YILDIZ TEKNİK ÜNİVERSİTESİ

Normalizasyon ve Karmaşık Veri Tipleri

Öğr. Gör. Dr. Yasemin Topuz

Yıldız Teknik Üniversitesi

BLM3041 – Veritabanı Yönetimi

19 Aralık 2025 | YTÜ Bilgisayar Mühendisliği

Neler konuşacağız?

- Birincil Anahtar (Primary Key) Tabanlı Normal Formlar
  - 1 NF
  - 2 NF
  - 3 NF
  - BNCF
- Karmaşık Veri Tipleri (Complex Data Types)
  - Yarı Yapılandırılmış Veri
  - Nesne Yönelimli
  - Metinsel Veri
  - Uzamsal Veri

Fonksiyonel Bağımlılıklar İçin Çıkarım Kuralları (Inference Rules)

- Bir FD kümesi F verildiğinde, bu kümeden yeni FD'ler türetebiliriz.
- Bu çıkarım işlemleri **Armstrong** kuralları ile yapılır.

Armstrong Kuralları

IR1 – Yansıtma (Reflexive Rule)

- Eğer Y, X'in alt kümesi ise: X → Y geçerlidir.
- Örnek: {A, B} → A

IR2 – Genişletme (Augmentation Rule)

- Eğer X → Y geçerliyse: XZ → YZ de geçerlidir.
  (Not: XZ = X ∪ Z)

IR3 – Geçişlilik (Transitive Rule)

- Eğer: X → Y ve Y → Z ise X → Z geçerlidir.
- TC_No → Kişi, Kişi → Adres ⇒ TC_No → Adres

- ✓ IR1, IR2 ve IR3 tam ve doğru (sound and complete) bir kural kümesidir.
- ✓ Bu üçü doğru sonuç üretir.
- ✓ Bu üçünden hareketle tüm diğer FD kuralları türetilebilir.

Fonksiyonel Bağımlılıklar İçin Çıkarım Kuralları (Inference Rules)

- IR1, IR2 ve IR3'e ek olarak, pratikte çok faydalı olan üç ek çıkarım kuralı daha vardır.
- Bu kurallar aslında Armstrong kurallarından (IR1–IR3) türetilebilir.

1. Ayrıştırma (Decomposition Rule)

- Eğer: X → YZ geçerliyse: X → Y ve X → Z de geçerlidir.

  TC_No → {İsim, Soyisim} ⇒ TC_No → İsim ve TC_No → Soyisim

2. Birleşim (Union Rule)

- Eğer: X → Y ve X → Z ise: X → YZ

  TC_No → İsim ve TC_No → Adres ⇒ TC_No → {İsim, Adres}

3. Sözde-Geçişlilik (Pseudotransitivity Rule)

- Eğer: X → Y ve WY → Z ise: WX → Z

  DersKodu → Hoca {Bölüm, Hoca} → Ofis ⇒ {Bölüm, DersKodu} → Ofis

Bir İlişkinin Anahtarını Belirleme (Key Bulma)

- Bir ilişkinin anahtarı, tüm öznitelikleri (R'nin tamamını) tek başına belirleyebilen en küçük öznitelik kümesidir. Anahtar bulmak için X^+ (kapsama-closure) hesabı yapılır.

Örnek – R Şeması ve (F) Fonksiyonel Bağımlılıklar

- R = {A, B, C, D, E, F, G, H, I, J}
- F = AB → C A → DE B → F F → GH D → IJ
- Amaç: Hangi öznitelik kümesi R'nin tamamını belirler?
- A^+ = ? A → DE D → IJ ⇒ A^+ = {A, D, E, I, J}. Daha fazla genişlemiyor → A anahtar değildir.
- B^+ = ? B → F ⇒ B^+ = {B, F}, F → GH ⇒ {B, F, G, H}
  R'nin tamamını kapsamıyor → B anahtar değildir.
- AB^+ = ? AB → C A → DE B → F F → GH D → IJ
- Hepsini toplarsak: $AB^+ = \{A, B, C, D, E, F, G, H, I, J\} = R$
- AB tüm R'yi kapsıyor → AB bir anahtardır. A veya B tek başına yetmediği için AB minimaldir.
- **Sonuç: Tek aday anahtar = AB**

Bir İlişkinin Anahtarını Belirleme (Key Bulma)

- Bir ilişkinin anahtarı, tüm öznitelikleri (R'nin tamamını) tek başına belirleyebilen en küçük öznitelik kümesidir. Anahtar bulmak için X^+ (kapsama) hesapları yaparız.

Örnek – R Şeması ve (G) Fonksiyonel Bağımlılıklar

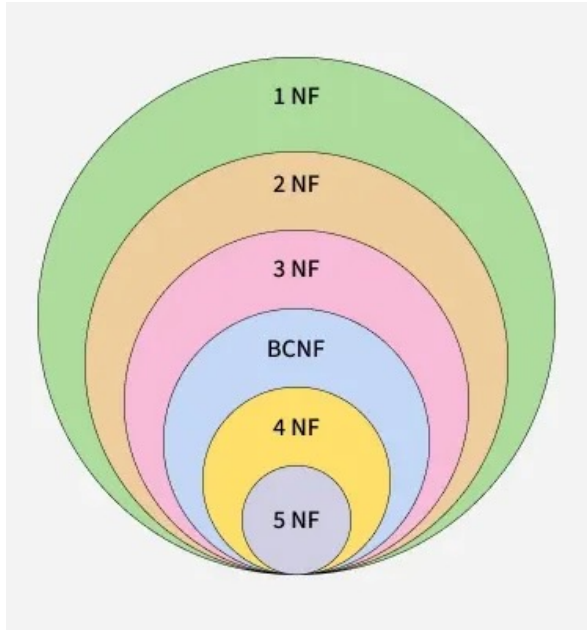
- R = {A, B, C, D, E, F, G, H, I, J}
- G = AB → C BD → EF AD → GH A → I H → J **Amaç:** Anahtarı bulmak.
- **A^+ :** A → I AD → GH (ama D yok) A^+ = {A, I} A tek başına yetmez → anahtar değil.
- **B^+ :** B'den başlayan kurallar yok → B^+ = {B} → anahtar değil.
- **AB^+:** Başlangıç: {A, B} AB → C ⇒ {A, B, C} A → I ⇒ {A, B, C, I}
- Burada takılıyoruz; EF, GH, J gibi özellikleri elde edemiyoruz çünkü D veya H yok. AB anahtar değildir.
- **ABD^+:** Başlangıç: {A, B, D} AB → C ⇒ {A, B, C, D} A → I ⇒ {A, B, C, D, I}
  BD → EF ⇒ {A, B, C, D, E, F, I} AD → GH ⇒ {A, B, C, D, E, F, G, H, I}.
  H → J ⇒ {A, B, C, D, E, F, G, H, I, J}
- **Hepsini toplarsak:** $ABD^+ = R$ ABD R'nin tamamını belirliyor → ABD bir aday anahtardır.

Birincil Anahtar (Primary Key) Tabanlı Normal Formlar

İlişkilerin Normalleştirilmesi

Normalization (Normalizasyon)

- **Tanım:** Kötü tasarlanmış (gereksiz tekrar içeren, anomali üreten) ilişkilerin, öznitelikleri daha küçük ve daha sağlıklı ilişkilere bölünerek iyileştirilmesi işlemidir.
  - Tekrarlanan bilgiyi azaltmak
  - Güncelleme / silme / ekleme anomallerini önlemek
  - Daha tutarlı, esnek ve doğru veri modeli oluşturmak



Normal Form (Normal Form / NF)

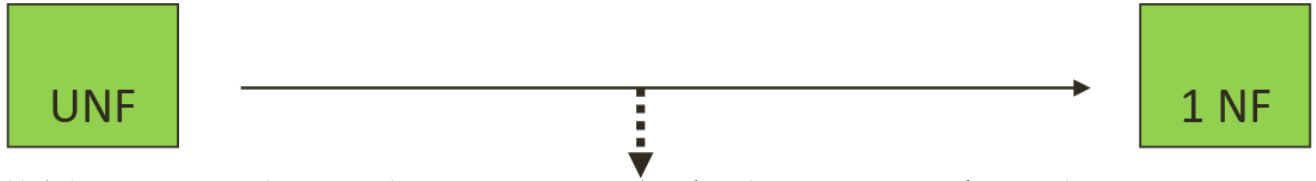
- **Tanım:** Bir ilişkinin hangi kurallara uyduğunu belirleyen kalite ölçüsüdür. Anahtarlar ve fonksiyonel bağımlılıklar kullanılarak, ilişkilerin:
  - 1NF, 2NF, 3NF, BCNF

  gibi normal formlardan hangisine uygun olduğu belirlenir.

