

Veritabanı Yönetim Sistemleri

1906003022015

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

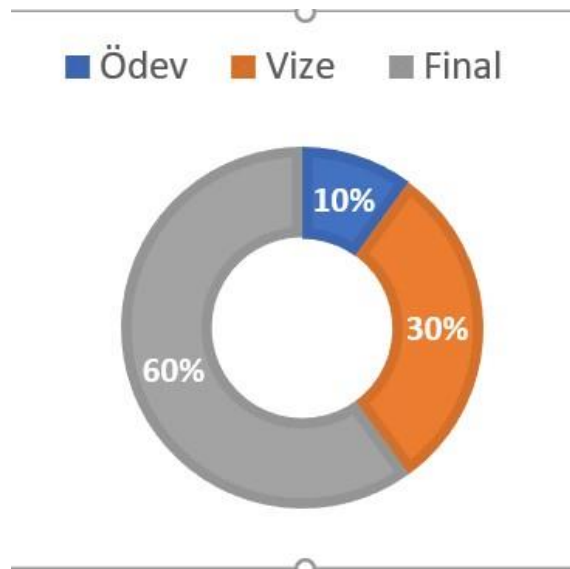


Giriş

Ders Günü ve Saati:

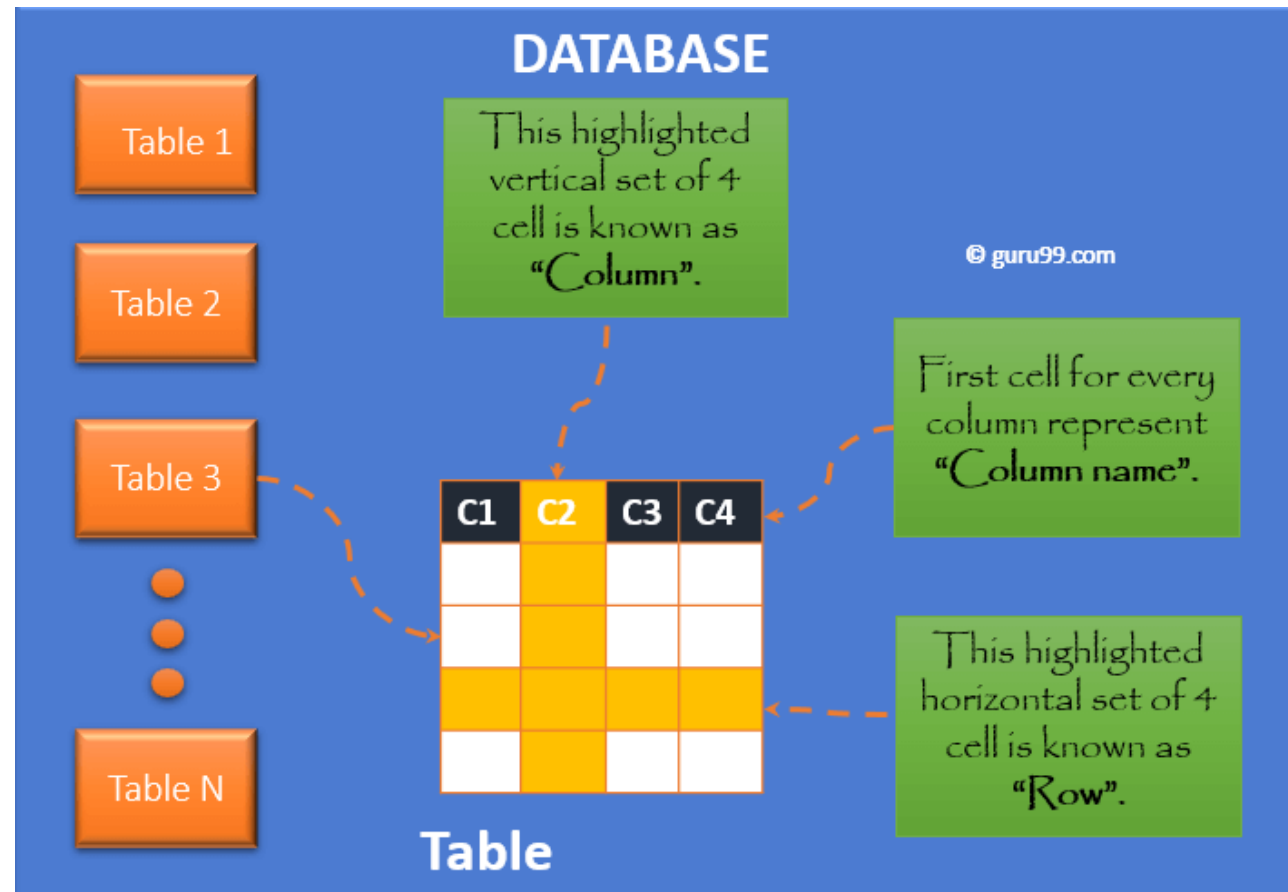
Pazartesi: 09:15-12:45

- Devam zorunluluğu %70
- Uygulamalar MS SQL ve MongoDB üzerinde gerçekleştirilecektir.



