ikinci normal form (2NF)
Normalleştirme sürecindeki ikinci aşama, bir ilişkinin 1NF içinde olduğu ve kısmi bağımlılıklar (bağımlılıkların yalnızca birincil anahtar).

Not

Bir tablo **aşağıdaki** durumlarda **ikinci normal formdadır (2NF)**:

- 1NF içinde *ve* Hiçbir kısmi bağımlılık içermez; hiçbir öznitelik birincil anahtarın yalnızca bir kısmına bağımlı değildir.

2NF'deki bir tablonun geçişli bağımlılık sergilemesi hala mümkündür. , birincil anahtar, birincil olmayan nitelikler arasında işlevsel bir bağımlılıkla gösterildiği gibi, diğer birincil olmayan nitelikleri işlevsel olarak belirlemek için bir veya daha fazla birincil olmayan niteliğe dayanabilir.

Şekil 6.4 hala geçişli bir bağımlılık göstermektedir ve bu da anomalilere yol açabilir. Örneğin, birçok çalışan tarafından sahip olunan bir iş sınıflandırması için saat başına ücret değişirse, bu değişikliğin bu çalışanların *her* biri için yapılması gerekir. Saat başı ücret değişikliğinden etkilenen bazı çalışan kayıtlarını güncellemeyi unutursanız, aynı iş tanımına sahip farklı çalışanlar farklı saatlik ücretler üretecektir.

6-3c Üçüncü Normal Forma (3NF) Dönüştürme

Şekil 6.4'te gösterilen veritabanı organizasyonunun yarattığı veri anomalileri aşağıdaki iki adımın tamamlanmasıyla kolayca ortadan kaldırılabilir:

determinant
Belirli bir değeri doğrudan o satırdaki diğer değerleri belirleyen herhangi bir nitelik.

Adım 1: Geçişli Bağımlılıkları Ortadan Kaldırmak için **Yeni Tablolar Oluşturun** Her geçişli **bağımlılık** için, belirleyicisinin bir kopyasını yeni bir tablo için birincil anahtar olarak yazın. **Belirleyici**, değeri bir satırdaki diğer değerleri belirleyen herhangi bir niteliktir. Üç farklı geçişli bağımlılığınız varsa, üç farklı belirleyiciniz olacaktır. 2NF'ye dönüştürmede olduğu gibi, belirleyicinin yabancı anahtar olarak hizmet etmesi için orijinal tabloda kalması önemlidir. Şekil 6.4'te geçişli bağımlılık içeren yalnızca bir tablo gösterilmektedir. Bu nedenle, bu geçişli bağımlılık için belirleyiciyi şu şekilde yazın: JOB_CLASS

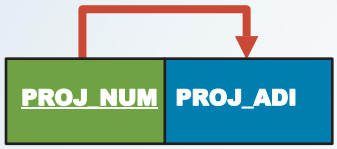
Adım 2: İlgili Bağımlı Öznitelikleri Yeniden Atayın Şekil 6.4'ü kullanarak, Adım 1'de tanımlanan her bir belirleyiciye bağımlı olan öznitelikleri belirleyin. Bağımlı öznitelikleri belirleyicileriyle birlikte yeni tablolara yerleştirin ve orijinal tablolarından kaldırın. Bu örnekte, Şekil 6.4'te gösterilen EMPLOYEE tablosundan CHG_HOUR öğesini kaldırarak EMPLOYEE tablosu bağımlılık tanımını şu şekilde bırakın: EMP_NUM→ EMP_NAME, JOB_CLASS

Adım 1 ve 2'de tanımladığınız tüm tabloları göstermek için yeni bir bağımlılık diyagramı çizin. Tabloyu, içeriğini ve işlevini yansıtacak şekilde adlandırın. Bu durumda, JOB uygun görünmektedir. Her tablonun bir belirleyicisi olduğundan ve hiçbir tablonun uygunsuz bağımlılıklar içermediğinden emin olmak için tüm tabloları kontrol edin. Bu adımları tamamladığınızda, Şekil 6.5'teki sonuçları göreceksiniz.
Başka bir deyişle, 3NF dönüşümü tamamlandıktan sonra, veritabanınız dört tablo içerecektir:

PROJE (**PROJE_SAYISI**, PROJE_ADI)
EMPLOYEE (**EMP_NUM**, EMP_NAME, **JOB_CLASS**) JOB
(**JOB_CLASS**, CHG_HOUR)
ATAMA (**PROJE_NUM, EMP_NUM**, ASSIGN_HOURS)

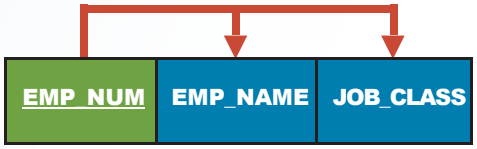
Bu dönüşümün orijinal EMPLOYEE tablosunun geçişli bağımlılığını ortadan kaldırdığına dikkat edin. Tabloların artık üçüncü normal formda (3NF) olduğu söylenmektedir.

Şekil 6.5 Üçüncü Normal Form (3NF) Dönüşüm Sonuçları



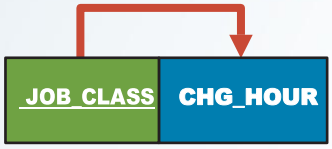
Tablo adı: PROJE

PROJE (PROJE_SAYISI, PROJE_ADI)



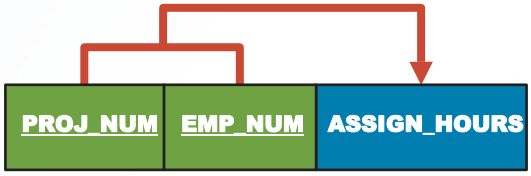
Tablo adı: EMPLOYEE

EMPLOYEE (EMP_NUM, EMP_NAME, JOB_CLASS)



Tablo adı: JOB

JOB (JOB_CLASS, CHG_HOUR)



Tablo adı: ASSIGNMENT

ATAMA (PROJE_NUM, EMP_NUM, ASSIGN_HOURS)

Not

Bir tablo aşağıdaki durumlarda üçüncü normal formdadır (3NF):

- 2NF içinde *ve* Geçişli bağımlılıklar içermez.

üçüncü normal form (3NF) Bir tablo 2NF'de olduğunda ve hiçbir anahtar olmayan nitelik başka bir anahtar olmayan niteliğe işlevsel olarak bağımlı olmadığında 3NF'dedir; yani geçişli bağımlılıklar içeremez.

2NF ve 3NF problemlerinin çözümü arasındaki benzerliklere dikkat etmek ilginçtir. Bir tabloyu 1NF'den 2NF'ye dönüştürmek için kısmi bağımlılıkları kaldırmak gerekir. Bir tabloyu 2NF'den 3NF'ye dönüştürmek için geçişli bağımlılıkları kaldırmak gerekir. "Problem" bağımlılığı ister kısmi bağımlılık ister geçişli bağımlılık olsun, çözüm aynıdır: her bir problem bağımlılığı için yeni bir tablo oluşturmak. Problem bağımlılığının belirleyicisi orijinal tabloda kalır ve yeni tablonun birincil anahtarı olarak yerleştirilir. Problem bağımlılığının bağımlıları orijinal tablodan kaldırılır ve yeni tabloya birincil olmayan öznitelikler olarak yerleştirilir.

Bununla birlikte, teknik aynı olsa da, 3NF'ye geçmeden önce 2NF'nin elde edilmesinin zorunlu olduğunu unutmayın; geçişli bağımlılıkları çözmeden önce kısmi bağımlılıkları çözdüğünüzden emin olun. Ayrıca, normalleştirme tartışmasının başında yapılan varsayımı hatırlayın - her tablonun yalnızca bir aday anahtarı vardır ve bu da birincil anahtardır. Bir tablonun birden fazla aday anahtarı varsa, genel süreç aynı kalır, ancak ek hususlar vardır.

Örneğin, bir tabloda birden fazla aday anahtar varsa ve bunlardan biri bileşik bir anahtarsa, seçilen birincil anahtar tek bir öznitelik olsa bile, tablo bu bileşik aday anahtara dayalı kısmi bağımlılıklara sahip olabilir. Bu gibi durumlarda, yukarıda açıklanan süreci takiben, bu bağımlılıklar geçişli bağımlılıklar olarak algılanacak ve 3NF'ye kadar çözülmeyecektir. Burada açıklanan basitleştirilmiş süreç tasarımcının doğru sonuca ulaşmasını sağlar, ancak pratik yaparak tüm aday anahtarları ve bağımlılıklarını bu şekilde tanımalı ve bunları uygun şekilde çözmelisiniz. Birden fazla aday anahtarın varlığı, geçişli bağımlılıkların tanımlanmasını da etkileyebilir. Önceden, bir asal olmayan nitelik başka bir asal olmayan niteliği belirlediğinde geçişli bir bağımlılığın var olduğu tanımlanıyordu. İçinde

Birden fazla aday anahtarın varlığında, asal olmayan özniteliğin herhangi bir aday anahtarın parçası olmayan bir öznitelik olarak tanımlanması kritik önem taşır. Bir işlevsel bağımlılığın belirleyicisi birincil anahtar değilse ancak başka bir aday anahtarın parçasıysa, o zaman asal olmayan bir değildir ve geçişli bir bağımlılığın varlığına işaret etmez.

6-4 Tasarımın Geliştirilmesi

Artık tablo yapıları sorun yaratan kısmi ve geçişli bağımlılıkları ortadan kaldıracak şekilde temizlendiğine göre, veritabanının bilgi sağlama yeteneğini geliştirmeye ve operasyonel özelliklerini iyileştirmeye odaklanabilirsiniz. Önümüzdeki birkaç paragrafta, iyi bir normalleştirilmiş tablo kümesi oluşturmak için ele almanız gereken çeşitli sorun türleri hakkında bilgi edineceksiniz. Kısa olması açısından her bölümde sadece bir örnek sunulmuştur; tasarımcı bu prensibi tasarımda kalan tüm tablolara uygulamalıdır. Normalleştirmenin tek başına iyi tasarımlar yapmak için kullanılamayacağını unutmayın. Bunun yerine normalleştirme, kullanımı veri fazlalıklarını ortadan kaldırmaya yardımcı olduğu için değerlidir.

Veri Giriş Hatalarını En Aza İndirin

EMPLOYEE tablosuna her yeni çalışan girildiğinde, bir JOB_CLASS değeri . Ne yazık ki, tasarım belirleyici bir öznitelikte "açıklayıcı" girişlere izin verdiğinde, referans bütünlüğü ihlallerine yol açan veri girişi hataları yapmak çok kolaydır. Örneğin, EMPLOYEE tablosundaki JOB_CLASS özniteliği için *Veritabanı Tasarımcısı* yerine *DB Tasarımcısı* girilmesi böyle bir ihlali tetikleyecektir. Bu nedenle, benzersiz bir tanımlayıcı oluşturmak için bir JOB_CODE özniteliği eklemek daha iyi olacaktır. JOB_CODE özniteliğinin eklenmesi aşağıdaki bağımlılığı oluşturur:JOB_CODE→ JOB_CLASS, CHG_HOUR JOB_CODE'un uygun bir birincil anahtar olduğunu varsayarsanız, bu yeni öznitelik aşağıdaki bağımlılığı doğurur: JOB_CLASS→ CHG_HOUR

Ancak, bu bağımlılık geçişli bir bağımlılık değildir çünkü belirleyici bir can- didate anahtarıdır. Ayrıca, JOB_CODE'un varlığı referans bütünlüğü ihlalleri olasılığını büyük ölçüde azaltır. Yeni JOB tablosunun artık iki aday anahtarı olduğuna dikkat edin: JOB_CODE ve JOB_CLASS. Bu durumda, JOB_CODE seçilen birincil anahtarın yanı sıra bir vekil anahtardır. Hatırlayacağınız gibi, vekil anahtar, tablolara birincil anahtarların atanmasını basitleştirmek amacıyla tasarımcı tarafından eklenen yapay bir PK'dır. Vekil anahtarlar genellikle sayısaldır, genellikle VTYS tarafından otomatik olarak oluşturulurlar, anlamsal içerikten yoksundurlar (özel bir anlamları yoktur) ve genellikle son kullanıcılardan gizlenirler. Kısacası, tüm birincil anahtarların Bölüm 5, Gelişmiş Veri Modelleme'de öğrendiğiniz birincil anahtar yönergelerine uygun olduğundan olun.

