YILDIZ TEKNİK ÜNİVERSİTESİ

E-R Kavram Tasarım, Fonksiyonel Bağımlılıklar ve Normalizasyon Temelleri

Öğr. Gör. Dr. Yasemin Topuz

Yıldız Teknik Üniversitesi

BLM3041 – Veritabanı Yönetimi

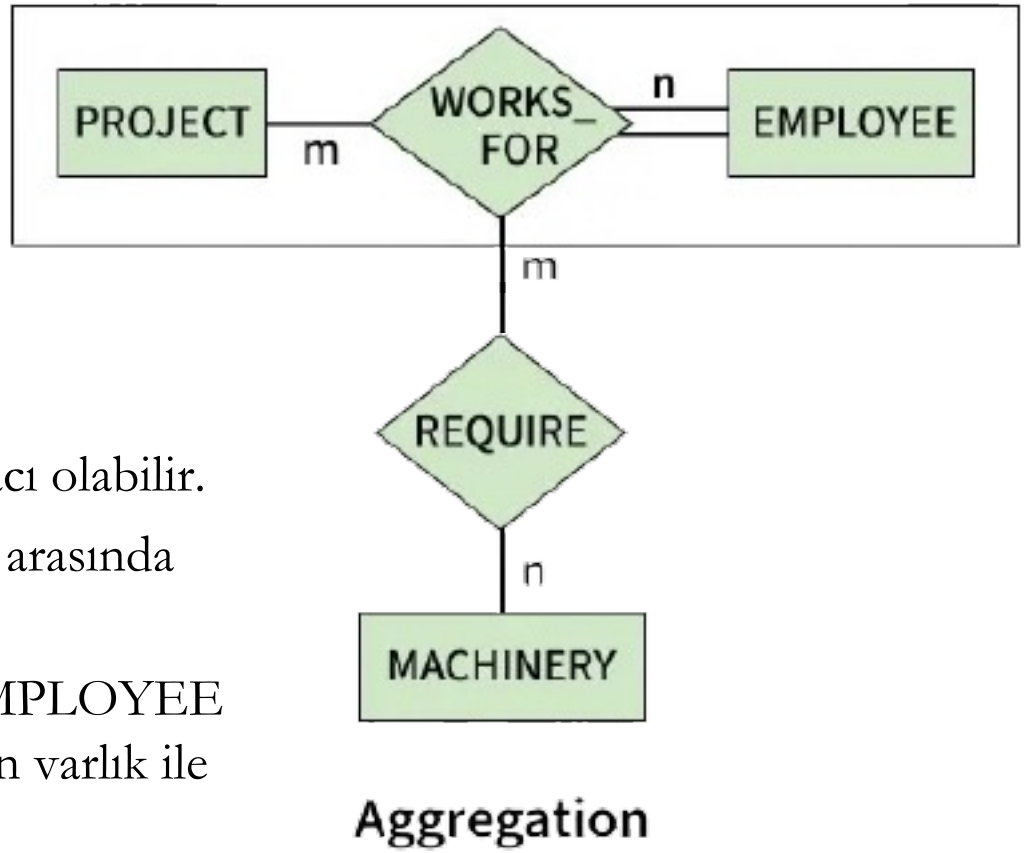
12 Aralık 2025 | YTÜ Bilgisayar Mühendisliği

Neler konuşacağız?

- Extended ER
- E-R Diyagramlarında Yaygın Hatalar (Common Mistakes)
- UML Birleşik Modelleme Dili (Unified Modeling Language)
- Fonksiyonel Bağımlılıklar ve Normalizasyon
- İlişkisel Veritabanları için Informal Tasarım İlkeleri
- Fonksiyonel Bağımlılıklar

İlişkiler Arasında İlişki Kurma İhtiyacı: Birleştirme (Aggregation)

- Bir ER diyagramı, bazı senaryolarda gerekli olabilecek varlık ile ilişki arasındaki bir ilişkiyi temsil edemez.
- Bu gibi durumlarda, ilgili varlıklarla birlikte bir ilişki, daha üst düzey bir varlıkta birleştirilir.
- Birleştirme, ilişkileri daha üst düzey varlık kümeleri olarak temsil edebileceğimiz bir soyutlamadır.



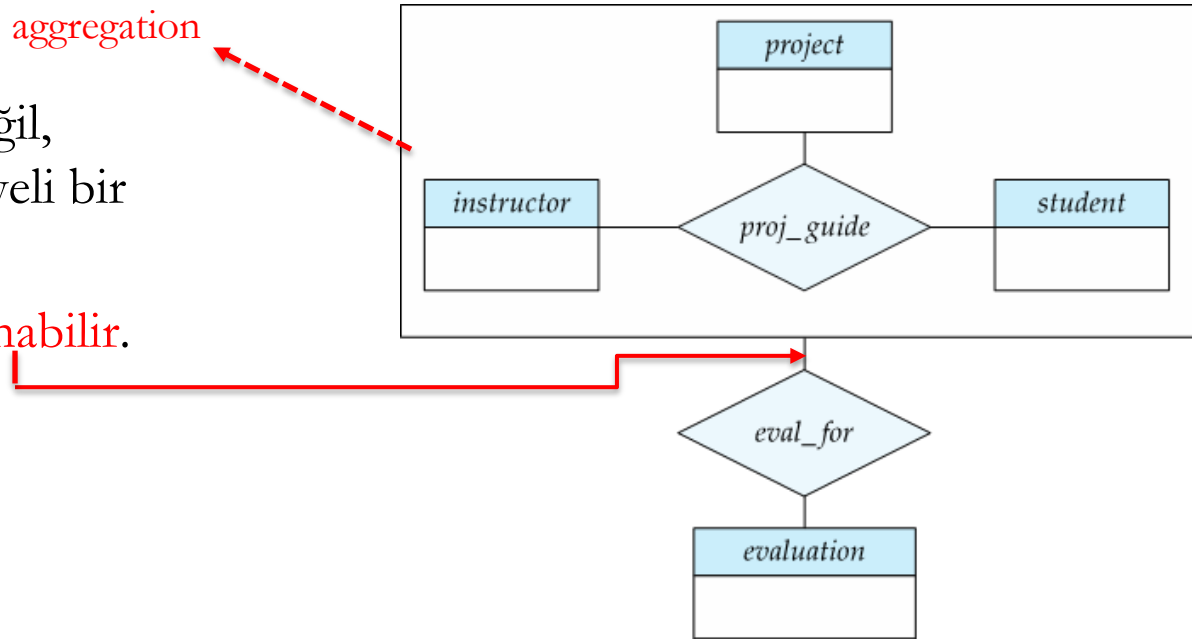
Aggregation

- Bir proje üzerinde çalışan bir Çalışanın bazı makinelere ihtiyacı olabilir.
- Bu nedenle, WORKS_FOR ilişkisi ile MACHINERY varlığı arasında REQUIRE ilişkisi gereklidir.
- Aggregation yöntemi kullanılarak, WORKS_FOR ilişkisi, EMPLOYEE ve PROJECT varlıklarıyla tek bir varlıkta toplanır ve toplanan varlık ile MACHINERY arasında REQUIRE ilişkisi oluşturulur.

İlişkiler Arasında İlişki Kurma İhtiyacı: Birleştirme (Aggregation)

Aggregation Kullanarak Gereksiz Tekrarın Giderilmesi

- Aşağıdaki diyagram, aggregation'ın nasıl çalıştığını göstermektedir:
  - Bir öğrenci, belirli bir proje üzerinde belirli bir instructor (rehber) tarafından yönlendirilir.
  - Bu öğrenci–öğretim üyesi–proje üçlüsü bir değerlendirmeye sahip olabilir veya olmayabilir.
- Bu nedenle:
  - proj_guide, artık sadece bir ilişki değil, aggregation sayesinde daha üst seviyeli bir "bileşik varlık" gibi davranır.
- Bu birleşik yapı, eval_for ilişkisine bağlanabilir.



proj_guide artık birleştirilmiş bir ilişkidir ve daha üst düzey bir «varlık kümesidir»

İlişkisel Şemalara Dönüştürme (Aggregation için)

- Aggregation'ı ilişkisel modelde gösterebilmek için oluşturulan şema şu bileşenleri içerir:
  - Birleştirilmiş (aggregated) ilişkinin birincil anahtarı
  - İlişkiye bağlı varlığın birincil anahtarı
  - Varsa tanımlayıcı (descriptive) öznitelikler
- Örnekte:
  - eval_for şeması: eval_for(s_ID, project_id, i_ID, evaluation_id)
- Bu öznitelikler şunları temsil eder:
  - s_ID → öğrenci (student) PK'si
  - project_id → proje PK'si
  - i_ID → instructor PK'si
  - evaluation_id → değerlendirme varlığının PK'si
- Bu dört anahtar, proj_guide ilişkisinin aggregation ile birleşip tek bir şema olarak aktarılmasını sağlar.

Aggregation, eval_for ilişkisine tam katılım (TOTAL participation) gösteriyorsa proj_guide şeması gereksiz (redundant) olur.

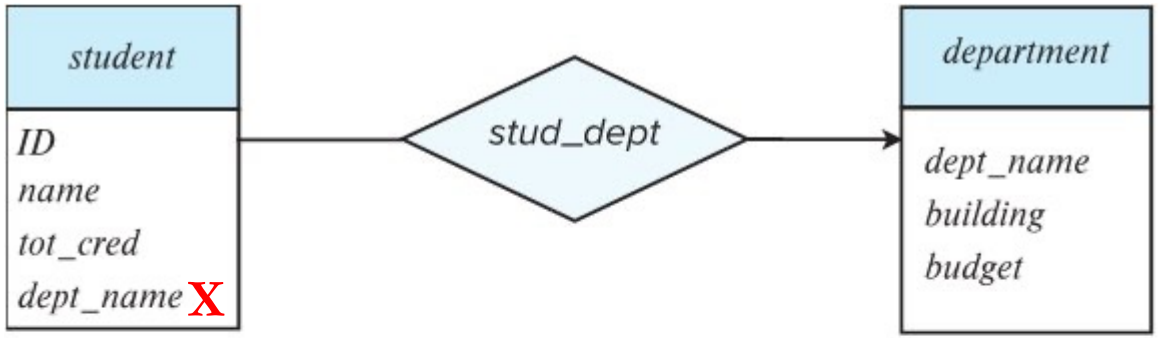
Yani:

- Eğer her proj_guide ilişkisinin mutlaka bir evaluation kaydı varsa → proj_guide tablosuna gerek yoktur (çünkü eval_for tüm bilgiyi kapsar).
- Eğer bazı proj_guide ilişkileri değerlendirme içermiyorsa → proj_guide tablosu korunmak zorundadır.