HAFTA	KONULAR
Hafta 1	VT ve VTYS'ne giriş
Hafta 2	ER Veri Modeli
Hafta 3	İlişkisel Modeller, İlişkisel model tasarımı
Hafta 4	İlişkisel Cebir ve Hesaplamalar
Hafta 5	İlişkisel Sorgular, SQL giriş
Hafta 6	SQL ile veri tabanı programlama
Hafta 7	SQL-Kısıtlar:Veri-tipi,birincil-anahtar,ikinci-anahtar,
Hafta 8	Vize
Hafta 9	İlişkisel Veri Tabanı Tasarımı ve Normalizasyon
Hafta 10	yarı-yapısal veri modelleri, XML
Hafta 11	JSON
Hafta 12	İlişkisel olmayan DB, NoSQL
Hafta 13	NoSQL
Hafta 14	DBMS -Eşzamanlılık (Concurrency) Kontrolü

SQL Server Table: CREATE, ALTER, DROP



MySQL Introduction

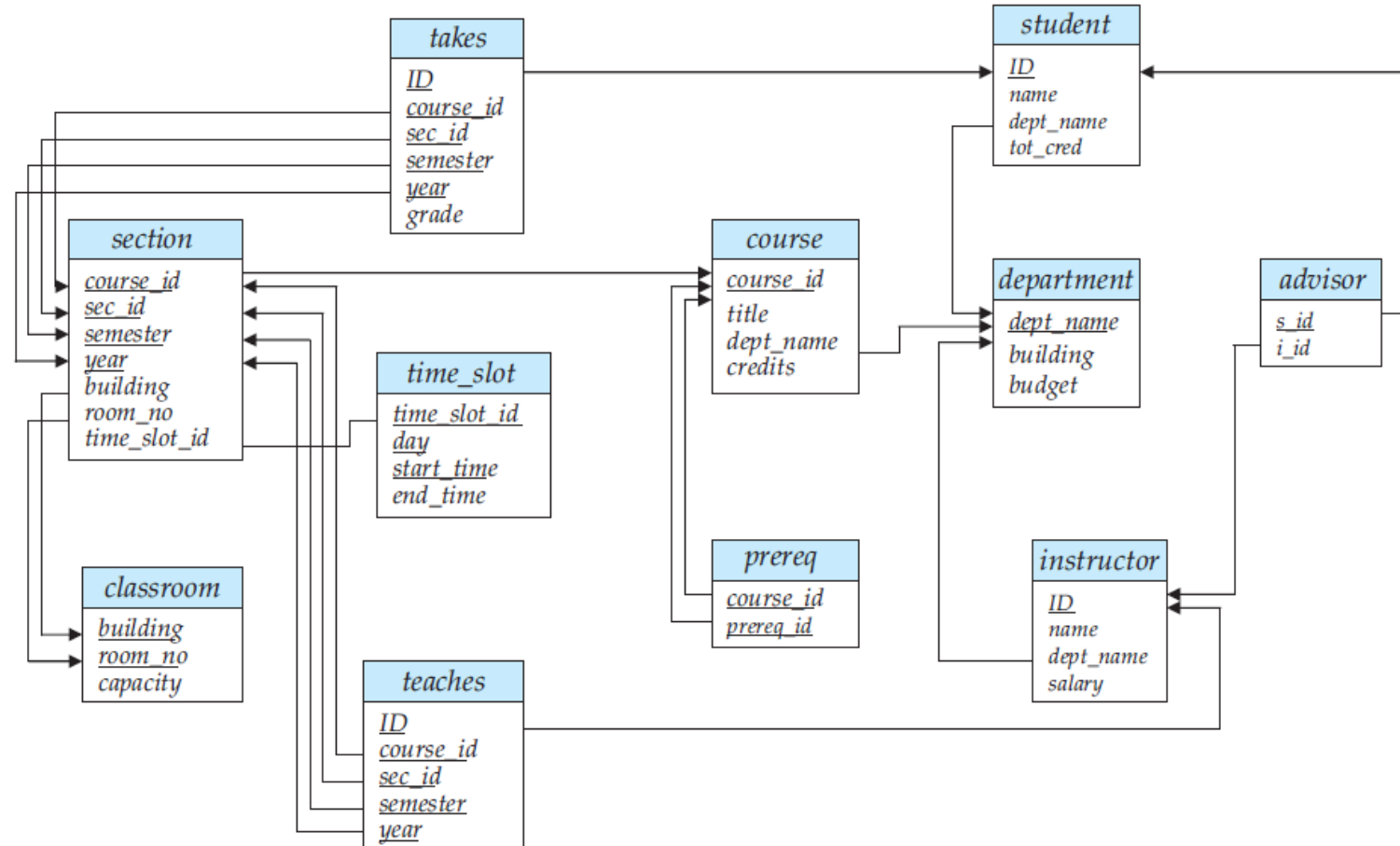


Figure 2.8 Schema diagram for the university database.