Adlandırma Kurallarını Değerlendirin

Bölüm 2, Veri Modelleri'nde özetlenen adlandırma kurallarına uymak en iyisidir. nedenle, CHG_HOUR, JOB tablosuyla ilişkisini belirtmek için JOB_CHG_HOUR olarak değiştirilecektir. Ayrıca, JOB_CLASS öznitelik adı Sistem Analisti, Veritabanı Tasarımcısı gibi girişleri tam olarak tanımlamamaktadır; JOB_DESCRIPTION etiketi girişlere daha iyi uymaktadır. Ayrıca, 1NF'den 2NF'ye dönüşümde HOURS'un ASSIGN_HOURS olarak değiştirildiğini fark etmiş olabilirsiniz. Bu değişiklik, çalışılan saatleri ASSIGNMENT tablosu ile ilişkilendirmenizi sağlar.

Öznitelik Atomikliğini İyileştirin

Atomiklik gerekliliğine dikkat etmek genellikle iyi bir uygulamadır. **Atomik** bir **nitelik**, daha fazla alt bölümlere ayrılamayan bir **niteliktir**. Böyle bir özniteliğin **atomiklik** gösterdiği söylenir. EMP_NAME'in EMPLOYEE tablosunda kullanılmasının atomik olmadığı açıktır çünkü EMP_NAME bir soyadı, bir ad ve bir baş harf olarak ayrıştırılabilir. Atomiklik derecesini artırarak, sorgulama esnekliği de kazanırsınız. Örneğin, EMP_LNAME, EMP_FNAME ve EMP_INITIAL kullanırsanız, soyadları, adları ve baş harfleri sıralayarak kolayca telefon listeleri oluşturabilirsiniz. İsim bileşenleri tek bir öznitelik içinde olsaydı böyle bir görev çok zor olurdu. Genel olarak, tasarımcılar iş kuralları ve işleme gereksinimlerinde belirtildiği gibi basit, tek değerli öznitelikler kullanmayı tercih eder.

atomik nitelik
Anlamlı sonuçlar üretmek için daha fazla alt bölümlere ayrı bir nitelik bileşenler. Örneğin,Bir kişinin soyadı özelliği anlamlı bir şekilde alt bölümlere ayrılamaz.

atomiklik
Daha küçük birimlere bölünememe.

Yeni Nitelikleri Tanımlayın

EMPLOYEE tablosu gerçek dünya ortamında kullanılsaydı, birkaç başka özniteliğin daha eklenmesi gerekirdi. Örneğin, yılbaşından bugüne brüt maaş ödemeleri, Sosyal Güvenlik ödemeleri ve Medicare ödemeleri istenebilir. Bir çalışanın işe alınma tarihi özniteliği (EMP_HIREDATE), bir çalışanın iş ömrünü takip etmek için kullanılabilir ve uzun süreli çalışanlara ikramiye verilmesi ve diğer moral artırıcı önlemler için bir temel oluşturabilir. Aynı ilke tasarımınızdaki diğer tüm tablolara da uygulanmalıdır.

Yeni İlişkiler Belirleyin

Orijinal rapora göre, kullanıcıların her bir projenin yöneticisi olarak hangi çalışanın görev yaptığını takip etmesi gerekmektedir. Bu, ÇALIŞAN ve PROJE arasında bir ilişki olarak uygulanabilir. Orijinal rapordan, her projenin yalnızca bir yöneticisi olduğu anlaşılmaktadır. Bu nedenle, sistemin her bir projenin yöneticisi hakkında ayrıntılı bilgi sağlayabilmesi, EMP_NUM'un PROJECT'te yabancı anahtar olarak kullanılmasıyla sağlanır. Bu eylem, gereksiz ve istenmeyen veri çoğaltması oluşturmadan her PROJE'nin yönetici verilerinin ayrıntılarına erişebilmenizi sağlar. Tasarımcı, normalleştirme ilkelerini kullanarak doğru öznitelikleri doğru tablolara yerleştirmeye özen göstermelidir.

Veri Ayrıntılandırması için Birincil Anahtarları Gerektiği Gibi Hassaslaştırın

Ayrıntı düzeyi, bir tablo satırında depolanan değerlerin temsil ettiği ayrıntı düzeyini ifade eder. Daha önce açıklandığı gibi, en düşük ayrıntı düzeyinde saklanan verilerin *atomik veriler* olduğu söylenir. Şekil 6.5'te, 3NF'deki ASSIGNMENT tablosu, belirli bir çalışanın belirli bir projede çalıştığı saatleri temsil etmek için ASSIGN_HOURS özniteliğini kullanır. Ancak, bu değerler en düşük ayrıntı düzeyinde mi kaydedilmektedir? Başka bir deyişle, ASSIGN_HOURS *saatlik* toplamı mı, *günlük* toplamı mı, *haftalık* toplamı mı, *aylık* toplamı mı yoksa *yıllık* toplamı mı temsil eder? ASSIGN_HOURS'un daha dikkatli bir tanımlama gerektirdiği açıktır. Bu durumda, ilgili soru şu şekilde olacaktır: ASSIGN_HOURS verilerini hangi zaman dilimi için-saat, gün, hafta, ay vb. kaydetmek istiyorsunuz?

Örneğin, EMP_NUM ve PROJ_NUM kombinasyonunun ASSIGNMENT tablosunda kabul edilebilir (bileşik) bir birincil anahtar olduğunu varsayın. Bu birincil anahtar, yalnızca bir çalışanın başlangıcından bu yana bir projede çalıştığı toplam saat sayısını temsil etmek için kullanışlıdır. Bu gibi durumlarda, kullanılmak istenen veri ayrıntı düzeyini belirlemek için son kullanıcıya danışmalısınız (günlük toplamlar, haftalık toplamlar, vb.) Cevap sadece veritabanı tasarımını değil, aynı zamanda operasyonel yönleri de etkileyecektir; yani, bir çalışanın bir projede çalıştığı saatleri ne sıklıkla ve hangi ayrıntı düzeyinde (günde birçok kez, günde bir kez, haftada bir kez, ayda bir kez, vb: 1) veri ayrıntı düzeyini belirlemeli, 2) uygun bir birincil anahtar belirlemeli (vekil anahtar ihtiyacını değerlendirmeli) ve 3) aday anahtarın benzersizliğini sağlamalısınız Bkz. Bölüm 6-5).

granülerlik
Bir tablonun satırında depolanan değerlerin temsil ettiği ayrıntı düzeyi. En düşük seviyede saklanan veriler tanecikli olduğu söylenir *atomik veri*.

ASSIGN_NUM gibi bir vekil birincil anahtar kullanmak daha düşük ayrıntı düzeyi sağlar ve daha fazla esneklik sağlar. Örneğin, EMP_NUM ve PROJ_NUM kombinasyonunun birincil anahtar olarak kullanıldığını ve ardından bir çalışanın ASSIGNMENT tablosunda iki "çalışılan saat" girişi yaptığını varsayalım. Bu eylem varlık bütünlüğü gerekliliğini ihlal eder. ASSIGN_DATE'i bileşik PK'nın bir parçası olarak ekleseniz bile, herhangi bir çalışan aynı gün aynı proje için iki veya daha fazla giriş yaparsa yine de varlık bütünlüğü ihlali oluşur. (Çalışan proje üzerinde sabah birkaç saat çalışmış ve daha sonra günün ilerleyen saatlerinde tekrar çalışmış olabilir). ASSIGN_NUM birincil anahtar olarak kullanıldığında aynı veri girişi hiçbir sorun yaratmaz. Bu nedenle, ayrıntı düzeyini gün başına "birden fazla çalışılan saat" girişi olarak belirledikten sonra, ASSIGNMENT tablosu için birincil anahtar olarak bir vekil anahtar kullanmaya karar verirsiniz.

Not

İdeal bir veritabanı tasarımında, istenen ayrıntı düzeyi kavramsal tasarım sırasında veya gereksinimler toplanırken belirlenir. Ancak, bu bölümde daha önce gördüğünüz gibi, birçok veritabanı tasarımı mevcut veri gereksinimlerinin iyileştirilmesini içerir ve böylece tasarım değişikliklerini tetikler. Gerçek dünya ortamında, değişen ayrıntı düzeyi gereksinimleri birincil anahtar seçiminde değişiklikler gerektirebilir ve bu değişiklikler sonuçta vekil anahtarların kullanılmasını gerektirebilir.

Tarihsel Doğruluğu Koruyun

Saat başına iş ücretinin ASSIGNMENT tablosuna yazılması, tablodaki verilerin tarihsel doğruluğunun korunması açısından çok önemlidir. Bu özniteliğe ASSIGN_ CHG_HOUR adını vermek uygun olacaktır. Bu öznitelik JOB_CHG_ HOUR ile aynı değere sahip gibi görünse de, bu *yalnızca* JOB_CHG_HOUR değeri sonsuza kadar aynı kalırsa doğrudur. Saat başına iş ücretinin zaman içinde değişeceğini varsaymak mantıklıdır. Ancak, her bir proje için ücretlerin ASSIGNMENT tablosundan çalışılan saatler ile JOB tablosundan saat başına ücret çarpılarak hesaplandığını ve faturalandırıldığını varsayalım. Bu ücretler, atama sırasında geçerli olan saat başına ücret yerine her zaman İŞ tablosunda kayıtlı olan geçerli saat başına ücreti gösterecektir.

Türetilmiş Öznitelikleri Kullanarak Değerlendirme

Son olarak, bir projeye yapılan gerçek ücretlendirmeyi saklamak için ASSIGNMENT tablosunda türetilmiş bir öznitelik kullanabilirsiniz. ASSIGN_CHARGE olarak adlandırılan bu türetilmiş öznitelik, ASSIGN_HOURS ile ASSIGN_CHG_HOUR'un çarpımının sonucudur. Bu, şu şekilde geçişli bir bağımlılık yaratır:

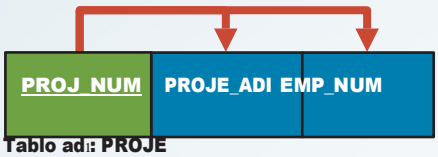
(ASSIGN_CHARGE 1 ASSIGN_HOURS)→ ASSIGN_CHG_HOUR

Sistem işlevselliği açısından bakıldığında, bu tür türetilmiş öznitelik değerleri rapor veya fatura yazmak için gerektiğinde hesaplanabilir. Bununla birlikte, türetilmiş özniteliğin tabloda saklanması, istenen sonuçları üretmek için uygulama yazılımının yazılmasını kolaylaştırır. Ayrıca, çok sayıda işlemin raporlanması ve/veya özetlenmesi gerekiyorsa, türetilmiş özniteliğin kullanılabilir olması raporlama süresinden tasarruf sağlayacaktır. (Hesaplama veri girişi sırasında yapılırsa, son kullanıcı Enter tuşuna bastığında tamamlanacak ve böylece süreç hızlanacaktır). Türetilmiş özniteliklerin bir veritabanı tablosunda saklanmasının sonuçları hakkında bilgi edinmek için Bölüm 4, Varlık İlişkisi (ER) Modellemesi'ni inceleyin.