E-R Diyagramlarında Yaygın Hatalar (Common Mistakes)

Hata 1 - Özniteliğin yanlış yerde gösterilmesi

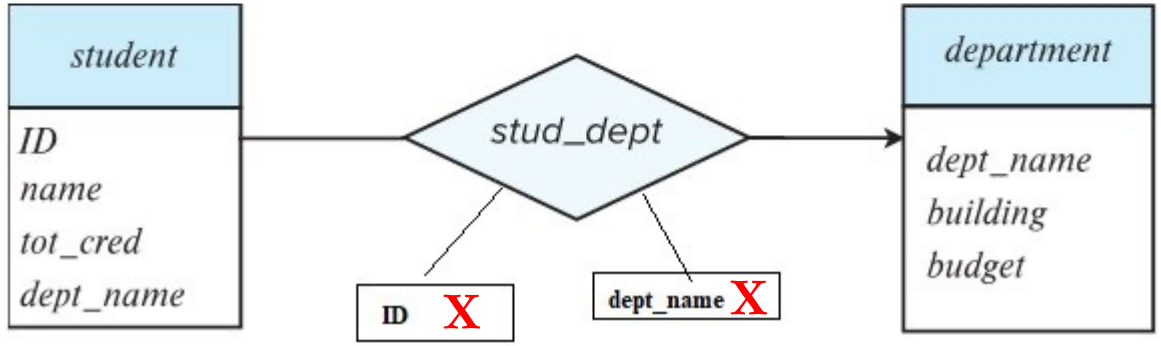
- student varlık kümesinde dept_name özniteliği yer alıyor.
- Ancak bir öğrencinin bölüm bilgisi, department varlığından gelmelidir; öğrenci varlığının kendi özniteliği değildir.
- Öğrenci bir bölüme ilişki (stud_dept) yoluyla bağlıdır.
- Bu nedenle dept_name özniteliği student içinde bulunamaz.



(a) Incorrect use of attribute

Hata 2 - İlişki kümesinin özniteliği gibi gösterimi

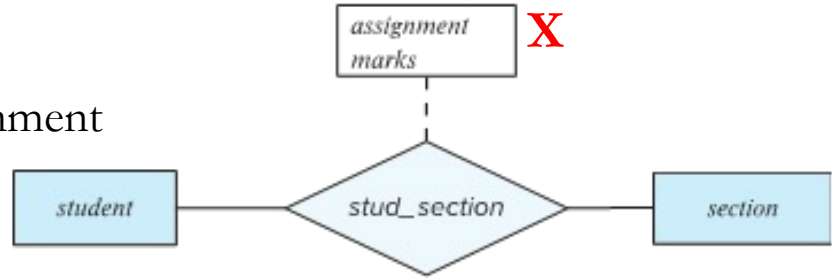
- stud_dept ilişkisinin altına ID ve dept_name eklenmiş.
- Oysa bunlar ilişkiyi tanımlayan participation keys (katılım anahtarlarıdır).
- İlişki kümesine "öznitelikmiş gibi" yazmak doğru değildir.
- Bir ilişki kümesi katılan varlıkların anahtarlarını taşır ama bunlar ilişki özniteliği değildir; bu nedenle kutucuk içinde öznitelik olarak gösterilmez.



E-R Diyagramlarında Yaygın Hatalar (Common Mistakes)

Hata 3 - İlişki Özniteliklerini Yanlış Kullanmak

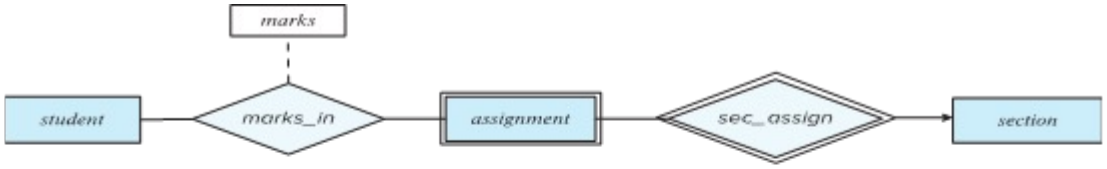
- stud_section ilişkisine assignment ve marks öznitelikleri eklenmiş.
- Bu durumda E-R modeli şunu ima eder: Bir öğrenci–section ikilisi yalnızca bir assignment kaydına sahip olabilir.
- Fakat gerçek hayatta: Bir öğrenci, aynı section için birden fazla assignment alabilir.
- Yani assignment, student–section ilişkisi ile bire çok ilişki içindedir. Bu nedenle, assignment ilişki üzerinde bir öznitelik olarak duramaz.



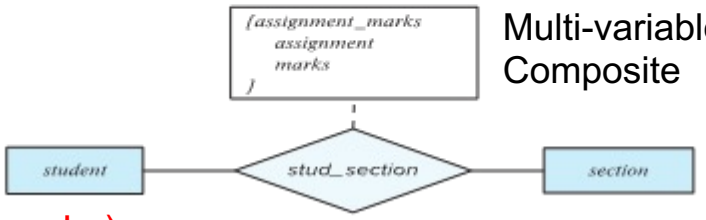
- **Çözüm:**

  1) Assignment bilgisini ayrı bir varlık yap assignment bağımsız bir entity olmalıdır.

  2) Öğrenci–Section ikilisine assignment'ı bağlayan yeni bir ilişki (veya tablo) oluştur



Student(sid, …)
Section(sectid, …)
Stud_section(sid,sectid,…)
Std_sect_assign (**sid,sectid,assig**,marks)



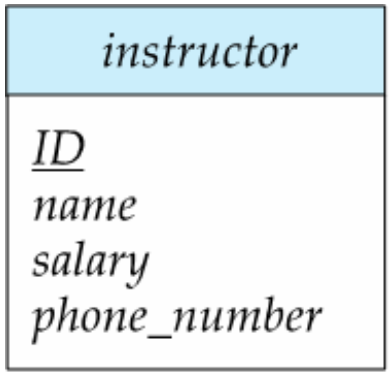
Multi-variabled Composite

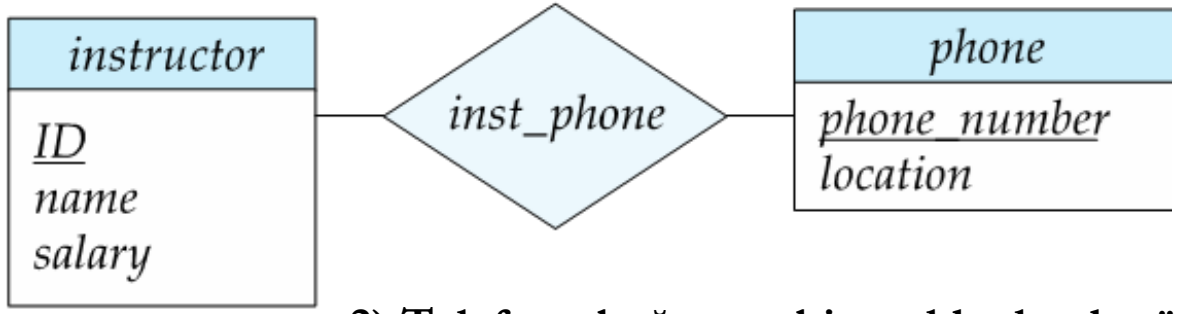
Entities vs. Attributes

Basit ve tek değerli → öznitelik

Çok değerli, ek bilgisi olan veya başka ilişkilerde yer alacak → varlık kümesi

- Bir bilgiyi öznitelik olarak mı, yoksa bağımsız bir varlık kümesi olarak mı göstermemiz gerektiği E-R modellemesinde önemli bir karardır.





1) Telefonu öznitelik olarak göstermek

- Instructor varlığı şu özniteliklere sahiptir:
  - ID, name, salary, phone_number
- Bu kullanım basittir fakat sınırlıdır.

→ Instructor'ın birden fazla telefon numarası olamaz.

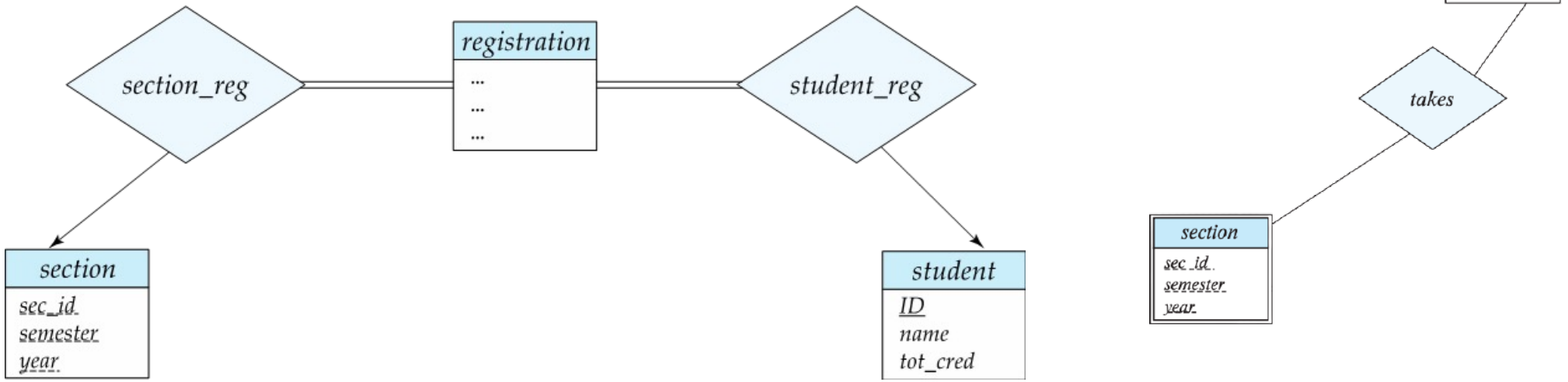
→ Telefonla ilgili ek bilgi (ör. konum, tür, operatör) tutamayız.