Denormalizasyon: Normalizasyon sırasında ayırdığımız tabloları, performans veya kullanım kolaylığı için tekrar birleştirme işlemidir.

Birinci Normal Form (1NF): Yinelenen Kayıtların Ortadan Kaldırılması

- Birinci Normal Form (1NF), bir veritabanı tablosunun yapısının, yönetilmesini ve sorgulanmasını kolaylaştıracak şekilde düzenlenmesini sağlar.
- Tekrarlayan veri gruplarının elimine edilmesi ve atomic'lik (veri kendini içerir ve bağımsızdır) sağlanmasıdır.
- Bu, ilişkisel veritabanı tasarımında gereksiz tekrarları azaltmanın, veri bütünlüğünü iyileştirmenin ve anormallikleri azaltmanın ilk ve temel adımıdır.
- Bir ilişki (tablo), aşağıdaki durumlarda Birinci Normal Formda (1NF) olarak adlandırılır:



  - Tüm özellikler (sütunlar) yalnızca atomik (bölünemez) değerler içerir.
  - Her sütun tek bir türe ait değerler içerir.
  - Her kayıt (satır) benzersizdir, yani birincil anahtar aracılığıyla tanımlanabilir.
  - Hiçbir satırda tekrar eden grup veya dizi bulunmamaktadır.

Birinci Normal Form (1NF): Yinelenen Kayıtların Ortadan Kaldırılması

COURSES Table

ID	Name	Courses
1	A	c1, c2
2	E	c3
3	M	c2, c3

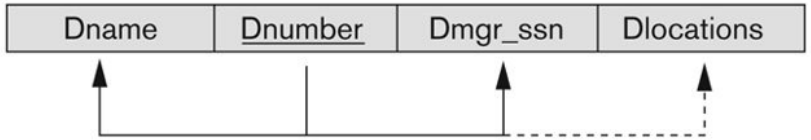
COURSES Table

ID	Name	Courses
1	A	c1
1	A	c2
2	E	c3
3	M	c2
3	M	c3

Birinci Normal Form (1NF): Yinelenen Kayıtların Ortadan Kaldırılması

(a)

DEPARTMENT



(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

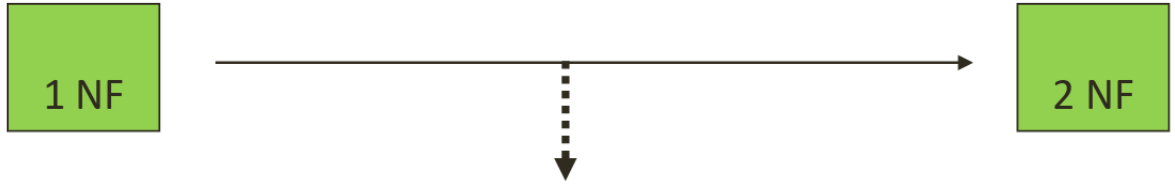
(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

İkinci Normal Form (2NF): Kısmi Bağımlılığın Ortadan Kaldırılması

- İkinci Normal Form (2NF), tam işlevsel bağımlılık kavramına dayanır. Fonksiyonel bağımlılıklar göz önünde bulundurularak tabloların bölünmesidir.
- **Tam İşlevsel Bağımlılık**, bölünen tablolardan birinin birincil anahtarı ile bölünen diğer tablodaki birincil olamayan bir alan arasındaki bağımlığa denir.
- **Bir tablonun 2NF'de olması için öncelikle aşağıdaki şartları karşılaması gerekir.**



- **1NF Gereksinimlerini Karşılayın:** Tablo öncelikle Birinci Normal Formu (1NF) karşılamalıdır , yani
  - Tüm sütunlar tek ve bölünemez değerler içerir. Sütun grupları tekrar etmemelidir.
- **Kısmi Bağımlılıkları Ortadan Kaldırın:** Kısmi bağımlılık, asal olmayan bir özniteliğin (aday anahtarın parçası olmayan) bileşik birincil anahtarın tamamına değil, yalnızca bir kısmına bağlı olması durumunda ortaya çıkar.
  - Birincil anahtar üzerindeki kısmi bağımlılıkları oluşturan özellikler yeni bir ilişkiye taşınır

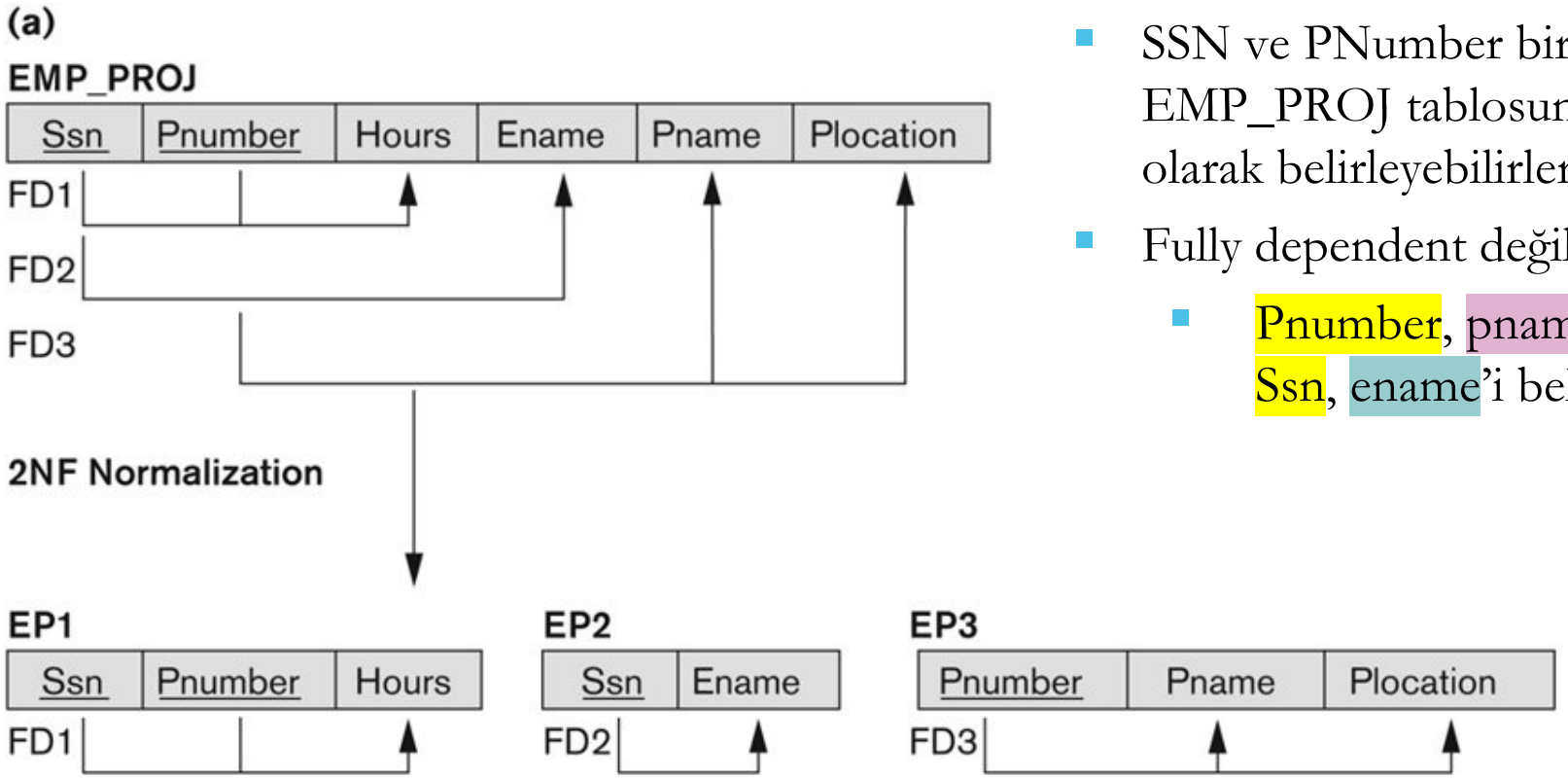
İkinci Normal Form (2NF): Kısmi Bağımlılığın Ortadan Kaldırılması

Kısmi Bağımlılık Nedir?

- A → B fonksiyonel bağımlılığı, B'nin A'ya fonksiyonel olarak bağımlı olması ve B'nin A'nın herhangi bir alt kümesi tarafından belirlenebilmesi durumunda kısmi bağımlılık olarak adlandırılır.

    - **Örnek:** Bileşik bir anahtar (ÖğrenciID, DersID) için, "ÖğrenciAdı" yalnızca "ÖğrenciID"ye bağlıysa ve anahtarın tamamına bağlı değilse, 2NF'yi ihlal eder. Normalleştirmek için, ÖğrenciAdı'nı yalnızca "ÖğrenciID"ye bağlı olduğu ayrı bir tabloya taşıyın.