MySQL Introduction

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

Figure 2.9 Schema of the university database.

MySQL Introduction

	course_id	title	dept_name	credits
►	123	Deneme	Comp. Sci.	5
	BIO-101	Intro. to Biology	Biology	4
	BIO-301	Genetics	Biology	4
	BIO-399	Com Genetics jology	Biology	3
	CS-101	Intro. to Computer Science	Comp. Sci.	4
	CS-190	Game Design	Comp. Sci.	4
	CS-315	Robotics	Comp. Sci.	3
	CS-319	Imag Processing	Comp. Sci.	3
	CS-347	Database System Concepts	Comp. Sci.	3
	EE-181	Intro. to Digital Systems	Elec. Eng.	3
	FIN-201	Investment Banking	Finance	3
	HIS-351	World History	History	3
	MU-199	Music Video Production	Music	3
	PHY-101	Physical Principles	Physics	4

course

	dept_name	building	budget
►	Biology	Watson	90000.00
	Comp. Sci.	Taylor	100000.00
	Elec. Eng.	Taylor	85000.00
	Finance	Painter	120000.00
	History	Painter	50000.00
	Music	Packard	80000.00
	Physics	Watson	70000.00

department

	ID	name	dept_name	salary
►	10101	Srinivasan	Comp. Sci.	65000.00
	12121	Wu	Finance	90000.00
	15151	Mozart	Music	40000.00
	22222	Einstein	Physics	95000.00
	32343	El Said	History	60000.00
	33456	Gold	Physics	87000.00
	45565	Katz	Comp. Sci.	75000.00
	58583	Califieri	History	62000.00
	76543	Singh	Finance	80000.00
	76766	Crick	Biology	72000.00
	83821	Brandt	Comp. Sci.	92000.00
	98345	Kim	Elec. Eng.	80000.00

instructor

MySQL Introduction

	course_id	sec_id	semester	year	building	room_number	time_slot_id
►	BIO-101	1	Summer	2009	Painter	514	B
	BIO-301	1	Summer	2010	Painter	514	A
	CS-101	1	Fall	2009	Packard	101	H
	CS-101	1	Spring	2010	Packard	101	F
	CS-190	1	Spring	2009	Taylor	3128	E
	CS-190	2	Spring	2009	Taylor	3128	A
	CS-315	1	Spring	2010	Watson	120	D
	CS-319	1	Spring	2010	Watson	100	B
	CS-319	2	Spring	2010	Watson	3128	C
	CS-347	1	Fall	2009	Taylor	3128	A
	EE-181	1	Spring	2009	Taylor	3128	C
	FIN-201	1	Spring	2010	Packard	101	B
	HIS-351	1	Spring	2010	Painter	514	C
	MU-199	1	Spring	2010	Packard	101	D
	PHY-101	1	Fall	2009	Watson	100	A

section

	ID	course_id	sec_id	semester	year
►	76766	BIO-101	1	Summer	2009
	76766	BIO-301	1	Summer	2010
	10101	CS-101	1	Fall	2009
	45565	CS-101	1	Spring	2010
	83821	CS-190	1	Spring	2009
	83821	CS-190	2	Spring	2009
	10101	CS-315	1	Spring	2010
	45565	CS-319	1	Spring	2010
	83821	CS-319	2	Spring	2010
	10101	CS-347	1	Fall	2009
	98345	EE-181	1	Spring	2009
	12121	FIN-201	1	Spring	2010
	32343	HIS-351	1	Spring	2010
	15151	MU-199	1	Spring	2010
	22222	PHY-101	1	Fall	2009

teaches

SQL GROUP BY

```
SELECT country, COUNT(*) AS number  
FROM Customers  
GROUP BY country;
```

```
1 select dept_name, avg (salary) as avg_salary  
2 from instructor  
3 group by dept_name;
```

dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	77333.333333
Elec. Eng.	80000.000000
Finance	85000.000000
History	61000.000000
Music	40000.000000
Physics	91000.000000

Nested Subqueries(SQL ANY and ALL)

```
SELECT *  
FROM Teachers  
WHERE t_age = ANY (  
    SELECT s_age  
    FROM Students  
);
```

```
SELECT *  
FROM Teachers  
WHERE t_age > ALL (  
    SELECT s_age  
    FROM Students  
);
```

t_id	t_name	t_age
1	Peter	32
2	Megan	43
3	Rose	29
4	Linda	30
5	Mary	41

s_id	s_name	age
1	Harry	23
2	Jack	42
3	Joe	32
4	Dent	23
5	Bruce	40

```
SELECT *  
FROM Teachers  
WHERE t_age = ANY (  
    SELECT s_age  
    FROM Students  
);
```

t_id	t_name	t_age
1	Peter	32

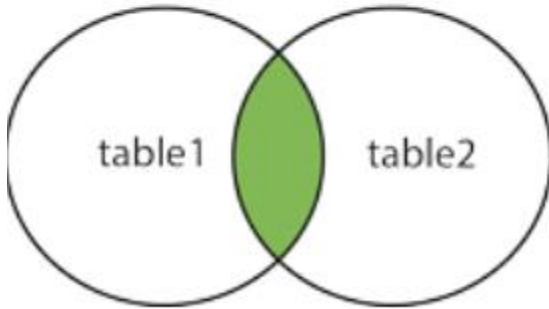
DBMS – Birleştirmeler (Join)