2) Telefonu bağımsız bir varlık olarak göstermek

- phone bir entity set olarak modellenir. phone varlığının öznitelikleri: phone_number, location
- instructor ile phone arasındaki ilişki: inst_phone
  - Bir eğitmenin birden fazla telefon numarası olabilir.
  - Telefon numarasına ilişkin ek bilgiler (konum, cihaz türü vb.) saklanabilir.
  - Telefon varlığı, başka ilişkilerde de tekrar kullanılabilir.

Entities vs. Relationship Sets

- Eğer iki varlık arasında gerçekleşen şey bir eylem (action) ise, bunu bir ilişki (relationship set) ile modellemek daha uygundur.
- "section — takes — student" ilişkisine alternatif tasarım, "student takes section" ilişkisi yerine:
  - section_reg → section için kayıt ilişkisidir. student_reg → student için kayıt ilişkisidir.



- Bu iki ilişki, ortada registration adında bir varlık ile birbirine bağlanmıştır.
- Bu yapı sayesinde: registration bir entity set olduğu için
  → kayıt ile ilgili ek bilgiler (örneğin kayıt tarihi, ödeme durumu, sınıf türü vb.) buraya eklenebilir.

Binary vs. Non-Binary Relationships

- Her ne kadar herhangi bir non-binary (n-ary, n > 2) ilişki kümesini birden fazla ikili ilişki ile modellemek mümkün olsa da, n-ary bir ilişki, birden fazla varlığın aynı anda tek bir ilişkide yer aldığını daha açık biçimde gösterir.

- Bazı ilişkiler ilk bakışta non-binary gibi görünse de, aslında ikili ilişkilerle daha doğru veya daha sade şekilde ifade edilebilir.
  - Örnek: Bir çocuğu anne ve babasıyla ilişkilendiren üçlü bir ilişki (parents),
    → iki ayrı ikili ilişki ile (father ve mother) daha iyi temsil edilir.
- Bu yaklaşım ayrıca kısmi bilgiyi temsil etmeye izin verir. Örneğin: Sadece annenin bilindiği durumlar.

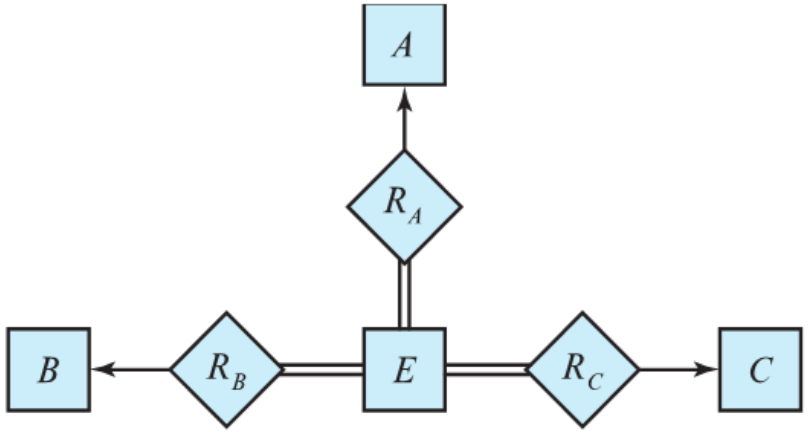
- Ancak bazı ilişkiler vardır ki doğal olarak non-binary'dir ve ikili ilişkilere bölünmesi anlamlı değildir.
  - Örnek: proj_guide (öğrenci – danışman – proje üçlü ilişkisi)

Non-Binary (N-ary) İlişkileri Binary İlişkilere Dönüştürme

- Bir üçlü ilişkiyi (A–B–C) doğrudan binary ilişkilere bölmek çoğu zaman yanlış sonuç verir.
- **Çözüm:** İlişkiyi bir yapay (artificial) entity set'e dönüştürmek ve her bir katılımcı ile ayrı binary ilişkiler kurmaktır.

Dönüştürülmüş Binary Model (b)

- **Adım 1 –** Yeni bir entity set oluştur: İlişkiden türetilmiş yeni entity set: E
- **Adım 2 –** Üç tane binary ilişki oluştur: R_A : E ↔ A R_B : E ↔ B R_C : E ↔ C
- **Adım 3 –** R üzerindeki tüm öznitelikler E'ye taşınır.
- **Adım 4 –** R içindeki her ilişki kaydı için: Bir (a_i, b_i, c_i) ilişkisi varsa:
  Yeni bir entity e_i oluştur (E içinde)
  - (e_i, a_i) → R_A
  - (e_i, b_i) → R_B
  - (e_i, c_i) → R_C



(b)

Non-Binary (N-ary) İlişkileri Binary İlişkilere Dönüştürme

- Kısıtların da dönüşüme dâhil edilmesi gerekir
- Non-binary bir ilişkiyi (ör. üçlü ilişki R) binary ilişkilere dönüştürdüğümüzde yalnızca yapının kendisi değil, ilişkideki kısıtlar da (cardinality, participation vb.) dönüşmelidir.
- Ancak, tüm kısıtlar eksiksiz dönüştürülemez
  - Örneğin: 1–N–N gibi üçlü ilişki kısıtları binary ilişkilere bölündüğünde tam olarak karşılık bulmaz. Bu nedenle bazı durumlar yeni modelde geçersiz / anlamsız kombinasyonlar üretebilir.
- Dönüştürülen şemada; A, B ve C'den gelen bir üçlü ilişkinin gerçek bir R satırına karşılık gelmeyebilir.

Alıştırma: Yeni oluşturulan bir varlığın A, B ve C varlık kümelerinin her birinde tam olarak bir varlığa karşılık gelmesini sağlamak için RA, RB ve RC ilişkilerine kısıtlamalar ekleyin.

E'yi, üç ilişki kümesi tarafından tanımlanan zayıf bir varlık kümesi (kısa bir açıklama ile) yaparak tanımlayıcı bir öznitelik oluşturmaktan kaçınabiliriz.

Konu	Açıklama
Non-binary ilişkiler binary'ye çevrilebilir	E varlığı + 3 binary ilişki
Kısıtlar her zaman tam dönüşmez	Özellikle *1-N-N* gibi üçlü kardinaliteler
Ek kısıt eklemeliyiz	E her A, B, C ile tam 1 kez ilişkilendirilmeli
ID eklemeyi önlemek mümkün	E → weak entity yapılabilir

E-R Tasarım Kararları (E-R Design Decisions)

1. Bir nesnenin bir öznitelikle mi yoksa bir varlık kümesiyle mi temsil edileceği

- Bazı bilgiler tek değerli bir öznitelik olarak yeterlidir; bazıları ise kendi kimliği ve ilişkileri olan ayrı bir varlık kümesi olmayı gerektirir.

  Örnek: Telefon numarası → öznitelik olabilir; çoklu numaralar gerekiyorsa → ayrı varlık.

2. Gerçek dünya kavramının daha iyi bir varlık kümesiyle mi yoksa ilişki kümesiyle mi ifade edildiği

- Bazen bir bilgi, varlığın özelliği olarak tutulmalıdır: bazen iki varlık arasındaki etkileşimdir ve ilişki olarak gösterilmelidir.

  Örnek: İşe başlama tarihi → çoğu zaman bir ilişki (çalışma ilişkisi) üzerinde özniteliktir.

3. Üçlü (ternary) ilişki mi yoksa iki ikili (binary) ilişki mi kullanılacağı

- Bazı durumlarda binary ilişkiler bilgi kaybına yol açar ve ternary ilişki zorunludur.

  Örnek: Öğrenci–Eğitmen–Proje ilişkisi.

E-R Tasarım Kararları (E-R Design Decisions)

4. Güçlü (strong) veya zayıf (weak) varlık kümesi kullanımı

- Bir varlık, kendi birincil anahtarına sahip değilse ve başka bir varlık tarafından tanımlanıyorsa zayıf varlık olarak modellenmelidir.

5. Uzmanlaşma / genelleştirme (specialization / generalization) kullanımı

- Hiyerarşik yapıyı ifade etmek ve tasarımı modüler kılmak için kullanılır.

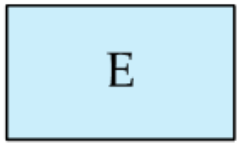
  Örnek: Person → Student / Employee.

6. Birleştirme (aggregation) kullanımı

- Bir ilişki kümesini soyutlayarak daha yüksek seviyeli bir birim gibi göstermek mümkündür.
  Bu, karmaşık ilişkileri sadeleştirir ve tasarımı daha okunabilir kılar.

Tasarım Kararı	Açıklama
Öznitelik mi varlık mı?	Nesnenin karmaşıklığına göre seçilir.
Varlık mı ilişki mi?	Kavramın doğasına göre belirlenir.
Ternary vs binary	Bilgi kaybı olmadan ifade edebilme durumu belirleyicidir.
Strong vs weak	Varlığın kendi anahtarına sahip olup olmaması önemli.
Specialization	Üst-alt sınıf yapıları için gereklidir.
Aggregation	İlişkiler arasında ilişki göstermek gerektiğinde kullanılır.