Önceki bölümlerde açıklanan geliştirmeler Şekil 6.6'da gösterilen tablolarda ve bağımlılık diyagramlarında gösterilmektedir.

Şekil 6.6 Tamamlanmış Veritabanı

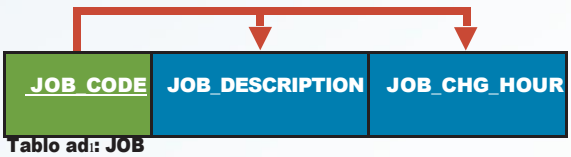
Tablo adı: PROJE



Tablo adı: PROJE

PROJ_NUM	PROJ_NAME	EMP_NUM
15	Evergreen	105
18	Amber Wave	104
22	Rolling Tide	113
25	Starflight	101

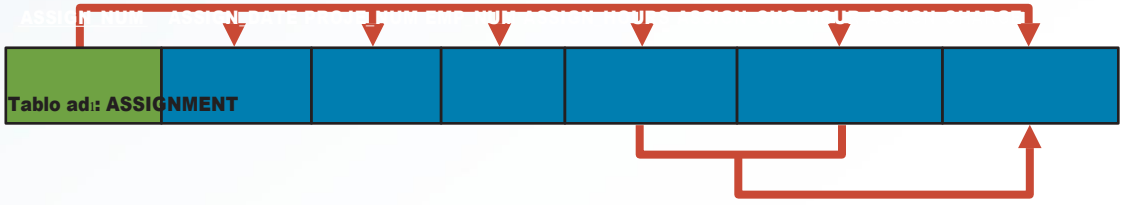
Tablo adı: JOB **Veritabanı adı: Ch06_ConstructCo**



Tablo adı: JOB

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
500	Programmer	35.75
501	Systems Analyst	96.75
502	Database Designer	105.00
503	Electrical Engineer	84.50
504	Mechanical Engineer	67.90
505	Civil Engineer	55.78
506	Clerical Support	26.87
507	DSS Analyst	45.95
508	Applications Designer	48.10
509	Bio Technician	34.55
510	General Support	18.36

Tablo adı: ASSIGNMENT

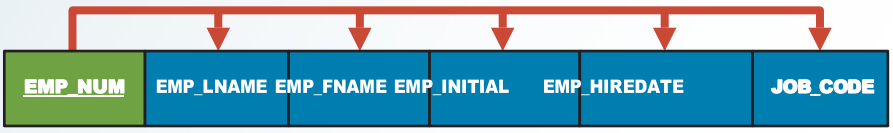


ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-22	15	103	2.6	84.50	219.70
1002	04-Mar-22	18	118	1.4	18.36	25.70
1003	05-Mar-22	15	101	3.6	105.00	378.00
1004	05-Mar-22	22	113	2.5	48.10	120.25
1005	05-Mar-22	15	103	1.9	84.50	160.55
1006	05-Mar-22	25	115	4.2	96.75	406.35
1007	05-Mar-22	22	105	5.2	105.00	546.00
1008	05-Mar-22	25	101	1.7	105.00	178.50
1009	05-Mar-22	15	105	2.0	105.00	210.00
1010	06-Mar-22	15	102	3.8	96.75	367.65
1011	06-Mar-22	22	104	2.6	96.75	251.55
1012	06-Mar-22	15	101	2.3	105.00	241.50
1013	06-Mar-22	25	114	1.8	48.10	86.58
1014	06-Mar-22	22	111	4.0	26.87	107.48
1015	06-Mar-22	25	114	3.4	48.10	163.54
1016	06-Mar-22	18	112	1.2	45.95	55.14
1017	06-Mar-22	18	118	2.0	18.36	36.72
1018	06-Mar-22	18	104	2.6	96.75	251.55
1019	06-Mar-22	15	103	3.0	84.50	253.50
1020	07-Mar-22	22	105	2.7	105.00	283.50
1021	08-Mar-22	25	108	4.2	96.75	406.35
1022	07-Mar-22	25	114	5.8	48.10	278.98
1023	07-Mar-22	22	106	2.4	35.75	85.80

(devam ediyor)

Şekil 6.6 Tamamlanan Veritabanı *(Devamı)*

Tablo adı: EMPLOYEE **Veritabanı adı: Ch06_ConstructCo**



Tablo adı: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-97	503
104	Ramoras	Anne	K	15-Nov-88	501
105	Johnson	Alice	K	01-Feb-94	502
106	Smithfield	William		22-Jun-05	500
107	Alonzo	Maria	D	10-Oct-94	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501
110	Olenko	Gerald	A	11-Dec-96	505
111	Wabash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-95	507
113	Joenbrood	Delbert	K	15-Nov-94	508
114	Jones	Annelise		20-Aug-91	508
115	Bawangi	Travis	B	25-Jan-90	501
116	Pratt	Gerald	L	05-Mar-95	510
117	Williamson	Angie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-06	510

Şekil 6.6 orijinal veritabanı tasarımına göre büyük bir gelişmedir. Uygulama yazılımı düzgün bir şekilde tasarlanırsa, en aktif tablo (ASSIGNMENT) yalnızca PROJ_NUM, EMP_NUM ve ASSIGN_HOURS değerlerinin girilmesini gerektirir. ASSIGN_NUM ve ASSIGN_DATE nitelikleri için değerler uygulama tarafından oluşturulabilir. ÖrneğinASSIGN_NUM bir sayaç kullanılarak oluşturulabilir ve ASSIGN_DATE uygulama tarafından okunan ve ASSIGNMENT tablosuna otomatik olarak girilen sistem tarihi olabilir. Buna ek olarak, uygulama yazılımı uygun JOB tablosunun JOB_CHG_HOUR değerini ASSIGNMENT tablosuna yazarak doğru ASSIGN_CHG_HOUR değerini otomatik olarak ekleyebilir. (JOB ve ASSIGNMENT tabloları JOB_CODE niteliği aracılığıyla ilişkilidir.) JOB tablosunun JOB_CHG_HOUR değeri değişirse, bu değerin ASSIGNMENT tablosuna bir sonraki eklenmesi değişikliği otomatik olarak yansıtacaktır. Böylece tablo yapısı insan müdahalesi ihtiyacını en aza indirir. Aslında, sistem çalışanların kendi çalışma saatlerini girmelerini gerektiriyorsa, EMP_NUM'larını kimliklerini giren bir manyetik kart okuyucu kullanarak ASSIGNMENT tablosuna tarayabilirler. Böylece, ASSIGNMENT tablosunun yapısı istenen güvenlik seviyesinin korunmasına zemin hazırlayabilir.

6-5 Vekilin Dikkat Etmesi Gereken Temel Hususlar

Bu tasarım önemli varlık ve referans bütünlüğü gereksinimlerini karşılasa da tasarımcının yine de bazı endişeleri gidermesi gerekir. Örneğin, bileşik birincil anahtar, öznitelik sayısı arttıkça kullanımı çok zahmetli hale gelebilir. (İlgili tablo bileşik bir birincil anahtar kullandığında uygun bir yabancı anahtar oluşturmak zorlaşır. Ayrıca, bileşik bir birincil anahtar arama rutinlerinin yazılmasını da zorlaştırır). Ya da bir birincil anahtar özniteliği kullanılamayacak kadar fazla açıklayıcı içeriğe sahip olabilir; bu nedenle JOB_CODE özniteliği JOB tablosuna birincil anahtar olarak kullanılmak üzere eklenmiştir. Birincil anahtarın herhangi bir nedenle uygun olmadığı düşünüldüğünde, tasarımcılar önceki bölümde tartışıldığı gibi vekil anahtarlar kullanırlar.

Uygulama düzeyinde, bir vekil anahtar genellikle DBMS aracılığıyla oluşturulan ve yönetilen sistem tanımlı bir özniteliktir. Genellikle, sistem tanımlı bir vekil anahtar sayısaldır ve değeri her yeni satır için otomatik olarak artırılır. Örneğin, Microsoft Access bir AutoNumber veri türü, Microsoft SQL Server bir kimlik sütunu, Oracle bir sıra nesnesi ve MySQL bir otomatik artırma kısıtlaması kullanır.
Bölüm 6-4'ten JOB_CODE özniteliğinin JOB tablosunun birincil anahtarı olarak belirlendiğini hatırlayın. Ancak, Tablo 6.4'teki JOB tablosunda gösterildiği gibi, JOB_CODE özniteliğinin yinelenen girişleri engellemediğini unutmayın.

Tablo 6.4 İş Tablosunda Yinelenen Girişler

Job_Code	İş_Açıklaması	Job_Chg_Hour
511	Programcı	$35.75
512	Programcı	$35.75

Tablo 6.4'teki veri girişlerinin, mevcut kayıtları tekrarladıkları için uygunsuz olduğu açıktır; ancak varlık bütünlüğü ya da referans bütünlüğü ihlal edilmemiştir. Bu çoklu yinelenen kayıt sorunu, JOB_CODE özniteliği PK olarak eklendiğinde ortaya çıkmıştır. (JOB_DESCRIPTION başlangıçta PK olarak belirlendiğinde, DBMS, varlık bütünlüğünü uygulaması istendiğinde tüm iş tanımı girişleri için benzersiz değerler sağlayacaktır. Ancak, bu seçenek ilk etapta JOB_CODE özniteliğinin kullanılmasına neden olan sorunları yaratmıştır!) Her halükarda, JOB_CODE vekil PK olacaksa, *benzersiz bir dizin kullanarak* JOB_DESCRIPTION içinde *benzersiz* değerlerin var olmasını sağlamanız gerekir.
Kalan tüm tabloların (PROJECT, ASSIGNMENT ve EMPLOYEE) aynı sınırlamalara tabi olduğunu unutmayın. Örneğin, EMPLOYEE tablosundaki EMP_NUM özniteliğini PK olarak kullanırsanız, aynı çalışan için birden fazla giriş yapabilirsiniz. Bu sorunu önlemek için EMP_LNAME, EMP_FNAME ve EMP_ INITIAL için benzersiz bir dizin oluşturabilirsiniz, ancak Joe B. Smith adlı iki çalışanla nasıl başa çıkacaksınız? Bu durumda, benzersiz bir dizinin temelini oluşturmak başka bir (tercihen harici olarak tanımlanmış) öznitelik kullanabilirsiniz.
Veritabanı tasarımının genellikle ödünleşim ve profesyonel muhakeme gerektirdiğini tekrarlamakta fayda var. Gerçek dünya ortamında, tasarım bütünlüğü ile esneklik arasında bir denge kurmanız gerekir. Örneğin, bir çalışanı tarih başına yalnızca bir ASSIGN_HOURS girişi ile sınırlamak istiyorsanız ASSIGNMENT tablosunu PROJ_NUM, EMP_NUM ve ASSIGN_DATE üzerinde benzersiz bir dizin kullanacak şekilde tasarlayabilirsiniz. Bu sınırlama, çalışanların herhangi bir tarih için aynı saatleri birden fazla kez girememesini sağlayacaktır. Ne yazık ki, bu sınırlama yönetimsel bakış açısından muhtemelen istenmeyen bir durumdur. Sonuçta, bir çalışan herhangi bir gün içinde bir projede birkaç farklı zamanda çalışıyorsa, o gün içinde aynı çalışan ve aynı proje için birden fazla giriş yapmak mümkün olmalıdır. Bu durumda en iyi çözüm, benzersizliği sağlamak için koçan, fiş veya bilet numarası gibi harici olarak tanımlanmış yeni bir eklemek olabilir. Her durumda, sık sık veri denetimleri yapılması uygun olacaktır. Bir çalışanın aynı projede aynı tarih için üç çalışılan saat girdisi (sırasıyla 8, 9 ve 8 saat) girebileceğini düşünün. Veritabanı bu değerleri kabul edecektir. Şu soruyu sormanız gerekir: Bir çalışan bir projede belirli bir günde 25 saat çalışabilir mi? Ya da bir çalışan birden fazla projede bir günde toplam 36 saat çalışabilir mi? Veritabanı bu girişleri kabul edecektir, ancak bu durum gerçekçi midir? Muhtemel cevap hayırdır. Çoğu durumda, bir çalışanın bir veya daha fazla projede bir zaman diliminde (gün, hafta, ay) çalışabileceği saat sayısını tanımlayan ve sınırlayan iş kuralları olmalıdır. Büyük olasılıkla, bu tür iş kuralları programlama kodu veya veritabanı tetikleyicileri tarafından uygulanmalıdır (Bkz. Bölüm 8-7a.)