- İki ilişkinin Kartezyen çarpımını almanın faydalarını anlıyoruz, bu da bize birlikte eşleştirilmiş tüm olası tupl'ları verir. Ancak bazı durumlarda, hatırı sayılır sayıda niteliğe sahip binlerce demet ile büyük ilişkilerle karşılaştığımız bir Kartezyen ürünü almak bizim için mümkün olmayabilir.
 - Birleştirme, seçim işlemini takip eden bir Kartezyen çarpımın kombinasyonudur. Bir Birleştirme işlemi, ancak ve ancak belirli bir birleştirme koşulu sağlanırsa farklı ilişkilerden iki demeti eşler.
 - Join DBMS'de Join , birleştirme çarpımı ve seçimi tek bir ifadede birleştirmenizi sağlayan ikili bir işlemdir. Bir birleştirme koşulu oluşturmanın amacı, iki veya daha fazla DBMS tablosundaki verileri birleştirmenize yardımcı olmasıdır. DBMS'deki tablolar, birincil anahtar ve yabancı anahtarlar kullanılarak ilişkilendirilir.
1. Inner Joins: Theta, Natural, EQUI
 2. Outer Join: Left, Right, Full

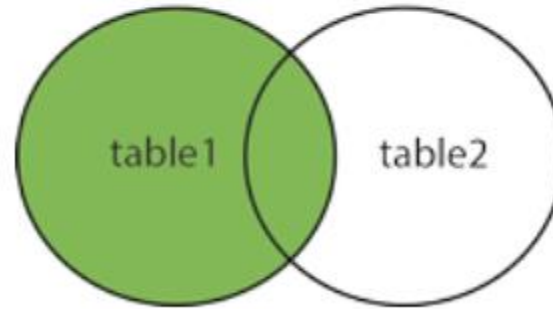


DBMS – Birleştirmeler (Join)

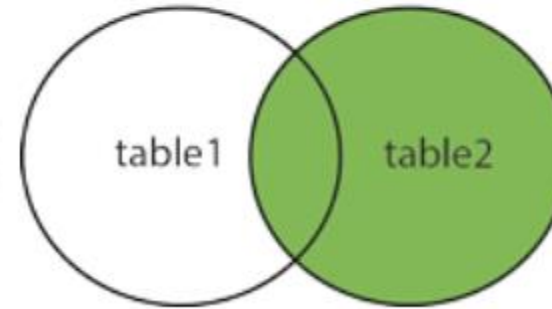
INNER JOIN



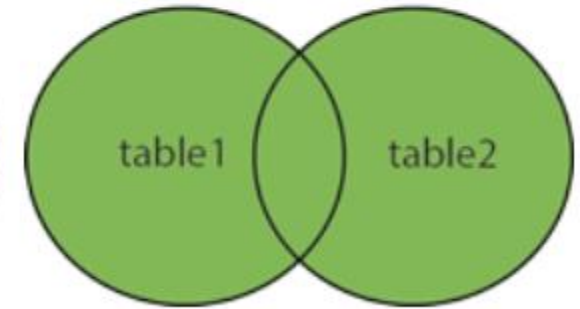
LEFT JOIN



RIGHT JOIN



FULL OUTER JOIN



DBMS – Inner Join

```
SELECT columns  
FROM table_1  
INNER JOIN table_2  
ON table_1.column = table_2.column;
```

```
use Eyon304;  
SELECT dbo.Ogrenci.oAdi, [dbo].[Ogrenci].[oNo], [dbo].[Ogrenci].[oYas]  
      , [dbo].[Ders].[dKod], [dbo].[Ders].[dAdi]  
FROM [dbo].[Ogrenci] INNER JOIN [dbo].[Ders]  
ON [dbo].[Ogrenci].[oDers]=[dbo].[Ders].[dKod];
```

DBMS – Inner Join

```
SELECT * FROM university.instructor INNER JOIN university.teaches  
ON university.instructor.ID=university.teaches.ID;
```