İkinci Normal Form (2NF): Kısmi Bağımlılığın Ortadan Kaldırılması

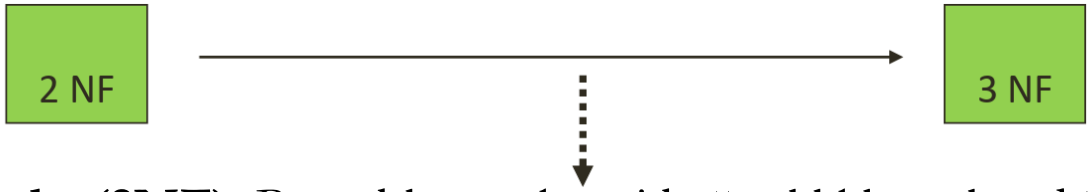


- SSN ve PNumber birlikte primary key olduğundan EMP_PROJ tablosundaki tüm sütunları fonksiyonel olarak belirleyebilirler.
- Fully dependent değildir.
  - Pnumber, pname ve plocation'ı belirlerken Ssn, ename'i belirleyebilir.

Üçüncü Normal Form (3NF): Geçişli Bağımlılığın Ortadan Kaldırılması

- 3NF'ye ulaşmak, veritabanı yapısının geçişli bağımlılıklardan arındırılmasını sağlayarak veri anormalliklerinin olasılığını azaltır. 2NF'deki tablolar 1NF'ye kıyasla daha az fazlalığa sahip olsa da, yine de güncelleme anormallikleri gibi sorunlarla karşılaşabilirler .

Bir ilişki, aşağıdaki iki koşulu sağlıyorsa Üçüncü Normal Formdadır (3NF):



- **İkinci Normal Formdadır (2NF)**: Bu, tablonun kısmi bağımlılıkları olmadığı anlamına gelir (yani, asal olmayan hiçbir öznitelik, aday anahtarın bir parçasına bağımlı değildir).
- **Asal olmayan öznitelikler için geçişli/dolaylı bağımlılık yoktur:** Daha basit bir ifadeyle, anahtar olmayan hiçbir öznitelik başka bir anahtar olmayan özniteliğe bağlı olmamalıdır. İlişkide birincil anahtara dolaylı bağımlı özellikler, yeni bir ilişkiye taşınır.

Geçişli bağımlılık, bir asal olmayan özniteliğin doğrudan birincil anahtara bağlı olmak yerine, başka bir asal olmayan özniteliğe bağlı olursa ortaya çıkar. Bu durum veritabanında fazlalık ve tutarsızlıklara yol açabilir.

Üçüncü Normal Form (3NF): Geçişli Bağımlılığın Ortadan Kaldırılması

CANDIDATE Table

CAND_NO	CAND_NAME	CAND_STATE	CAND_COUNTRY	CAND_AGE
1	Amayra	West Bengal	India	18
2	Rihaan	Haryana	India	17
3	Manya	Haryana	India	19

CAND_NO (X) ve CAND_COUNTRY (Z) arasındaki geçişli bağımlılıktan kaynaklanmaktadır:

- CAND_NO (X) → CAND_STATE (Y)
- CAND_STATE (Y) → CAND_COUNTRY (Z)

Bu, CAND_COUNTRY'nin CAND_STATE aracılığıyla CAND_NO'ya dolaylı olarak bağımlı olduğu anlamına gelir.

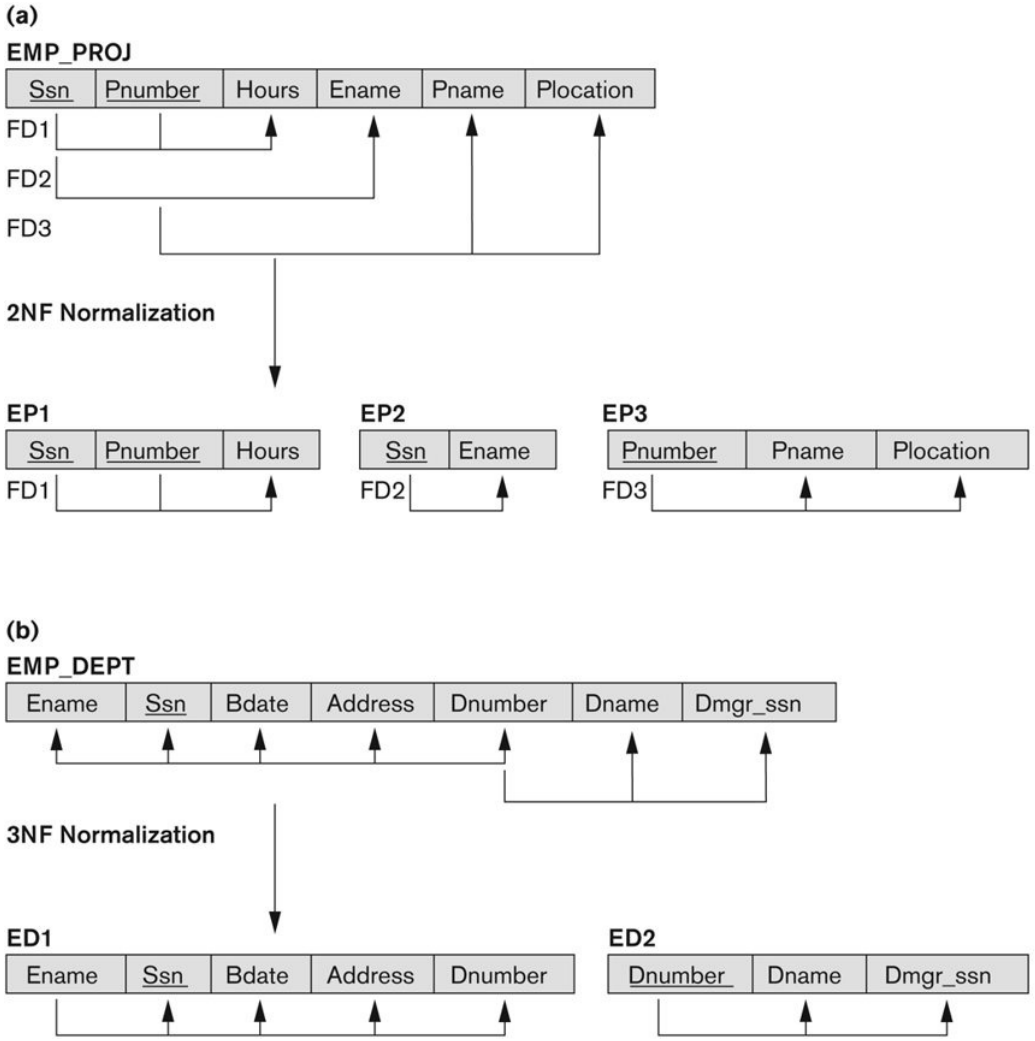
CANDIDATE Table

CAND_NO	CAND_NAME	CAND_STATE	CAND_AGE
1	Amayra	West Bengal	18
2	Rihaan	Haryana	17
3	Manya	Haryana	19

STATE_COUNTRY Table

CAND_STATE	CAND_COUNTRY
West Bengal	India
Haryana	India

Üçüncü Normal Form (3NF): Geçişli Bağımlılığın Ortadan Kaldırılması



- Ssn → Ename, Bdate, Address (çalışana ait bilgiler)
- Dnumber → Dname, Dmgr_ssn (departmana ait bilgiler)

Sorun: Departman bilgileri (Dname, Dmgr_ssn) çalışanın SSN'i ile transitif olarak bağlıdır.

- Ssn → Dnumber → Dname, Dmgr_ssn
- Bu bir transitif bağımlılıktır ve 3NF kuralını ihlal eder, çünkü:
  - Çalışan bilgileri → çalışan anahtarına bağlı olmalı
  - Departman bilgileri → departman anahtarına bağlı olmalı

Tam İşlevsel Bağımlılık, Kısmi Bağımlılık ve Dolaylı Bağımlılık

- **Tam İşlevsel (Fonksiyonel) Bağımlılık**: A ve B bir ilişki, eğer B işlevsel olarak A' ya bağımlı ise, bu durumda B özellik kümesi A özellik kümesine tam işlevsel bağımlıdır.
- A → B ise A fonksiyonel olarak B 'yi tanımlar. A'nın değeri biliniyorsa, B'nin değeri tek ve kesin olarak belirlenebilir. A'dan herhangi bir özelliği çıkarırsan, bağımlılık bozulur.