6-6 Yüksek Seviyeli Normal Formlar

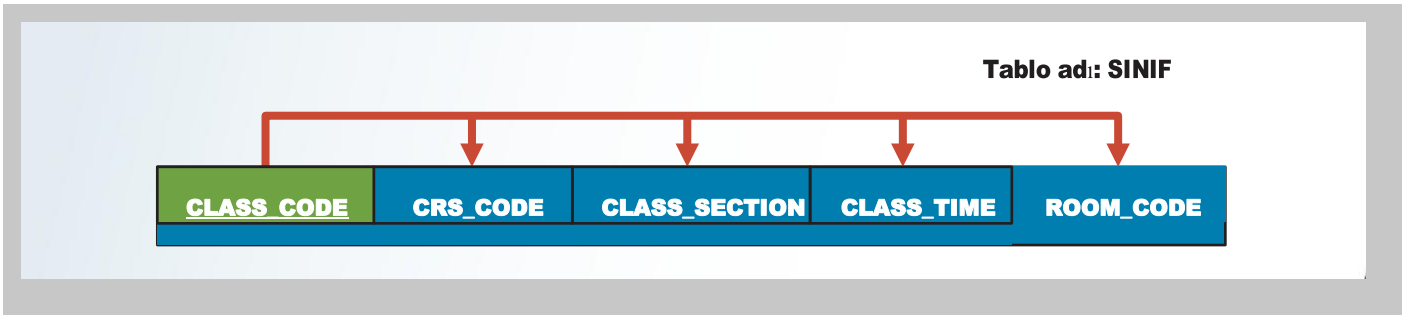
3NF'deki tablolar ticari işlem veritabanlarında uygun bir performans gösterecektir. Ancak, daha yüksek normal formlar bazen kullanışlıdır. Bu bölümde, Boyce-Codd normal formu, dördüncü normal form (4NF) ve beşinci normal form (5NF) olarak bilinen 3NF'nin özel bir durumu hakkında bilgi edineceksiniz.

6-6a Boyce-Codd Normal Formu

Boyce-Codd normal formu (BCNF)
Her belirleyicinin bir aday anahtar olduğu özel bir üçüncü normal form (3NF) türü. BCNF'deki bir tablo 3NF' olmalıdır. Ayrıca bkz. *belirleyici*.

Bir tablo 3NF'de olduğunda ve tablodaki her belirleyici bir aday anahtar olduğunda **Boyce-Codd normal formundadır (BCNF).** (Bölüm 3'ten, aday anahtarın birincil anahtarla aynı özelliklere sahip olduğunu, ancak herhangi bir nedenle birincil anahtar olarak seçilmediğini hatırlayın). Açıkçası, bir tablo yalnızca bir aday anahtar içerdiğinde, 3NF ve BCNF eşdeğerdir. Başka bir deyişle, BCNF yalnızca tablo birden fazla aday anahtar içerdiğinde ihlal edilebilir. Önceki normal form örneklerinde, açıklamaları basitleştirmek için yalnızca bir aday anahtar içeren tablolar kullanılmıştır. Ancak birden fazla aday anahtarın her zaman mümkün olduğunu ve nor- malleştirme kurallarının sadece birincil anahtara değil aday anahtarlara odaklandığını unutmayın. Şekil 6.7'de gösterilen tablo yapısını düşünün.

Şekil 6.7 Birden Fazla Aday Anahtarı Olan Tablolar



CLASS tablosunun iki aday anahtarı vardır:

- CLASS_CODE
- CRS_CODE 1 CLASS_SECTION

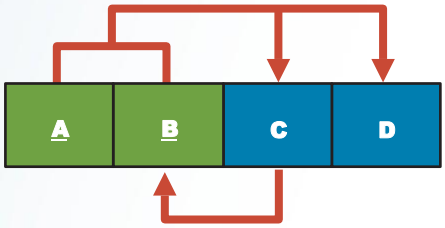
Tablo 1NF'dedir çünkü anahtar nitelikler tanımlanmıştır ve anahtar olmayan tüm nitelikler tarafından belirlenir. Bu durum her iki aday anahtar için de geçerlidir. Her iki aday anahtar da tanımlanmıştır ve diğer tüm öznitelikler her iki aday tarafından da belirlenebilir. Tablo 2NF'dedir çünkü 1NF'dedir ve her iki aday da kısmi bağımlılık yoktur. CLASS_CODE tek bir öznitelik aday anahtarı olduğundan, kısmi bağımlılıklar sorunu geçerli değildir. Ancak, CRS_CODE 1 CLASS_SECTION bileşik aday anahtarı potansiyel olarak kısmi bağımlılığa sahip olabilir, bu nedenle bu aday anahtar için 2NF değerlendirilmelidir. Bu durumda, bileşik anahtarı içeren kısmi bağımlılıklar yoktur. Son olarak, tablo 3NF'dedir çünkü geçişli bağımlılıklar yoktur. CRS_ CODE 1 CLASS_SECTION bir aday anahtar olduğu için, bu bileşiğin CLASS_TIME ve ROOM_CODE'u belirleyebildiği gerçeğinin geçişli bir bağımlılık olmadığını unutmayın. *Anahtar olmayan* bir öznitelik başka bir anahtar olmayan özniteliği belirleyebiliyorsa ve CRS_CODE 1 CLASS_SECTION bir anahtar ise geçişli bir bağımlılık söz konusudur.

Not

Bir tablo 3NF'de olduğunda ve tablodaki her belirleyici bir aday anahtar olduğunda Boyce- Codd normal formundadır (BCNF).

Çoğu tasarımcı BCNF'yi 3NF'nin özel bir durumu olarak görmektedir. Aslında, bu bölümde gösterilen teknikler kullanılırsa, 3NF'ye ulaşıldığında çoğu tablo BCNF gerekliliklerine uyar. Peki, bir tablo nasıl hem 3NF'de olup hem de BCNF'de olmayabilir? Bu soruyu yanıtlamak için, asal olmayan bir nitelik asal olmayan başka bir niteliğe bağımlı olduğunda geçişli bir bağımlılığın var olduğunu aklınızda tutmalısınız.
Başka bir deyişle, bir tablo 2NF'de olduğunda ve geçişli bağımlılıklar olmadığında 3NF'dedir, ancak bir anahtar niteliğin başka bir anahtar niteliğin belirleyicisi olduğu bir durum ne olacaktır? Bu durum 3NF'yi ihlal etmez, ancak BCNF gerekliliklerini karşılamaz (bkz. Şekil 6.8), çünkü BCNF tablodaki her belirleyicinin bir aday anahtar olmasını gerektirir.

Şekil 6.8 3NF'de Olan Ancak BCNF'de Olmayan Bir Tablo



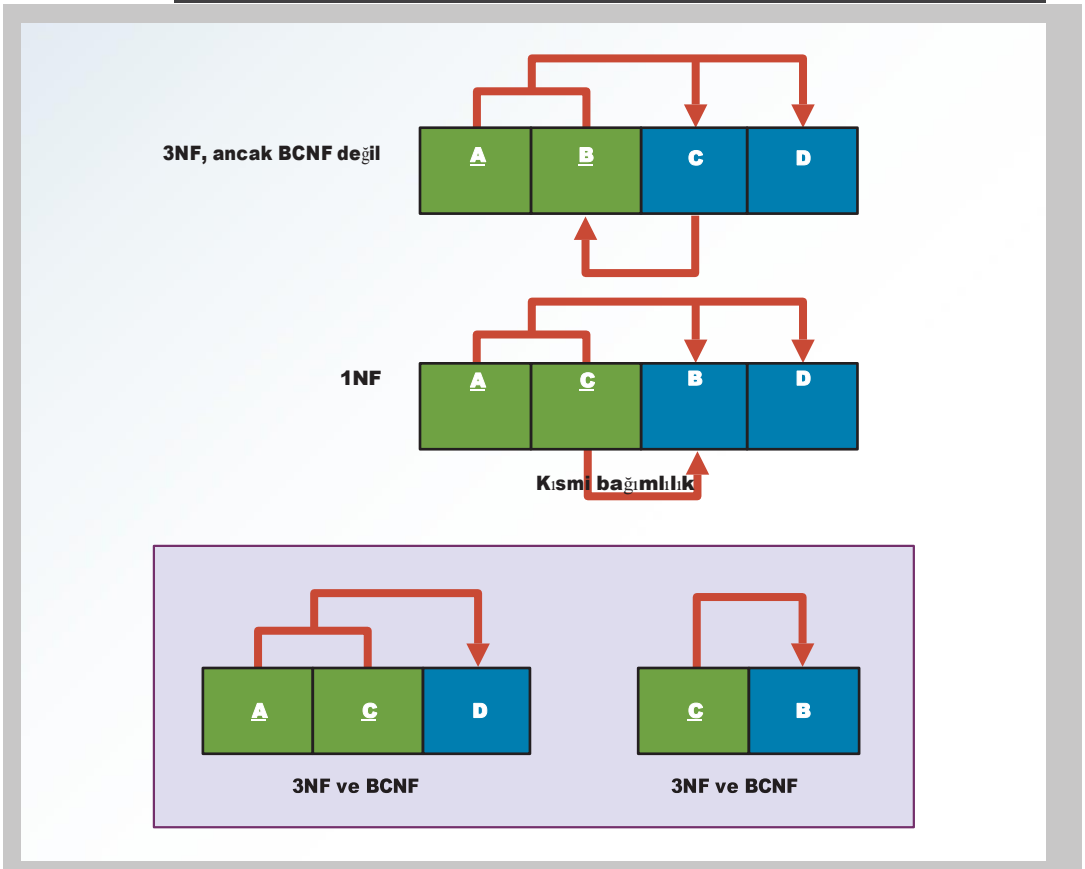
Şekil 6.8'deki şu işlevsel bağımlılıklara dikkat edin:

A 1 B→ C, D
A 1 C→ B, D
C→ B

Bu yapının iki aday anahtarı olduğuna dikkat edin: (A 1 B) ve (A 1 C). Şekil 6.8'de gösterilen tablo yapısında kısmi bağımlılıklar olmadığı gibi geçişli bağımlılıklar da yoktur. (C→ B koşulu, *bir anahtar niteliğinin birincil anahtarın bir parçasını belirlediğini* gösterir ve *bu* bağımlılık geçişli veya kısmi *değildir* çünkü bağımlı bir asal !) Dolayısıyla, Şekil 6.8'deki tablo yapısı 3NF gerekliliklerini karşılar, ancak C→ B koşulu tablonun BCNF gerekliliklerini karşılayamamasına neden olur.
Şekil 6.8'deki tablo yapısını 3NF ve BCNF tablo yapılarına dönüştürmek için, önce birincil anahtarı A 1 C olarak değiştirin. Bu değişiklik uygundur çünkü C→ B bağımlılığı, C'nin etkin bir şekilde B'nin bir üst kümesi olduğu anlamına gelir. Bu noktada, tablo 1NF'dedir çünkü kısmi bir bağımlılık içerir, C→ B. Ardından, Şekil 6.9'da gösterilen sonuçları üretmek için standart ayrıştırma prosedürlerini izleyin.
Bu prosedürün gerçek bir probleme nasıl uygulanabileceğini görmek için Tablo 6.5'teki örnek verileri inceleyin.
Tablo 6.5 aşağıdaki koşulları yansıtmaktadır:

