmalıdır.

5) Kullanıcıların diyetisyenlerle görüşme yapabilmek için satın aldıkları paketlerin kayıt altına alınması gerekmektedir.

6) Her bir paket diyetisyene özeldir ve görüşme sayısı limiti içermelidir.

7) Bu durum için veritabanı tasarımını gerçekleştiriniz.

8.1.2. Çözüm Önerisi

Bir süre sonra tasarlama aşamasındaki birçok adımı doğallıkla gerçekleştirebilecek olsanız da bu işe ilk başladığınızda adımlarınızı temkinli atmanızda büyük fayda var. Problem başlığı altındaki anlatılar ve aldığınız farz edilen notlar, tasarım aşaması için büyük bir yol göstericidir. Hatırlarsanız önceki başlıklardan biri altında ad avına çıkmıştık. Nereden başlayacağınızı bilemediğiniz durumlarda sistemi özetleyen metin üzerindeki tüm adların bir listesini çıkararak gözden geçirin. Böyle bir metin yoksa siz yazın. Metnin dil ya da etik kurallara uyumlu olmasına gerek yok. Tüm özellikleri içerecek kadar detaylı ve mümkün olan en özet haliyle hazırlanması bu işlem için yeterli olacaktır.

Problem başlığı altında verilen metindeki adların bir listesini çıkaralım: Klinik, sistem, diyetisyen, kişi, sekreter, randevu, görüşme, fiziksel, çevrimiçi, rezervasyon, yönetici, kullanıcı, yetki, paket.

Şimdi bu adları irdeleyerek varlıklarımızı tanımlamaya başlayacağız. Sistemler genellikle kullanıcılar ve bunların rolleriyle ilgili varlıklar içerirler. Dolayısıyla öncelikle sistem kullanıcılarıyla ilgili kelimeleri seçip yorumlayarak başlayabiliriz. Kişi kelimesi kullanıcıyı kastetmekle birlikte diyetisyen, yönetici, sekreter ve kullanıcı kelimeleri kullanıcıların yetki seviyelerini belirtmektedir. Özel durumlarda açıklandığı gibi biz burada Kisi ya da Kullanici adında bir varlık tanımlayabilir ve tüm yetkileri bu varlık altında toplayabiliriz. İhtiyaç halinde daha sonra güncellemek üzere şimdilik bir varlık önerisinde bulunalım.

Kullanici: id, ad, soyad, yonetici, diyetisyen, sekreter

Kullanici varlığına, ihtiyaç duyulacağı öngörüsüyle id, ad ve soyad nitelikleri eklenmiştir. Bu varlığa ekli bir kayıt dolayısıyla bir kullanıcıdır. Kullanıcı, eğer yonetici niteliği 1 değeri alırsa yönetici, diyetisyen

niteliğinin 1 değeri alması durumunda ise diyetisyen yetkisine sahip olacaktır. Benzer şekilde sekreter niteliği de 1 değeri almasıyla birlikte kullanıcıya bu yetkiyi kazandırır.

Kullanıcı ve yetki işini hallettikten sonra listemizdeki adları sırayla ele alabiliriz. Klinik ve sistem kelimelerini es geçiyorum. Bu kelimeler sistemin bütününe atıfta bulunan kelimeler. Bir varlık adı olmalarına ihtiyaç duymuyoruz. Randevu, görüşme ve rezervasyon kelimelerini de birlikte inceleyebiliriz. Bu kelimeler, bir diyetisyen ile kullanıcının yapacağı görüşmeyi referans alan kelimelerdir. Gerçekleşmiş ya da gerçekleşecek tüm görüşmelerin kayıt altında olması isteğinden bahsedilmişti. Burada görüşme kelimesini seçerek bir varlık tanımlayabilir, özellikleri bu varlığa bağlı şekilde belirleyebiliriz. Önerim aşağıdaki gibidir.

Gorusme: id, diyetisyen, kullanici, sekreter, gorusmeZamani, diyetisyenKatilimi, kullaniciKatilimi, ortam

Gorusme varlığı içerisinde bir ayrıntı dikkatinizi çekmiş olmalı. diyetisyen, kullanici ve sekreter nitelikleri birer kullanıcıyı işaret eden nitelikler. Tahmin edeceğiniz üzere bunlar ikincil anahtar olarak görev yapacaklar. Burada bir tablonun (Kullanici) kendi kendisiyle çoğa çok bağlantılı olmasına bir örnek görmekteyiz. Lütfen bu ayrıntıyı tasarımın tamamı üzerinde inceleyiniz.

Bu varlıkta görüşmenin tarafları olan diyetisyen ve kullanicinın yanı sıra bu kaydı oluşturmuş olması muhtemel olan sekreter de bulunmaktadır. gorusmeZamani niteliği görüşmenin gerçekleşeceği zamanı içermektedir. Bu niteliğin bir zaman damgası olarak değer aldığını varsayabiliriz. diyetisyenKatilimi ve kullaniciKatilimi ise 0 ya da 1 değeri alan (boolean) ve bu kullanıcıların görüşmeye katılıp katılmadığı bilgisini içeren nitelikler olacaktır. Burada gorusmeZamaninin, içinde bulunulan andan önce ya da sonra olması görüşmenin geçmiş, gerçekleşmekte olan ya da gelecek olacak bir görüşme olduğu anlamını taşıyabilir. Bunun yanında diyetisyenKatilimi ve kullaniciKatilimi bu kullanıcıların görüşmeye katılım durumlarını içerdikleri için, geçmişte kalan ancak katılım gerçekleşmeyerek tamamlanamamış bir görüşmenin de kayıt altına alınabilmesini mümkün hale getirmektedir.

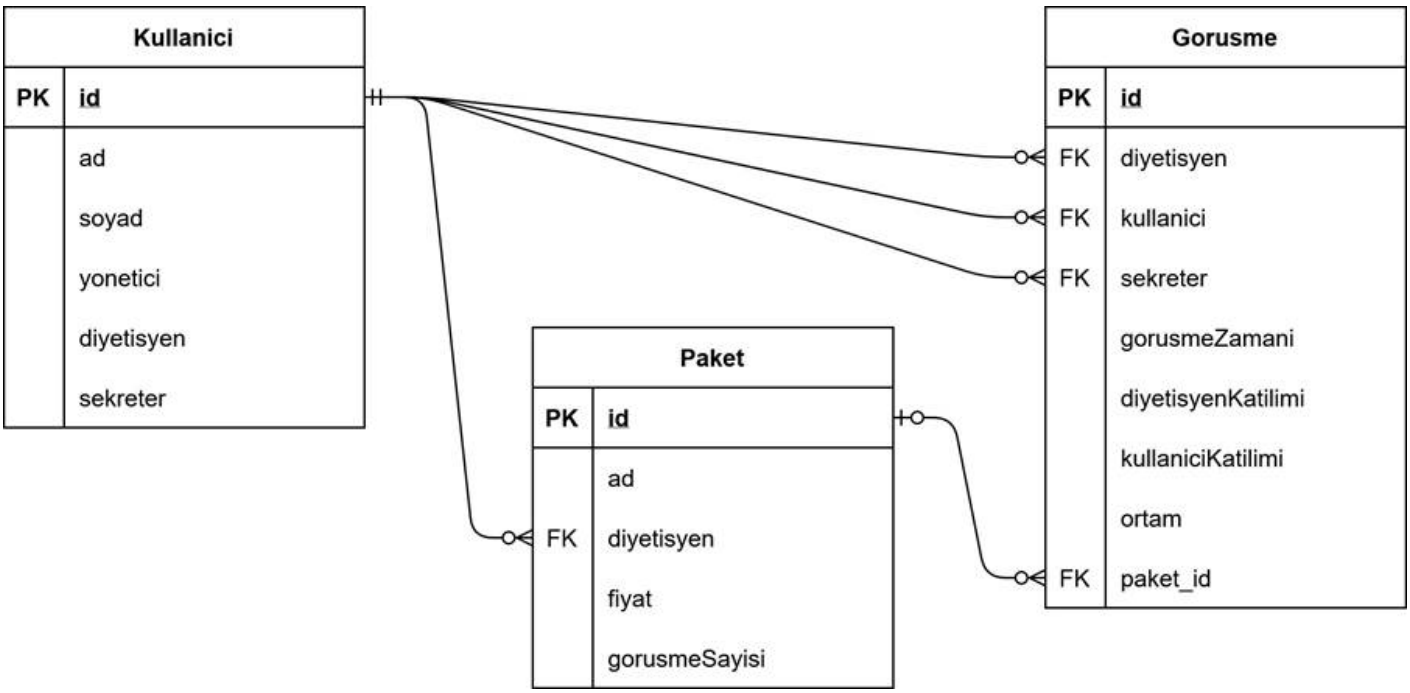
Son nitelik olan ortam, seçtiğimiz fiziksel ve elektronik kelimeleri için eklenmiştir. Bu kelimeler görüşmenin yapılacağı ortamın türünü belirtmektedir. Dolayısıyla yapılacak görüşmenin bir özelliği ve Gorusme tablosunun bir niteliğidir. ortam niteliği elektronik ya da fiziksel değerlerinden birini alabilir. Eğer burada saklama kapasitesinden tasarruf edilmek isteniyorsa kısaca e ve f harfleri de kullanılabilir. Bunun ötesinde bir diğer seçenek ortam niteliği yerine fiziksel ya da cevrimici adlı bir nitelik kullanılabilir. Örneğin, fiziksel niteliği kullanılması durumunda, 1 değeri görüşmenin fiziksel ortamda gerçekleştirileceğini; 0 değeri ise çevrimiçi ortamda gerçekleştirileceğini gösterecektir. Bu yaklaşımların tümü işe yarayacaktır.

Son olarak paket kelimesini ele alalım. Açıklamalar sayesinde her diyetisyen için birden fazla paket tanımlanabileceğini görmekteyiz. Öncelikle Paket adında bir varlık tanımlamalı, sonrasında da öngördüğümüz nitelikleri eklemeliyiz. Aşağıda bu varlık için önerimi sunuyorum.

Paket: id, ad, diyetisyen, fiyat, gorusmeSayisi

Paket varlığı içerisinde id ve ad nitelikleri paketi tanımlamayı sağlayan niteliklerdir. diyetisyen niteliği paketin hangi diyetisyene ait olduğunu gösteren bir ikincil anahtardır. fiyat niteliği paketin satın alma fiyatını, gorusmeSayisi ise bu paket satın alındığında gerçekleştirilebilecek görüşme sayısını içermektedir.

Tüm bu incelemeler sonucunda veri tabanı tasarımımız için bir öneri çizebilecek durumdayız. Önerilerimle aynı doğrultuda çizdiğim tasarım Şekil 8.1’de sunulduğu gibidir.



Üzerinde tartıştıklarımıza ek olarak burada Gorusme varlığı içerisinde paket_id niteliği göreceksiniz. Bunun sebebi Gorusme ve Paket varlıkları arasındaki olası bağlantıdır. Görüşme, satın alınmış bir paket dahilinde gerçekleştiriliyor olabilir. O halde bu görüşmenin paket ile ilişkilendirilmesi normaldir.

8.2. Araç Satışı Portalı

8.2.1. Problem

Kullanılmış arabaları alan ve satan bir işletme bir bilgi sistemine ihtiyaç duyarak sizinle iletişime geçti. İşletme sorumlusu, her türlü marka ve model aracın alınıp satıldığından bahsetti ve bunların kayıt altına alınmasını istediklerini belirtti. Bazı durumlarda eski müşterilerine avantajlar sunabilmek için geriye dönük de sorgulamalar yapmak istediklerinden bahsedildi. Sistemin genel hatlarını öğrendikten sonra aşağıdaki maddeleri not aldınız.

1) Her bir araç birbirinden farklı olduğu için aynı marka ya da model olsa bile yeni kayıt girilmeli. (İkinci el ürün pazarı gibi)

2) Firma tüm marka ve modelleri alıp satıyor.

3) Alış ve satış kayıtları birbirinden bağımsız kayıt altına alınmalı.

4) Araç alış ya da satış işlemiyle ilgilenen müşteri temsilcisi de kayıt altında olsun.

5) Alış ve satış işlemlerinde yapılan ya da alınan tüm ödemeler kayıt altına alınmalı.

6) Ödemeler nakit, kredi kartı ya da banka havalesi ile gerçekleştirilebilmeli.

Bahsedilen sistem için bir veritabanı tasarımı gerçekleştiriniz.

8.2.2. Çözüm Önerisi

Çözüme başlarken problemin tarifinde ve ayrıntı içeren notlardaki olguların neler olduğunu belirlemeye çalışalım. Tüm problem başlığını dikkate aldığımızda sistemin aşağıdaki bileşenlere sahip olduğunu düşünüyorum.

Araba, İşletme, Marka, Model, Müşteri, Firma, Müşteri Temsilcisi,

Nakit, Kredi Kartı, Banka Havalesi

Bunun yanında aşağıdaki işlevlerden bahsedildiğini de görmekteyiz.

Alış, Satış, Kayıt, Ödeme

Olgu ve işlevlerin tamamını bir defada belirleyip, sonrasında türleri üzerine yoğunlaşmak da ele alınabilecek bir yöntemdir. Bu süreci tekrarladıkça olguları tanımanız ve nasıl kullanacağınız konusundaki bakış açınız daha da netleşecektir. Burada daha çok ad olarak geçen olguların birer varlık adı olmaya, işlev adlarının ise muhtemelen çoğa çok bağlantının varlığı olmaya aday olduğunu hatırlamak gereklidir. Şimdi ele aldığımız tüm kelimelerden hangilerinin işe yarayacağını belirlemeye çalışalım.

İşletme ve Firma aynı yapıyı ifade ediyor olmakla birlikte sistemin tamamını yöneten yapıda olduğu için bir varlık olarak tanımlama ihtiyacımız bulunmamaktadır. İşletmenin şubeleri ya da bölgeleri gibi bir şekilde ilişkili bileşenleri olsaydı bu konuyu biraz daha farklı değerlendirebilirdik. Ancak şu durumda işletme bilgisi bizim için herhangi bir varlık ile doğrudan bağlantılı değil, tüm sistemi kapsayıcı niteliktedir. Bu sebeple bu iki kelimeyi değerlendirme dışı bırakıyoruz.