ID	name	dept_name	salary	ID	course_id	sec_id	semester	year
76766	Crick	Biology	72000.00	76766	BIO-101	1	Summer	2009
76766	Crick	Biology	72000.00	76766	BIO-301	1	Summer	2010
10101	Srinivasan	Comp. Sci.	65000.00	10101	CS-101	1	Fall	2009
45565	Katz	Comp. Sci.	75000.00	45565	CS-101	1	Spring	2010
83821	Brandt	Comp. Sci.	92000.00	83821	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000.00	83821	CS-190	2	Spring	2009
10101	Srinivasan	Comp. Sci.	65000.00	10101	CS-315	1	Spring	2010
45565	Katz	Comp. Sci.	75000.00	45565	CS-319	1	Spring	2010
83821	Brandt	Comp. Sci.	92000.00	83821	CS-319	2	Spring	2010
10101	Srinivasan	Comp. Sci.	65000.00	10101	CS-347	1	Fall	2009
98345	Kim	Elec. Eng.	80000.00	98345	EE-181	1	Spring	2009
12121	Wu	Finance	90000.00	12121	FIN-201	1	Spring	2010
32343	El Said	History	60000.00	32343	HIS-351	1	Spring	2010
15151	Mozart	Music	40000.00	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000.00	22222	PHY-101	1	Fall	2009

DBMS - Inner Joins: Theta (θ) Join

Theta join, teta koşulunu sağladıkları sürece farklı ilişkilerden tupleları birleştirir. Birleştirme koşulu θ sembolü ile gösterilir.

$$R1 \bowtie_{\theta} R2$$

$R1$ ve $R2$, $(A1, A2, \dots, An)$ ve $(B1, B2, \dots, Bn)$ özniteliklere sahip ilişkilerdir, öyle ki özniteliklerin ortak bir yanı yoktur, yani $R1 \cap R2 = \Phi$.

DBMS - Inner Joins: Theta (θ) Join

 $R1 \bowtie_{\theta} R2$

Table A		Table B	
column 1	column 2	column 1	column 2
1	1	1	1
1	2	1	3

 $A \bowtie A.\text{column 2} = B.\text{column 2} (B)$ $A \bowtie A.\text{column 2} > B.\text{column 2} (B)$

column 1	column 2
1	2

DBMS - Inner Joins: EQUI Join

Bir Theta birleştirmesi yalnızca eşdeğerlik koşulunu kullandığında yapılır. EQUI birleştirme, bir RDBMS'de verimli bir şekilde uygulanması en zor işlemdir ve RDBMS'nin temel performans sorunlarına sahip olmasının bir nedeni.

Student		
SID	Name	Std
101	Alex	10
102	Maria	11

Subjects	
Class	Subject
10	Math
10	English
11	Music
11	Sports

DBMS - Inner Joins: EQUI Join

STUDENT ⋈_{Student.Std = Subject.Class} SUBJECT

Student_detail				
SID	Name	Std	Class	Subject
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equijoin: Theta join sadece eşitlik karşılaştırma operatörü kullandığında, equijoin olduğu söylenir. Yukarıdaki örnek equijoin'e karşılık gelir.



DBMS - Natural Join (\bowtie)

Doğal birleştirme, herhangi bir karşılaştırma operatörü kullanmaz. Kartezyen bir çapımın yaptığı gibi birleştirmez.

Yalnızca iki ilişki arasında en az bir ortak özellik varsa Doğal Birleştirme gerçekleştirebiliriz.

Ek olarak, özniteliklerin aynı ada ve alana sahip olması gerekir.

Doğal birleştirme, her iki ilişkideki özniteliklerin değerlerinin aynı olduğu eşleşen özniteliklere etki eder.

DBMS - Natural Join (\bowtie)

Courses		
CID	Course	Dept
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

HoD	
Dept	Head
CS	Alex
ME	Maya
EE	Mira

DBMS - Natural Join (\bowtie)

Courses \bowtie HoD			
Dept	CID	Course	Head
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

SQL SELECT –Natural Join

```
1 SELECT * FROM instructor NATURAL JOIN teaches NATURAL JOIN course;
```

+ Seçenekler

dept_name	course_id	ID	name	salary	sec_id	semester	year	title	credits
Biology	BIO-101	76766	Crick	72000.00	1	Summer	2009	Intro. to Biology	4
Biology	BIO-301	76766	Crick	72000.00	1	Summer	2010	Genetics	4
Comp. Sci.	CS-101	10101	Srinivasan	65000.00	1	Fall	2009	Intro.to Computer Science	4
Comp. Sci.	CS-101	45565	Katz	75000.00	1	Spring	2010	Intro.to Computer Science	4
Comp. Sci.	CS-190	83821	Brandt	92000.00	1	Spring	2009	Game Design	4
Comp. Sci.	CS-190	83821	Brandt	92000.00	2	Spring	2009	Game Design	4
Comp. Sci.	CS-315	10101	Srinivasan	65000.00	1	Spring	2010	Robotics	3
Comp. Sci.	CS-319	45565	Katz	75000.00	1	Spring	2010	Imag Processing	3
Comp. Sci.	CS-319	83821	Brandt	92000.00	2	Spring	2010	Imag Processing	3
Comp. Sci.	CS-347	10101	Srinivasan	65000.00	1	Fall	2009	Database System Concepts	3
Elec. Eng.	EE-181	98345	Kim	80000.00	1	Spring	2009	Intro. to Digital Systems	3
Finance	FIN-201	12121	Wu	90000.00	1	Spring	2010	Investment Banking	3
History	HIS-351	32343	El Said	60000.00	1	Spring	2010	World History	3
Music	MU-199	15151	Mozart	40000.00	1	Spring	2010	Music Video Production	3
Physics	PHY-101	22222	Einstein	95000.00	1	Fall	2009	Physical Principles	4

DBMS - Outer Joins

OUTER JOIN, iki birleştirme tablosundaki her kaydın eşleşen bir kayda sahip olmasını gerektirmez. Bu tür birleştirmede, tablo, eşleşen başka bir kayıt olmasa bile her kaydı tutar.