- Her CLASS_CODE bir sınıfı benzersiz bir şekilde tanımlar. Bu durum, bir kursun birçok sınıf oluşturabileceği durumu göstermektedir. Örneğin, INFS 420 etiketli bir kurs, kayıt işlemlerini kolaylaştırmak için her biri benzersiz bir kodla tanımlanan iki sınıf (bölüm) halinde verilebilir. Böylece, SINIF_KODU 32456 INFS 420, sınıf bölümü 1'i tanımlarken SINIF_KODU 32457 INFS 420, sınıf bölümü 2'yi tanımlayabilir. Ya da SINIF_KODU 28458, QM 362, sınıf bölümü 5'i tanımlayabilir.
- Bir öğrenci birçok ders alabilir. Örneğin, 125 numaralı öğrencinin hem 21334 hem de 32456 derslerini aldığını ve sırasıyla A ve C notlarını aldığını unutmayın.
- Bir personel birçok sınıfa ders verebilir, ancak her sınıf yalnızca bir personel tarafından verilir. Personel 20'nin 32456 ve 28458 olarak tanımlanan sınıflara ders verdiğine dikkat edin.

Şekil 6.9 BCNF'ye ayrıştırma



Tablo 6.5 BCNF Dönüşümü için Örnek Veriler

Stu_ID	Personel_ID	Sınıf_kodu	Enroll_grade
125	25	21334	A
125	20	32456	C
135	20	28458	B
144	25	27563	C
144	20	32456	B

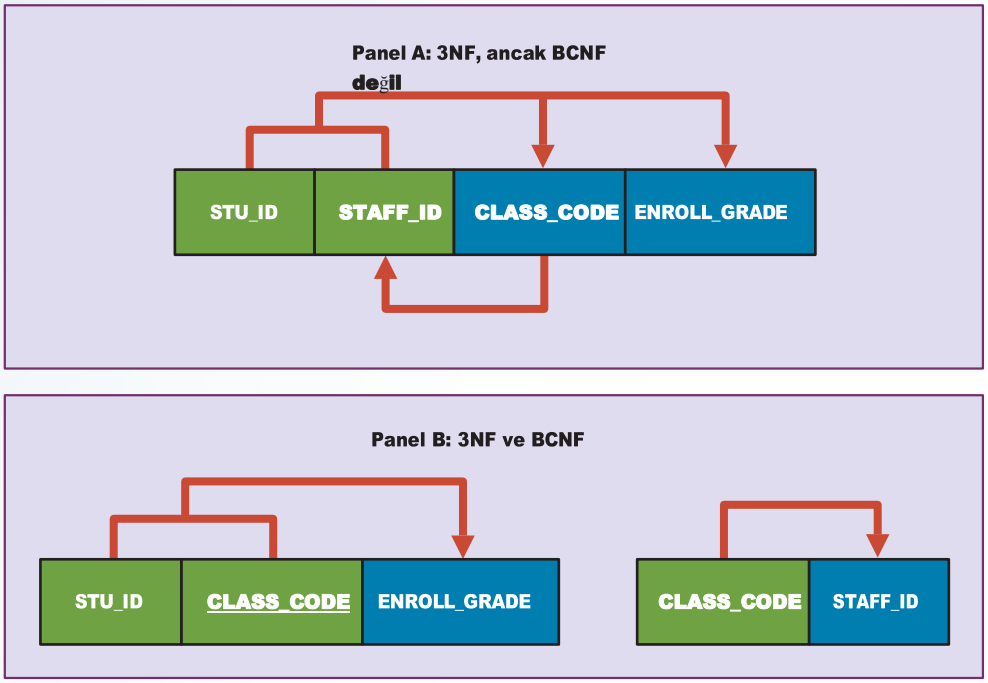
Tablo 6.5'te gösterilen yapı Şekil 6.10'un A Panelinde yansıtılmıştır:

STU_ID 1 STAFF_ID→ CLASS_CODE, ENROLL_GRADE

CLASS_CODE→ STAFF_ID

Şekil 6.10'daki Panel A, açıkça 3NF'de olan bir yapıyı göstermektedir, ancak bu yapının temsil tabloda büyük bir sorun vardır çünkü iki şeyi tanımlamaya çalışmaktadır: sınıflara personel atamaları ve öğrenci kayıt bilgileri. Böyle çift amaçlı bir tablo yapısı anomalilere neden olacaktır. Örneğin, 32456 numaralı sınıfa ders vermek üzere farklı bir personel atanırsa, iki satırın güncellenmesi gerekecek ve böylece bir güncelleme anomalisi ortaya çıkacaktır. Ayrıca, 135 numaralı öğrenci 28458 numaralı dersi bırakırsa, bu dersi kimin verdiği bilgisi kaybolur ve böylece bir silme anomalisi ortaya çıkar. Sorunun çözümü, daha önce özetlenen prosedürü izleyerek tablo yapısını ayrıştırmaktır. Şekil 6.10'da gösterilen Panel B'nin ayrıştırılması, hem 3NF hem de BCNF gerekliliklerine uyan iki tablo yapısı ortaya çıkarır.

Şekil 6.10 Başka Bir BCNF Ayrıştırması



Bir tablodaki her belirleyici bir aday anahtar olduğunda o tablonun BCNF'de olduğunu unutmayın. Bu nedenle, bir tablo yalnızca bir aday anahtar içerdiğinde, 3NF ve BCNF eşdeğerdir.

6-6b Dördüncü Normal Form (4NF)

Kötü tasarlanmış veritabanlarıyla karşılaşabilirsiniz veya elektronik tabloları birden fazla çok değerli niteliğin bulunduğu bir veritabanı formatına dönüştürmeniz istenebilir. Örneğin, bir çalışanın birden fazla görevi olabileceği ve aynı zamanda birden fazla hizmet kuruluşunda yer alabileceği olasılığını göz önünde bulundurun. 10123 numaralı çalışanın Kızıl Haç ve United Way için gönüllü olduğunu varsayalım. Buna ek olarak, aynı çalışan üç projede çalışmak üzere atanmış olabilir: 1, 3 ve 4. Şekil 6.11, bu gerçekler kümesinin çok farklı şekillerde nasıl kaydedilebileceğini göstermektedir.

Şekil 6.11 Çok Değerli Bağımlılıklara Sahip Tablolar

Veritabanı adı: Ch06_Service

Tablo adı: VOLUNTEER_V1

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	UW	3
10123		4

Tablo adı: VOLUNTEER_V2

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	
10123	UW	
10123		1
10123		3
10123		4

Tablo adı: VOLUNTEER_V3

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	RC	3
10123	UW	4

Şekil 6.11'deki tablolarda bir sorun vardır. ORG_CODE ve ASSIGN_NUM her biri birçok farklı değere sahip olabilir. Normalleştirme terminolojisinde bu durum, bir anahtar diğer iki özniteliğin birden fazla değerini belirlediğinde ve bu öznitelikler bağımsız olduğunda ortaya çıkan çok değerli bağımlılık olarak adlandırılır. (Bir çalışan birçok hizmet girişine ve birçok atama girişine sahip olabilir. Bu nedenle, bir EMP_NUM birden fazla ORG_CODE ve birden fazla ASSIGN_NUM belirleyebilir; ancak ORG_CODE ve ASSIGN_NUM bağımsızdır). Çok değerli bir bağımlılığın varlığı, tablo sürümleri 1 ve 2 uygulandığında, tabloların oldukça az sayıda boş değer içereceği anlamına gelir; aslında, tabloların uygun bir aday anahtarı bile yoktur. (EMP_NUM değerleri benzersiz değildir, bu nedenle PK olamazlar. Tablo sürüm 1 ve 2'deki özniteliklerin hiçbir kombinasyonu bir PK oluşturmak için kullanılamaz çünkü bazıları null içerir). Böyle bir durum, özellikle de birden fazla iş ataması ve birçok hizmet faaliyeti olabilecek binlerce çalışan olduğunda arzu edilen bir durum değildir. Sürüm 3 en azından bir PK'ya sahiptir, ancak bu PK tablodaki tüm özniteliklerden oluşur. Aslında, 3. versiyon 3NF gereksinimlerini karşılamaktadır, ancak açıkça istenmeyen birçok fazlalık içermektedir.

Çözüm, çok değerli bağımlılığın neden olduğu sorunları ortadan kaldırmaktır. Bunu, çok değerli bağımlılığın bileşenleri için yeni tablolar oluşturarak yaparsınız. Bu örnekte, Şekil 6.12'de gösterilen ASSIGNMENT ve SERVICE_V1 tabloları oluşturularak çok değerli bağımlılık çözülür ve ortadan kaldırılır. Bu tabloların 4NF içinde olduğu söylenir.

Şekil 6.12 4NF'de Bir Tablolar Kümesi

Veritabanı adı: CH06_Service

Tablo adı: PROJE

PROJ_CODE	PROJ_NAME	PROJ_BUDGET
1	BeThere	1023245.00
2	BlueMoon	20198608.00
3	GreenThumb	3234456.00
4	GoFast	5674000.00
5	GoSlow	1002500.00

Tablo adı: EMPLOYEE

EMP_NUM	EMP_LNAME
10121	Rogers
10122	O'Leery
10123	Panera
10124	Johnson

Tablo adı: ASSIGNMENT

ASSIGN_NUM	EMP_NUM	PROJ_CODE
1	10123	1
2	10121	2
3	10123	3
4	10123	4
5	10121	1
6	10124	2
7	10124	3
8	10124	5

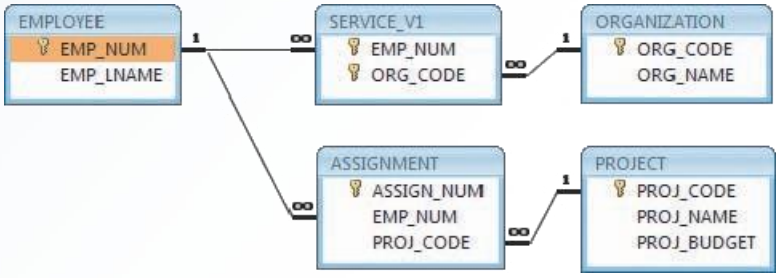
Tablo adı: ORGANİZASYON

ORG_CODE	ORG_NAME
RC	Red Cross
UW	United Way
WF	Wildlife Fund

Tablo adı: SERVICE_V1

EMP_NUM	ORG_CODE
10123	RC
10123	UW
10123	WF

İlişkisel diyagram



Bu kitapta gösterilen uygun tasarım prosedürlerini takip ederseniz, Şekil 6.11'de gösterilen sorunla karşılaşmazsınız. Özellikle, tablolarınızın aşağıdaki iki kurala uygun olduğundan emin olursanız, 4NF tartışması büyük ölçüde akademiktir:

1. Tüm öznitelikler birincil anahtara bağımlı olmalı, ancak birbirlerinden bağımsız olmalıdır.
2. Hiçbir satır bir varlık hakkında iki veya daha fazla çok değerli olgu içeremez.