Araba, probleme konu olan ve alış ve satışı yapılan varlığın ta kendisi olduğundan bir varlık olarak tanımlanması yerinde olacaktır. Bununla birlikte Marka ve Model kelimeleri, Araba varlığının bir özelliğini temsil ettiği için bu varlık altında nitelik olarak yapılandırılmaları uygun olacaktır.

Müşteri, alış ve satış işlerindeki muhatap kişi olmakla birlikte Müşteri Temsilcisi de bu işlemleri gerçekleştirmek üzere görevli işletme yetkilisidir. İki ifade de birer kişiyi temsil etmekle birlikte bir kişinin müşteri ya da müşteri temsilcisi sıfatı kazanması, o kişi için nitelik olarak görülebilir. Dolayısıyla bu iki olgu Kişi ya da Kullanıcı olarak birleştirilebilir ve görevi bir nitelik olarak ele alınabilir.

Nakit, Kredi Kartı ve Banka Havalesi olguları Ödeme işleminin bir özelliğidir. Bu sebeple bu üç olguya ödemeyle ilgili bir varlık içerisinde nitelik olarak yer vermek daha mantıklı olacaktır.

Sistemi işlevler açısından incelediğimizde temelde tek bir işlem olduğunu görmekteyiz: Araç satışı. Alış ve Satış işlemleri aynı anlama gelmekte, sadece satışın yönü değişmektedir. Kayıt işlevi de alış ve satış işlemlerin kaydı olduğu için ayrıca belirtilmesine gerek bulunmamaktadır. Ödeme işlevi ayrı ve tek başına varlık olmayı hak eden bir varlık gibi gözükse de aslında alış ve satış işlemlerinden türeyen bir işlev olduğunu unutmamak gerekmektedir. Dolayısıyla Ödeme yeni bir varlık olarak yapılandırılacak ancak mutlaka satış işleviyle ilişkilendirilecektir.

Yaptığımız analizin sonuçlarını toplarsak neticede aşağıdaki varlıkları elde ederiz. Ayrıca bu varlıklar için bazı nitelik önerilerini de aşağıda bulacaksınız.

Araba: id, marka, model, plaka, motorGucu, yakitTuru

Kisi: id, ad, soyad, telefon, ePosta, yetki

Odeme: id, tutar

Satis: id, timestamp

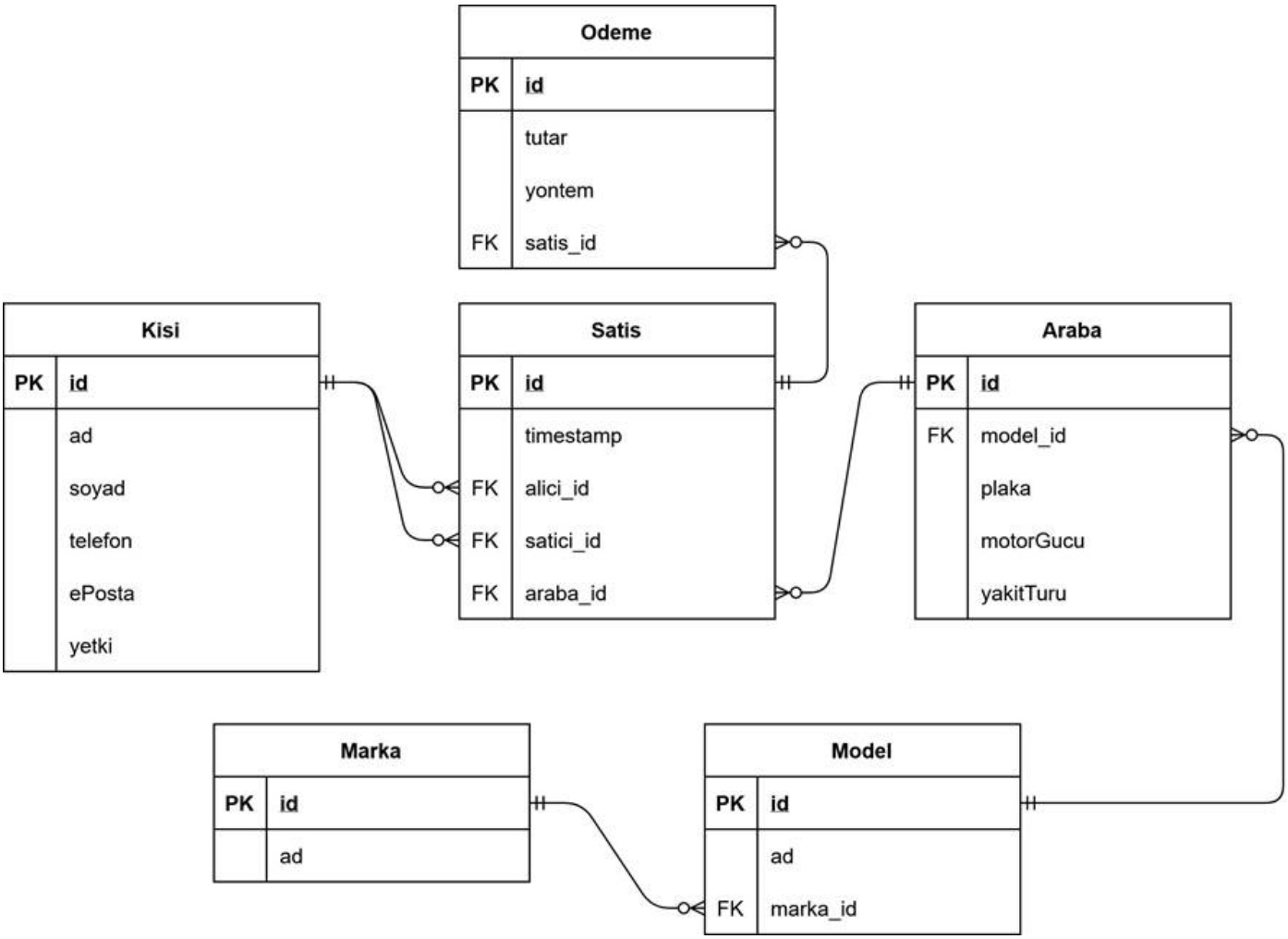
Varlık önerilerini sunarken Odeme ve Satis varlıklarının birer işlevi temsil ettiğinden ve bunların çoğa çok bağlantı yapısındaki bağlantı varlığı özelliği taşıyor olmasının muhtemel olduğundan bahsetmiştik. Yukarıdaki varlık tanımlaması önerilerine dikkat ederseniz bu iki varlığın diğer iki varlığa ait bilgiler içermesi gerektiğini, dolayısıyla bir ilişki kurulması gerektiğini fark edebilirsiniz. Örneğin Satis varlığı alıcı ve satıcıya ait bilgiler içermelidir. Ayrıca ödemeye ait de bilgiler içermesi gerekmektedir. Benzer şekilde Odeme varlığı da bir satış ve bu satışta yer alan kişilerle ilişkili olmalıdır.

Araba ve Kisi varlıklarını ele alalım. Bu iki varlık birbiriyle ilişkili olmalıdır. Bu ilişki, işlevsel olarak satış işlemi olarak yorumlanabilir. Kontrol etmek açısından ilişki türü belirlemeyle ilgili kontrol sorularımızı yöneltebiliriz. Bir araba birden fazla kişiyle ilişkili olabilir. Bir kez satış işlemi olsa bile bir alıcı bir satıcı olduğu için iki kişiyle bağlantılıdır. Bunun yanında birden fazla kez satış işlemi de gerçekleşebilir. Benzer

şekilde kişi de birden fazla araba ile ilişkili olabilir. Böylece bu iki varlığın çoğa çok ilişkili olduğunu teyit etmiş olduk. Aradaki ilişkinin de satış işlemi olduğunu bildiğimiz için bağlantı varlığı olarak doğrudan Satis varlığını kullanabiliriz.

Ödeme işleminin de bir satış ile bağlantılı olması gerektiğinden bahsetmiştik. Aradaki ilişki ise bizim bir varsayımda bulunmamızı gerektiriyor. Bir ödeme işleminin tek bir satış işlemiyle ilgili olduğunu varsaymak kolay olacaktır. Ancak bir satış işleminin tek bir ödemeyle ilişkili olduğunu söylemek o kadar da kolay değildir. Eğer her satış işleminin tek türde tek bir ödemeyle yapıldığını varsayarsak buradaki ilişki türü bire bir olacaktır. Ancak parçalı ödeme gibi yöntemlerin kullanıldığını düşünüyorsak bu durumda bir satış işlemi birçok ödemeyle ilişkilendirilebilir. Bu durumda da bire çok ilişki ile karşı karşıya kalacağız. Ödeme işlemleri genellikle hassas olduğu için varsayımlarımızı mümkün olduğunca herhangi bir hataya sebep olmayacak şekilde belirlememiz gerekmektedir. Bire bir bağlantı bizim için aynı zamanda bir kısıt sayılacaktır çünkü bu durumda bir satış işlemi ancak tek bir ödeme ile ilişkilendirilebilecektir. Bire çok bağlantıda ise bir satış işlemiyle birden fazla ödemeyi ilişkilendirebileceğimiz gibi tek bir ödemeyle de ilişkilendirebiliriz. Burada bire çok ilişki, bire bir ilişkinin bir kapsayanı olarak görev yapacaktır. Böylece veri yapısını olası karar değişikliğinde bile bozulmayacak olarak yapılandırmış olacağız. Bu sebeple Satis ve Odeme varlıkları arasındaki bağlantıyı bire çok türünde tanımlayabiliriz.

Tüm bu yorumlar neticesinde bir veri tabanı tasarımı önerisinde bulunalım. Çözüm önerisi için lütfen Şekil 8.2'yi inceleyiniz.



Şekilde, değerlendirmelerimizi yaparken konuştuğumuz varlıklara ek 2 varlık daha yer aldığını göreceksiniz. Bu varlıklar, Araba varlığının içerisindeki marka ve model nitelikleri tekrarlı veri içerecek olması sebebiyle eklendiler. Araba markaları belirli sayıdadır ve kendini tekrar edecektir. Benzer şekilde modeller de markalara bağlıdır ve onların değerleri de tekrarlıdır. Bu sebeple Marka ve Model iki varlık olarak eklenmiştir. Çözüm önerisinin başlarında bu iki olgu için Araba varlığının birer niteliği olduklarını ve ayrı varlık olmayı hak etmediklerini söylemiştik. Aslında görüşümüz hala geçerli. Veri yapısını incelediğimiz zaman bu iki olgu bizim için hala Araba varlığının nitelikleri. Burada sadece veri tekrarı olmaması ve daha etkin bir tasarım elde etmek amacıyla bu değerleri bir varlık içerisinde topladık ve Araba varlığına bir ikincil

anahtar ekleyerek bağlantıyı sağladık. Bu da dolaylı da olsa hala bu iki özelliğin Araba varlığı içerisinde yer aldığını göstermektedir.

Çözüm önerisini incelediğinizde bir ayrıntının daha dikkatinizi çekmesi gerekiyor. Şekil 8.2 ile verilen veri tabanı tasarımı, bir önceki bölümde bahsettiğimiz özel durumlardan birine uyuyor. Lütfen, bu ayrıntı hakkında bir fikriniz yoksa tasarıma geri dönerek hangi özel duruma uyduğuyla ilgili inceleme ve tahminde bulunmaya çalışınız.

Yanıttan önce şunu söylemek gerekmektedir. Veri tabanı tasarımları hiçbir zaman birebir aynı şablon ile karşımıza çıkmazlar. Özel durumlar başlığı altında gördüğünüz durumlar, bu özelliğe sahip her bir veri tabanı için birebir aynı biçimde karşınıza çıkabileceği anlamı taşımamaktadır. Verilen tasarım, yıldız şema olarak değerlendirilebilecek bir tasarımdır. Dikkatli incelediğinizde bu tasarımda bir veri akışının kayıt altına alındığını görebilirsiniz. Satis varlığı, bahsettiğimiz veri akışının kayıt altına alındığı varlıktır. Kisi ve Araba varlıkları bu tasarımda boyut varlığı olarak değerlendirilebilir. Ödeme varlığı da bir boyut varlığı olmakla birlikte Kayıt tablosu kadar, belki daha fazla kayıt içeriyor olması sebebiyle ilgi çekici bir ayrıntı sunar. Marka ve Model varlıkları ise Araba varlığındaki veri tekrarını aşmak için uygulanmış bir çözümün neticesidir.

8.3. Para Transferi Yazılımı

8.3.1. Problem

Bir bankanın yazılım geliştirme departmanını çalışanısınız. Araştırma - geliştirme projeleri kapsamında, banka müşterisi olsun ya da olmasın herkesin birbirine para göndermesini mümkün hale getirebilecek bir yapı üzerine çalışıyorsunuz. Bu yapıya göre kişiler telefon numaraları tarafından temsil edilmekte ve buna göre birbirlerine para gönderebilmektedir. Telefon numarası bilgisi kişilerin kimliği yerine geçmekle birlikte aynı zamanda doğrulayıcı rolü de üstlenmektedir. Bu projede veri tabanı geliştiricisi olarak yer almaktasınız. Gerçekleştirilen çok sayıda toplantı neticesinde özet olarak aşağıdaki notları aldınız.

1) Sistem kullanıcılarının banka müşterisi olma zorunluluğu yok.

2) Telefon numarası kişiler için kimlik niteliğinde.

3) Telefon numarası mutlaka SMS kodu yöntemiyle doğrulanmalı.

4) Gönderilecek para limiti veri tabanı üzerinde tutulmalı ve gerektiğinde güncellenebilmeli.

5) Para göndermek bir zaman aşımına sahip olmalı. Zaman aşımına uğrama durumunda göndericiye iade gerçekleşmeli.

Bu proje için bir veri tabanı tasarımı gerçekleştiriniz.

8.3.2. Çözüm Önerisi

Problemi analiz ederken öncelikle varlık adayı olgularımızı ve işlevlerimizi belirlemeye çalışalım. Problem başlığı altındaki ad özelliğindeki kelimeleri toparladığımızda aşağıdaki listeyi elde ediyoruz.

Banka, Müşteri, Telefon Numarası, Kişi, Doğrulayıcı, Kullanıcı,

SMS Kodu

Bunun yanında yine problem başlığı altında sistemin işlevi olarak sayılabilecek kelimeler de aşağıda listelenmektedir.