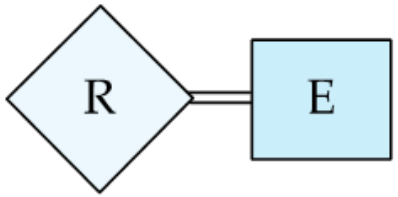
E-R Notasyonunda Kullanılan Sembollerin Özeti



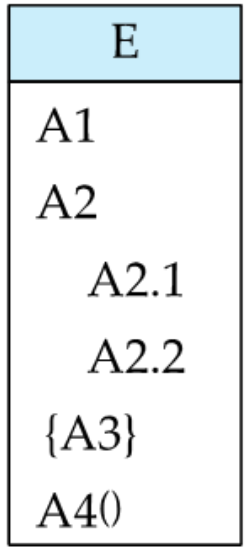
Varlık Kümesi
(Entity Set)

İlişki Kümesi
(Relationship Set)

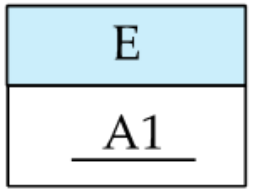
Zayıf Varlığı Tanımlayan İlişki
(Identifying Relationship)



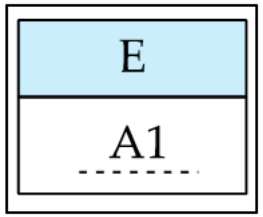
Tam Katılım
(Total Participation)



Öznitelikler (Attributes)
(A1) Simple (Basit) öznitelik
(A2) Composite (Bileşik) öznitelik
(A3) Multi-valued (Çok değerli) öznitelik
(A4) Derived (Türetilmiş) öznitelik



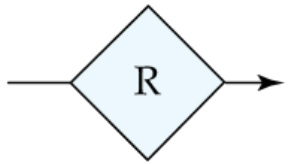
Birincil Anahtar
(Primary Key)



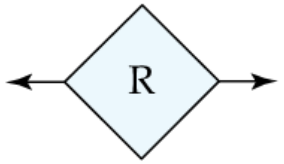
Zayıf Varlığın Ayırt Edici Özniteliği
(Discriminating Attribute)

E-R Notasyonunda Kullanılan Sembollerin Özeti

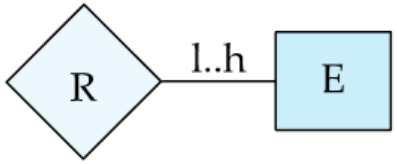
Many-to-Many (N–N) Relationship



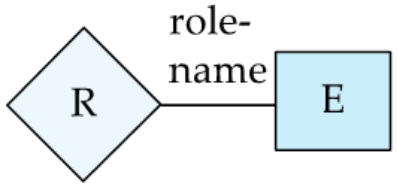
Many-to-One (N–1) Relationship



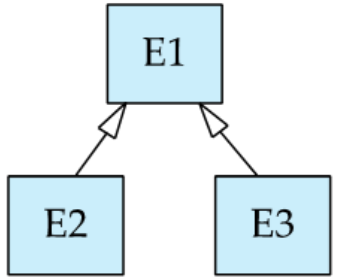
One-to-One (1–1) Relationship



Kardinalite Sınırları (1..h Notasyonu)

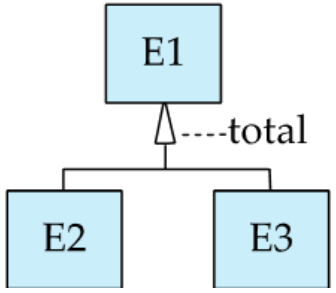


Role Indicator (Rol Göstergesi)

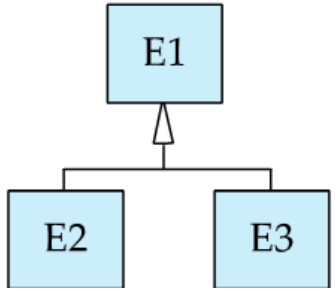


overlap

ISA – Generalization / Specialization (Kalıtım Yapısı)



Total Specialization (Zorunlu Alt Sınıf Üyeliği)

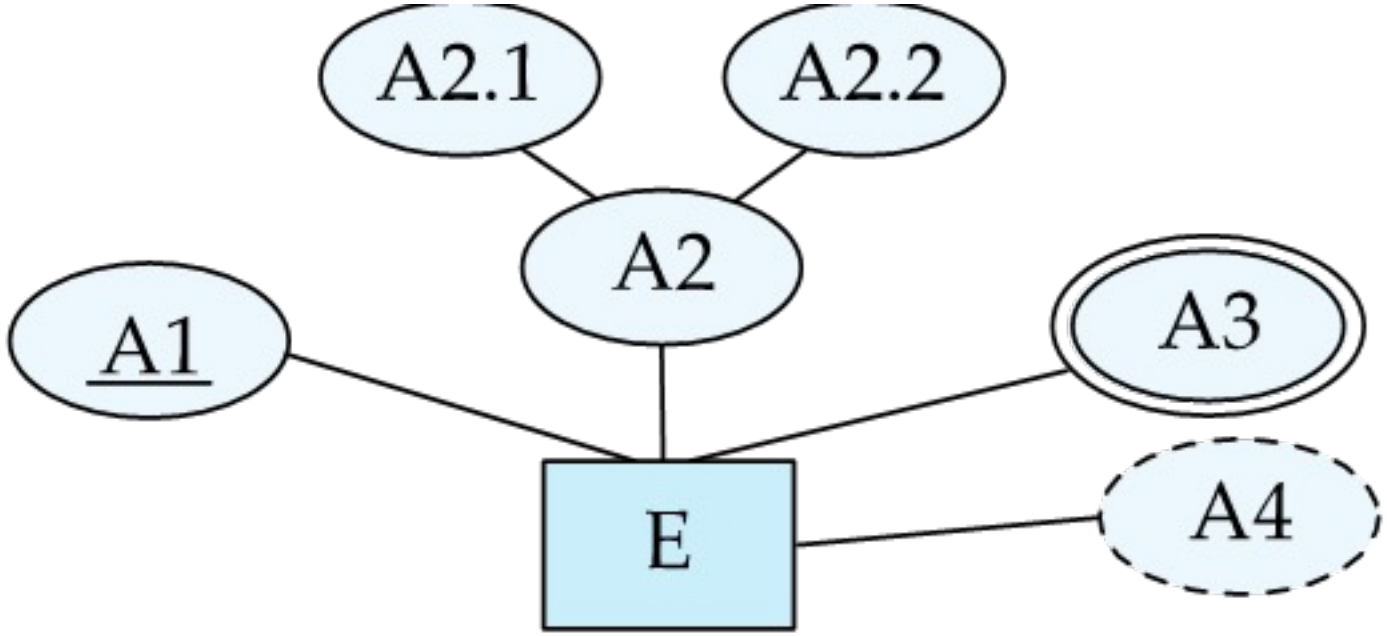


Disjoint (Ayrık Alt Sınıflar)

Alternatif ER Gösterimleri

Entity Set E — Farklı Öznitelik Türleri

(A1) Simple (Basit) öznitelik, primary key
(A2) Composite (Bileşik) öznitelik,
alt öznitelikleri A2.1 ve A2.2
(A3) Multi-valued (Çok değerli) öznitelik
(A4) Derived (Türetilmiş) öznitelik

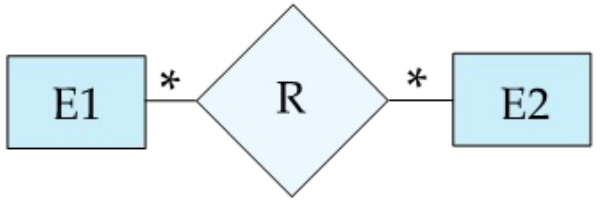
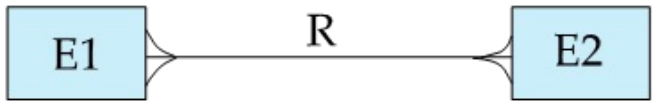
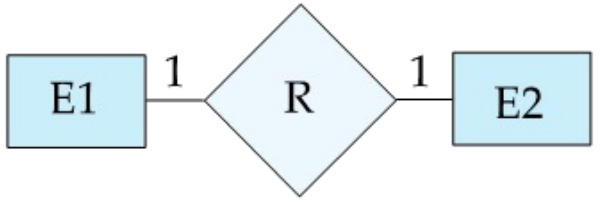
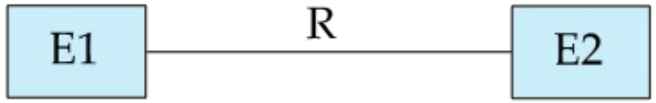
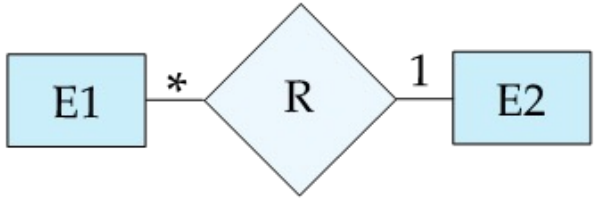
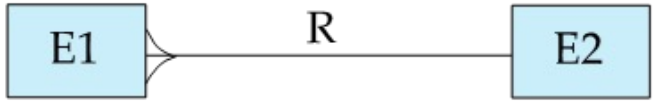
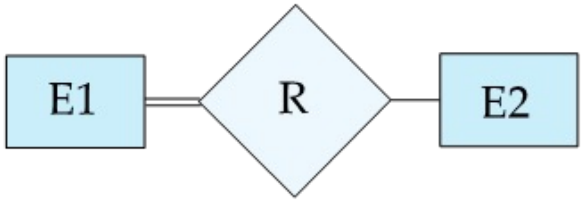
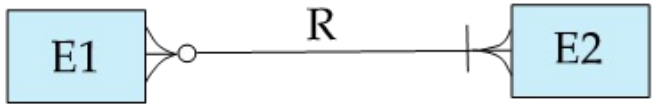


weak entity set 

generalization 

total generalization 

Alternatif ER Gösterimleri

	Chen	IDE1FX (Kaz ayağı notasyonu)
Many-to-Many Relationship (E1 ↔ E2, M:N)		
One-to-One (E1 1 ↔ 1 E2)		
Many-to-One (E1 N → 1 E2)		
Participation in R: Total (E1) / Partial (E2)		

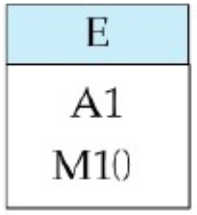
UML (Unified Modeling Language): Birleşik Modelleme Dili

- UML, bir yazılım sisteminin tamamının (entire software system) farklı yönlerini grafiksel olarak modellemek için birçok bileşen içerir.

- UML Sınıf Diyagramları, E-R Diyagramlarına karşılık gelir; ancak aralarında birkaç önemli fark vardır.