Not

Bir tablo 3NF'de olduğunda ve çok değerli bağımlılıkları olmadığında **dördüncü normal formdadır (4NF).**

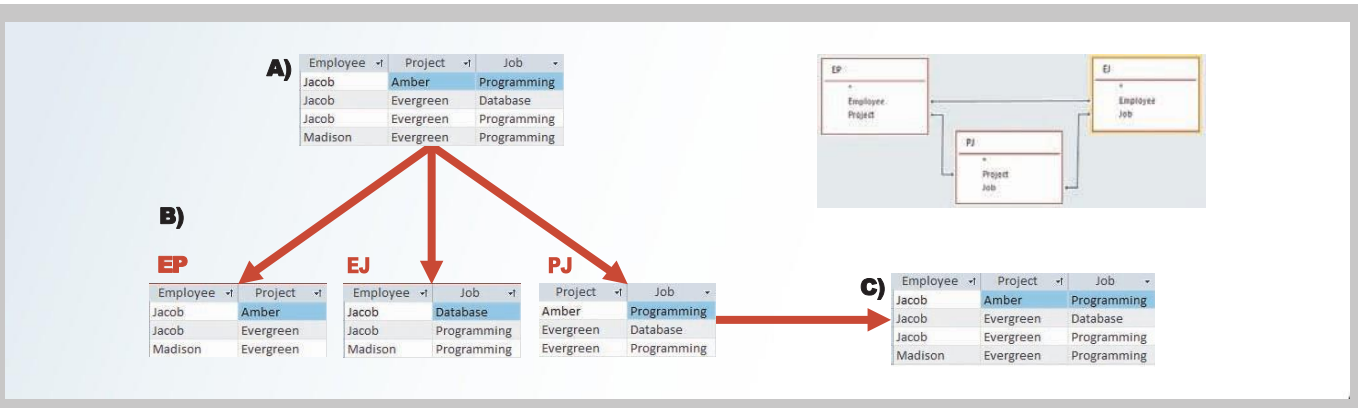
dördüncü normal form (4NF)
Bir tablo BCNF'de ise ve çok değerli bağımlılıkların çoklu bağımsız kümelerini içermiyorsa 4NF'dedir.

6-6c Beşinci Normal Form (5NF)

Öğrendiğiniz gibi normalleştirme, bir tabloyu daha küçük tablolara bölerek veri fazlalıklarını en aza indirir. Ancak, yeni tabloların modellemeye çalıştığınız gerçekleri doğru bir şekilde temsil edip etmediğini nasıl anlarsınız? **Proje** birleştirme **normal formu (PJNF)** olarak da bilinen beşinci normal **form**, bir tablonun veri kaybetmeden veya yanlış bilgi oluşturmadan daha fazla ayrıştırılamaması sorununu ele alır.
Örneğin, bu bölümde daha önce kullanılan aynı örneği kullanarak, Şekil 6.13'te gösterilen ÇALIŞAN (E), PROJE P) ve İŞ (J) durumunu göz önünde bulundurun. Bu ilişki, bir çalışanın bir veya daha fazla projede çalıştığı, bir veya daha fazla iş becerisine sahip olduğu ve her projede bir veya daha fazla iş becerisi kullandığı bir iş kuralını temsil eder.

proje birleştirme normal formu (PJNF)
Beşinci normal form (5NF) için başka bir terim. Bir tablonun zaten 4NF'de olduğu ve kayıpsız ayrışımlara sahip olmadığı bir durum.

Şekil 6.13 5NF Dönüşümü



Şekil 6.13'te gördüğünüz gibi, A panelindeki orijinal tablo B panelinde gösterildiği gibi üç ilişkiye ayrıştırılabilir: Çalışan-Proje (EP), Çalışan-İş (EJ) ve Proje-İş (PJ). Her tablo, iş kuralında belirtildiği gibi her bir nitelik kümesi için olası değer kombinasyonunu temsil eder. EP, EJ ve PJ arasındaki üç ilişkinin doğal birleşimi panelde yer alan sonucu verir C. C panelindeki verilerin orijinal verilerle aynı olduğuna dikkat edin; hiçbir veri kaybolmamış veya yeni hatalı veri eklenmemiştir. Bu durumda, tabloların **kayıpsız ayrıştırmaya** sahip olduğunu söyleyebilirsiniz. Orijinal tablonun 5NF'de olmadığı ve ayrıştırılmış tabloların 5NF'de olduğu sonucuna varabilirsiniz.

Kayıpsız ayrıştırma, ayrıştırılmış tablolar birleştirildiğinde orijinal tabloyu yeniden oluşturur. Eksik veri veya yeni hatalı .

Beşinci normal form (5NF) Bir tablo 4NF'de olduğunda ve daha fazla kayıpsız ayrıştırmaya sahip olamadığında beşinci normal formdadır (5NF).

Not

Bir tablo 4NF'de olduğunda **beşinci normal formdadır (5NF)** ve daha fazla kayıpsız ayrıştırmaya sahip olamaz.

Beşinci normal form çoğunlukla akademik bir alıştırmadır ve gerçek dünyadaki iş uygulamalarında yaygın değildir. Aşağıda 4NF ve 5NF için bazı özel gereksinimler verilmiştir:

- Dördüncü ve beşinci normal formlar en az üç nitelik gerektirir ve başka hiçbir bağımlı nitelik gerektirmez.
- 4NF'deki bir tablo 5NF'de olabilir veya olmayabilir, ancak 5NF'deki bir tablo her zaman 4NF'dedir.
- 4NF'de asal nitelik diğer iki sütunun birden fazla değerini belirler, ancak iki sütun birbirinden bağımsızdır.
- 5NF'de orijinal tablo öznitelikleri birincil özniteliklerdir; yani birincil anahtarın parçasıdırlar. Bu önemlidir ve iş kuralları tarafından belirlenir.
- 5NF'de, ayrıştırılmış tablolar arasındaki ilişki Şekil 6.13'teki ilişkisel diyagramla temsil edilir. Üç ilişki vardır: EP-PJ, PJ-EJ ve EP-EJ.
- Bir 4NF'nin 5NF'ye uymadığı çok az örnek vardır.
- Veritabanı tasarımı ve uygulama geliştirme açısından bakıldığında, 4NF'deki tablolarla çalışmak 5NF'ye göre çok daha az zahmetlidir. EMPLOYEE- PROJECT-JOB tablosuna yeni bir satır eklemek, EP, EJ ve PJ adlı üç ilişkiyi güncellemekten daha kolaydır.

Daha yüksek normal formlar değer sağlayabilirken, değerin verilerle çalışmak için gerekli ek işlemlerle sınırlı olduğunu unutmayın. 4NF ve 5NF problemlerinin ortaya çıkardığı anomaliler tipik olarak verimsizlik sorunlarıdır. Bu sorunlara 4NF ve 5NF tarafından sağlanan çözümler de DBMS için daha fazla birleştirme şeklinde ek bir yük oluşturur. Bu nedenle, bu normal formlar bir verimsizliğin diğeriyle takas edildiği durumlar olma eğilimindedir. Daha düşük normal formlar, istenmeyen bir olgunun kaldırılmasının ilgili, istenen bir olgunun da kaybolmasına neden olması gibi veri kaybı sorunlarıyla ilgilenir. Alt normal formlar genellikle çok arzu edilir ve veritabanı tasarım sürecinde her zaman göz önünde bulundurulmalıdır.

6-7 Normalleştirme ve Veritabanı Tasarımı

Şekil 6.6'da gösterilen tablolar, normalleştirme prosedürlerinin zayıf tablolardan iyi tablolar elde için nasıl kullanılabileceğini göstermektedir. Gerçek dünya veritabanlarıyla çalışmaya başladığınızda bu beceriyi uygulamaya koymak için muhtemelen bolca fırsatınız olacaktır. *Normalleştirme tasarım sürecinin bir parçası olmalıdır*. Bu nedenle, tablo yapıları oluşturulmadan *önce* önerilen varlıkların gerekli normal biçimi karşıladığından emin olun. Bölüm 3 ve 4'te tartışılan tasarım prosedürlerini izlerseniz, veri anormallikleri olasılığının düşük olacağını unutmayın. Ancak, en iyi veritabanı tasarımcılarının bile normalleştirme kontrolleri sırasında ortaya çıkan hatalar yaptığı bilinmektedir. Ayrıca, karşılaştığınız gerçek dünya veritabanlarının birçoğu yanlış tasarlanmış ya da zaman içinde uygunsuz bir şekilde değiştirilmişse anomalilerle dolu olacaktır. Bu, aslında anomali tuzakları olan mevcut veritabanlarını yeniden tasarlamanız ve değiştirmeniz istenebileceği anlamına gelir. Bu nedenle, iyi tasarım ilkeleri ve prosedürlerinin yanı sıra normalleştirme prosedürlerinin de farkında olmalısınız.

İlk olarak, bir ERD yinelemeli bir süreçle oluşturulur. İlgili varlıkları, bunların niteliklerini ve ilişkilerini tanımlayarak işe başlarsınız. Ardından sonuçları ek varlıkları tanımlamak için kullanırsınız.

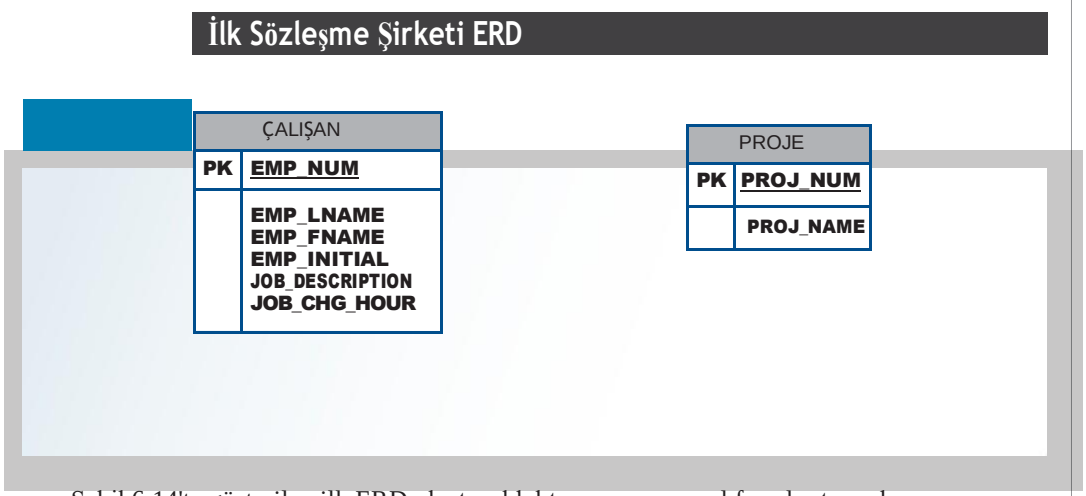
varlıklar ve öznitelikler. ERD, bir kuruluşun veri gereksinimlerinin ve işlemlerinin büyük resmini veya makro görünümünü sağlar.
İkinci olarak, normalleştirme belirli varlıkların özelliklerine odaklanır; yani, normalleştirme ERD içindeki varlıkların mikro bir görünümünü temsil eder. Ayrıca, bu bölümün önceki kısımlarında öğrendiğiniz gibi, normalleştirme işlemi ERD'ye dahil edilecek ek varlıklar ve nitelikler ortaya çıkarabilir. Bu nedenle, normalleştirmeyi ER modellemesinden ayırmak zordur; iki teknik de yinelemeli ve artımlı bir süreçte kullanılır.
Normalleştirmenin tasarım sürecindeki uygun rolünü anlamak için, önceki bölümlerde tabloları normalleştirilen müteahhitlik şirketinin işlemlerini yeniden incelemelisiniz. Bu işlemler aşağıdaki iş kuralları kullanılarak özetlenebilir:

- Şirket birçok projeyi yönetmektedir.
- Her proje birçok çalışanın hizmetini gerektirir.
- Bir çalışan birkaç farklı projeye atanabilir.
- Bazı çalışanlar bir projeye atanmaz ve özellikle bir ilgili olmayan görevleri yerine getirir. Bazı çalışanlar, tüm proje ekipleri tarafından paylaşılacak bir işgücü havuzunun parçasıdır. Örneğin, şirketin yönetici sekreteri belirli bir projeye atanmayacaktır.
- Her çalışanın saatlik faturalandırma oranını belirleyen tek bir birincil iş sınıflandırması vardır.
- Birçok çalışan aynı iş sınıflandırmasına sahip olabilir. Örneğin, şirket birden fazla elektrik mühendisi istihdam etmektedir.