Para Göndermek, SMS Doğrulaması, Para Limiti, Zaman Aşımı, İade

Birçok sistemde farklı kullanıcı seviyeleriyle ilgili kayıtlar, daha önce bahsettiğimiz sebeplerden dolayı tek bir Kisi ya da Kullanici varlığı altında toplanırlar. Bu tür varlıklar her zaman bir kişiyi işaret etmekle birlikte

sistemlerdeki farklı kullanıcı tipleri bu kişi varlığında bir nitelik olarak kolaylıkla tutulabilir ve veri bütünlüğünü mümkün hale getirebilirler. Ayrıca tüm kişileri tek bir varlık içerisinde toparlamak, gerçekleşmesi olası yetki değişikliklerinde bir kaydın varlıktan varlığa taşınmasındansa bir kaydın tek bir niteliğinin güncellenmesi ile sağlanabilmektedir. Bu sebeplerden dolayı gerçekleştireceğiniz veritabanı tasarımlarında eğer kullanıcılar işin içindeyse, her ne kadar farklı yetki seviyeleri ya da kullanım şekli olsa da kullanıcılara ait tek bir varlık iş görecektir. Bu çözümün riskli olduğunu düşündüğünüz durumlarda elbette farklı bir çözüm uygulayabilirsiniz ancak rahatlıkla söyleyebilirim ki bu yöntem yüzde 90'ın üzerinde iş görecektir.

Bu açıklama üzerine, sistemimizin kullanıcıları olduğu gerçeğini hatırlayarak bu tabloyu tanımlamak üzere hangi olguları listeden elediğimize bir bakabiliriz. Musteri, Kisi ve Kullanici yerine ortak bir ad seçerek devam edelim: Kullanici. Banka, sistemin tamamına ait bir tanımlama olduğu için bir varlık olarak belirtmeye ihtiyacımız yok. Eğer problem açıklaması içerisinde Bir bankanın ... şeklinde açıklama yapılması yerine birden fazla bankanın ortak gerçekleştirdiği bir süreç olarak tanımlansaydı elimizde birden fazla banka olacağı için Banka varlığını kullanmak gerekli olacaktı. Telefon Numarasi, Kullanici niteliğine ait bir özellik gibi görünüyor. Ancak bu konuda biraz daha ayrıntı var. Banka kullanıcılarının, bankanın müşterisi olmayan kişilere telefon numarası üzerinden para gönderilebileceğinden bahsettik. Dolayısıyla para gönderimi yapılacak telefon numarası hali hazırda bir tablo içerisinde kayıtlı olmayabilir. Sistemde kayıtlı olmayan bir kişiye para gönderimi yapabileceğimiz için Telefon Numarasi bir kullanıcının değil ödeme işleminin bir niteliği olmalı. Bu durumu tasarımı gerçekleştirdikten sonra ayrıntılı yorumlayacağız. Dogrulayici, kişinin telefon numarasının kendisine ait olduğunu kanıtlaması için kullanılan, banka tarafından oluşturulan ve telefon numarasına gönderilen özel bir kodu ifade etmektedir. Kişi, telefon numarası gerçekten kendisine aitse gelen kodu bankaya geri bildirir ve numaranın kendisine ait olduğunu ispatlar. Bu kod, ödeme işleminin bir niteliği olmalıdır. Her bir ödeme işleminde yeni bir kod üretilmeli ve ödeme kaydıyla birlikte saklanmalıdır.

Şimdi işlevleri inceleyelim. Para gönderimi, iki kişi arasında gerçekleştirilen bir işlemdir. Kişi varlığının kendi kendisiyle çoğa çok bağlanması sonucunda oluşabilecek bağlantı varlığı olarak da karşımıza çıkmaktadır. Bu varlığa Odeme adını verelim. Bir kişi birden fazla kişiye para gönderebilirken, bir kişi birden fazla kişiden para alabilir. Ancak burada alıcının Kullanici tablosunda kayıt olma zorunluluğu yoktur. Tasarımı gerçekleştirirken bu ayrıntıyı tasarım üzerinde mutlaka belirtmemiz gerekmektedir. SMS doğrulamasının Odeme varlığında yer alması gereken bir nitelik olduğundan bahsetmiştik. Para limiti ve zaman aşımı değerleri sisteme ait bir ayar varlığında saklanmalı, bunun yanında zaman aşımı her bir ödeme için de kayıt altına alınmalıdır. Ayar varlığı zaman aşımının ne kadar süreceğini kayıt altında tutmalı, Odeme varlığı ise kayda alınan ödemenin hangi zamana kadar geçerli olduğu bilgisini bulundurmalıdır. İade işlemi yeni bir varlık yerine mevcut ödeme kaydı üzerinde nitelik ile tutulabilecek bir bilgidir. Yeni bir varlık olarak tanımlamak hata olmayacağı gibi bu konuda ek bilgilerin kayıt altına alınması istenilmediği için mevcut varlık içerisinde bu bilgiyi tutmak çözümü pratikleştirecektir.

Bu bilgilere göre varlık tanımlarımız için aşağıdaki planlamayı kullanabiliriz.

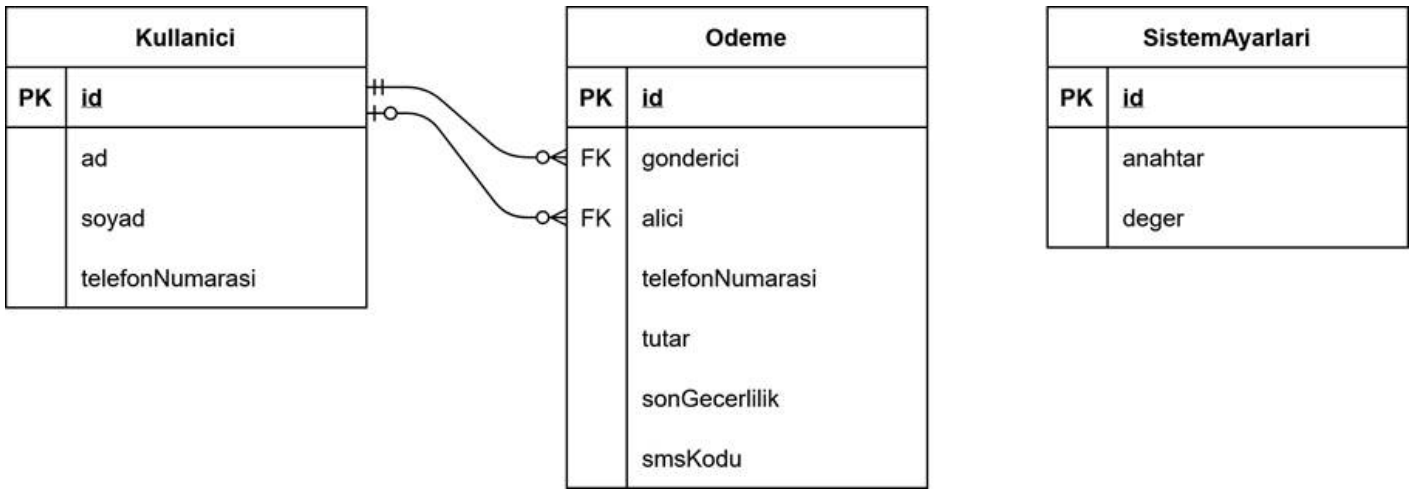
Kullanici: id, ad, soyad, telefonNumarasi

Odeme: id, gonderici, alici, telefonNumarasi, tutar,

sonGecerlilik, smsKodu

SistemAyarlari: id, anahtar, deger

Bu tanımlamalara ve yapılan açıklamalara uygun olarak hazırlanan veri tabanı tasarımı Şekil 8.3 ile sunulmuştur.



Tasarım önerisi üzerindeki bazı ayrıntıların üzerinde duralım. Öncelikle bu tasarım da yıldız şemanın bir örneği olabilir. Her ne kadar tek boyuta sahip olsa da ödemeyle ilgili tüm kayıtları içeren bir kayıtsal veri yapısı ile karşı karşıyayız. Odeme varlığı, Kullanici varlığının kendi kendisiyle yaptığı çoğa çok bağlantının bağlantı varlığı olarak da görülebilir. Bir kullanıcının hiçbir ödeme yapmaması ya da almaması muhtemel olduğu için bağlantıların Odeme varlığı ile birleşim noktalarında zorunluluk olmaması normaldir. Ancak gönderici kullanıcı bağlantısında zorunluluk olduğuna, alıcı kullanıcıda ise herhangi bir zorunluluk olmadığına dikkatinizi çekmek isterim. Bunun sebebi problem içerisinde banka müşterisi olmayanlara da para gönderilebileceğinin belirtilmiş olmasıdır. Bu ayrıntı sebebiyle Odeme varlığı alici ikincil anahtarı üzerinden her zaman Kullanici varlığı ile ilişki kuramayabilir.

Bir diğer ayrıntı telefonNumarasi niteliğinin iki varlık içerisinde birden bulunması. Odeme varlığı hem alici hem de gonderici üzerinden Kullanici varlığı ile bağlantılı. Dolayısıyla telefon numarası bilgisi erişilebilir. Ancak burada ayrıntı şu ki; telefon numarası ödeme işlemi için büyük önem taşıyor. Özellikle alıcının telefon numarası. Az önce bahsettiğimiz üzere ödeme alıcısının Kullanici varlığında herhangi bir kaydı olmayabilir. Ancak bizim bu bilgiyi boş geçmemiz mümkün değil. Dolayısıyla telefon numarası bilgisinin mutlaka ödeme ile ilişkilendirilmesi gerekiyor. Eğer gönderici ve alıcılar Kullanici varlığı içerisinde tanımlı olsaydı bile ödeme yapılmak istenilen telefon numarası kullanıcının varsayılan iletişim hattından farklı olabilirdi. Bu durumda da ödeme kaydı oluşturulacak telefon numarasının farklı olarak kaydedilebilmesi gerekiyordu. Her halükarda bu çözüm riskleri en aza indirebiliyor. Unutmayın, veri bütünlüğünü sağlamak birincil önceliğimizdir.

Son olarak sistem ayarları ve zaman aşımı yönetimini ele alalım. Sistem ayarları, özel durumlarda da bahsettiğimiz üzere diğer varlıklardan ayrık bir şekilde veri tabanına kaydedilebilir. Burada istenilen anahtara bağlı değerler tutularak sistem ayarı olarak kullanılabilir ve ihtiyaç halinde güncellenebilir. Problemdeki taleplerden biri ödemelerin bir geçerlilik tarihine sahip olabilmesiydi. Bu bilgi SistemAyarlari varlığına aşağıdaki gibi bir kayıtla eklenebilir:

id: 1, anahtar: gecerlilik, deger: 86400

Geçerlilik süresi ayar varlığı içerisinde gecerlilik anahtarı ile saklanmış olmaktadır. Değeri ise görüldüğü üzere 86400'dür. Bu değer 1 güne karşılık gelen saniye sayısıdır. Bu kayıt sayesinde yeni bir ödeme kaydı oluşturulduğunda Odeme varlığındaki sonGecerlilik niteliğinin değeri şu şekilde belirlenebilir:

Odeme.sonGecerlilik = simdi() + 86400

Burada simdi() fonksiyonu şu ana ait zamanı timestamp cinsinden veren fonksiyon olsun. 86400 değeri ise SistemAyarlari varlığından çekilen güncel gecerlilik değeridir. Bu durumda ödemeye ait son geçerlilik, kaydın oluşturulduğu andan tam bir gün sonrasını işaret edecektir. SistemAyarlari varlığında yapılacak bir güncelleme yeni eklenecek kayıtlarda etki edecek ancak geçmiş kayıtlar için herhangi bir etkisi olmayacaktır.

8.4. Nakliye Firması Organizasyon Yazılımı

8.4.1. Problem

Bir kargo firması için kargo takip sistemi geliştirme ekibine dahil oldunuz. Projedeki göreviniz veri tabanı tasarımını gerçekleştirmek. Firma, Türkiye'deki tüm şehirlere hizmet veren bir kargo firmasıdır. Bir gönderi, kargo firmasının herhangi bir şubesinden yolculuğa başlar, çeşitli aktarma merkezlerinden geçerek varış şubesine ulaşır. Varış şubesi personelinin de adrese teslim etmesiyle sonlandırılır. Geliştirilecek projeyle ilgili tüm ayrıntıları konuştuktan sonra bazı notlar aldınız. Aldığınız notlar şu şekilde:

1) Kargonun tüm hareketlerinin dökümü alınabilmeli.

2) Sistem, müşteriler ve çalışanlar tarafından erişilebilir olmalıdır.

3) Müşteriler bilgileri yalnızca görüntüleyebilmeli, çalışanlar yeni kayıt ekleyebilmeliler.

4) Mevcut kayıtlar silinememelidir.

Taleplere uygun bir sistem geliştirilebilmesi için uygun veri tabanı tasarımını gerçekleştiriniz.

8.4.2. Çözüm Önerisi

Daha çözüm üzerine herhangi bir adım atmadan yıldız şema biçiminde bir çözüme ulaşacağımızı tahmin edebileceğimiz bir örnek ile karşı karşıyayız. Çoğumuzun aşina olduğu kargo hizmeti, bir malın iki kişi ve/veya kurum arasındaki nakliyesini konu almaktadır. Bu hizmet için oluşturulmuş bir sistem, sürekli yeni teslimatların sisteme eklenmesi ve tüm teslimatların süreçlerinin kayıt altında olması sebebiyle geriye dönük işlemlerin de kayıt altında bulunması neticesinde bir kayıtsal veri tabanı biçiminde değerlendirilebilecektir. Dolayısıyla kargo hareketlerini içeren bir varlığa sahip olacağımızı ve bunun yıldız şemadaki kayıt (fact) varlığına denk geleceğini savunabiliriz. Bunları öngörüyor olabilsek de herhangi bir hata yapmamak için izlememiz gereken adımları izleyelim.

Problem başlığı altındaki kelimeleri incelediğimizde ad özelliğine sahip olanların varlık, fiil olanların ise bağlantı varlığı olmaya aday olduğunu gördük. Bu noktada varsa sıfatların da genellikle bir nitelik adayı olduğunu belirtelim. Şimdi adları ve fiilleri seçerek aday varlıklarımızı bulmaya çalışalım.