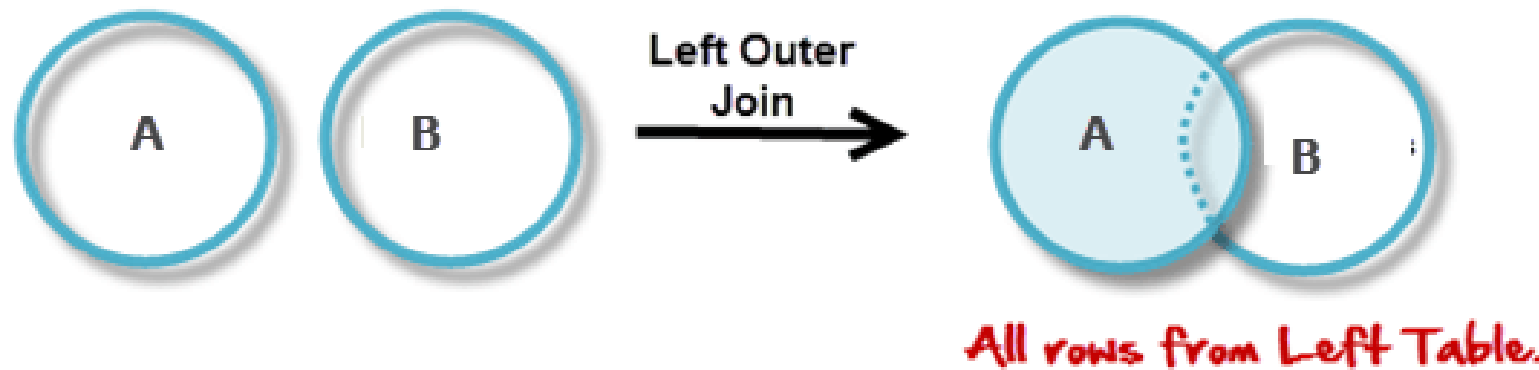
Theta Join, Equijoin ve Natural Join iç birleşimler olarak adlandırılır. Bir iç birleşim, yalnızca eşleşen özniteliklere sahip tuple'ları içerir ve geri kalanlar ortaya çıkan ilişkide atılır. Bu nedenle, ortaya çıkan ilişkiye katılan ilişkilerden gelen tüm demetleri dahil etmek için dış birleşimleri kullanmamız gerekir.

Üç tür dış birleşim vardır - sol dış birleşim, sağ dış birleşim ve tam dış birleşim.

DBMS Left Outer Join(R S)


LEFT JOIN, sağdaki tabloda eşleşen satır bulunmasa bile soldaki tablodan tüm satırları döndürür. Sağdaki tabloda eşleşen kayıt bulunamadığında NULL döndürülür.

Sol ilişkiden gelen tüm tuplelar, sonuçta ortaya çıkan ilişkiye dahil edilir. Sağ bağıntısında S herhangi bir eşleşen tuple olmadan R'de tuplelar varsa, ortaya çıkan ilişkinin S öznitelikleri NULL yapılır.




DBMS Left Outer Join(R S)

A	
Num	Square
2	4
3	9
4	16

A  B

B	
Num	Cube
2	8
3	18
5	75

A  B		
Num	Square	Cube
2	4	8
3	9	18
4	16	-



DBMS Left Outer Join(R S)

```
SELECT Students.admission, Students.firstName, Students.lastName, Fee.amount_paid
FROM Students
LEFT OUTER JOIN Fee
ON Students.admission = Fee.admission
```

```
= use Eyon304;
= SELECT dbo.Ogrenci.oAdi, [dbo].[Ogrenci].[oNo], [dbo].[Ogrenci].[oYas]
, [dbo].[Ders].[dKod], [dbo].[Ders].[dAdi]
FROM [dbo].[Ogrenci] LEFT OUTER JOIN [dbo].[Ders]
ON [dbo].[Ogrenci].[oDers]=[dbo].[Ders].[dKod];
```

DBMS Right Outer Join (A \bowtie B)

Sağ bağıntısından gelen tüm tuplelar, sonuçta ortaya çıkan ilişkiye dahil edilir. S'de R'de eşleşen herhangi bir tuple olmayan tuplealar varsa, sonuçta ortaya çıkan ilişkinin R-nitelikleri NULL yapılır. RIGHT JOIN, soldaki tabloda eşleşen satır bulunmasa bile sağdaki tablodan tüm sütunları döndürür. Soldaki tabloda herhangi bir eşleşme bulunamadığında, NULL döndürülür. RIGHT dış JOIN, LEFT JOIN'in tersidir