OGRENCI

numara	adsoyad	bolum	sınıf	tck
1	Ali	Bilgisayar	1	11
2	Fatma	Elektronik	2	22
3	Arda	Makine	1	33

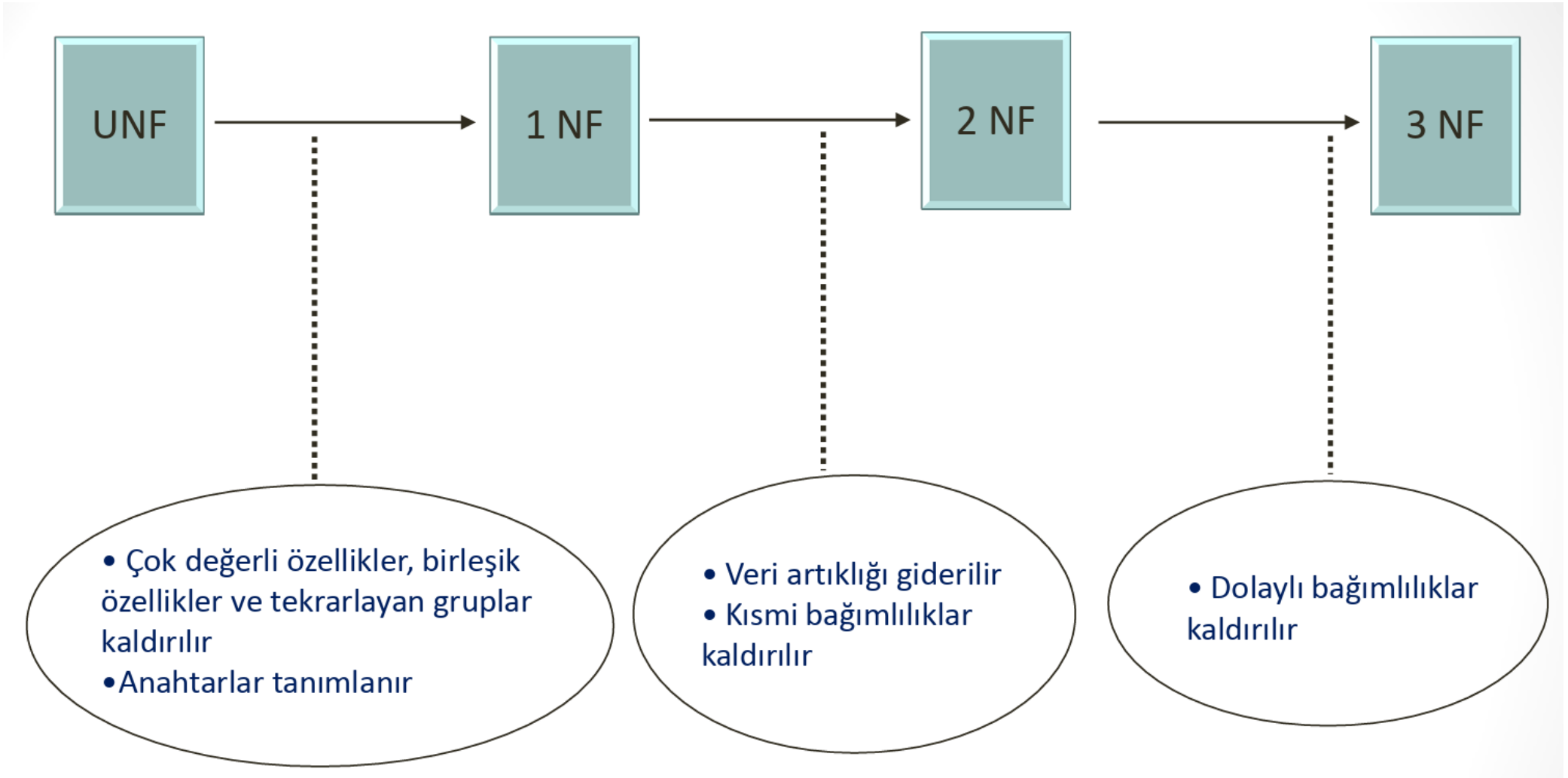
- Yukarıdaki OGRENCI tablosunu ele aldığımız zaman hangi bağımlılıklardan söz edebiliriz?
  - numara → adsoyad
  - numara → adsoyad, bolum, sınıf, tck
  - tck → numara, adsoyad, bolum, sınıf
  - bolum → sınıf diyemeyiz
    Aynı bölümde farklı sınıflarda öğrenciler olabilir.
  - sınıf → bolum diyemeyiz
    Aynı sınıfta farklı bölümler olabilir.

Tam İşlevsel Bağımlılık, Kısmi Bağımlılık ve Dolaylı Bağımlılık

- **Kısmi Bağımlılık (Partial Dependency):** A ve B özellik kümeleri işlevsel bağımlı ise (A → B) ve A özellikler kümesinden herhangi bir özelliğin çıkarılması bu bağımlılığı bozmazsa,
  - A → B bağımlılığına kısmi bağımlılık denir.
  - A'nın değeri biliniyorsa, B'nin değeri tek ve kesin olarak belirlenebilir.
  - A'nın bir parçası bile B'yi belirleyebiliyorsa, bu kısmi bağımlılıktır.
  - Genelde bileşik anahtar (birden fazla alanlı anahtar) varsa görülür.

- **Dolaylı Bağımlılık (Transitive Dependency):** A, B ve C özellik kümelerini içeren bir ilişkide
  - A → B ve B → C işlevsel bağımlılıkları bulunmakta ise,
  - A, C'yi doğrudan değil B aracılığıyla belirler.

Normalizasyon Aşamaları



Normalizasyon Aşamaları

- County_name ve Lot# birlikte aday anahtardır. AB → C
- Ancak county_name tek başına Tax_rate'i belirleyebiliyor. Bu kısmi bağımlılıktır.
- Area anahtar olmadığı halde Price'ı belirleyebiliyor. Dolaylı bağımlılıktır.

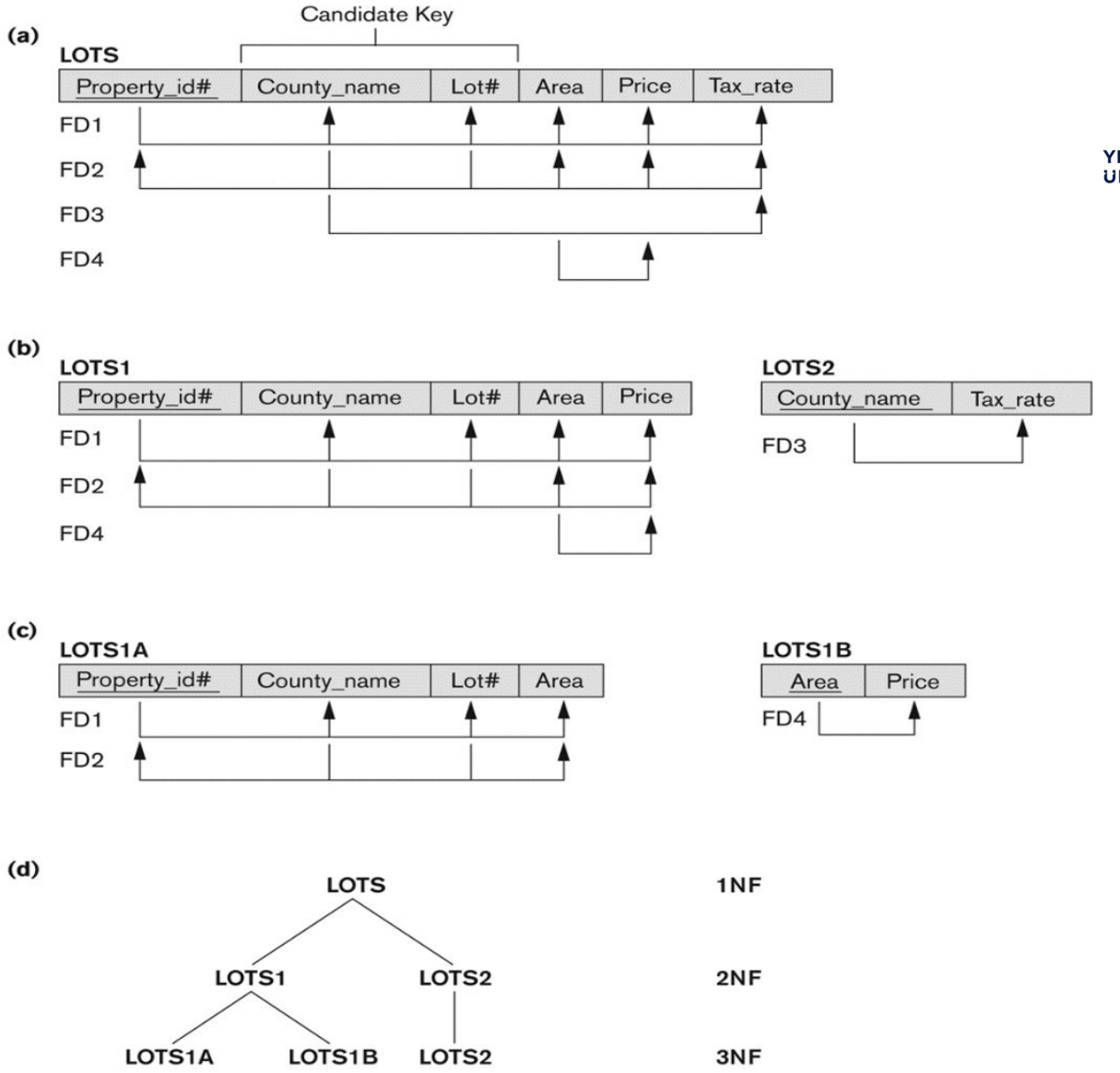


Figure 10.11
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

Normalleştirme Örneği (1NF)