Kargo, Gönderi, Aktarma Merkezi, Şube, Personel, Müşteri,

Çalışan

Bunlara ek olarak aşağıdaki ifadeleri fiiller olarak değerlendirebilir ve sistemin işlevleri olarak dikkate alabiliriz.

Takip, Aktarma, Teslim, Görüntülemek, Kayıt

İlk kelimeyle incelememize başlayalım. Kargo, bu sistemde takibi yapılacak ürünlerin kayıtlarının tutulması için gerekli olan bir varlıktır. Bununla birlikte listedeki Gonderi ifadesi de aynı amaçla kullanılmış; bir kargonun nakliye edilmesi sürecini kasteden bir kelimedir. Kargo adı sistemin geneli için de kullanılan bir kelime olduğundan burdaki varlık adını Gonderi olarak seçmeyi tercih edebiliriz.

AktarmaMerkezi gönderilerin nakliyeleri esnasında kullanılan merkez birimlerini, Sube ise gönderi nakliyatının başlangıç ve bitiş noktalarındaki ofisleri ifade etmektedir. Bu iki ifade birleştirilerek Birim adı altında tek bir varlıkta toplanabilir. Tüm nakliye süreci için gönderinin uğradığı noktalar Birim adıyla anılabilir.

Personel, Musteri ve Calisan ifadeleri temelde sistem kullanıcılarını ifade etmekle birlikte kullanıcının sahip olduğu yetkiye bağlı bir farklılık içermektedir. Önceki atölyelere benzer şekilde burada bir Kullanici varlığı tanımlayabilir, kullanıcının türünü bir nitelik ile kayıt altına alabiliriz.

Kelime çıkartırken ayırdığımız fiillerin bizim için sistemin işlevlerini temsil ettiğini söylemiştik. Şimdi bunları ele alalım. Takip, her bir gönderi için nakliye sırasındaki işlem kayıtlarının tutulduğu bir varlık olarak kurgulanabilir. Böylece gerçekten de gönderinin her bir adımının takip edilmesini sağlayan bir veri yapısı elde edebiliriz. Aktarma, yine gönderinin hareketleriyle ilgili bir eylem olduğu için Takip tarafından

kapsanan bir özellik olacaktır. Dolayısıyla bu ad için yeni bir varlığa ihtiyaç duymayacağız. Benzer şekilde Kayıt da işlevsel olarak bu varlığın işlevi kastedilerek kullanıldığı için göz ardı edilebilir.

Teslim süreci, bir gönderinin hedefe ulaşmasıyla ilgili bilgidir. Dolayısıyla bu bilgi bir Gonderinin niteliği olmalıdır. Yeni bir varlık oluşturmak yerine Gonderi varlığına teslimle ilgili nitelik ya da nitelikler oluşturarak bu bilgiyi tutabiliriz. Bu durum karmaşık geldiyse test etmek üzere bir deneme yapabiliriz.

Gonderi varlığıyla birlikte Teslim adında da bir varlık oluştursaydık bunların arasındaki ilişki tipi ne olurdu?

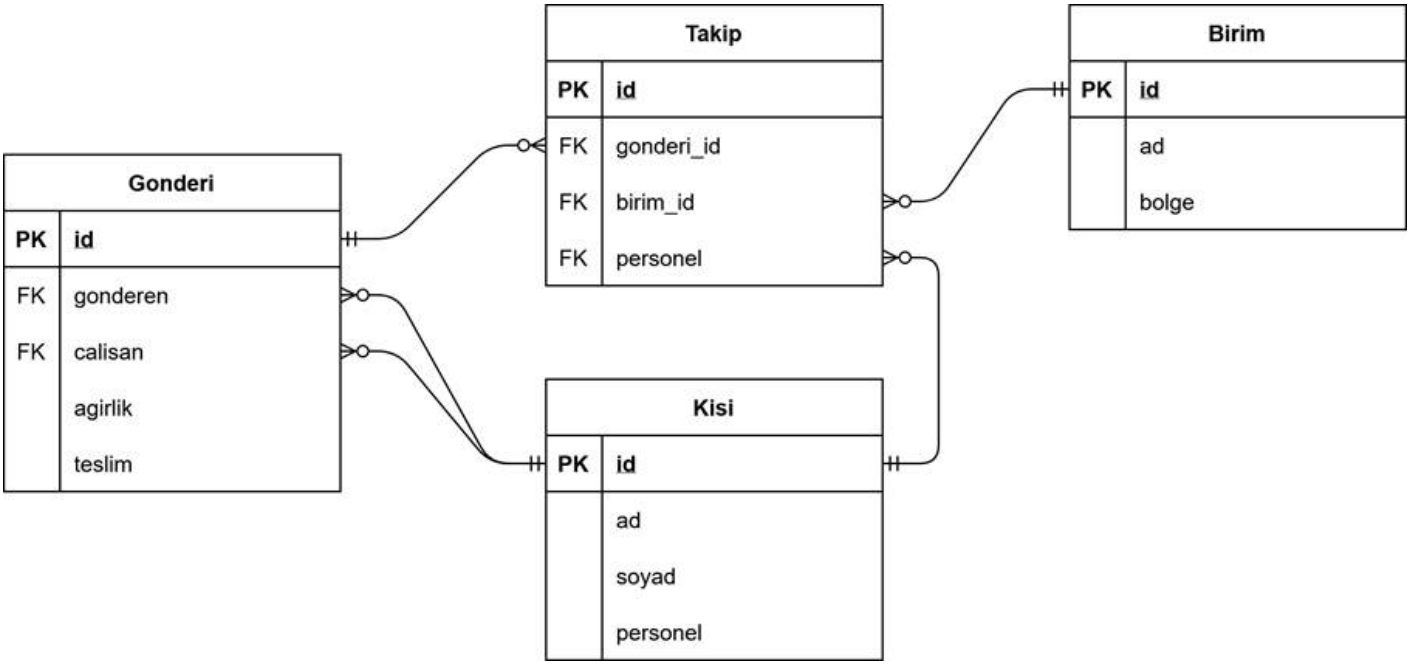
Bir gönderi, yalnızca tek bir teslim bilgisine sahip olmalıdır. Bir teslim bilgisi de yalnızca tek bir gönderiye ait olmalıdır. Burada bire bir ilişki ile karşılaşıyoruz. Bire bir ilişkilerin genellikle ihtiyaç duyulan ya da tercih edilen bir ilişki türü olmadığından bahsetmiştik. Bu ilişki türü genellikle özellikle ihtiyaç duyulduğunda kullanılan, zorunluluk gerektirmeyen ilişki türleridir. Bir tasarımda bire bir ilişki ile karşılaştıysanız ve bunu planlayarak yapmadıysanız bir hata yapmış olmanız olasıdır. Bu sebeple ilişki türünü belirlerken dikkatli olmakta fayda var.

Peki burada bire bir ilişki kullanmak her zaman hata mıdır? Yanıt: Hayır. Eğer ki biz teslimat ile sadece teslim edildi ya da edilmedi bilgisini tutmuyor, bunun yerine teslimatla ilgili çok fazla nitelik barındırıyorsak Gonderi varlığını kullanmak istemeyebiliriz. Bu durumda yeni bir varlık oluşturarak Gonderi tarafında zorunluluk olan, Teslim tarafında olmayan bire bir bağlantı da çözüm olabilir. Ele aldığımız problemde böyle bir ayrıntı sunulmadığı için teslimat bilgisini tamamlandı ya da tamamlanmadı bilgisi içerecek şekilde kurgulayabilir, dolayısıyla Teslim varlığına ihtiyaç duymayabiliriz.

Son olarak görüntülemek kelimesini ele alalım. Sistemi talep eden kargo firması, bu işin uzmanı olmadıkları için hangi özelliği kimden talep edeceklerini bilemeyebilirler. Bilgilerin müşteriler tarafından yalnızca görüntülenmesini, personelin sadece giriş yapıp silme işlemi gerçekleştirememesini talep edebilir ve bunun veritabanı sorumlusu olduğunuz için sizin göreviniz olduğunu düşünebilirler. Ancak bu tarz arayüz kısıtlama işlemlerin genellikle yazılımın içerisinde gerçekleştirilmektedir. Dolayısıyla siz veri tabanında kullanıcının yetkisini tutarsınız, yazılım ve arayüz geliştiricileri de sağladığınız yetki bilgisine göre ilgili ekranları kullanıcıya gösterir.

Veri tabanları farklı kullanıcılar tarafından kullanılabilecek bir yapıya da sahip olsa da veri tabanı için tanımlanan kullanıcılar genellikle sistemin kendisidir. Geliştirilen bir sistem hem web tabanlı hem masaüstü hem de mobil uygulama olabilir. Bu durumda veri tabanı için 3 kullanıcı oluşturmak bazı avantajlar sağlayabilir. Ancak sistemi kullanan her bir kullanıcı için veri tabanı seviyesinde ve bu erişimi sağlayan bir kullanıcı tanımlanması pek de karşılaşılan bir uygulama değildir. Buradaki ayrımı oluşturan çizgi biraz bulanık gibi görülebilir. Değerlendirmemize genel çerçeveden bakarsak şu yorum ile toparlayabiliriz: Veri tabanı veriyi saklar ve sunar. Bu verinin nasıl kullanılacağı yazılım içerisinde çözülmesi gereken bir problemdir.

Analiz ve yorumlamalar neticesinde ilgili problem için veri tabanı tasarımı Şekil 8.4 ile sunulmuştur.



8.5. Operatör Koordinasyon Yazılımı

8.5.1. Problem

Teknik tarafta meydana gelen gelişmeler sebebiyle bir GSM operatörünün altyapısının yenilenmesi kararı alındı. Yenileme çalışmalarını gerçekleştirecek ekip içerisindesiniz. Tüm fizibilite aşamasında ekip ile birlikte hareket ettiniz ve geliştirmeye yönelik ayrıntılı bilgiye sahipsiniz. Geliştirme sürecinde ilgili sistemin veri tabanı tasarımını hazırlamaktan sorumlu olacaksınız. Fizibilite aşamasında edindiğiniz bilgilere ait notlar aşağıdaki gibidir.

1) Tüm telefon hattı sahiplerinin kişisel bilgileri de kayıt altına alınmalı.

2) Hatlar faturalı ya da faturasız (bakiye yüklemeli) olabilir.

3) Hatlara belirli konuşma süresi, sms ve internet kotası tanımlı paketler atanabilir. Bu paketler belirli tarihler arasında aktifleştirilebilir olmalı.

4) Paketlerin kalan bakiyeyi sonraki aya devretme özelliği olabilir.

Bu duruma uygun veri tabanı tasarımını hazırlayınız.

8.5.2. Çözüm Önerisi

Kelimeleri analiz ederek çözüme başlayalım. Öncelikle varlık aday olan adları yakalamaya çalışalım. Önerilerim şu şekilde:

Telefon, Hat, Sahip, Kişisel Bilgi, Faturalı/Faturasız, Bakiye,

Konuşma, Sms, İnternet, Kota, Paket,

Fiil olan ve sistem için işlevsel özellik taşıma potansiyeline sahip kelimelere de bir göz atalım.

Paket Atama, Aktifleştirme, Devretme

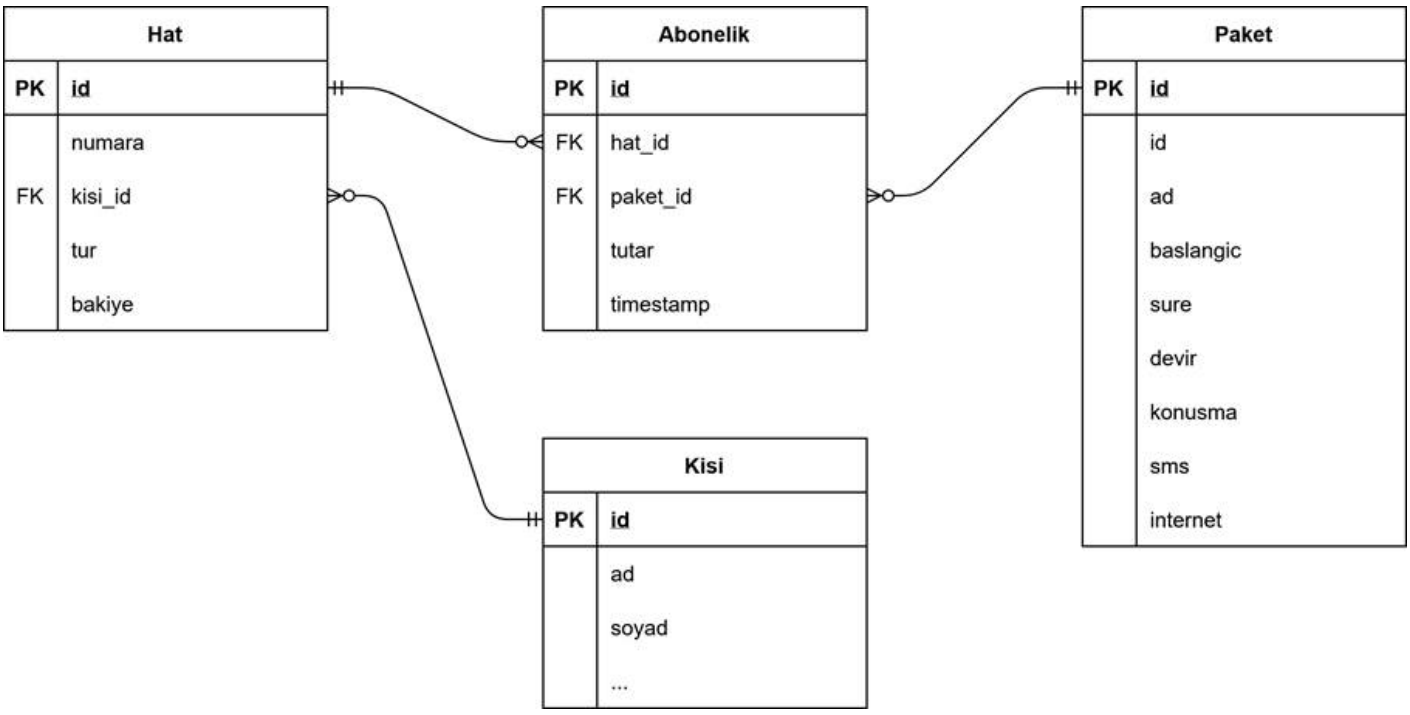
Tasarlanan sistemler genellikle en az bir kullanıcıya sahip olacakları için benim gözüm öncelikle kullanıcı kelimesini arıyor. Farklı yetki seviyeleri için farklı kelimelerle anılan kullanıcılar sayesinde birden fazla kelimeyi bu havuzdan çıkararak kullanıcı kelimesi altında rahatlıkla birleştirebiliyorum. Örneğimizde sahip kelimesi hattın sahibini, dolayısıyla kullanıcının ta kendisini işaret etmektedir. Bununla birlikte bir de kişisel

bilgi ifadesi mevcut. Kişisel bilgiler, kullanıcıyla ilgili varlık içerisinde bir araya getirilebilir. Ancak bir önceki atölyede ele aldığımız üzere bu tarz birçok nitelik içeren bir ek varlık ayrıca oluşturularak bire bir bağlantı ile de tasarlanabilir. İki yaklaşım da yanlış olmayacaktır. Kullanici varlığı oluşturup hem kullanıcı bilgileri hem de kullanıcının kişisel bilgilerini bu varlık içerisinde tutabiliriz. İkinci yaklaşımda ise Kullanici ve KisiselBilgi varlıklarını tanımlayabilir, bunları bire bir bağlantı ile kurgulayabiliriz.