DBMS Right Outer Join (A B)

A	
Num	Square
2	4
3	9
4	16

A  B

B	
Num	Cube
2	8
3	18
5	75

A  B		
Num	Cube	Square
2	8	4
3	18	9
5	75	-

DBMS Right Outer Join (A B)

```
SELECT Students.admission, Students.firstName, Students.lastName, Fee.amount_paid  
FROM Students  
RIGHT OUTER JOIN Fee  
ON Students.admission = Fee.admission
```

```
= use Eyon304;  
= SELECT dbo.Ogrenci.oAdi, [dbo].[Ogrenci].[oNo], [dbo].[Ogrenci].[oYas]  
  , [dbo].[Ders].[dKod], [dbo].[Ders].[dAdi]  
FROM [dbo].[Ogrenci] RIGHT OUTER JOIN [dbo].[Ders]  
ON [dbo].[Ogrenci].[oDers]=[dbo].[Ders].[dKod];
```

DBMS Full Outer Join (A ⋈ B)

Her iki katılımcı ilişkiden tüm demetler ortaya çıkan ilişkiye dahil edilir. Her iki ilişki için eşleşen tuple yoksa, bunların eşleşmeyen öznitelikleri NULL yapılır.

Bir TAM DIŞ BİRLEŞTİRME'de, her iki ilişkiden gelen tüm tuplelar, eşleşen koşuldan bağımsız olarak sonuca dahil edilir.

A ⋈ B

A ⋈ B		
Num	Square	Cube
2	4	8
3	9	18
4	16	-
5	-	75

DBMS Full Outer Join (A B)

```
SELECT Students.admission, Students.firstName, Students.lastName, Fee.amount_paid
FROM Students
FULL OUTER JOIN Fee
ON Students.admission = Fee.admission
```

```
= use Eyon304;
= SELECT dbo.Ogrenci.oAdi, [dbo].[Ogrenci].[oNo], [dbo].[Ogrenci].[oYas]
, [dbo].[Ders].[dKod], [dbo].[Ders].[dAdi]
FROM [dbo].[Ogrenci] FULL OUTER JOIN [dbo].[Ders]
ON [dbo].[Ogrenci].[oDers] = [dbo].[Ders].[dKod];
```

Hafta 9

DBMS - Normalleştirme



İlişkisel Tasarımı İyileştirme

- Genel olarak, ilişkisel veritabanı tasarımının amacı, gereksiz fazlalık olmadan bilgileri depolamamıza izin veren bir dizi ilişki şeması oluşturmaktır, ancak aynı zamanda bilgileri kolayca almamıza da izin verir. Bu, uygun bir normal formda olan şemalar tasarlanarak gerçekleştirilir.
- Bir ilişki şemasının istenen normal formlardan birinde olup olmadığını belirlemek için, veritabanıyla modellediğimiz gerçek dünya girişimi hakkında bilgiye ihtiyacımız var. Bu bilgilerin bir kısmı iyi tasarlanmış bir E-R diyagramında bulunur.