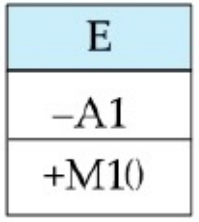
UML (Unified Modeling Language): Birleşik Modelleme Dili

ER Diagram Notation

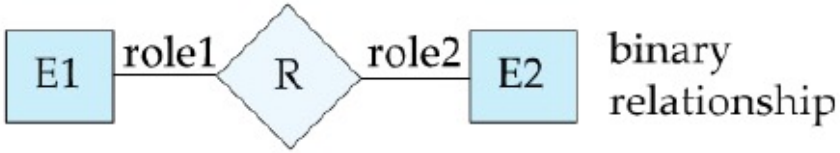


E tablosu içinde:
A1 → basit öznitelik
M{ } → çok değerli öznitelik

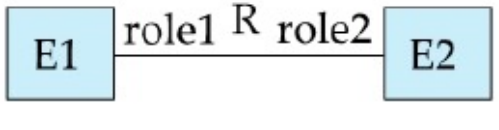
Equivalent in UML

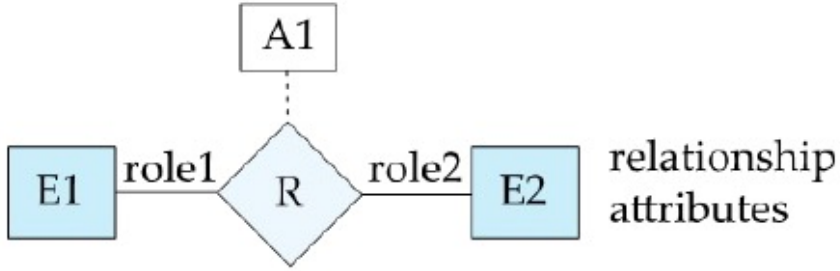


Class (sınıf) kutusu içinde:
−A1 → private attribute
+M{ } → public multivalued attribute
(UML'de + public, − private, # protected anlamına gelir.)

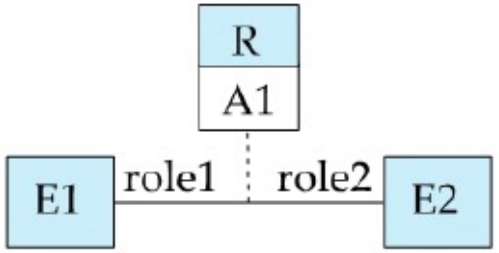


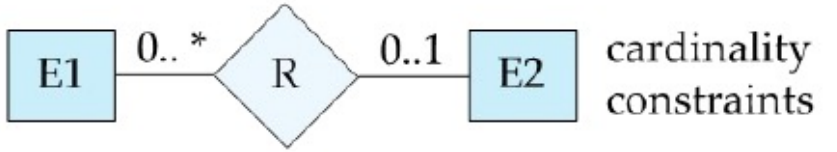
binary relationship



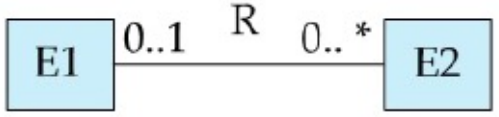


relationship attributes

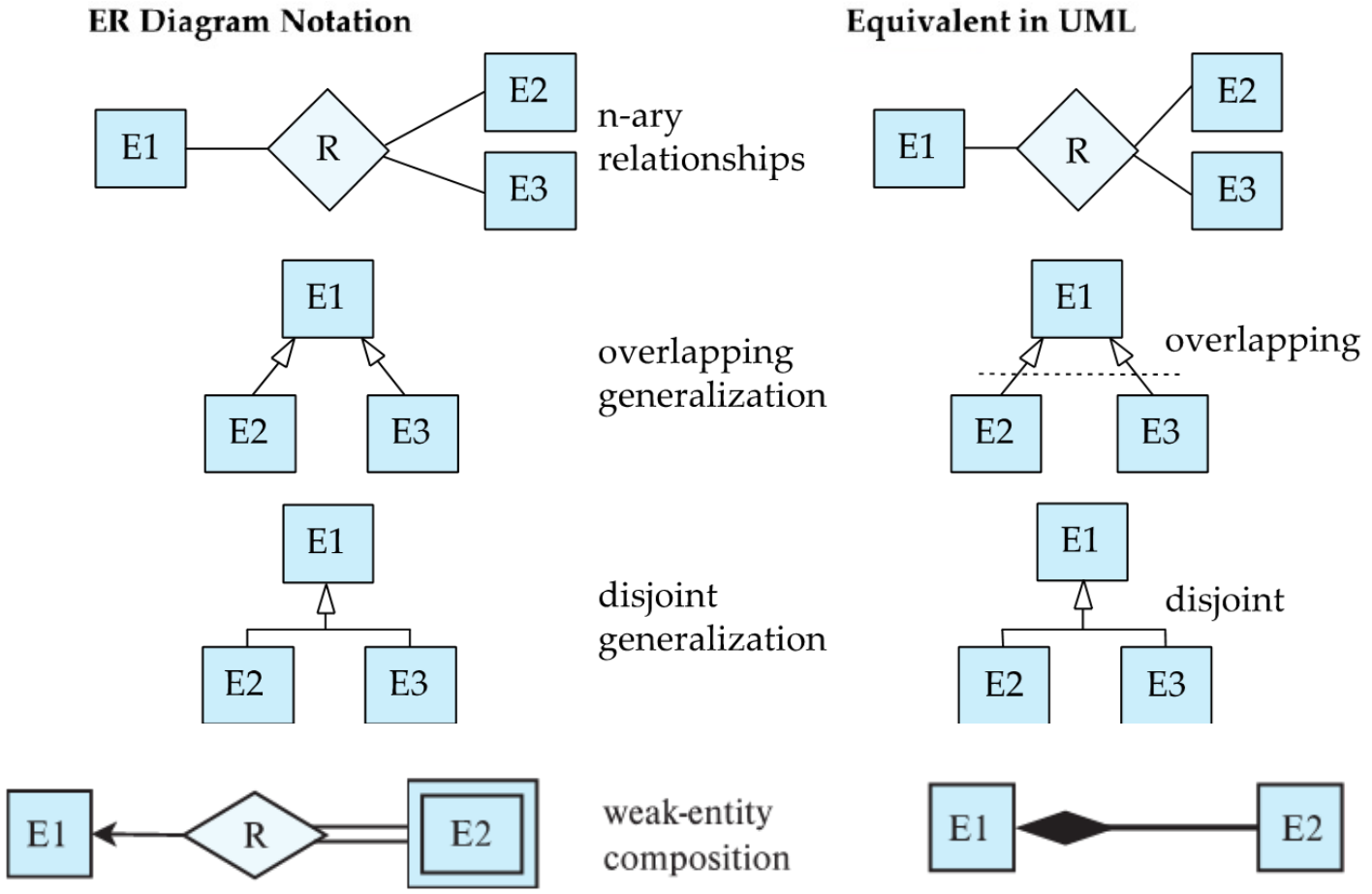




cardinality constraints



UML (Unified Modeling Language): Birleşik Modelleme Dili



UML'de overlapping, noktalı çizgi ile gösterilir.

Genelleme (Generalization), disjoint/overlapping) oklardan bağımsız olarak birleştirilmiş veya ayrı oklar kullanabilir.

UML (Unified Modeling Language): Birleşik Modelleme Dili

1. Binary ilişkiler UML'de nasıl gösterilir?

- UML'de ikili (binary) ilişkiler, iki sınıfı birbirine bağlayan düz bir çizgi ile gösterilir.
- İlişkinin adı çizginin yanına yazılır.

2. Rol isimleri nasıl gösterilir?

- Bir sınıfın ilişkide üstlendiği rol, çizginin üzerinde o sınıfa yakın bir yerde rol adı yazılarak belirtilir.

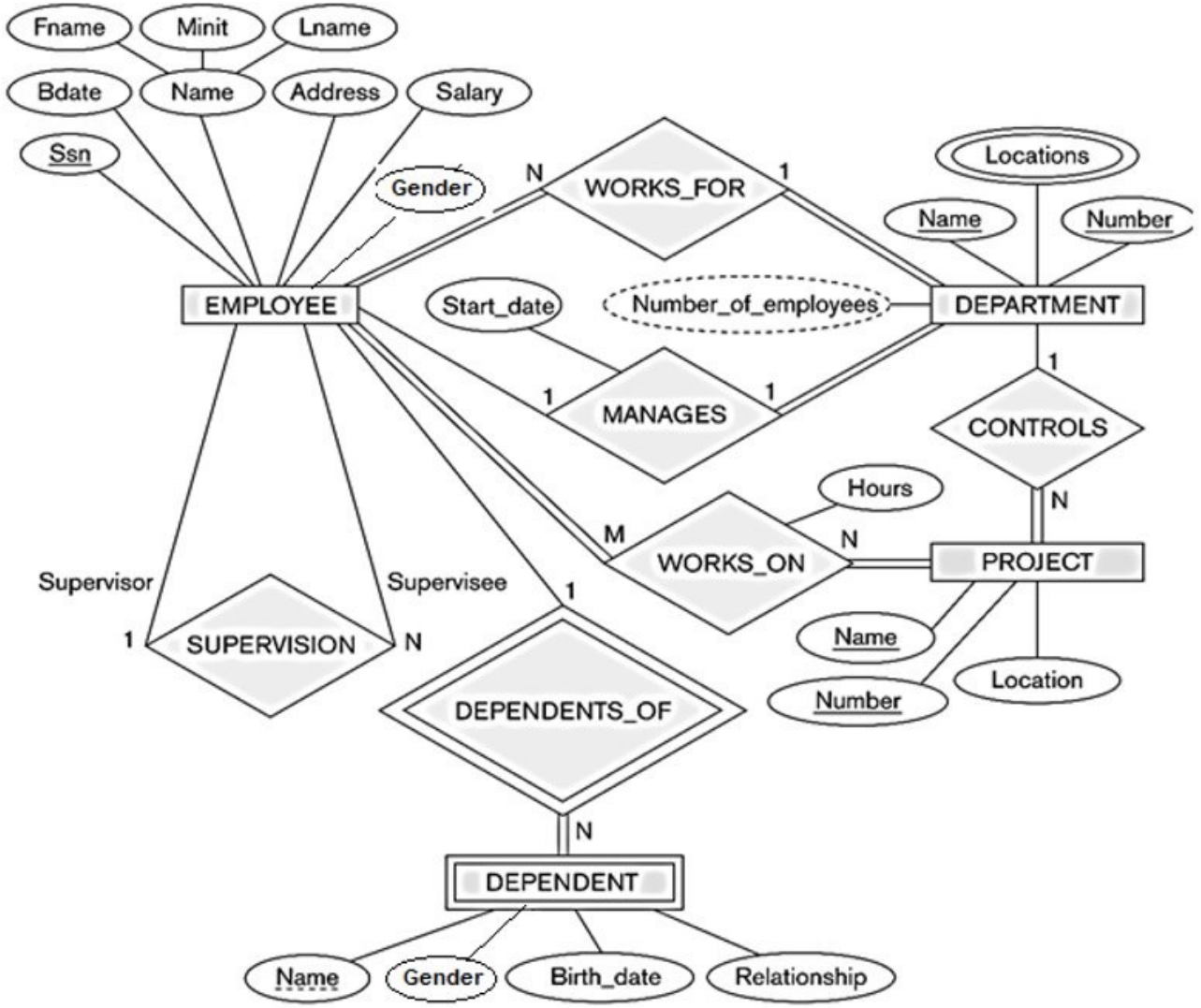
3. İlişki öznitelikleri nasıl gösterilir?