Kullanici ve KisiselBilgi varlıkları arasında oluşturulacak bire bir bağlantı üzerinde zorunluluk durumları nasıl seçilmeli ve ikincil anahtar hangi varlık içerisine yerleştirilmelidir?

Telefon ve Hat kelimeleri neredeyse aynı olgu için kullanılıyor olsa da kurgu için benim önerim, Hat adlı bir varlık tanımlamak ve numara adlı bir niteliği bu varlık içerisine yerleştirmektir. Bir hattın faturalı ya da faturasız olduğu yine o hat için bir nitelik ile tutulabilecek bir bilgidir. Bu sebeple tur adlı bir niteliğin oluşturulması yerinde bir karar olacaktır. Bakiye bilgisi de yine hatta ait bir özelliktir. Bu bilgi de bakiye niteliği olarak Hat varlığına eklenebilir. Bunların yanında konuşma, sms ve internet kotalarının da hatta ait olduğu düşünülebilir olsa da bunlar aslında hat için abone olunan pakete ait bilgilerdir. Bu sebeple öncelikle Paket adında bir varlık tanımlanmalı; konusma, sms ve internet nitelikleri bu varlık içerisine yerleştirilmelidir.

İşlevleri belirleyebilmek için seçtiğimiz fiilleri ele aldığımızda, paket atama ifadesini bir paket için gerçekleştirilen abonelik olarak değerlendirilebilir ve Abonelik adlı bir varlık olarak tanımlayabiliriz. Paket atama ile kastedilen de abonelik durumudur. Devretmek ise bir paketle ilgili tanımlanabilecek bir özelliktir. Dolayısıyla Paket varlığına devir adında bir nitelik ekleyerek bu bilgiyi sağlayabiliriz.



Problem için önerilen veri tabanı tasarımı Şekil 8.5 ile sunulmuştur. Burada, Kisi varlığı içerisindeki .. ifadesi, kişisel bilgilerle ilgili niteliklerin burada yer alabileceğini ya da yeni bir tablo ile bu tabloya bağlayabileceğini ifade etmek için eklenmiştir. Kendi çözümünüzde bu iki yaklaşımdan birini uygulayabilirsiniz. Bunların yanında problemde belirtilmeyen ama bulunmasının gerekli olduğu tahmin edilebilecek bazı nitelikler de çözüme dahil edilmiştir.

8.6. Elektronik İletişim Yazılımı

8.6.1. Problem

Kişisel verilerin yabancı ülkelere gitmesini engellemek üzerine bir ulusal anlık mesajlaşma uygulaması geliştirme ekibi kuruldu ve bu ekipte veri tabanı uzmanı olarak yer almaktasınız. Geliştirilecek sisteme ait ayrıntılar, aşağıda maddeler olarak sunulmuştur.

1) Her bir kullanıcı için telefon numarası yazılımdaki kimlik numarası olarak kullanılmalı.

2) İki kişi arasında özel mesajlaşma yapılabilecek.

3) İkiden fazla kişiyi içeren mesajlaşma grupları kurulabilecek.

4) Mesajlar; metin, görsel, video ya da ses kaydı biçimlerinde hazırlanabilir.

5) Mesajlar kullanıcılar tarafından silinse bile veri tabanında tutulmaya devam edilmeli.

Bu sistem için uygun bir veri tabanı tasarımı hazırlayınız.

8.6.2. Çözüm Önerisi

Oldukça popüler olan bu uygulama türü için elbette biz de bir veri tabanı tasarımı önerisi getirebiliriz. Problem başlığı altındaki kelimeleri incelediğimizde olgular için şu önerileri sunabiliriz:

Kullanici, Telefon Numarası, Kimlik Numarası, Mesaj, Özel

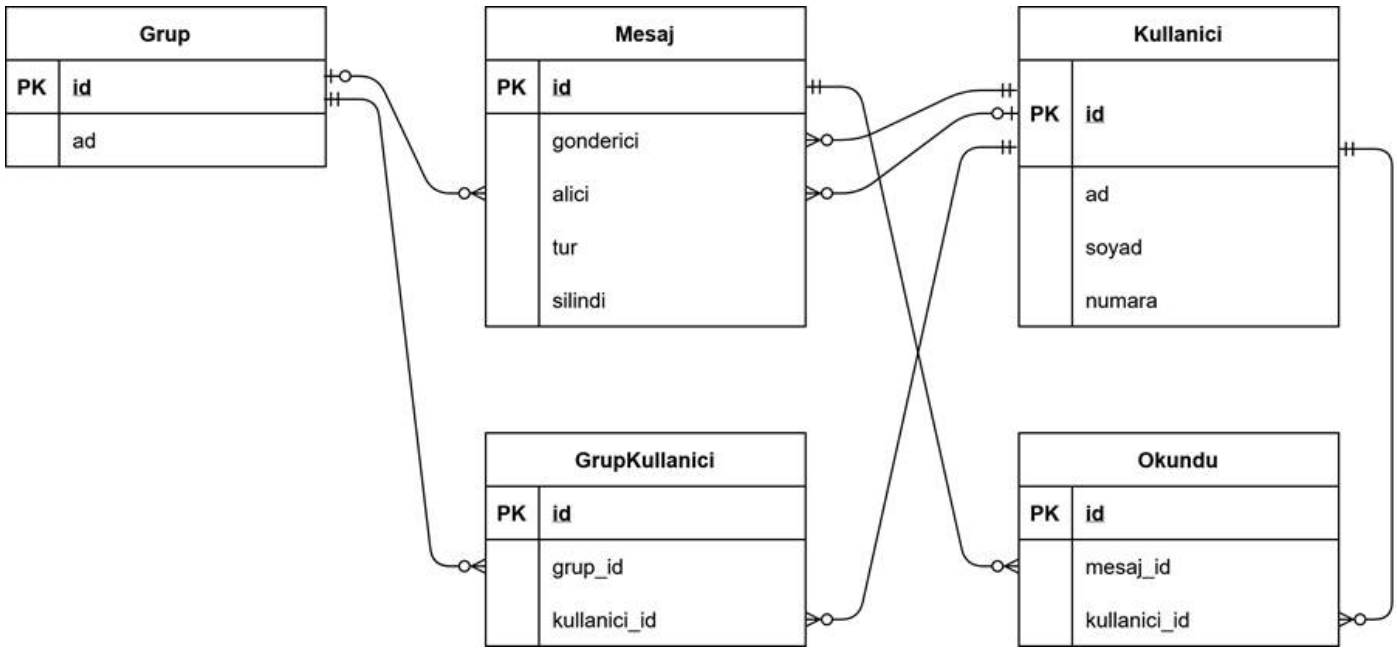
Mesajlaşma, Mesajlaşma Grubu, Metin, Görsel, Video, Ses

Sistemin işlevlerini tanımlayabilmek için ise aşağıda verilen, problem başlığı incelenerek çıkartılmış kelimeler fayda sağlayabilir.

Mesaj Gönderme, Silme, Okundu

Burada Kullanici varlığını hiç tartışmadan, doğrudan tanımlayabiliriz. Bununla birlikte numara niteliğini de bu varlığa eklemek gerekecektir. Kimlik Numarası ile kastedilen zaten bu telefon numarasının kullanıcı için ayırt edici olarak kullanılmasını sağlamaktı. Bu normalde birincil anahtar olarak kullanmak anlamına gelse de biz varlıkları tanımlarken her türlü riskten kaçınmak adına yalnızca id birincil anahtarını kullanacağımızdan bahsetmiştik. Telefon numarasının eşsiz bir kimlik bilgisi olarak tutulabilmesi yazılım katmanında sağlanabilecek bir durumdur. Bir mesajlaşma uygulaması için Mesaj adlı bir varlığı elbette tanımlanmalıdır. Bu varlık, sistem üzerinde gönderilen tüm mesajları saklayabilecek olan bir kayıt varlığı olarak da görülebilir. Özel mesajlaşma, mesajın iki kişi arasında gerçekleştirilebileceğini gösteren bir ifadedir. Tasarımımızı yapılandırırken bu ayrıntıya dikkat etmemiz gerekecektir. Mesajlaşma grubu özelliği için öncelikle grupların saklanması, sonrasında ise hangi kullanıcıların hangi gruplarda yer aldığının bilgisini saklamak üzerine birer varlığa ihtiyacımız bulunmaktadır. Bu sebeple Grup ve GrupKullanici varlıklarını oluşturabiliriz. Burada Grup varlığının oluşturulması kararı önemlidir. Ancak katılımcılarla ilgili ikinci tabloyu gözden kaçırmış olabilirsiniz. Eğer böyle olduysa bile tasarımı gerçekleştirirken Kullanici ve Grup arasında kurmak isteyeceğiniz ilişkinin çoğa çok olduğunu farkedecek, bir bağlantı varlığına ihtiyaç duyacaktınız. Bu bağlantı varlığı grup katılımı bilgisini içerecek olan varlıktır. Dolayısıyla ilk adımda kaçırılması muhtemel olsa bile tasarım tamamlanmadan eksikliği farkedilecek türden bir varlıktır. Metin, görsel, video ve ses ifadeleri mesajın türünü göstermektedir. Dolayısıyla Mesaj varlığına eklenecek bir tur niteliği bu bilginin saklanabilmesini sağlayacaktır.

İşlevlere geldiğimizde mesaj gönderme Mesaj varlığına kayıt eklendiği anda gerçekleştirilmiş olacak bir işlemdir. Dolayısıyla Mesaj varlığı bizim için hem bir olgu hem de kayıt tablosu olarak kullanılabilecek niteliktedir. Mesajların silinme bilgisi, eğer biz mesajı gerçekten silmek istemiyorsak, ilgili kayda ait bir silindi niteliği kullanılmasıyla tutulabilir. Bu sebeple Mesaj varlığına bir silindi niteliği eklemek bizim için iyi bir çözümdür. Okundu bilgisi için ise ele almamız gereken iki olası durum var. Birincisi özel (bire bir) mesajlaşma. Mesajlaşmanın sadece iki kişi arasında olduğu durumda, mesaj gönderici tarafından zaten okundu kabul edilebileceği için yalnızca alıcının okuyup okumadığı bilgisini kayıt altına almak yeterlidir. Ancak bir grup mesajlaşmasında mesajın okunma bilgisi çok kişi için tutulmalıdır. Bu da Mesaj ile Kullanici varlıkları arasında çoğa çok bağlantıya sebep olacak, dolayısıyla bir bağlantı varlığı gerekecektir. Okundu adı verebileceğimiz bu bağlantı varlığı sayesinde tüm kullanıcıların tüm mesajlar için okuma bilgisini kayıt altına alabiliriz. Bu yorumlara göre tasarım önerisi Şekil 8.6 ile sunulmuştur.



Burada Mesaj varlığı üzerinde biraz durmak gerektiğini düşünüyorum. Sistemin en önemli parçası olmakla birlikte sistemin kullanıcılar arasında ya da grup mesajlaşmasına müsade etmesini sağlamak; bu varlık üzerinde bir özel durumun oluşmasına sebep oldu. Mesaj.gonderici her zaman bir kullanıcıyı gösteren ikincil anahtardır. Çünkü bir mesaj her zaman bir kullanıcı tarafından gönderilir. Ancak alıcı, bir kişi ya da grup olabilir. Bu sebeple Mesaj.alici ikincil anahtarı hem Grup.id birincil anahtarı, hem de Kullanici.id birincil anahtarıyla ilişkili gözükmektedir. Fark ettiyseniz bu iki ilişkide birincil anahtara yapılan bağlantıda zorunluluk bulunmamaktadır. Bunun sebebi ikincil anahtarın, birincil anahtarlardan birine gidecek olmasıdır. Dolayısıyla aynı anda iki tarafa da bağlanmayacağı için iki bağlantının da oluşmama olasılığı vardır.

Peki sistem kurgusunda bu ikincil anahtarın Grup ya da Kullanici varlığına bağlanacağını nasıl anlayabiliriz? Mesaj.tur niteliğinin amacı bunu belirlemektir. Bu nitelik kişisel ya da benzeri bir değer aldığında bağlantının Kullanici varlığıyla yapılması gerektiğini; grup değerini aldığında ise Grup varlığıyla yapılması gerektiğini gösterebilir. Bu durum, pek de sık karşılaşılan bir durum olmamakla birlikte pek deneyimli olmayan tasarımcıların genellikle birden fazla Mesaj varlığıyla çözmeye çalıştığı bir problemdir. Ancak önerimizde gördüğünüz üzere tek bir Mesaj varlığı tüm sistem mesajlarını yönetmek için yeterlidir. Ayrıca nesne tabanlı yaklaşım açısından bakıldığında da doğrusu budur.

Sistem önerisinde bir mesajın okunma bilgisini tüm kullanıcılar için tuttuğumuz bir yapı önerdik. Peki sadece alıcının okuma bilgisini kaydetmek isteseydik?

Okundu varlığını tamamen ortadan kaldırarak, özel mesajlaşma durumunda alıcının mesajı okuyup okumadığını kayıt altına alacak; grup mesajlaşması için herhangi bir okundu bilgisi tutmayacak bir değişiklik önerisinde bulunun.

