2. VERİ SAKLAMA YÖNTEMLERİ

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

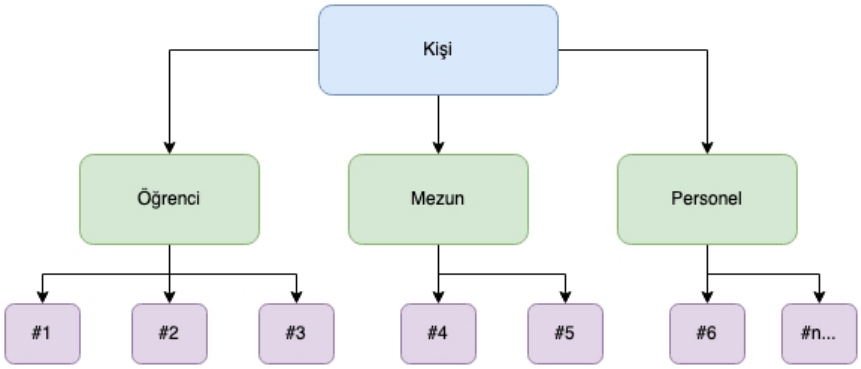
toplama ve kaydetme aşamaları zaman, insan kaynağı ve operasyonel açıdan maliyetli süreçlerdir. Ayrıca süreç tamamlandığında bir eksikliğin fark edilmesi de geri dönülmez bir riski içerisinde barındırmaktadır.

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.

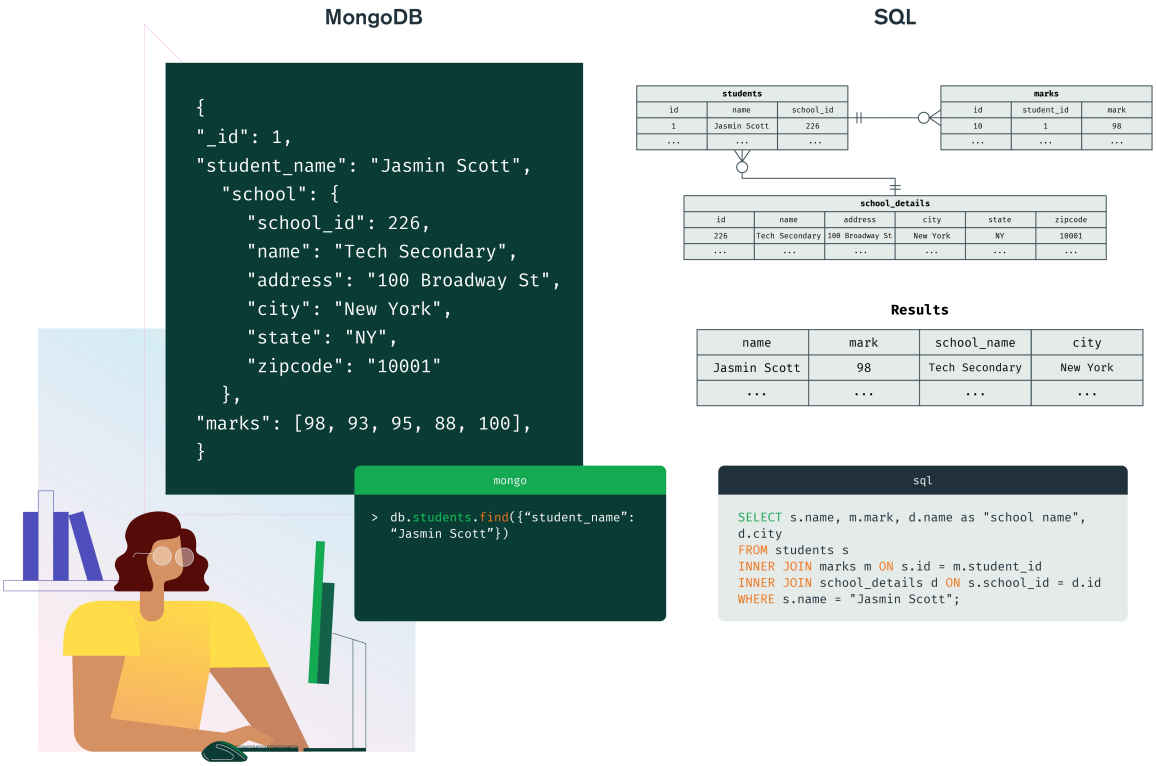
Microsoft SQL Server: İnternet tabanlı uygulamalar için sıklıkla kullanılan iki platform bulunmaktadır. Bunlardan biri Microsoft'a ait olan .NET (dotnet) platformudur. C# programlama dili kullanılan bu platform üzerinde veri tabanı yönetim sistemi olarak Microsoft SQL Server sıklıkla tercih edilmektedir. 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.