- İlişki adını ve ilişkiye ait öznitelikleri içeren bir kutu çizilebilir.
- Bu kutu, noktalı bir çizgi ile ilişki çizgisine bağlanır.
- Böylece UML'de ilişki öznitelikleri, ER diyagramındaki gibi ilişkiye bağlı bir kutu yapısıyla gösterilmiş olur.

Örnek – COMPANY Database – Gereksinimler

- Şirket, **DEPARTMAN**LARA ayrılmıştır.
  - Her departmanın bir adı, numarası ve departmanı yöneten bir çalışanı (manager) vardır.
  - Departman yöneticisinin göreve başlama tarihini tutarız.
  - Bir departmanın birden fazla lokasyonu olabilir.
- Her departman birden fazla **PROJE** yürütür (controls). Her projenin benzersiz bir adı, benzersiz bir numarası vardır ve tek bir lokasyonda yürütülür.
- Her **ÇALIŞAN**'ın sosyal güvenlik numarasını (SSN), adresini, maaşını, doğum tarihini sistemde saklarız.
  - Her çalışan bir departmanda çalışır, ancak birden fazla projede görev alabilir (works on).
  - Her çalışanın, üzerinde çalıştığı her proje için haftalık çalışma saatini tutarız.
  - Ayrıca her çalışanın doğrudan amirini (direct supervisor) kayıt ederiz.
- Bu ilişki çalışanlar arasında hiyerarşik bir ilişki oluşturan öz yinelemeli (recursive) bir ilişkidir.
- Her çalışanın birden fazla **DEPENDENT**'ı olabilir.
- Her bağımlı için: adı, cinsiyeti, doğum tarihi, çalışana olan yakınlığı (eş, çocuk vb.) bilgilerini saklarız.

Örnek – COMPANY Database E–R Diagramı



https://www.geeksforgeeks.org/dbms/dbms/

Entity Relationship Model

1. ER Model
2. Enhanced ER Model
3. Minimization of ER Diagram
4. Generalization, Specialization and Aggregation
5. Recursive Relationships

Fonksiyonel Bağımlılıklar ve Normalizasyon Temelleri

Neler konuşacağız?

1. İlişkisel Veritabanları İçin Informal Tasarım İlkeleri
   - **1.1.** Öznitelik Anlamları (Semantics of the Relation Attributes)
   - **1.2.** Gereksiz Bilgi ve Güncelleme Anomalileri (Redundant Information & Update Anomalies)
   - **1.3.** Null Değerleri (Null Values in Tuples)
   - **1.4.** Yanıltıcı Kayıtlar (Spurious Tuples)

2. Fonksiyonel Bağımlılıklar (Functional Dependencies – FDs)
   - **2.1.** Fonksiyonel Ayrıştırmanın Tanımı
   - **2.2.** Fonksiyonel Ayrıştırmalar için Çıkarım Kuralları
   - **2.3.** Fonksiyonel Ayrıştırma Kümelerinin Eşdeğerliği
   - **2.4.** Fonksiyonel Ayrıştırmaların Minimal Kümeleri

İlişkisel Veritabanları için İnformal Tasarım İlkeleri

İlişkisel veritabanı tasarımı nedir?

- Bir tabloda hangi özniteliklerin (kolonların) birlikte bulunacağına karar verme sürecidir.
- **Amaç:** "iyi" ilişki şemaları (doğru tanımlanmış tablolar) oluşturmaktır.

İlişki şemalarının iki seviyesi

1) Mantıksal seviye (logical "user view"): Kullanıcının gördüğü, verinin anlamına odaklanan soyut tasarım.
2) Fiziksel seviye (storage "base relation"): Verinin sistemde nasıl saklandığıyla ilgili daha teknik düzey.

- Tasarım süreci çoğunlukla temel (base) ilişkiler üzerinde yoğunlaşır.

1.1. İlişki Özniteliklerinin Anlamı (Semantics of the Relation Attributes)

- Bir tabloda (relation) her satır tek bir varlığı veya tek bir ilişki örneğini temsil etmelidir.

Temel İlkeler

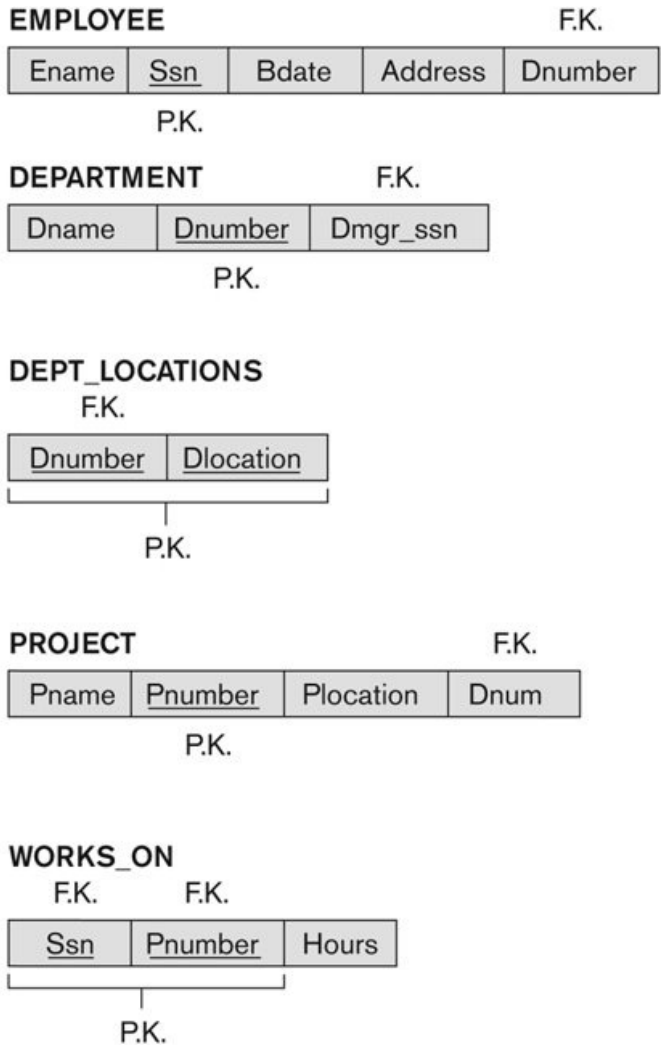
- Farklı varlıklara ait öznitelikler aynı tabloda karıştırılmamalıdır.
  - Örneğin: EMPLOYEE, DEPARTMENT, PROJECT alanları tek bir tabloda olmamalı.

- Başka varlıklara referans vermek için sadece "foreign key" kullanılmalıdır.
  - Yani ilişkiler yabancı anahtar üzerinden kurulmalıdır.

- Varlık öznitelikleri ile ilişki öznitelikleri mümkün olduğunca ayrı tutulmalıdır.

Özetle: İlişki bazında kolayca açıklanabilen bir şema tasarlayın. Niteliklerin anlamı kolayca yorumlanabilir olmalıdır.

Basitleştirilmiş bir COMPANY İlişkisel Veritabanı Şeması



EMPLOYEE

Ename	Ssn	Bdate	Address	Dnumber
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1

DEPARTMENT

Dname	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Ssn	Pnumber	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	Null

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

1.2. Tablolarda Gereksiz (Redundant) Bilgi ve Güncelleme Anomalileri

- Bir tabloda aynı bilginin tekrar tekrar tutulması gereksiz veri tekrarına yol açar.
  - Depolama alanını boşa harcar.
  - Güncelleme anomalilerine neden olur.

Güncelleme anomalisi türleri

- **Ekleme anomalisi** (Insertion anomaly): Bir bilgi parçasını eklemek için gereksiz başka alanları da doldurmak zorunda kalmak.
- **Silme anomalisi** (Deletion anomaly): Bir satırı silince, aslında kaybetmek istemediğimiz başka bilgilerin de silinmesi.
- **Güncelleme anomalisi** (Modification anomaly): Aynı bilginin birçok satırda bulunması nedeniyle, bir güncellemenin hepsine uygulanması gerekmesi — uygulanmazsa tutarsızlık oluşması.