Bu seviyeye gelmiş bir okuyucunun bu problemi rahatlıkla çözeceğinden emin olduğum için son atölyenin yanıtını bulmayı sizlere bırakıyorum.

8.7. Ses Tabanlı Sosyal Medya Yazılımı

8.7.1. Problem

Bir girişim (start-up) ekibinde yer alıyorsunuz. Girişimin konusu yeni bir sosyal medya platformu. Bu platformda tüm kullanıcı gönderileri birer ses kaydı olmalı. Bu yeni girişimde veri tabanı tasarımcısı olarak yer almaktasınız. Geliştirilecek sosyal medya platformuna ait ayrıntılar aşağıda verilmiştir.

1) Platform kullanıcıları ses kaydı içeren iletiler paylaşabilirler. İletiler metin içerik alamazlar. Yalnızca ses kaydı içermelidirler.

2) Gönderiler altına metin ya da ses içeren yorumlar yapılabilir.

3) Gönderiler için beğenme ya da beğenmeme işareti koyulabilir.

Belirtilen özelliklere sahip bir platform geliştirmek için gerekli veri tabanı tasarımını gerçekleştirin.

8.7.2. Çözüm Önerisi

Popülerliği ve sayıca çokluğu sebebiyle böyle bir girişim ve bunun için hazırlanacak yazılım projesinde yer alma şansınız görece daha fazladır. Girişim projeleri için yazılım geliştirmek konusundaki en büyük risk, genellikle daha önce yapılmamış işler denendiği için olası aksiliklerin öngörülmesinin zor olmasıdır. Önünüzde daha önce aynı amaçla geliştirilmiş bir proje olduğunda bu projeyi örnek alarak geliştirme yapmak süreci hızlandıracaktır.

Ancak tüm kuralları siz koyuyorsanız ve bazı yanıtlanması gereken sorular henüz yanıta sahip değilse geliştirme sürecinde öngörülemez zorluklar yaşanması muhtemeldir. Ele aldığımız girişimcilik örneği için kendi yöntemimizi kullanarak kelime analizimize başlayalım. Problem başlığı altında gördüğümüz ad özelliğindeki kelimelerden seçtiklerimiz aşağıda verilmiştir.

Kullanıcı, ses kaydı, ileti, metin, ses

İşlevsel özellikleri belirlemek adına seçtiğimiz fiil özellikli kelimeler de şu şekilde seçilmiştir:

Yorum yapmak, beğenmek, beğenmemek

Sistem içerisinde kullanıcıların kaydının tutulması gerekliliği kolayca kabul görecektir diye düşünüyorum. Bu sebeple ilk kelimemiz olan Kullanici, ilk varlığımız olarak değerlendirilebilir. Ses kaydı ifadesi, bu platformda gönderilecek iletilerin türü olarak karşımıza çıkmaktadır. Bu sebeple Ileti adlı bir varlık tanımlamalı ve bu iletinin bir ses kaydı içerdiğini unutmamalıyız. Böylece Ileti varlığını şu şekilde tanımlayabiliriz:

Ileti: id, dosyaAdi, timestamp

Ses kaydı olarak bahsedilen içerik, bir multimedya içeriğidir ve ayrıca bir dosya olarak saklanmaktadır. Bu tarz dosyaların veri tabanı içerisinde saklanması imkansız olmasa da pratikte neredeyse hiç tercih edilmeyen bir yöntemdir. Metin içeriğe göre devasa sayılabilecek boyuttaki dosyaları olduğu gibi bir veri tabanı içerisine yerleştirmek, bir süre sonra veri tabanını yönetmeyi oldukça zorlaştıracaktır. Hantallaşma, beraberinde yedekleme, işleme ve diğer bazı sorunları da getirebilecektir. Bu sebeple multimedya dosyaları genellikle bir sunucu üzerine kaydedilir ve bu dosyaya erişim yolu veri tabanı içerisinde saklanır. Ileti.dosyaAdi niteliği, bu ileti için oluşturulmuş olan ses kaydı dosyasının sunucu üzerindeki yolunu gösteren metinsel bir ifadedir. Böylece ses kaydı dolaylı olarak veri tabanında saklanmış olur ancak dosyanın kendisini taşımanın getireceği yükten uzak kalmak mümkün hale gelir.

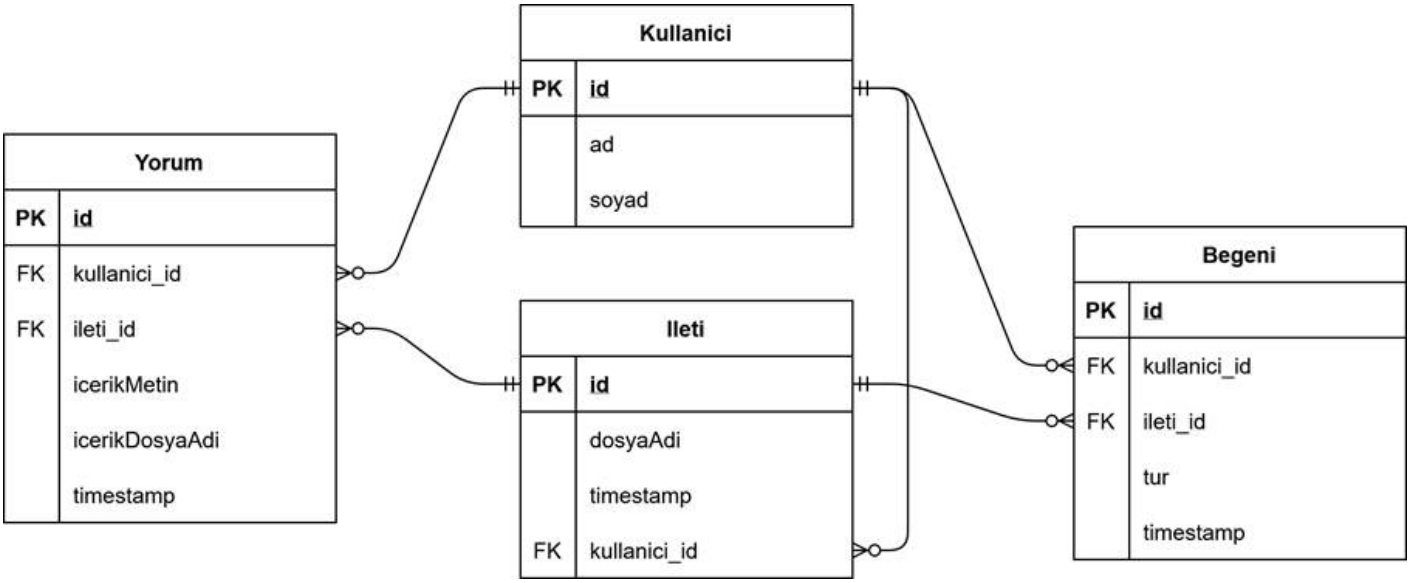
Metin ve ses ifadeleri, yorum için kullanılan kelimelerdi. Dolayısıyla onları yorumlarla ilişkilendirmek üzere bekletelim bir süre. Fiillere geçtiğimizde ilk kelimemiz yorum. İletilere yapılacak yorumların mutlaka bir varlık içerisinde tutulması gerekiyor. Bu işlevsel bir tablo olduğu için hangi ilişkiden doğacağını da ele almak gerekir. Bir yorum, bir ileti için bir kullanıcı tarafından yapılmalıdır. Dolayısıyla Yorum varlığı, Kullanici ve Ileti varlıklarının çoğa çok bağlantısının sonucunda oluşacak bir bağlantı varlığı olarak da görülebilir. Az önce beklemeye aldığımız nitelikleri de işin içine katarsak Yorum varlığımızı şu şekilde kurgulayabiliriz:

Yorum: id, icerikMetin, icerikDosyaAdi, timestamp

Yorum varlığını bu şekilde kurgulamak, yorumların hem metin içerikli hem de ses kaydı eklenebilir olmasını sağlayacaktır. icerikMetin niteliği metin içeren yanıtı, icerikDosyaAdi ise ilgili ses kaydı dosyasının sunucu üzerindeki yolunu saklayacak olan niteliktir.

Son olarak beğenme ve beğenmeme durumlarını ele alalım. Begeni adlı tek bir varlıkla hem beğenme hem de beğenmeme durumlarını kayıt altında tutabiliriz. Bu varlık da Yorum varlığına benzer şekilde Kullanici ve Ileti varlıklarının çoğa çok bağlantısının bağlantı varlığı olarak görülebilir. Begeni varlığına ekleyeceğimiz

bir tur niteliği, etkileşimin beğenmek ya da beğenmemek olduğu bilgisini tutabilir. Tüm bu değerlendirmeler neticesinde oluşturduğumuz veri tabanı tasarımımız Şekil 8.7'de sunulmuştur.



Burada Kullanici ve Ileti adlı iki temel varlıktan söz edebiliriz. Yorum ve Begeni varlıkları, Kullanici ve Ileti varlıklarının çoğa çok bağlantılı olması neticesinde ortaya çıkan bağlantı varlıkları olarak da değerlendirilebilirler. Daha önce de değindiğimiz üzere, iki varlık birden fazla kez çoğa çok bağlantılı olabilir. Çoğa çok bağlantı genellikle bu iki varlık arasındaki bir işlevi konu almaktadır. Burada da kullanıcılar ile iletilerin iki farklı etkileşim durumu iki bağlantı varlığıyla kayıt altına alınmıştır.

Burada bir ayrıntı üzerinde duralım. Eğer iki varlık arasındaki ilişkiyi arıyor ve çoğa çok ilişki ile karşılaştıysanız, bu iki varlığın hangi işlev ile bu ilişkiyi kurduğu üzerinde biraz durun. Sistemde yer alması gereken ancak öngöremediğiniz bir varlık, karşılaştığınız ilişki için oluşturacağınız bağlantı varlığı olabilir.

8.8. Video Paylaşım Platformu

8.8.1. Problem

Bir video içerik yayınlama platformunun geliştirme ekibinde yer alıyorsunuz. Yönetim tasarından sistemin yenilenmesine karar verildi ve yeniden yazılacak bu sistem için veri tabanı tasarımcısı olarak ekipte yer almaktasınız. Kullanıcıların ücretsiz ve sınırsız şekilde video paylaşmalarına olanak sağlayan bu sistemin mevcut çalışma yapısına ait ayrıntılarıyla birlikte geliştirilecek yeni özelliklere ait bilgiler birleştirilerek liste olarak, aşağıda verildiği şekliyle size sunulmuştur.

1) Kullanıcılar bir ya da daha fazla içerik kanalı oluşturabilir. Kanallar, kişilerin paylaştığı videoları içeren özel sayfalardır.

2) Kullanıcılar kanalları takibe alabilirler.

3) Yalnızca kanal sahibi kullanıcı, kanala içerik yükleyebilir.

4) Tüm içerik izlemeler kayıt altına alınmalıdır.

5) Kullanıcılar videolara beğenme ya da beğenmeme işareti koyabilirler.

6) Videoların toplam izlenme sayıları kayıt altında olmalıdır.

7) Tüm video içerikler bir video dosyasına, başlığa ve açıklamaya sahiptirler.

Verilen ayrıntılar ışığında bir veri tabanı tasarımı hazırlayınız.

8.8.2. Çözüm Önerisi

Talep edilen veri tabanı tasarımı için problem başlığı altındaki ayrıntıları inceleyerek kelime analizimizi gerçekleştirdiğimizde elde ettiğimiz kelimeler, aşağıda listelediğimiz şekildedir. Önceki örneklerden farklı olarak burada fiil özelliğine sahip yani sistem için bir işlev belirten kelimeleri ayrıca listelemeyip, aşağıdaki liste içerisinde kelime başında bir * işareti kullanarak vurguladık.

Kullanıcı, içerik, kanal, video, *paylaşmak, *takip, sahip,

*içerik yükleme, *izleme, *beğenme, *beğenmeme, izleme sayısı, dosya, başlık, açıklama

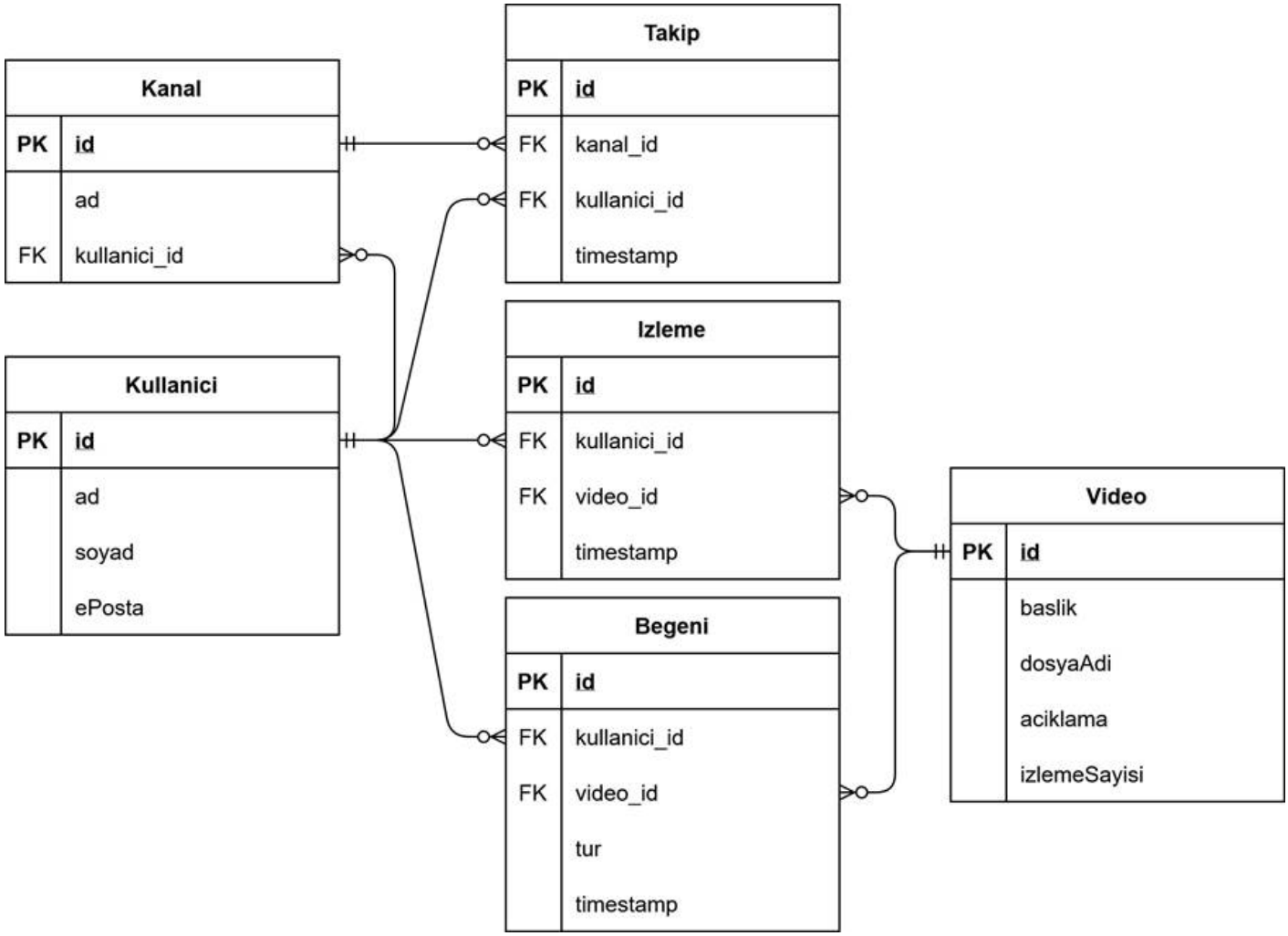
Burada kullanıcı ve sahip kelimelerini tek bir Kullanici varlığı tanımlayarak ifade edebileceğimiz konusunda kolayca hemfikir olacağımızı düşünüyorum. İçerik kelimesiyle kastedilen video içerikler olduğu için bu iki kelimeyi birlikte ele alarak Video varlığının tanımlanabileceğini de söyleyebiliriz. Bu videoların bazı kanallara yükleneceğinden bahsedilmektedir. Kolayca bunu da bir Kanal varlığı oluşturarak ele alabiliriz. İzleme sayısı, dosya, başlık ve açıklama kelimelerinin paylaşılan video içeriğe ait nitelikler olduğunu kabul edebiliriz. Böylece sistemimizde yer alan olgulara ait varlıkları tanımlamış olmaktayız.