Müşteri no	ad	soyad	tel	adres	şehir	Plaka kodu	Ürün	miktar	fiyat
1	Fatma	Kara	11111	Fatmanın Adresi	Muğla	48	Tablet	1	600
2	Ali	Can	22222	Alinin adresi	Burdur	15	RAM	2	80
2	Ali	Can	22222	Alinin adresi	Burdur	15	DVD	100	5
2	Ali	Can	22222	Alinin adresi	Burdur	15	Mouse	10	120
3	Veli	Koç	33333	Velinin adresi	İstanbul	34	RAM	10	350
3	Veli	Koç	33333	Velinin adresi	İstanbul	34	Klavye	20	200
4	Hamdi	Turan	44444	Hamdinin Adresi	İzmir	35	RAM	1	50
5	Hande	Ata	5555	Handenin adresi	Antalya	07			

- Alışveriş yapmaksızın tabloya yeni müşteri eklenirse?
- Bir müşterinin adresi değişirse?
- Müşteri tek bir alışveriş yapmış ise ve alışverişler silinirse?

Normalleştirme Örneği (2NF)

Müşteri no	ad	soyad	tel	adres	şehir	Plaka kodu
1	Fatma	Kara	11111	Fatmanın Adresi	Muğla	48
2	Ali	Can	22222	Alinin adresi	Burdur	15
3	Veli	Koç	33333	Velinin adresi	İstanbul	34
4	Hamdi	Turan	44444	Hamdinin Adresi	İzmir	35
5	Hande	Ata	5555	Handenin adresi	Antalya	07

- **Kısmi Bağımlılıkları Ortadan Kaldırın**
  - Fonksiyonel bağımlılık göz önünde bulundurularak tablolar bölünmelidir.

Satış Id	Müşteri no	Ürün	miktar	fiyat
1	1	Tablet	1	600
2	2	RAM	2	80
3	2	DVD	100	5
4	2	Mouse	10	120
5	3	RAM	10	350
6	3	Klavye	20	200
7	4	RAM	1	50

Normalleştirme Örneği (3NF)

- **Dolaylı Bağımlılıkları Ortadan Kaldırın**
  - Anahtar olmayan hiçbir öznitelik başka bir anahtar olmayan özniteliğe bağlı olmamalıdır.
  - İlişkide birincil anahtara dolaylı bağımlı özellikler, yeni bir ilişkiye taşınır.

Müşteri no	ad	soyad	tel	adres	Şehir id
1	Fatma	Kara	11111	Fatmanın Adresi	1
2	Ali	Can	22222	Alinin adresi	2
3	Veli	Koç	33333	Velinin adresi	3
4	Hamdi	Turan	44444	Hamdinin Adresi	4
5	Hande	Ata	5555	Handenin adresi	5

Şehir İd	şehir	Plaka kodu
1	Muğla	48
2	Burdur	15
3	İstanbul	34
4	İzmir	35
5	Antalya	07

Satiş ıd	Müşteri no	Ürün	miktar	Fiyat
1	1	Tablet	1	600
2	2	RAM	2	80
3	2	DVD	100	5
4	2	Mouse	10	120
5	3	RAM	10	350
6	3	Klavye	20	200
7	4	RAM	1	50

Normalleştirme Örneği

numara	Adı	Soyadı	Bölüm No	Bölüm Adı	Ders kodu	Ders adı	Sicil_no	Öğretim Görevlisi	E amil
1	Ali	Can	09	Bilgisayar	CS101	İng.	101	Demet Örmeci	Ormeci
1	Ali	Can	09	Bilgisayar	CS102	Matematik	102	Ali Çalışkan	Caliskan
1	Ali	Can	09	Bilgisayar	CS103	Veri Tabanı-1	103	M.ilkuçar	İlkucar
2	Fatma	Kara	01	Elektrik	EL101	İng.	101	Demet Örmeci	Oremci
2	Fatma	Kara	01	Elektrik	EL103	Matematik	102	Ali Çalışkan	Caliskan
3	Kazım	Koç	03	Makine	MK104	Teknik Resim	104	Yusuf Altındal	Altindal
3	Kazım	Koç	03	Makine	MK103	Bil.Des.Tas	105	Sualp Deniz	deniz

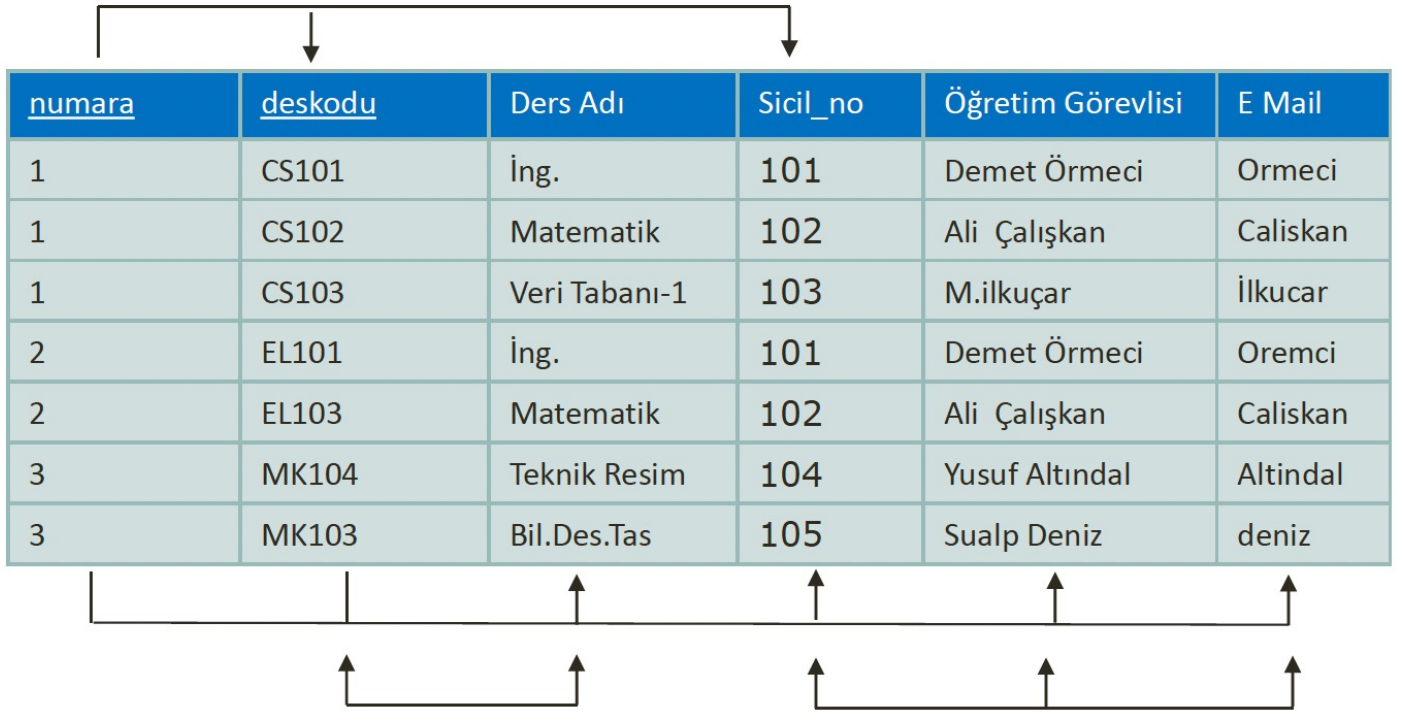
- 1NF
- Birincil Anahtar → numara, Bölüm No, ders kodu
- **İşlevsel Bağımlılıklar**
  - numara → {Adı, Soyadı, Bölüm No} --- Kısmi Ad, soyad, bölüm_no sadece numara ile.
  - Bölüm No → Bölüm Adı --- Kısmi
  - sicil_no → {Öğretim Görevlisi, e mail} --- Dolaylı
  - Ders adı → {ders_kodu} --- Dolaylı