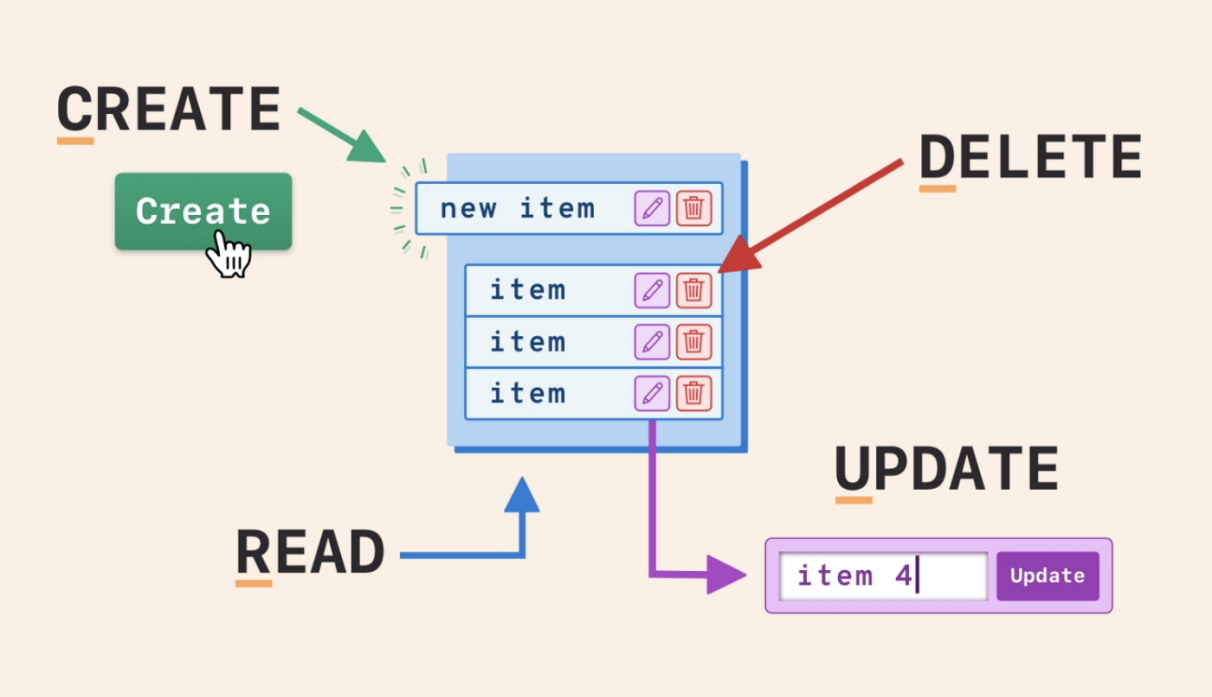
SQL evrensel bir dildir. Hangi veri tabanıyla çalıştığınızda bağımsız olarak sorgulama ve komutlarınızı SQL formatında vermeniz mümkündür. Bazı veri tabanı yönetim sistemleri, SQL üzerinde kendi özelleştirmelerini yapmaktadır. Ancak bu özelleştirmeler, standart SQL komutları üzerindeki değişiklikler 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 oluşturumasından başlayarak, tüm tabloların, değişkenlerin tipleriyle birlikte belirlenebildiği; veri yönetimi sürecinde birçok işlemi gerçekleştirebilmektedir. Bu ders kapsamında SQL'in yalznıca 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