İşlev anlamı taşıyan kelimeleri ele aldığımızda bazı kelimelerin yazılım katmanında ele alınması gereken kelimeler olduğunu görmekteyiz. Örnek olarak paylaşmak ve içerik yükleme ifadeleri yazılım ve dolayısıyla arayüz katmanında ele alınması gereken sistem özelliklerindendir. Bunlarla ilgili veri tabanında gerçekleştireceğimiz herhangi bir kayıt işlemi bulunmamaktadır. Takip kelimesi kullanıcıların kanalları takip etmesi özelliği sebebiyle kullanılmıştı. Bu kelime gerçekten de kullanıcının gerçekleştireceği takip işlevini temsil etmekle birlikte Kullanici ve Kanal varlıkları arasındaki çoğa çok bağlantı için kullanılabilecek bir bağlantı varlığı olma potansiyeli de taşımaktadır.

İzleme işlevi, bir kullanıcı ile bir video etkileşimiyle oluşmaktadır. Kullanıcıların videoları izlemesi ya da videoların kullanıcılar tarafından izlenmesi olarak düşünüldüğünde burada da çoğa çok bağlantı ve bu bağlantıyı sağlayan bir bağlantı varlığından bahsedilebilir. Elbette burada bağlantı varlığı İzleme olarak tanımlanmalıdır.

Beğenme ve beğenmeme durumları iki benzer işlevin türünün farklı olması olarak düşünülebilir. Beğenme ya da beğenmeme durumu aynı zamanda bir içeriğe pozitif ya da negatif puan vermekle teknik olarak aynı durumdur. Bu sebeple iki işlevi ayrı varlıklar yapmak yerine tek bir Begeni varlığı olarak ele alabiliriz. Bu varlık kullanıcılar ile videoların bir başka etkileşimi olarak görülebilir.

Tüm bu yorumlar neticesinde elde edeceğimiz veri tabanı tasarımı Şekil 8.8 ile sunulmuştur.



Burada Kullanici ve Kanal varlıkları arasındaki sahiplik bağlantısı dışında tüm bağlantıların çoğa çok olduğunu dikkatinize sunarım. Kullanici ve Video varlıklarının iki kez, Kullanici ve Kanal varlığının bir kez çoğa çok bağlantı kurması neticesinde 3 tane bağlantı, dolayısıyla işlev varlığı tanımlanmıştır. Sistem işlevlerini içeren bu varlıkların, olgu varlıkları olmasa da sistemin ana işlevlerini üstlenen, çok önemli varlıklar olduklarını görmekteyiz. Bu da çoğa çok bağlantı için mecburen tanımlanan bağlantı varlıklarının aslında önemli rollere sahip; sistemin işleyişi açısından önem arz eden varlıklar olduklarını göstermektedir.

8.9. Bulut Tabanlı Dosya Barındırma Yazılımı

8.9.1. Problem

Kullanıcıların dosyaları arşivleyebilmesini sağlayacak bir bulut hizmetin geliştirilmesinde ekip üyesi olduğunuzu farzedin. Sizden, aşağıda özellikleri verilmiş olan sistem için bir veri tabanı tasarımı gerçekleştirmeniz beklenmektedir.

1) Sisteme kaydolan kullanıcılar, farklı kapasite seçenekleri içeren paketlerden birini satın almak zorundadır.

2) Her kullanıcı, satın aldığı paket doğrultusunda bir kapasite kullanma hakkına sahiptir.

3) Kullanıcılar, kendi arşivleri içerisinde istedikleri sayıda ve kırılımda klasör açabilir, istedikleri türde dosyalar yerleştirebilir.

4) Kullanıcı tarafından silinen dosyalar doğrudan silinmez, çöp kutusuna taşınırlar.

5) Çöp kutusuna taşınan dosyalar 30 gün sonra geri çekilmedikleri takdirde tamamen silinirler.

6) Dosyalar paylaşıma açılabilirler. Paylaşım ve paylaşılan dosyalara erişim, e-posta adresleri kullanılarak gerçekleştirilir.

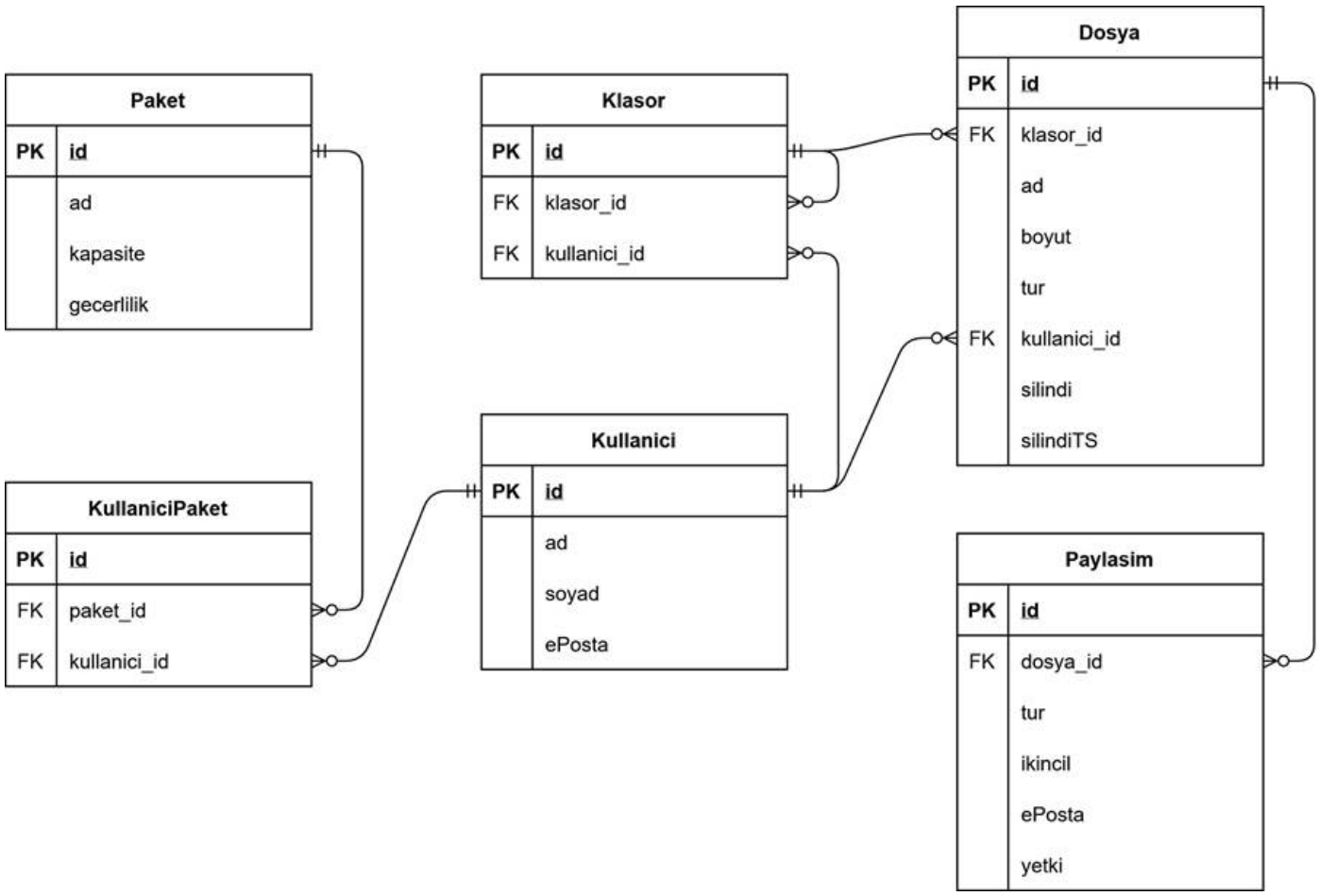
Bu şartlara uyan bir veri tabanı tasarımı gerçekleştiriniz.

8.9.2. Çözüm Önerisi

Ele alınan problem için kelime analizini gerçekleştirdiğimizde aşağıdaki kelimeleri elde etmekteyiz. İşlev niteliğinde olabilecek kelimeler, kelimenin başına eklenen bir * işareti ile vurgulanmıştır.

Kullanıcı, dosya, arşiv, kapasite, paket, *satın almak, klasör, tür, *silmek, çöp kutusu, *taşımak, *geri döndürmek, *paylaşmak, e-posta

Sistem kullanıcılarının kaydedilmesi için Kullanici adlı varlık, tanımlayacağımız ilk varlıktır. Dosya ve klasörlerle ilgili yapı daha önce özel durumlar altında örneği verilmiş bir yapı idi. Klasör yapılandırması sonsuz sayıda kırılıma sahip olabilecek bir hiyerarşi sunduğu için farklı seviyedeki kategorilerin farklı varlıklar ile sunulması etkili bir çözüm değildir. Bu sebeple klasörlerin kayıt altına alınacağı varlık kendi kendisiyle bire çok bağlantı kurarak sınırsız sayıda alt kırılıma sahip olabilir. Bu yapı Klasor adlı varlık ile kayıt altına alınacaktır. Dosyalar ise klasörlere bağlı olarak kayıt altına alınması gerektiği için Dosya adlı bir varlık ile kayıt altına alınacaktır. Arşiv kelimesi yine sistemin tamamını niteleyen, bir kullanıcının klasör ve dosyaları bütününe verilen bir ad olduğu için varlık olarak tanımlama ihtiyacı oluşturmayan bir kelimedir. Kapasite ve paket kelimeleri, kullanıcıların bu sistemde faydalanabilme limitlerini göstermektedir. Bir Paket varlığı kullanarak kullanılabilecek kapasite türleri gruplandırılabilir. Kapasite, Paket varlığının bir niteliği olmalıdır. Ayrıca kullanıcılar zaman içerisinde farklı paketlere abone olabileceği için Paket ile Kullanici varlıkları arasında çoğa çok bağlantı olduğunu belirtmek gerekir. Bu ilişkideki bağlantı varlığı KullaniciPaket olarak tanımlanabileceği gibi işlevsel bir ad olarak Abonelik de tercih edilebilir. Aynı zamanda kelime listemizdeki satın alma işlevi de bu bağlantı varlığı ile karşılanabilmektedir. Tür ifadesi bir olgunun özelliği olduğu için ilgili varlık altına nitelik olarak kolaylıkla eklenebilir. Bir dosyanın silinip silinmeme bilgisi eklenecek bir nitelik ile kayıt altına alınabilir. silindi adlı bir niteliğin 1 değeri alması neticesinde ilgili dosyanın silinip, çöp kutusuna gönderildiği düşünülebilir. Böylece bir dosyayı çöp kutusuna göndermek için yapılması gereken sadece Dosya.silindi = 1 tanımlaması yapmaktır. Ayrıca bir süre geçtikten sonra çöp kutusundaki dosyanın tamamen silinmesi için dosyanın çöp kutusuna ne zaman gönderildiğinin de bilinmesi gerekmektedir. Bunun için ise silindiTS adlı bir nitelik yardımıyla dosyanın silinme zamanına ait timestamp değeri kayıt altına alınabilir. Böylece yazılım katmanında, tanımlı kurtarma süresi dolan dosyalar otomatik olarak sistemden tamamen silinebilirler. Dosyaların paylaşılabilmesi, doğal olarak dosyaların bir niteliği olarak görülebilir. Ancak bir dosyanın birden fazla kez paylaşılabilmesi mümkün olabileceği için paylaşım işlevinin Dosya varlığı ile bire çok ilişkili olması gerekmektedir. Paylaşım bilgisinin kayıt altına alınabilmesi için Paylasim adlı bir varlık tanımlamamız uygun olacaktır. Bu ayrıntılı açıklamalar neticesinde önerdiğimiz veri tabanı tasarımı Şekil 8.9 ile sunulmuştur.



Tasarım, üç önemli bileşenden oluşmaktadır. Birincisi Klasor-Dosya yapılanmasının kayıt altına alınmasıdır. Bu özel bir durumdur ve dinamik bir kayıt gerçekleştirilebilmesi için kullanılması gereken yapı şekildeki gibidir. Özellikle Klasor varlığının kendi kendisiyle yaptığı bire çok ilişkiyi inceleyiniz. İkinci önemli bileşen üyeliklerin paketlere abone olmasıdır. Abonelik bilgisi KullaniciPaket varlığı ile kayıt altına alınmaktadır.