Normalleştirme Örneği

2NF

numara	Adı	Soyadı	Bölüm No
1	Ali	Can	01
2	Fatma	Kara	03
3	Kazım	Koç	09

Bölüm No	Bölüm Adı
01	Elektrik
03	Makine
09	Bilgisayar



numara	deskodu	Ders Adı	Sicil_no	Öğretim Görevlisi	E Mail
1	CS101	İng.	101	Demet Örmeci	Ormeci
1	CS102	Matematik	102	Ali Çalışkan	Caliskan
1	CS103	Veri Tabanı-1	103	M.ilkuçar	İlkucar
2	EL101	İng.	101	Demet Örmeci	Oremci
2	EL103	Matematik	102	Ali Çalışkan	Caliskan
3	MK104	Teknik Resim	104	Yusuf Altındal	Altindal
3	MK103	Bil.Des.Tas	105	Sualp Deniz	deniz

Normalleştirme Örneği

3NF

ÖĞRENCİ

numara	Adı	Soyadı	Bölüm No
1	Ali	Can	01
2	Fatma	Kara	03
3	Kazım	Koç	09

BÖLÜM

Bölüm No	Bölüm Adı
01	Elektrik
03	Makine
09	Bilgisayar

DERS

deskodu	Ders Adı
CS101	İng.
CS102	Matematik
CS103	Veri Tabanı-1
MK104	Teknik Resim
MK103	Bil.Des.Tas

ÖĞRENCİ_DERS

numara	deskodu	Sicil no
1	CS101	101
1	CS102	102
1	CS103	103
2	EL101	101
2	EL103	102
3	MK104	104
3	MK103	105

ÖĞRETİM ELEMANI

Sicil no	Öğretim Görevlisi	E Mail
101	Demet Örmeci	Ormeci
102	Ali Çalışkan	Caliskan
103	M.ilkuçar	İlkucar
104	Yusuf Altındal	Altindal
105	Sualp Deniz	deniz

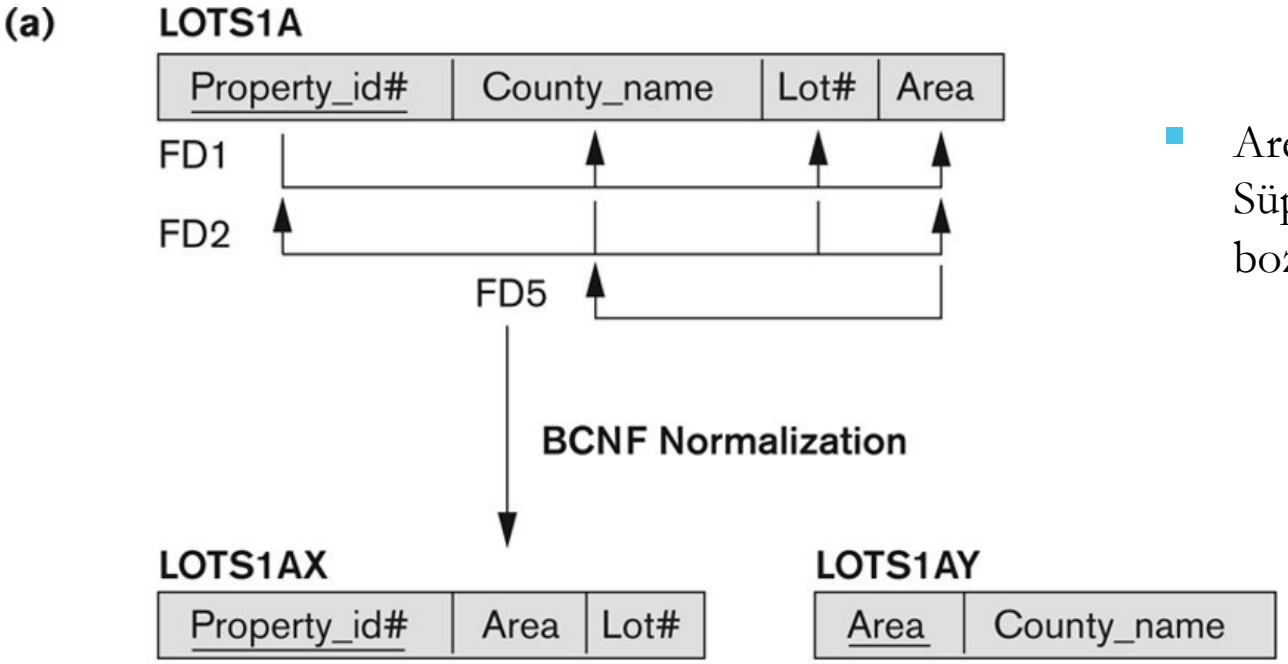
Normalizasyon Özet Tablosu (1NF–2NF–3NF)

Normal Form	Amaç	Sorunlu Durum	Gereken Kural	Çözüm
1NF (Birinci Normal Form)	Tekrarlayan grup ve çok değerli öznitelikleri kaldırmak	Bir hücrede birden fazla değer bulunması (örn. `{Tel1, Tel2}`)	Tüm öznitelikler **atomik** olmalı (tek değer)	Çok değerli alanları ayır, tekrar eden grupları satırlara böl
2NF (İkinci Normal Form)	Kısmi bağımlılıkları ortadan kaldırmak	Bir **bileşik anahtarın** sadece bir kısmının, non-prime attribute'u belirlemesi → *Partial Dependency*	Her non-prime attribute **bileşik anahtarın tamamına** bağlı olmalı	Tablonun bağımlılıklara göre parçalanması (özellikle ilişkiyi bölen tablo oluştur)
3NF (Üçüncü Normal Form)	Transitif bağımlılıkları kaldırmak	A → B ve B → C varsa, A → C oluşması → *Transitive Dependency*	Non-prime attribute'lar **yalnızca anahtara** bağlı olmalı; başka attribute üzerinden değil	Transitif bağımlılık oluşturan öznitelikleri ayrı tabloya taşı

BCNF (Boyce-Codd Normal Form)

- İlişki şeması R'nin süper anahtarı - R'nin bir anahtarını içeren R'nin öznitelik kümesi S
- Bir ilişki şeması R, üçüncü normal formda (3NF) ise, R'de bir FD X -> A geçerli olduğunda, aşağıdakilerden biri geçerlidir:
  - (a) X, R'nin bir süper anahtarıdır veya
  - (b) A, R'nin bir asal özniteliğidir.
- Boyce-Codd normal formu, yukarıdaki (b) koşulunu engeller.
- Bir ilişki şeması R, Boyce-Codd Normal Formunda (BCNF) ise, R'de bir fonksiyonel bağımlılık X -> A geçerli olduğunda, X, R'nin bir süper anahtarıdır.
- Her normal form, bir öncekinden kesinlikle daha güçlüdür.
  - Her 2NF ilişkisi 1NF'dedir.
  - Her 3NF ilişkisi 2NF'dedir.
  - Her BCNF ilişkisi 3NF'dedir.
- 3NF'de olup BCNF'de olmayan ilişkiler vardır.

BCNF (Boyce-Codd Normal Form)



- Area'nın County_name'i belirlemesi ve Area'nın Süper Key olmamasından dolayı BCNF'yi bozar.

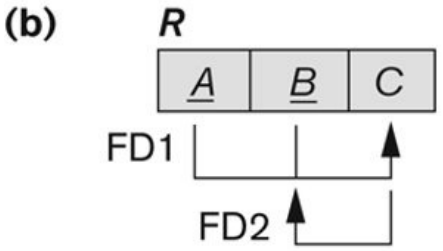


Figure 10.12
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

Karmaşık Veri Tipleri (Complex Data Types)

Karmaşık Veri Tipleri

- Günümüzde yaygın olarak kullanılan çeşitli **atomik olmayan** veri türleri (çok değerli, bileşik vb.) vardır.

- **4 kategori:**
  - Yarı Yapılandırılmış Veri
  - Nesne Yönelimli
  - Metinsel Veri
  - Uzamsal Veri

1) Yarı Yapılandırılmış Veri (Semi-Structured Data)

- Tamamen düzensiz olmayan, ancak katı bir ilişkisel şemayı da gerektirmeyen veri türüdür.
- Verinin yapısı vardır ama bu yapı esnektir ve zamanla değişebilir.