  Şirketin faaliyetlerinin bu basit tanımı göz önüne alındığında, başlangıçta iki varlık ve bunların nitelikleri tanımlanır:

- PROJE (**PROJE_SAYISI**, PROJE_ADI)
- EMPLOYEE (**EMP_NUM**, EMP_LNAME, EMP_FNAME, EMP_INITIAL, JOB_DESCRIPTION, JOB_CHG_HOUR)

  Bu iki varlık Şekil 6.14'te gösterilen ilk ERD'yi oluşturur.



Şekil 6.14'te gösterilen ilk ERD oluşturulduktan sonra normal formlar tanımlanır:

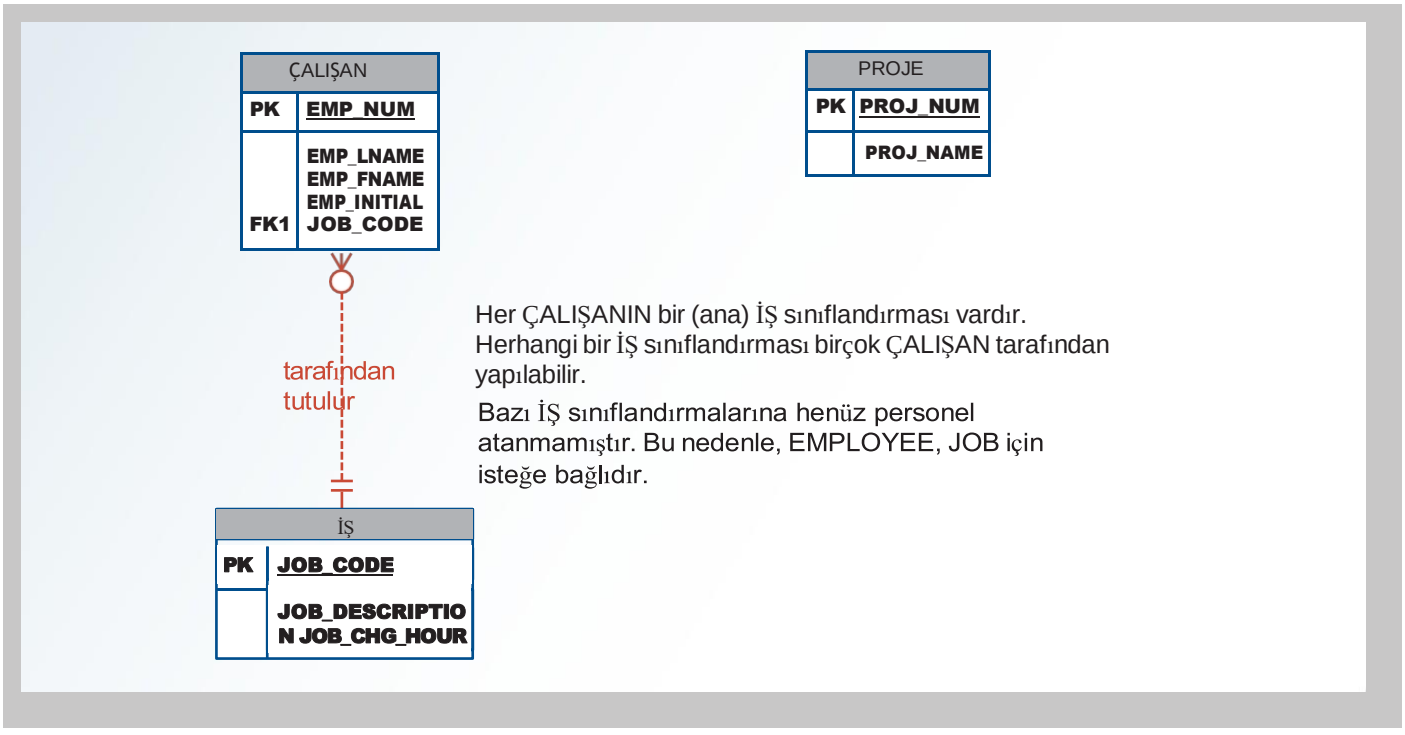
- PROJECT 3NF'dedir ve bu noktada herhangi bir değişikliğe ihtiyaç duymamaktadır.
- EMPLOYEE ek inceleme gerektirir. JOB_DESCRIPTION özniteliği Sistem Analisti, Veritabanı Tasarımcısı ve Programcı gibi iş sınıflandırmalarını tanımlar. Bu sınıflandırmalar da faturalandırma oranını, JOB_CHG_HOUR, belirler. Bu nedenle, EMPLOYEE geçişli bir bağımlılık içerir.

EMPLOYEE'nin geçişli bağımlılığı kaldırıldığında üç varlık ortaya çıkar:

- PROJE (**PROJE_SAYISI**, PROJE_ADI)
- ÇALIŞAN (**EMP_NUM**,EMP_LNAME,EMP_FNAME,EMP_INITIAL, JOB_CODE)
- JOB (**JOB_CODE**, JOB_DESCRIPTION, JOB_CHG_HOUR)

Normalleştirme işlemi ek bir varlık (JOB) ortaya çıkardığından, ilk ERD Şekil 6.15'te gösterildiği gibi değiştirilir.

Şekil 6.15 Modifiye Müteahhitlik Şirketi ERD

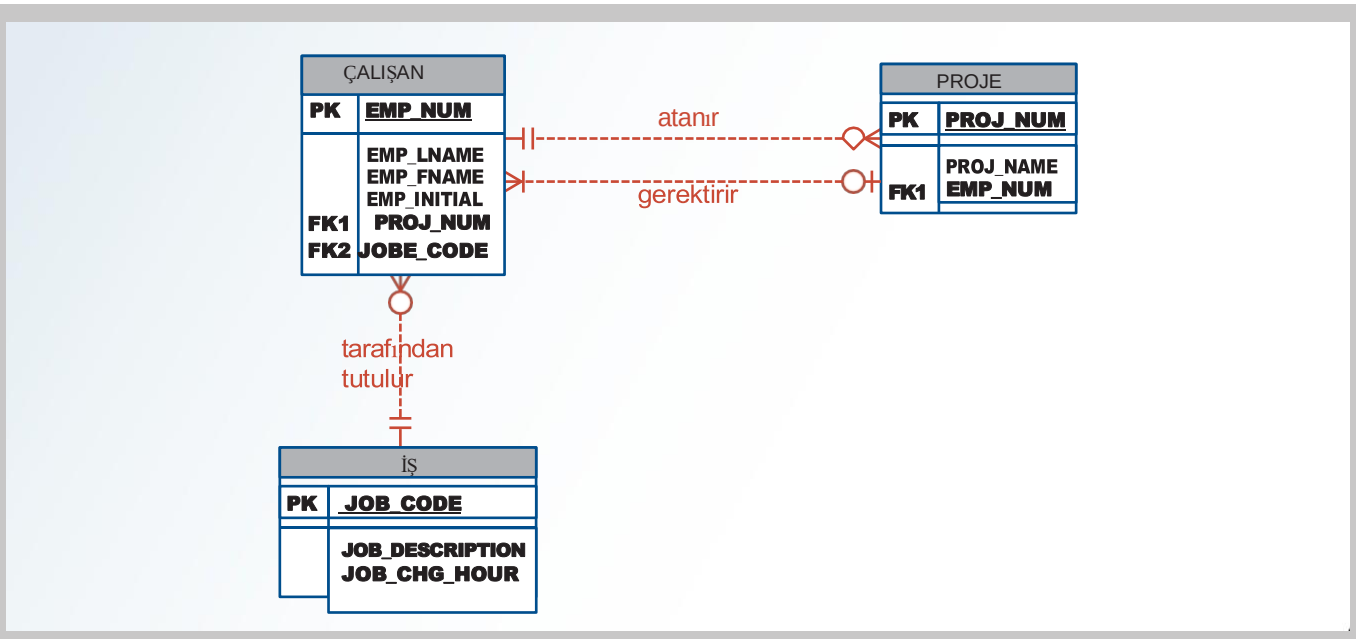


ÇALIŞAN ve PROJE arasındaki M:N ilişkisini temsil etmek için iki 1:M ilişkisinin kullanılabileceğini düşünebilirsiniz - bir çalışan birçok projeye atanabilir ve her projeye birçok çalışan atanabilir. (Bkz. Şekil 6.16.) Ne yazık ki, bu gösterim doğru şekilde uygulanamayacak bir tasarım ortaya çıkarmaktadır.

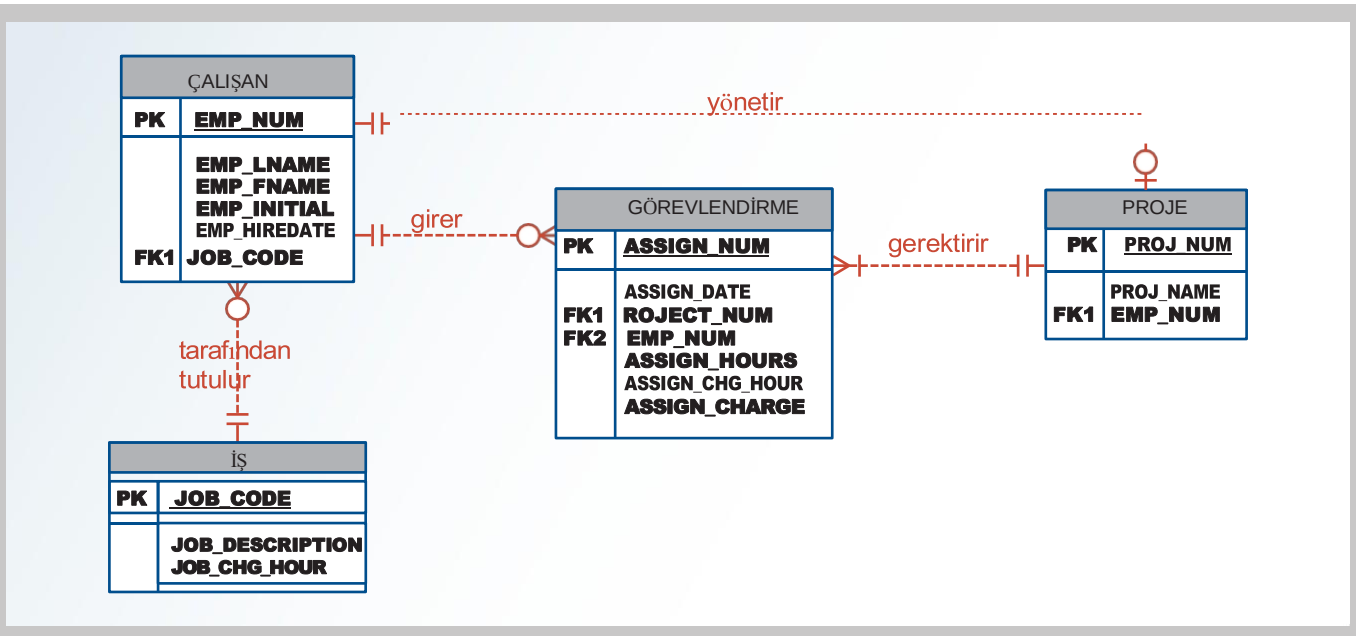
EMPLOYEE ve PROJECT arasındaki M:N ilişkisi uygulanamayacağından, Şekil 6.16'daki ERD, çalışanların projelere atanmasını izlemek için ASSIGNMENT varlığını içerecek şekilde değiştirilmelidir, böylece Şekil 6.17'de gösterilen ERD ortaya çıkar. Şekil 6.17'deki ASSIGNMENT varlığı, yabancı anahtarları olarak PROJECT ve EMPLOYEE varlıklarının birincil anahtarlarını kullanır. Ancak, bu uygulamada, bileşik bir birincil anahtar kullanımından kaçınmak ve seçilen veri ayrıntı düzeyini desteklemek için ASSIGNMENT varlığının vekil birincil anahtarının ASSIGN_NUM olduğuna dikkat edin. Bu nedenle, EMPLOYEE ile ASSIGNMENT arasındaki "enters" ilişkisi ve PROJECT ile ASSIGNMENT arasındaki "requires" ilişkisi zayıf veya tanımlayıcı olmayan olarak gösterilir.