Güncelleme Anomalisi (Update Anomaly) – Örnek

Örnek İlişki (Tablo)

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

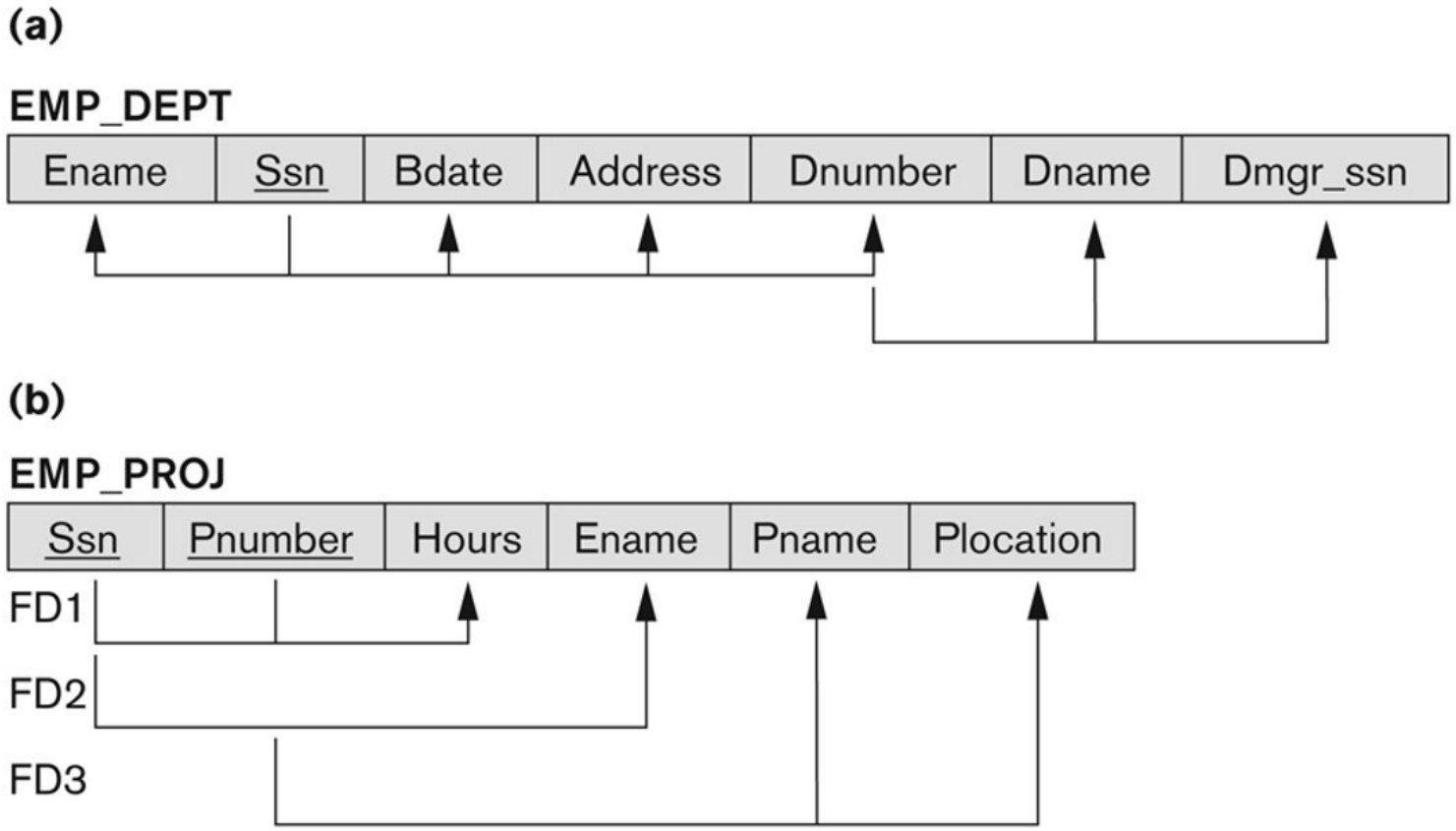
Bu tabloda çalışanlar ve çalıştıkları projeler birlikte tutulmaktadır.

- Projelerin adları (Pname) her satırda tekrar ettiği için bir proje adını değiştirmek zorlaşır.
  Örnek: Proje_no P1 için proje adı "Billing" → "Customer-Accounting" olarak değiştirilecekse:
  - P1 üzerinde çalışan 100 farklı çalışan satırının tek tek güncellenmesi gerekir.

- Fazladan iş yükü oluşturur, bazı satırlar güncellenmezse tutarsız veri ortaya çıkar.

Güncelleme Anomalisi Yaşayan İki İlişki Şeması

- Çalışan sayısı kadar aynı departman tekrar eder.



Ekleme Anomalisi (Insert Anomaly) – Örnek

Örnek İlişki (Tablo)

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

Çalışan–proje ilişkisini tek bir tabloda tutmaktadır.

- Bir projeyi ekleyemezsiniz, çünkü projeyi eklemek için tabloda bir çalışan bilgisi de olması gerekir.
  - Projeye henüz çalışan atanmadıysa tabloya ekleme yapılamaz.
- Tersi durumda da sorun vardır:
  - Bir çalışanı ekleyemezsiniz, çünkü çalışanı eklemek için onun bir projeye atanmış olması gerekir.
- Bu yapı, bağımsız varlıkların (EMPLOYEE ve PROJECT) ayrı tablolarda olması gerektiğini gösterir.

Silme Anomalisi (Delete Anomaly) – Örnek

Örnek İlişki (Tablo)

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

Bu tablo yapısında silme işlemi istenmeyen veri kayıplarına yol açabilir.

- Bir proje silindiğinde, o projede çalışan tüm çalışan kayıtları da silinir.
  - Yani çalışan bilgileri yanlışlıkla kaybolabilir.
- Tersi durum: Bir projede tek çalışan varsa ve o çalışan silinirse,
  - Proje bilgisi de tamamen kaybolur.

- Bu, projeler ve çalışanların bağımsız tablolar olması gerektiğini gösterir.

Anomali Yaşayan İki İlişki Şeması

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganization	Houston
888665555	20	NULL	Reorganization	Houston

Gereksiz Bilgi ve Güncelleme Anomalileri için Kılavuz

- Ekleme, silme ve güncelleme anomalileri oluşturmayan bir şema tasarlayın.
- Yani tablo yapısı, gereksiz tekrar ve tutarsızlık üretmemelidir.
- Eğer şemada kaçınılmaz anomaliler varsa, bunları açıkça belirtin.
- Böylece uygulamalar bu durumları göz önüne alarak tasarlanabilir.

1.3. NULL Değerler

- Tablolar mümkün olduğunca az sayıda NULL içerecek şekilde tasarlanmalıdır.
- Sık sık NULL gelen öznitelikler ayrı bir tabloya taşınabilir.

Örnek: Çalışanların yalnızca %15'inde bireysel ofis varsa, EMPLOYEE(..., Office_number) yerine

- EMP_OFFICES(Essn, Office_number) şeklinde ayrı bir tablo açmak daha doğrudur;

Null değerlerin ortaya çıkma nedenleri

- Öznitelik geçersiz veya uygulanamaz.
- Değer bilinmiyor ama sonradan girilebilir.
- Değerin var olduğu biliniyor fakat şu anda elde edilemiyor.

Null değerlerin oluşturduğu problemler

- Depolama israfı (çok sayıda boş alan).
- Sorguların zorlaşması: Filtreleme (selection), Gruplama ve toplama (aggregation) ve JOIN işlemleri NULL değerler nedeniyle karmaşık hale gelir.

1.4. Spurious Tuples (Yanıltıcı / Gerçek Dışı Kayıtlar)

- Kötü tasarlanmış ilişki şemaları, bazı JOIN işlemlerinde gerçekte var olmayan kayıtların oluşmasına yol açabilir. Bu hatalı sonuçlara spurious tuples (yanıltıcı satırlar) denir.

Lossless Join (Kayıpsız Birleştirme): Tablolara ayrılmış bir yapıyı tekrar JOIN ettiğimizde, hiçbir bilginin kaybolmaması ve yanlış kayıt üretilmemesi gerekir. Bu özellik yoksa tasarım hatalıdır.

- EMP_PROJ tablosu kötü bir tasarımdır.
  Bu nedenle tablo genelde iki ayrı tabloya bölünür: EMP_LOCS, EMP_PROJ1
- Bu ayrım, kayıpsız birleştirme sağlayarak yanlış kayıt oluşmasını engeller.

Çözüm

- İlişkiler, kayıpsız birleştirme koşulunu karşılayacak şekilde tasarlanmalıdır.
- Herhangi bir ilişkinin doğal birleştirmesi yapılarak hiçbir hatalı kayıt oluşturulmamalıdır.
- Hiçbir hatalı kayıt oluşturulmamasını garanti edecek şekilde anahtar (birincil anahtar, yabancı anahtar) çiftleri uygun şekilde ilişkilendirilmelidir.

1.4. Spurious Tuples (Yanıltıcı / Gerçek Dışı Kayıtlar)

Ayrıştırmaların iki önemli özelliği vardır:

a) Karşılık gelen birleştirmenin toplamsal olmaması veya kayıpsız olması

b) Fonksiyonel bağımlılıkların korunması.

(a) özelliği son derece önemlidir ve feda edilemez.

(b) özelliği daha az katıdır ve feda edilebilir.

2.1. Fonksiyonel Bağımlılıklar (Functional Dependencies – FD)

- Bir ilişkisel tasarımın ne kadar "iyi" olduğunu ölçmek için kullanılan resmi kriterlerdir.
- Normal formların tanımlanmasında anahtarlar ve fonksiyonel bağımlılıklar temel rol oynar.
- Bu bağımlılıklar, özniteliklerin anlamları ve aralarındaki ilişkilerden türetilen kısıtlamalardır.

Fonksiyonel Bağımlılık Tanımı

- Bir öznitelik kümesi X, başka bir öznitelik kümesi Y'yi fonksiyonel olarak belirliyorsa:
  - X'in değeri bilindiğinde, Y'nin değeri tek ve kesin olarak belirlenebilir.

$$X \rightarrow Y$$

- Örnek: TC_No → İsim, Soyisim
  (TC numarası bilindiğinde kişinin adı ve soyadı kesin olarak bulunur.)

2.1. Fonksiyonel Bağımlılıklar (Functional Dependencies – FD)

- Bir tablo içinde iki satır X için aynı değere sahipse, Y için de aynı değere sahip olmak zorundadır.

Formel İfade: Herhangi iki satır t1 ve t2 için:

- Eğer t1[X] = t2[X] ise t1[Y] = t2[Y] olmalıdır.
- Bu, fonksiyonel bağımlılığın zorunlu bir kısıtlama olduğunu gösterir.

- X → Y bir kısıtlamadır
- Bir ilişkide (R), X → Y varsa: Bu, tablonun tüm örneklerinde geçerli olmak zorunda olan bir veri bütünlüğü kuralıdır. R'deki X → Y, tüm ilişki örnekleri r(R) üzerinde bir kısıtlama belirtir.

- FD'ler, öznitelikler üzerindeki gerçek dünya kısıtlamalarından türetilir.