İki temel motivasyon vardır:

- **Veri Saklama ve İşleme (Data Storage & Processing)**
  - Günümüzde birçok uygulama karmaşık veriler üretir. Bu verilerin yapısı sık sık değişir (özellikle web uygulamalarında).
  - İlişkisel veritabanlarında her değişiklik: tablo ekleme, sütun değiştirme, normalizasyon işlemleri gerektirir → zaman harcanır ve maliyetlidir.
  - Yarı yapılandırılmış veri, bu değişikliklere daha esnek şekilde uyum sağlar.
- **Veri Paylaşımı (Data Exchange)**
  - Veri, farklı uygulamalar arasında, ya da backend–frontend katmanları arasında sıkça aktarılır.
  - Web servisleri (API'ler), karmaşık verileri, hızlı ve verimli şekilde istemciye (web/mobil) gönderir.
  - Yarı yapılandırılmış veri formatları bu aktarımı kolaylaştırır ve hızlandırır.
- JSON ve XML, yaygın olarak kullanılan yarı yapılandırılmış veri modelleridir.

1) Yarı Yapılandırılmış Veri Modellerinin Özellikleri

A. Esnek Şema (Flexible Schema)

- Yarı yapılandırılmış verilerde, katı bir tablo yapısı yoktur, verinin yapısı zamanla değişebilir.

Bu esneklik iki şekilde karşımıza çıkar:

- **Wide Column (Geniş Sütun) Yapısı:** Her kaydın (satırın) farklı alanları olabilir. Yeni bir özellik eklemek için tabloyu değiştirmeye gerek yoktur.
  - **Kullanıcı I:** ad, yaş, şehir
  - **Kullanıcı II:** ad, ilgiAlanları, sosyalMedya

- **Sparse Column (Seyrek Sütun) Yapısı:** Tanımlı çok sayıda alan vardır, ama her kayıt bu alanların sadece bir kısmını kullanır.
  - Alanlar sabit gibi görünür, ama çoğu kayıt birçok alanı boş bırakır. Bu yapı, gereksiz boş veri tutmayı önler.

1) Yarı Yapılandırılmış Veri Modellerinin Özellikleri

B. Çok Değerli Veri Tipleri (Multivalued Data Types)

- İlişkisel veritabanlarında zor olan bazı veri tipleri, burada doğal olarak desteklenir.

- **Küme / Liste (Set, Multiset):** Bir alan tek bir değer yerine birden fazla değer tutabilir.
  - **İlgi Alanları:** {basketbol, futbol, yemek, anime, caz}

- **Anahtar–Değer Yapısı (Key–Value / Map):** Veri, anahtar–değer çiftleri şeklinde tutulur. Her kaydın anahtarları farklı olabilir.
  - Ürün Bilgisi I: {marka: Apple, model: MacBook Air, ekran: 13, renk: silver }
  - Ürün Bilgisi II:{marka: Lenovo, model: ThinkPad, ram: 16GB}
  - Haritalar (map) üzerinde yaygın işlemler:
    - ✓ put(key, value) → ekle get(key) → oku delete(key) → sil

- **Diziler (Arrays):** Aynı türden verilerin sıralı şekilde tutulduğu yapıdır.
  - Özellikle bilimsel uygulamalarda, izleme (monitoring) sistemlerinde çok yaygındır.
    - ✓ sensör ölçümleri, zaman serileri, log kayıtları

1) Yarı Yapılandırılmış Veri Modellerinin Özellikleri

B. Çok Değerli Veri Tipleri (Multivalued Data Types)

- Yarı yapılandırılmış verilerde **diziler** çok yaygın kullanılır.
- Bu tür veriler düzenli aralıklarla üretilir ve ardışık değerler içerir.
- Eğer ölçümler düzenli zaman aralıklarında alınıyorsa, zaman bilgisini her seferinde tekrar yazmak gereksizdir.
  - Zaman–değer çiftleri yerine: (1,5), (2,8), (3,9), (4,11)
    Sadece değerleri tutmak yeterlidir: [5, 8, 9, 11]
- Daha sade, az yer kaplayan, hızlı işlenen bir yapı elde edilir.

Dizi Veritabanları (Array Databases)

- Bazı veritabanları diziler için özel olarak tasarlanmıştır.
- Bu tür veritabanları, dizileri çok verimli saklar ve dizi üzerinde hızlı sorgular yapabilirler.

Sağladıkları Avantajlar

- Sıkıştırılmış veri saklama (daha az disk kullanımı), diziye özel sorgu komutları, büyük veri setlerinde yüksek performans
  - Oracle GeoRaster, PostGIS, SciDB, MonetDB

1) Yarı Yapılandırılmış Veri Modellerinin Özellikleri

Çok Değerli Alanlar (Multi-Valued Attributes)

- Bazı alanlar, tek bir değer yerine birden fazla değer tutabilir.
  - Bir öğrencinin aldığı dersler, bir kullanıcının ilgi alanları, bir ürünün etiketleri

- Bu tür veriler, birinci normal forma uymayan, yani NFNF (Non First Normal Form) yapılarla modellenir.

- Klasik ilişkisel modelde bu önerilmez, ama yarı yapılandırılmış veri modellerinde doğal bir durumdur.

- Günümüzde çoğu modern veritabanı, çok değerli alanları doğrudan desteklemektedir.

Case Study: SciDB ve MonetDB

Ortak Noktaları

- Her ikisi de büyük veri analitiği (big data analytics) için tasarlanmıştır.
- Klasik OLTP (günlük işlem) veritabanları gibi değil, analiz ve sorgu performansı odaklıdırlar.

SciDB

- SciDB, dizi (array) tabanlı veriler için özel olarak geliştirilmiştir.
- İklim modelleme, uydu ve coğrafi veriler, genomik veriler, bilimsel simülasyonlar
- Veri tablo gibi değil, daha çok çok boyutlu diziler şeklindeyse SciDB daha uygundur.

MonetDB

- Sütun bazlı (columnar), SQL destekli bir analitik veritabanıdır.
- Tablosal veriler üzerinde çok hızlı sorgu sonucu döndürür.
- Büyük tablo analizleri, veri ambarı (data warehouse), analitik raporlama, SQL tabanlı analizler

İç İçe (Nested) Veri Tipleri

- Birçok uygulamada veriler düz tablo şeklinde değil, hiyerarşik (ağaç yapısında) tutulur.
- Yani bir veri başka verilerin içinde yer alabilir, alt–üst ilişkileri içerebilir.
- Bir kullanıcıya ait tüm bilgiler tek bir büyük nesne (doküman) içinde tutulabilir. Böylece join işlemlerine gerek kalmaz, veri daha hızlı okunur.

JSON (JavaScript Object Notation): Günümüzde en yaygın kullanılan veri formatıdır. Web ve mobil uygulamalarda standart haline gelmiştir. Okunması ve yazılması kolaydır. Nesneler, alt nesneler, diziler doğrudan tanımlanabilir.

XML (Extensible Markup Language): JSON'dan daha eski bir formattır. Hâlâ birçok sistemde aktif olarak kullanılır. Özellikle konfigürasyon dosyalarında, eski (legacy) sistemlerde yaygındır.

```json
{
  "userId": 101,
  "name": "Ahmet Yılmaz",
  "email": "ahmet@gmail.com",
  "address": {
    "city": "İstanbul",
    "district": "Kadıköy",
    "zip": "34710"
  },
  "interests": ["spor", "müzik", "yapay zeka"],
  "orders": [
    {
      "orderId": 5001,
      "date": "2024-05-12",
      "total": 1200,
      "items": [
        { "product": "Laptop", "price": 1000 },
        { "product": "Mouse", "price": 200 }
      ]
    }
  ]
}
```

address → alt nesne (nested object)

interests → dizi (array)

orders → dizi içinde nesne

items → daha da iç içe yapı

Tek sorgu ile kullanıcıya ait tüm bilgilere erişilebilir.