Şekil 6.17'de ASSIGN_HOURS özniteliği ASSIGNMENT adlı bileşik varlığa atanmıştır. Her projenin yöneticisi hakkında ayrıntılı bilgiye ihtiyacınız olacağından, bir "manages" ilişkisinin oluşturulması yararlıdır. "Manages" ilişkisi PROJECT içindeki yabancı anahtar aracılığıyla gerçekleştirilir. Son olarak, sistemin ek bilgi üretme kabiliyetini geliştirmek için bazı ek nitelikler oluşturulabilir. Örneğin, çalışanların işe alındıkları tarihi (EMP_HIREDATE) takip etmek için eklemek isteyebilirsiniz.

Şekil 6.16 Yanlış M:N İlişki Temsili



Şekil 6.17 Nihai Yüklenici Şirket ERD



çalışanların uzun ömürlülüğü. Bu son değişikliğe dayanarak, model dört varlık ve bunların niteliklerini içermelidir:

PROJE (**PROJE_NUM**, PROJE_ADI, EMP_NUM)

EMPLOYEE (**EMP_NUM**, EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_ HIREDATE, JOB_CODE)

JOB (**JOB_CODE**, JOB_DESCRIPTION, JOB_CHG_HOUR)

ASSIGNMENT (**ASSIGN_NUM**, , PROJECT_NUM, EMP_NUM, ASSIGN_ HOURS, ASSIGN_CHG_HOUR, ASSIGN_CHARGE)

Şekil 6.18 Uygulanan Veritabanı

Tablo adı: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-97	503
104	Ramoras	Anne	K	15-Nov-88	501
105	Johnson	Alice	K	01-Feb-94	502
106	Smithfield	William		22-Jun-05	500
107	Alonzo	Maria	D	10-Oct-94	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501
110	Olenko	Gerald	A	11-Dec-96	505
111	Wabash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-95	507
113	Joenbrood	Delbert	K	15-Nov-94	508
114	Jones	Annelise		20-Aug-91	508
115	Bawangi	Travis	B	25-Jan-90	501
116	Pratt	Gerald	L	05-Mar-95	510
117	Williamson	Angie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-06	510

Veritabanı adı: Ch06_ConstructCo

Tablo adı: JOB

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
500	Programmer	35.75
501	Systems Analyst	96.75
502	Database Designer	105.00
503	Electrical Engineer	84.50
504	Mechanical Engineer	67.90
505	Civil Engineer	55.78
506	Clerical Support	26.87
507	DSS Analyst	45.95
508	Applications Designer	48.10
509	Bio Technician	34.55
510	General Support	18.36

Tablo adı: PROJE

PROJ_NUM	PROJ_NAME	EMP_NUM
15	Evergreen	105
18	Amber Wave	104
22	Rolling Tide	113
25	Starflight	101

Tablo adı: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-22	15	103	2.6	84.50	219.70
1002	04-Mar-22	18	118	1.4	18.36	25.70
1003	05-Mar-22	15	101	3.6	105.00	378.00
1004	05-Mar-22	22	113	2.5	48.10	120.25
1005	05-Mar-22	15	103	1.9	84.50	160.55
1006	05-Mar-22	25	115	4.2	96.75	406.35
1007	05-Mar-22	22	105	5.2	105.00	546.00
1008	05-Mar-22	25	101	1.7	105.00	178.50
1009	05-Mar-22	15	105	2.0	105.00	210.00
1010	06-Mar-22	15	102	3.8	96.75	367.65
1011	06-Mar-22	22	104	2.6	96.75	251.55
1012	06-Mar-22	15	101	2.3	105.00	241.50
1013	06-Mar-22	25	114	1.8	48.10	86.58
1014	06-Mar-22	22	111	4.0	26.87	107.48
1015	06-Mar-22	25	114	3.4	48.10	163.54
1016	06-Mar-22	18	112	1.2	45.95	55.14
1017	06-Mar-22	18	118	2.0	18.36	36.72
1018	06-Mar-22	18	104	2.6	96.75	251.55
1019	06-Mar-22	15	103	3.0	84.50	253.50
1020	07-Mar-22	22	105	2.7	105.00	283.50
1021	08-Mar-22	25	108	4.2	96.75	406.35
1022	07-Mar-22	25	114	5.8	48.10	278.98
1023	07-Mar-22	22	106	2.4	35.75	85.80

Tasarım süreci artık doğru yoldadır. ERD işlemleri doğru bir şekilde temsil eder ve varlıklar artık 3NF'ye uygunluklarını yansıtır. Normalleştirme ve ER modelleme kombinasyonu, varlıkları artık uygun tablo yapılarına dönüştürülebilen kullanışlı bir ERD ortaya çıkarır. Şekil 6.16'da, PROJE'nin "yönetir" ilişkisinde ÇALIŞAN'a isteğe bağlı olduğuna dikkat edin. Bu opsiyonelliğin nedeni tüm çalışanların projeleri yönetmemesidir. Nihai veritabanı içeriği Şekil 6.18'de gösterilmektedir.

6-8 Denormalizasyon

En uygun ilişkisel veritabanı uygulamasının tüm tabloların en azından 3NF'de olmasını gerektirdiğini unutmamak önemlidir. İyi bir ilişkisel VTYS, normalleştirilmiş ilişkileri, veri anormalliklerine neden olabilecek gereksiz fazlalıklardan arındırılmış ilişkileri yönetmede mükemmeldir.

Normalleştirilmiş ilişkilerin oluşturulması önemli bir veritabanı tasarım hedefi olsa da, bu tür birçok hedeften yalnızca biridir. İyi bir veritabanı tasarımı, işleme (veya raporlama) gereksinimlerini ve işlem hızını da dikkate alır. Normalleştirme ile ilgili sorun, tablolar normalleştirme gereksinimlerine uyacak şekilde ayrıştırıldıkça, veritabanı tablolarının sayısının artmasıdır. Bu nedenle, bilgi üretmek için verilerin çeşitli tablolardan bir araya getirilmesi gerekir. Çok sayıda tablonun birleştirilmesi, ek giriş/çıkış (I/O) işlemleri ve işleme mantığı gerektirerek sistem hızını düşürür. Çoğu ilişkisel veritabanı sistemi birleştirme işlemlerini çok verimli bir şekilde gerçekleştirebilir. Bununla birlikte, nadiren ve ara sıra karşılaşılan durumlar bir dereceye kadar normalleştirmeye izin verebilir, böylece işlem hızı artırılabilir.

Daha yüksek işlem hızı avantajının, veri anomalilerinin dezavantajına karşı dikkatlice tartılması gerektiğini unutmayın. Öte yandan, bazı anomaliler yalnızca teorik olarak ilgi çekicidir. Örneğin, gerçek dünyadaki bir veritabanı ortamındaki kişiler, birincil anahtarı müşteri numarası olan bir CUSTOMER tablosunda ZIP_CODE'un CITY'yi belirlemesinden endişe etmeli midir? Aşağıdakiler için ayrı bir tablo üretmek gerçekten pratik midir ZIP (**ZIP_CODE**, ŞEHIR) CUSTOMER tablosundan geçişli bir bağımlılığı ortadan kaldırmak için? (Posta listeleri üretme işindeyseniz belki de bu soruya değişir). Daha önce de açıklandığı gibi, normalleştirilmemiş ilişkiler ve gereksiz verilerle ilgili sorun, ekleme, güncelleme ve silme anormallikleri olasılığı nedeniyle veri bütünlüğünün tehlikeye girebilmesidir. Tavsiye basittir: normalleştirme işlemi sırasında sağduyulu olun.

Ayrıca, veritabanı tasarım süreci, bazı durumlarda, önceki örnekte görüldüğü gibi, modele az da olsa gereksiz veri ekleyebilir. Bu da aslında "denormalize" ilişkiler yaratır. Tablo 6.6, genellikle veritabanı uygulamalarında bulunan bazı yaygın veri fazlalığı örneklerini göstermektedir.

Tablo 6.6 Yaygın Denormalizasyon Örnekleri

Dava	Örnek	Gerekçe ve Kontroller
Yedek veri	ZIP, CITY'yi belirlediğinde ZIP ve CITY özniteliklerinin AGENT tablosunda saklanması (bkz. Şekil 2.2)	Ekstra birleştirme işlemlerinden kaçının Program posta koduna göre şehri (açılır kutu) doğrulayabilir
Türetilmiş veriler	STU_HRS, STU_CLASS'ı belirlediğinde STU_HRS ve STU_CLASS'ın (öğrenci sınıflandırması) saklanması (bkz. Şekil 3.28)	Ekstra birleştirme işlemlerinden kaçının Program, öğrenci saatlerine göre sınıflandırmayı (arama) doğrulayabilir
Önceden toplanmış veriler (ayrıca türetilmiş veriler)	Öğrenci not ortalaması (STU_GPA) toplam değerinin ENROLL ve COURSE tablolarından hesaplanabildiği durumlarda STUDENT tablosunda saklanması (bkz. Şekil 3.28)	Ekstra birleştirme işlemlerinden kaçının Program, bir not her girildiğinde veya güncellendiğinde not ortalamasını hesaplar STU_GPA yalnızca idari rutin aracılığıyla güncellenebilir
Bilgi gereksinimleri	Rapor verilerini tutmak için geçici bir denormalize tablo kullanmak; bu, sütunların tabloda satır olarak depolanan verileri temsil ettiği bir tablo raporu oluştururken gereklidir (bkz. Şekil 6.18 ve 6.19)	Raporun gerektirdiği verileri düz SQL kullanarak oluşturmak imkansız Tablo tutmaya gerek yok Rapor tamamlandığında geçici tablo silinir İşleme hızı sorun değildir

Raporlama gereklilikleri nedeniyle normalleştirme ihtiyacının daha kapsamlı bir örneği, her satırda ders verilen son dört dönem boyunca alınan puanların listelendiği bir fakülte değerlendirme raporu örneğidir. (Bkz. Şekil 6.19.)

Bu rapor yeterince basit görünse de sorun, verilerin her satırın belirli bir dönemdeki belirli bir öğretim üyesi için farklı bir puanı temsil ettiği nor- malize bir tabloda saklanmasıdır. (Bkz. Şekil 6.20.)