2.1. Fonksiyonel Bağımlılıklar – Örnek

SSN → ENAME

- Çalışanın sosyal güvenlik numarası (SSN), çalışan adını belirler. Yani aynı SSN için isim değişemez.

PNUMBER → {PNAME, PLOCATION}

- Proje numarası, projenin adını ve konumunu belirler. Her **pnumber** tek bir isim ve lokasyonla ilişkilidir.

{SSN, PNUMBER} → HOURS

- Bir çalışan (SSN) ve proje (PNUMBER) birlikte: Çalışanın projede haftalık kaç saat çalıştığını belirler. Tek başına SSN veya tek başına PNUMBER saat bilgisini belirleyemez.

{State, Driver_license_number} → SSN

- (Eyalet + ehliyet numarası bir kişiyi tekil olarak belirleyebilir.)

ZipCode → AreaCode

- (Bir posta kodu, sabit bir telefon bölge koduna karşılık gelir.)

2.1. Fonksiyonel Bağımlılıklar – Örnek

- Fonksiyonel bağımlılık (FD), bir şemanın özniteliklerine ait bir özelliktir.
  - Yani FD, verinin yapısal bir kısıtlamasıdır; her tablo tasarımında doğal olarak bulunur.

- FD kısıtı, tablonun her örneğinde geçerli olmak zorundadır.
  - Tabloya eklenen tüm satırlar bu kurala uymalıdır. FD geçerli değilse, tablo bütünlük ihlali içerir.

- Eğer K, bir tablonun anahtarı ise: K → (tüm öznitelikler) geçerlidir.
  - **Çünkü:** aynı anahtar değeriyle iki farklı satır olamaz.

- Bu nedenle anahtar, tablodaki diğer tüm değerleri tekil olarak belirler.

- "İki farklı satırın anahtar değeri aynı olamayacağı için K, diğer tüm öznitelikleri belirler."

2.1. Fonksiyonel Bağımlılıklar – Örnek

- FD'ler verinin anlamına dayanır ve her zaman geçerlidir.
- Fonksiyonel bağımlılıklar, tabloya ait verilerin gerçek dünyadaki ilişkilerini ifade eder.

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

- TEXT → COURSE

  olabilir (çünkü her kitap tek bir dersle ilişkilendiriliyormuş gibi görünüyor).

- TEACHER → COURSE

  olamaz, çünkü Smith iki farklı dersi vermektedir.

Önemli Not

- Bir FD otomatik olarak tabloya bakılarak çıkarılamaz. Aynı tablo durumu, yanlış da olsa FD varmış gibi görünebilir.
- Doğru FD'ler veri uzmanı tarafından açıkça belirtilmelidir.

2.2. 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.

2.2. 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

2.2. Fonksiyonel Bağımlılıklar İçin Çıkarım Kuralları (Inference Rules)

FD kümesinin kapanımı (Closure of a set F of FDs)

- Bir FD kümesi F verildiğinde, F'den türetilebilecek tüm FD'lerin oluşturduğu kümeye: F^+ (F'nin kapanımı) denir.
- Bu, F'nin "genişletilmiş" hâlidir.

Bir öznitelik kümesinin kapanımı (Closure of attributes X)

- Bir öznitelik kümesi X için: X^+ = X'in F'ye göre belirlediği tüm özniteliklerin kümesidir.
- Yani X'den başlayarak, FD'leri kullanarak ulaşılabilen tüm öznitelikler X^+ içinde yer alır.

3. X^+ nasıl hesaplanır?

- X^+ hesaplamak için: IR1 (Reflexive), IR2 (Augmentation), IR3 (Transitive) kuralları tekrar tekrar uygulanır ve bu sırada F kümesindeki FD'ler kullanılır.

Örnek (kısa yaklaşım):

- $F = \{ A \rightarrow B, B \rightarrow C \}$
  $X = \{A\}$
- $A \rightarrow B \quad B \rightarrow C$
  $\Rightarrow A^+ = \{A, B, C\}$

2.3. Fonksiyonel Bağımlılıkların Denkliği (Equivalence of Sets of FDs)

İki FD kümesi F ve G ne zaman denk kabul edilir?

- F ve G eşdeğer (equivalent) ise: F içindeki her FD, G'den çıkarılabiliyorsa G içindeki her FD, F'den çıkarılabiliyorsa
- Yani: $F^+ = G^+$ (kapanımları eşittir)

"Kapsama" (Covers) Tanımı

- F, G'yi kapsar (F covers G) ⇢ Eğer G'deki her FD, F kullanılarak türetilebiliyorsa.
- Başka bir ifadeyle: G^+, F^+ nın alt kümesidir.

Denkliğin Sonuç Tanımı

- F ve G şu durumda denktir: F, G'yi kapsar. G, F'yi kapsar.
- Yani iki yönlü kapsama vardır.

2.4. Minimal Fonksiyonel Bağımlılık Kümeleri

- Minimal FD kümesi, gereksiz hiçbir bağımlılık veya öznitelik içermeyen en sade biçimdir.
- Bir FD kümesi minimal ise aşağıdaki üç koşulu sağlar.

1. Her FD'nin sağ tarafında (RHS) sadece tek bir öznitelik olmalıdır.

- Yani: X → YZ gibi çoklu sağ taraf olmaz.
  Bunun yerine: X → Y X → Z şeklinde ayrıştırılmış olmalıdır.

2. F içindeki herhangi FD'yi silince, geri kalan küme F'ye denk olmamalıdır.

- Hiçbir bağımlılık gereksiz olmamalıdır.
- Bir bağımlılığı kaldırırsak, kapanım değişiyorsa o bağımlılık zorunludur.

3. X → A şeklindeki bir FD'de X'in yerini daha küçük bir küme Y ile değiştiremeyiz.

- Yani: Eğer Y, X'in gerçek bir alt kümesi ise (Y ⊂ X), "Y → A" bağımlılığını koyduğumuzda F ile eşdeğerlik bozuluyorsa, X → A zorunludur.
- Bu koşul: Sol tarafta gereksiz öznitelik olmamasını sağlar.

Minimal FD kümesi:

- Sağ tarafta tek öznitelik,
- Gereksiz FD yok,
- Sol tarafta gereksiz öznitelik yok

Bu nedenle minimal kümeler normalizasyon ve şema tasarımında çok önemlidir.

2.4. Minimal Fonksiyonel Bağımlılık Kümeleri

1. Her FD kümesinin ona denk bir minimal kümesi vardır.

- Yani F kümesi için mutlaka minimal bir sürüm (minimal cover) bulunabilir.

2. Birden fazla denk minimal FD kümesi olabilir.

- Minimal küme tek değildir; farklı minimal kümeler aynı kapanımı verebilir.

3. Minimal küme bulmak için basit bir algoritma yoktur.

- Tam otomatik, kolay bir yöntem yok. Genellikle adım adım:
  - RHS ayrıştırma
  - Gereksiz FD kontrolü
  - Sol tarafı sadeleştirme

  gibi işlemler yapılır.

2.4. Minimal Fonksiyonel Bağımlılık Kümeleri – Örnek

Verilen FD kümesi: $E = \{B \rightarrow A,\ D \rightarrow A,\ AB \rightarrow D\}$ **Amaç:** E'nin minimal cover'ını bulmak.

Adım 1 — RHS Tekli mi?

Minimal cover'ın 1. şartı: Her FD'nin sağ tarafında tek bir öznitelik olmalı.

Bizim tüm FD'lerimiz zaten tek özniteliğe sahip. Adım 1 tamam.

Adım 2 — Sol tarafta gereksiz öznitelik var mı?

Özellikle AB → D. FD'sine bakılır:

Soru: AB → D yerine B → D veya A → D olabilir mi? Test: B → D elde edilebilir mi?

B → A var. B → A olduğu için sol tarafa B ekleyelim ⇒ IR2 (Augmentation): $B \rightarrow A \Rightarrow B \rightarrow AB$

Şimdi elimizde: B → AB AB → D

İkisinin transitifi (IR3): $B \rightarrow D$

Bu sonuç AB → D'nin yerine B → D yazılabileceğini gösterir. Yani AB'nin sol tarafındaki A gereksizdir.

Bu durumda yeni küme: $E' = \{B \rightarrow A,\ D \rightarrow A,\ B \rightarrow D\}$

2.4. Minimal Fonksiyonel Bağımlılık Kümeleri – Örnek

Verilen FD kümesi: $E = \{B, \rightarrow, A\ D \rightarrow A\ AB \rightarrow D\}$ **Amaç:** E'nin minimal cover'ını bulmak.

Adım 3 — Gereksiz FD var mı?

Bu aşamada FD'lerden biri diğerlerinden çıkarılabiliyorsa gereksizdir.

Test edelim: B → A gereksiz olabilir mi?

Elimizde: B → D. D → A B → D ve D → A'nın transitifi: B → A

Yani B → A zaten diğer FD'lerden elde edilebiliyor → gereksiz! Sil

Sonuç: Minimal FD Kümesi $\boxed{\{B, \rightarrow D\ D \rightarrow A\}}$ Bu set hem daha küçük hem de orijinal setle eşdeğerdir.

ÖZET

- Sağ taraflar tek → tamam. $E' = \{B, \rightarrow, A\ D \rightarrow A\ AB \rightarrow D\}$
- AB → D içinde A gereksiz → B → D yap. $E' = \{B, \rightarrow, A\ D \rightarrow A\ B \rightarrow D\}$
- B → A zaten B → D → A yoluyla türetiliyor → sil. $E'' = \{B, \rightarrow D\ D \rightarrow A\}$

3. 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**

3. 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.

3. 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 – G Kümesi

- G = AB → C BD → EF AD → GH A → I H → J

ABD minimal mi? Başka anahtar var mı?

- Kontrol edelim:
- AD^+: AD → GH. A → I Ama E,F çıkmıyor → anahtar değil.
- BD^+: BD → EF Ama A, I, G,H gelmez → anahtar değil.
- AB^+ zaten anahtar değil.

- Dolayısıyla ABD'nin herhangi bir altkümesi anahtar olamaz.
- ABD minimal ve tek anahtar gibi duruyor.

YILDIZ TEKNİK ÜNİVERSİTESİ

Yasemin Topuz

Yıldız Teknik Üniversitesi

ytopuz@yildiz.edu.tr