İç İçe (Nested) Veri Tipleri

Özellik	JSON	XML
Okunabilirlik	Daha sade ve kısa	Daha uzun ve karmaşık
Yazım biçimi	Anahtar–değer	Etiket (tag) tabanlı
Veri boyutu	Küçük	Daha büyük
Web kullanımı	Çok yaygın	Azalan kullanım
JavaScript uyumu	Doğrudan	Ek işlem gerekir
Güncel kullanım	Web & mobil API'ler	Konfigürasyon, eski sistemler

JSON (JavaScript Object Notation)

- JSON günümüzde veri alışverişinde (data exchange) neredeyse standart hâline gelmiştir.
- Modern uygulamaların çoğu web servisleri etrafında tasarlanır. Her işlem için sunucuya istek atılır.
  - **Örnek:** Bir e-posta uygulamasında; mail listeleme, mail okuma, mail silme gibi her işlem ayrı bir web servisi çağrısıdır ve bu servisler JSON ile haberleşir.
- JSON'un en büyük avantajlarından biri birçok programlama diliyle doğrudan uyumlu olmasıdır.
  - **JSON ↔ Nesne dönüşümü:** JavaScript, Java, Python, PHP gibi dillerde çok kolaydır.
  - Bunun için hazır kütüphaneler vardır, geliştirici ekstra uğraşmaz.
- Günümüzde birçok veritabanı JSON'u doğrudan destekler.
  - JSON verisi, özel JSON tiplerinde saklanabilir. JSON içindeki alanlara path ifadeleri ile erişilebilir.
  - Tablo verilerinden JSON nesnesi oluşturulabilir. Birden fazla satır, tek bir JSON dizi (array) hâline getirilebilir.
    - ✓ PostgreSQL'de json_agg gibi fonksiyonlar kullanılır.
- JSON, metin tabanlıdır. Bu yüzden, daha fazla yer kaplar ve işlenmesi biraz daha maliyetlidir.
  - **Çözüm:** Sıkıştırılmış JSON (BSON). Bu sorunu azaltmak için ikili (binary) formatlar kullanılır.

XML (Extensible Markup Language)

- XML, veriyi etiketler (tag) kullanarak tanımlayan bir veri gösterim biçimidir.
- Amaç verinin ne anlama geldiğini, verinin nasıl yapılandığını açıkça göstermektir.
- XML'de her veri: bir etiket ile başlar, aynı etiketle biter. Etiketler veriyi açıklar (Self-Documenting)
- Etiket isimleri sayesinde veriye bakıldığında neyin ne olduğu kolayca anlaşılır.
- XML iç içe etiketleri destekler, doğal olarak ağaç (tree) yapısı oluşturur. Bu da karmaşık ilişkileri düzenli şekilde ifade etmeyi sağlar.
- XML ile veri alışverişi yapan taraflar hangi etiketlerin kullanılacağını, bu etiketlerin ne anlama geldiğini önceden anlaşmak zorundadır.

```
<purchase order>
  <identifier> P-101 </identifier>
  <purchaser>
      <name> Cray Z. Coyote </name>
      <address> Route 66, Mesa Flats, Arizona 86047, USA </address>
  </purchaser>
  <supplier>
      <name> Acme Supplies </name>
      <address> 1 Broadway, New York, NY, USA </address>
  </supplier>
<itemlist>
    <item>
        <identifier> RS1 </identifier>
        <description> Atom powered rocket sled </description>
        <quantity> 2 </quantity>
        <price> 199.95 </price>
    </item>
    <item>…</item>
</itemlist>
<total cost> 429.85 </total cost>
    ….
</purchase order>
```

DTD (Document Type Definition)

- Bu anlaşmayı sağlamak için DTD dosyaları kullanılır.
  - Hangi etiketler olabilir? hangi sırayla gelmeli? hangileri zorunlu? gibi kuralları tanımlar.
- DTD: XML belgesinin şemasını tanımlar.

2) Nesne-ilişkisel (Object Orientation) Veri Modeli

- Günümüzde uygulamaların çoğu nesne yönelimli programlama dilleri ile yazılır (Java, Python, C#, vb.)
- Ancak veriler çoğunlukla ilişkisel veritabanlarında saklanır.

Object-Relational (Nesne–İlişkisel) Veri Modeli

- Nesne–ilişkisel veri modeli klasik ilişkisel modele göre daha zengin bir tip sistemi sunar.
- Bu sayede karmaşık veri tipleri ve nesneye benzer yapılar veritabanında tanımlanabilir.
- **Amaç:** Programlama dilindeki nesneler ile veritabanındaki veriler arasındaki farkı azaltmak.

Tip Uyumsuzluğu

- **Programlama dilleri:** sınıf (class), nesne (object), kalıtım (inheritance), metotlar gibi yapılar kullanır.
- **İlişkisel veritabanları:** tablo, satır, sütun mantığına dayanır.

Dil Değiştirmenin Zorluğu

- Uygulama geliştirirken sürekli şunlar arasında geçiş yapılır: imperative (buyurgan) diller → nasıl yapılacağını anlatıri SQL (declarative) → ne istendiğini söyler
- Bu geçiş geliştirici için zahmetlidir.

2) Nesne-ilişkisel (Object Orientation) Veri Modeli

Nesne Yönelimli Yaklaşımı Veritabanına Entegre Etme Yolları

1) **Nesne–İlişkisel Veritabanı Kullanmak**
   - Mevcut ilişkisel veritabanına nesne yönelimli özellikler eklenir.
   - **Örnek:** karmaşık tipler, JSON, array, kullanıcı tanımlı tipler
2) **Object-Relational Mapping (ORM)**

- Programlama dili ile veritabanı arasında otomatik dönüşüm yapılır.
  - sınıf ↔ tablo nesne ↔ satır alan ↔ sütun. eşlemesini otomatik yapar.
  - Örnek ORM'ler: Hibernate (Java), SQLAlchemy (Python), Entity Framework (.NET)
- Geliştirici SQL yazmadan veriye erişebilir.

3) **Nesne Yönelimli Veritabanı (OODB)**

- Veritabanı doğrudan nesne yönelimli çalışır. Veriler nesne olarak saklanır ve programlama dili doğrudan erişilir.
  - **Avantaj:** dil ile entegrasyon çok iyidir.
  - **Dezavantaj:** SQL desteği yoktur veya çok sınırlıdır, yaygın değildir, pointer tabanlı yapılar karmaşıktır.

Object-Relational Database Systems

- Birçok modern veritabanı array (dizi), çok değerli (multiset) gibi yapısal veri tiplerini doğrudan destekler.
    - PostgreSQL'de, «create table N (numbers integer [ ]);» komutu.
    - Oracle'da, «create table N (numbers varray(10) of integer);» komutu.

Klasik model → atomik alanlar
Nesne–ilişkisel model → yapısal alanlar

- Bu yüzden web uygulamaları, API tabanlı sistemler, nesne yönelimli yazılımlar için daha uygundur.

3) Metinsel Veri (Textual Data)

- Metinsel veri, genellikle yapılandırılmamış veridir ve bu tür veriler üzerinde yapılan aramalar Information Retrieval (Bilgi Erişimi) kapsamında ele alınır.
- Amaç, verilen anahtar kelime veya sorguya göre en uygun dokümanları bulmaktır.
- "Doküman" kavramı bağlama göre değişir:
  - bilgi erişimi alanında metinlerin tamamı,
  - veritabanlarında bir text alanı,
  - web ortamında ise her web sayfası bir doküman olarak kabul edilir.
- Bir sorgu genellikle çok sayıda dokümanla eşleştiği için, sonuçların alakaya göre sıralanması **(relevance ranking)** gerekir. Bu sıralama, sorgu ile dokümanlar arasındaki **benzerliğin** ölçülmesine dayanır.
  - **İstatistiksel yöntemler:**
    - ✓ **Sorguya bağlı:** TF-IDF, BM25,…
    - ✓ **Sorgudan bağımsız:** PageRanking,…
  - **Semantik modeller:** Günümüzde sadece kelime eşleşmesine değil, **anlama dayalı (semantic)** modellere de odaklanılmaktadır. Bu yaklaşım, bilgilerin etiketlendiği ve ilişkilerle tanımlandığı Semantic Web fikrine dayanır ve kullanıcının sorgusuna daha anlamlı sonuçlar üretmeyi amaçlar.

4) Uzamsal Veriler (Spatial Data)

- Uzamsal veritabanları, mekansal konumlarla ilgili bilgileri depolar ve uzamsal verilerin verimli bir şekilde depolanmasını, indekslenmesini ve sorgulanmasını destekler.
- Coğrafi ve geometrik verilerin temsili veritabanına göre değişebilse de, çoğu sistem Open Geospatial Consortium (OGC) standartlarını temel alır.
- Coğrafi veriler; yol haritaları, arazi kullanımı, yükseklik ve idari sınırlar gibi dünya üzerindeki konumları temsil eder ve genellikle enlem, boylam ve yükseklik gibi küresel koordinat sistemleriyle ifade edilir.
- Bu amaçla kullanılan Coğrafi Bilgi Sistemleri (GIS), coğrafi veriler için özelleşmiş veritabanlarıdır.
- Geometrik veriler ise binalar, uçaklar veya entegre devreler gibi nesnelerin şekil ve yapısını tanımlar ve genellikle 2 veya 3 boyutlu Öklid uzayında (X, Y, Z) modellenir.

YILDIZ TEKNİK ÜNİVERSİTESİ

Yasemin Topuz

Yıldız Teknik Üniversitesi

ytopuz@yildiz.edu.tr