Üçüncü ve son bileşen ise dosyaların paylaşıma açılabilmesidir. Bu özellik Paylasim varlığı ile kullanılabilmektedir. Paylasim varlığının adı ve yapısı gereği bir bağlantı varlığına benzediği dikkatinizi çekmiştir. Aslında bu yapıda da bağlantı varlığı olarak değerlendirilebilir. Dosyaların, paylaşılan kişiler ile ilişkisini sağlayan bir işleve sahiptir. Ancak dosyaların paylaşıldığı kişiler bu sistemde kayıt altında olmadığı; paylaşımın e-posta adresleri ile sağlandığı için Paylasim varlığının paylaşılan kişiyle ilgili bir varlık ilişkisi bulunmamaktadır. Bunun yerine Paylasim.ePosta niteliğinin mantıksal olarak bir ikincil anahtar özelliği taşıdığını söyleyebiliriz. Eğer paylaşılan kişilerin de sisteme kayıtlı olma zorunluluğundan bahsetseydik bu kişileri de Kullanici varlığı içerisine kaydedecektik ve Paylasim varlığı Dosya-Kullanici varlıklarının çoğa çok ilişkisinin bağlantı varlığı haline gelecekti.

8.10. Satranç Turnuvaları Yazılımı

8.10.1 Problem

Ülkenizin ulusal satranç fedarasyonunda bilişim bölümünde görev yapmaktasınız. Yönetim, tüm satranç turnuvalarının, bu maçlarda oynanan oyunların ve lisanslı oyuncuların kayıtlarını tutabilecekleri bir bilgi sistemi geliştirmek istiyor. Sistem, aşağıda listelenmiş özelliklere sahip olmalı. Sizden, veri tabanı tasarımı gerçekleştirme konusunda görev almanız isteniyor.

1) Satranç maçları iki taraftan birinin kazandığı ya da berabere kalındığı maçlardır. Kazanç durumu 1 puan, beraberlik 1/2, kayıp 0 puan getirmektedir.

2) Her bir oyuncu elo adı verilen bir oyuncu skoruna bağlıdır. Bu skor, oyuncunun güncel durumunu gösterir ve maç sonuçlarına göre güncellenir.

3) Maçlar puanlı ya da puansız olabilir. Yalnızca puanlı maçlar oyuncuların elo skorunun değişmesine sebep olur.

4) Oyuncuların bir turnuvaya ön kayıt yaptığı bilgisi kayıt altında olmalıdır.

5) Oyuncuların maç sonuçları turnuvaya bağlı olarak kayıt altında olmalıdır.

Bu şartlara uygun veri tabanı tasarımını gerçekleştiriniz.

8.10.2 Çözüm Önerisi

Varlıkların ve sistem işlevlerinin belirlenebilmesi için problem başlığında verilen ayrıntıları analiz ettiğimizde listelenen kelimeleri elde edebiliriz:

Turnuva, *maç, oyuncu, puan, elo, *güncellemek, puanlı,

puansız, *ön kayıt, maç sonuçları

Problem tarifi içerisinde turnuvalar düzenleneceği ve bu turnuvalar kapsamında maçlar gerçekleştirileceği açıkça ifade edilmektedir. Bu sebeple Turnuva ve Mac varlıklarını tanımlamamız gerektiğinden bahsedebiliriz.

Oyuncuları ve gerekli olması durumunda sisteme dahil olabilecek diğer kişileri de kaydetmek üzere Kisi adlı bir varlık tanımlayabiliriz. Burada puan ve diğer adıyla elo ifadesi oyuncunun bir özelliği olduğu için Kisi varlığı içerisinde nitelik olarak tanımlanabilir.

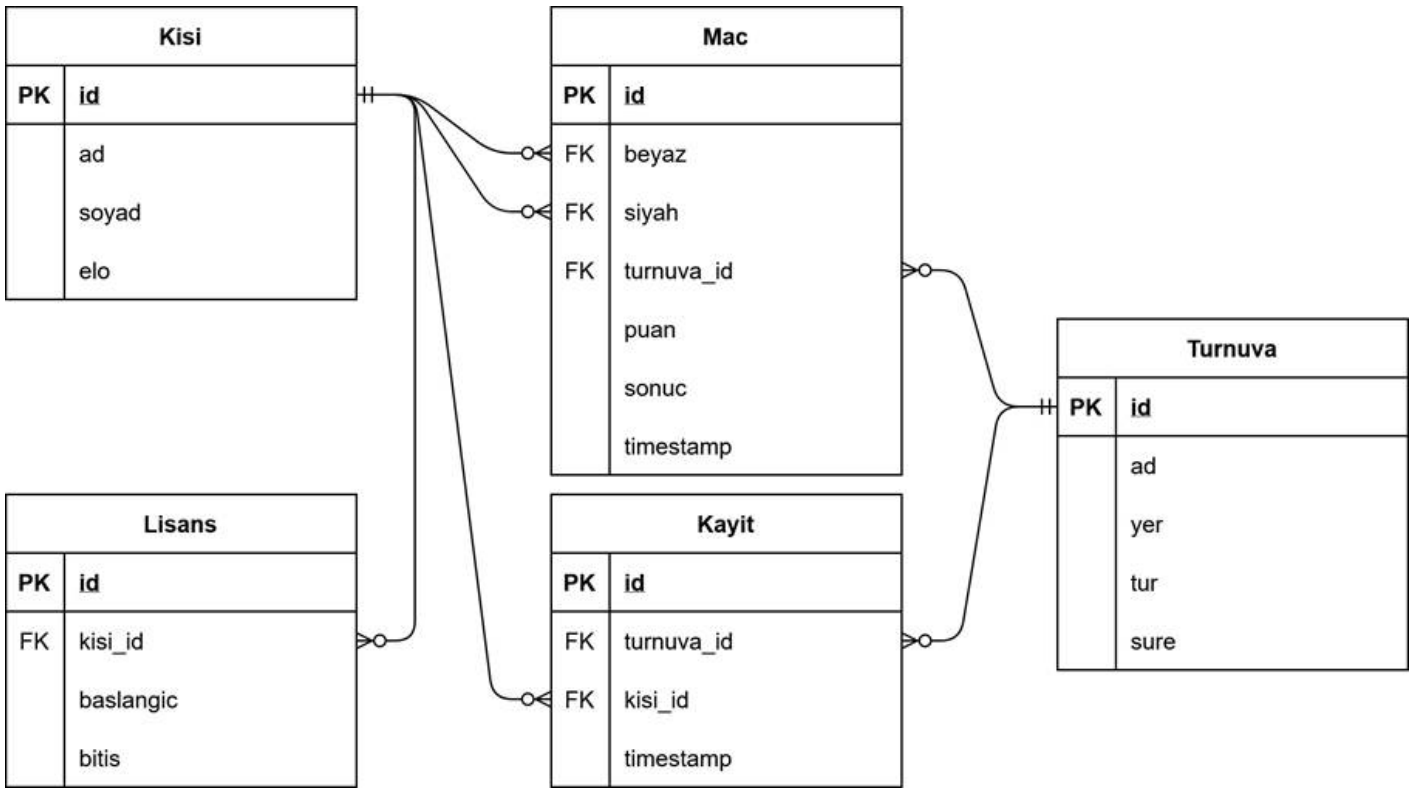
Güncellemek ifadesi, oyuncuların maç sonuçlarına göre puanlarının güncellenmesi anlamına gelmektedir. Bu işlev, yazılım katmanında gerçekleştirilmesi gereken bir işlem olduğu için ve sonucu Kisi.elo niteliği içine kaydedilebileceği için ek bir müdahale gerekmemektedir.

Puanlı ve puansız ifadeleri gerçekleştirilen maçın özelliği olmakla birlikte aslında turnuva şartları ile tanımlanan bir kuraldır. Dolayısıyla bu durum Turnuva varlığı içerisinde tur adlı bir nitelik ile kayıt altına alınabilir. Bu bilginin Mac varlığı altında tutulması veri bütünlüğü açısından bir fark yaratmazdı. Ancak bir turnuvaya bağlı tüm maç kayıtlarında tur niteliğinin aynı değeri alması beklenirdi. Bu da aynı değerin maç sayısı kadar tekrar etmesi anlamına gelecektir. İlgili kaydın Turnuva varlığı altında tutulması bu veri tekrarı problemini ortadan kaldıracaktır.

Ön kayıt işlemleri, Turnuva varlığı ile Kisi varlığı arasındaki çoğa çok ilişki sonucunda oluşan bir bağlantı varlığı olarak ele alınabilir. Bir turnuvaya birçok kişi ön kayıt yaptırabileceği gibi bir kişi birçok farklı turnuvaya kaydolabilir. Bu durumda kayıtları içeren varlık, ilişkili olduğu bu iki varlığa bağlı bir varlık olarak karşımıza çıkacaktır.

Maç sonuçları, Mac varlığıyla bire bir ilişkili olduğu için ayrıca bir varlık olarak tutulmasına gerek bulunmamaktadır. Burada Mac ve MacSonuc gibi bire bir ilişkili iki varlık tanımlamanın da hata olmayacağını belirterek önerilen çözümde bu iki varlığın tek bir varlık olarak ele alındığını belirtelim.

Değerlendirmeler neticesinde önerilen veri tabanı tasarımı Şekil ile sunulmuştur.



Bölüm Özeti

Bölüm içerisinde 10 özel probleme yönelik 10 farklı veri tabanı tasarımı gerçekleştirdik. Bunları yaparken aşağıdaki yaklaşımlar bizim için çok önemliydi.

1. İhtiyacı dinlemek: Sistem analizi ve tasarımı aşamasında bu sistemi kullanacak kişileri dikkatle dinlemek ve gerekli soruları belirleyip sormak çok önemlidir. Bu yazılım yokken süreci nasıl işlettiklerini öğrenmek en faydalı yöntemlerden biridir.

2. Kelime analizi yapmak: Aldığınız notlar üzerinde kelimeleri ayıklamak, bunları ad ve fiil olarak ikiye ayırmak iyi bir başlangıç noktası elde etmenizi sağlayacaktır. Adlar tablo adlarını, fiiller ise bağlantı tablolarının adlarını belirlememizde yol göstericidir.

3. İlişki tiplerini doğru belirlemek, veri bütünlüğünü sağlamak açısından oldukça önemlidir.

Bölüm içerisinde aşağıdaki konularda veri tabanı örnekleri sunulmuştur:

1. Diyetisyenler için platform

2. Araç satış portalı

3. Para transferi yazılımı

4. Nakliye firması organizasyon yazılımı

5. Operatör Koordinasyon yazılımı

6. Elektronik iletişim yazılımı

7. Ses tabanlı sosyal medya yazılımı

8. Video paylaşım platformu

9. Bulut tabanlı dosya barındırma yazılımı

10. Satranç turnuvaları yazılımı

Tüm bu örnekler için önce kendiniz bir tasarım üretmeyi deneyip, sonrasında yorumları incelediyseniz bu konuda büyük gelişme kaydettiğinize emin olabiliriz. Artık bilgisayar programcılığında bir adım daha öndesiniz.

Kaynakça

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

Ünite Soruları

Soru-1 :

Bir kütüphane için geliştirilen kitap ödünç alma sistem için aşağıdaki tablolardan hangisinin kullanılması uygun değildir?

(Çoktan Seçmeli)

(A) Kitap

(B) Yazar

(C) Bayi

(D) Kanal

(E) Satis

Cevap-1 :

Kanal

Soru-2 :

Bir otoyol geçiş sistemi için hazırlanan veri tabanı tasarımında aşağıdaki tablolardan hangisinin kullanılması uygun değildir?

(Çoktan Seçmeli)

(A) Arac

(B) Yazilim

(C) Gise

(D) Bolge

(E) Ulke

Cevap-2 :

Yazilim

Soru-3 :

Bir teknik servis yazılımı için hazırlanan veri tabanı tasarımında aşağıdaki tablolardan hangisinin kullanımı uygun değildir?

(Çoktan Seçmeli)

(A) Kisi

(B) Banka

(C) Tarife

(D) ArizaKaydi

(E) Urun

Cevap-3 :

Banka

Soru-4 :

Bir toplu taşıma yazılımı için aşağıdaki tablolardan hangisinin kullanımı uygun değildir?

(Çoktan Seçmeli)

(A) Vergi

(B) Kisi

(C) Arac

(D) Kart

(E) Hat

Cevap-4 :

Vergi

Soru-5 :

Bir şans oyunu uygulaması için aşağıdaki tablolardan hangisinin kullanımı uygun değildir?

(Çoktan Seçmeli)

(A) Oyun

(B) Kisi

(C) Ikramiye

(D) BankaHesabi

(E) Araba

Cevap-5 :

Araba

Soru-6 :

Bir film ve dizi izleme platformu için aşağıdaki tablolardan hangisinin kullanımı uygun değildir?

(Çoktan Seçmeli)

(A) Icerik

(B) Kisi

(C) Bilgisayar

(D) Izleme

(E) Odeme

Cevap-6 :

Bilgisayar

Soru-7 :

Bir hastane bilgi sistemi için aşağıdaki tablolardan hangisinin kullanımı uygun değildir?

(Çoktan Seçmeli)

(A) Personel

(B) Doktor

(C) Hasta

(D) Oda

(E) OtoparkGirisi

Cevap-7 :

OtoparkGirisi

Soru-8 :

Bir online görüşme platformu için aşağıdaki tablolardan hangisinin kullanımı uygun değildir?

(Çoktan Seçmeli)

(A) Araba

(B) Kisi

(C) Gorusme

(D) SatinAlma

(E) Kategori

Cevap-8 :

Kategori

Soru-9 :

Aşağıdakilerden hangisi tablo olarak tanımlamak için en uygundur?

(Çoktan Seçmeli)

(A) Arac

(B) Plaka

(C) UretimYeri

(D) Fiyat

(E) Mensei

Cevap-9 :

Arac

Soru-10 :

Aşağıdakilerden hangisi tablo olarak tanımlamak için en kötü seçenektir?

(Çoktan Seçmeli)

(A) Satis

(B) Kisi

(C) Tarife

(D) KayitTarihi

(E) Kategori

Cevap-10 :

KayitTarihi