İlişkisel Tasarım İyileştirme

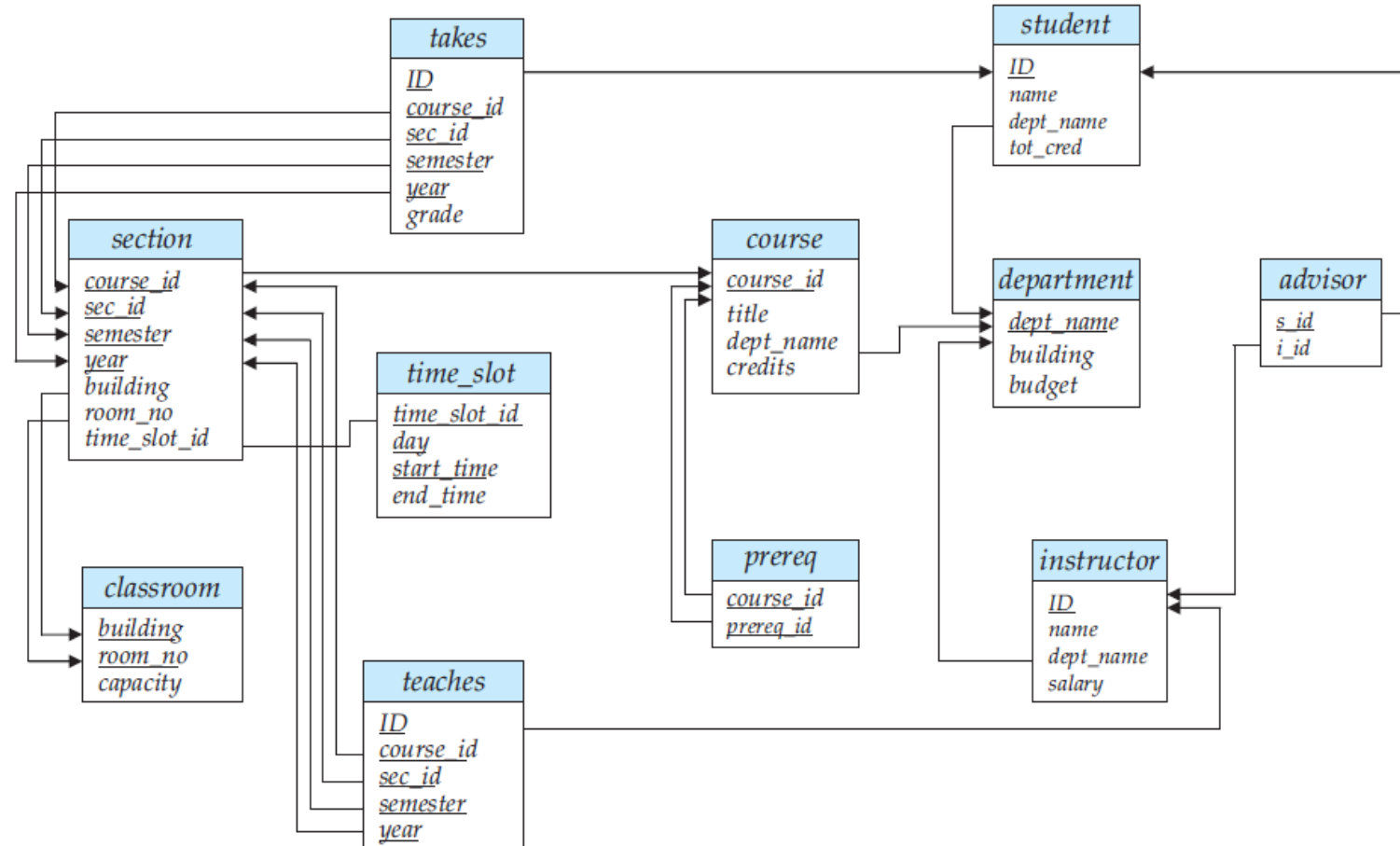


Figure 2.8 Schema diagram for the university database.

İlişkisel Tasarımı İyileştirme

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

Figure 2.9 Schema of the university database.

Design Alternative: Larger Schemas

inst_dept (ID, name, salary, dept_name, building, budget)

<i>ID</i>	<i>name</i>	<i>salary</i>	<i>dept_name</i>	<i>building</i>	<i>budget</i>
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

- E-R tasarımıımıza yol açan üniversite hakkındaki gerçekler hakkında dikkatlice düşünene kadar bazı sorgular daha az birleştirme kullanılarak ifade edilebilir.

Design Alternative: Smaller Schemas

inst_dept (ID, name, salary, dept_name, building, budget)

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

- Bilgi tekrarı
- Tekrar olmamasının sadece “şanslı” bir özel durum mu yoksa genel bir kuralın tezahürü mü olduğunu belirlememize izin vermez.
- “dept_name için her belirli değer en fazla bir _bud_get'e karşılık gelir” gibi kurallar belirlemesine izin vermemiz gerekir. Yani bir şema (dept_name, budget) olsaydı dept_name birincil anahtar görevi görebilir diye bir kural yazmamız gerekiyor. Bu kural, **işlevsel bağımlılık** olarak belirtilir.

İşlevsel Bağımlılık

- İşlevsel bağımlılık (FD), bir ilişkideki iki öznitelik arasındaki bir dizi kısıtlamadır. İşlevsel bağımlılık, iki demet A_1, A_2, \dots, A_n öznitelikleri için aynı değerlere sahipse, bu iki dizinin B_1, B_2, \dots, B_n öznitelikleri için aynı değerlere sahip olması gerektiğini söyler.
- İşlevsel bağımlılık, X 'in işlevsel olarak Y 'yi belirlediği bir ok işareti (\rightarrow), yani $X \rightarrow Y$ ile temsil edilir. Sol taraftaki öznitelikler, sağ taraftaki özniteliklerin değerlerini belirler.

$$X \rightarrow Y$$

- FD'nin sol tarafı belirleyici, üretimin sağ tarafı bağımlı olarak bilinir.

İşlevsel Bağımlılık

- Bir tablonun özniteliği aynı tablonun başka bir özniteliğini benzersiz şekilde tanımladığında, bir tablonun özniteliklerinin birbirine bağlı olduğu söylenir.
- Örneğin: Stu_Id, Stu_Name, Stu_Age niteliklerine sahip bir öğrenci tablomuz olduğunu varsayalım. Burada Stu_Id özniteliği, öğrenci tablosunun Stu_Name özniteliğini benzersiz bir şekilde tanımlar çünkü öğrenci kimliğini bilirsek, onunla ilişkili öğrenci adını söyleyebiliriz. Bu, işlevsel bağımlılık olarak bilinir ve Stu_Id-> Stu_Name olarak yazılabilir veya kelimelerle Stu_Name'in işlevsel olarak Stu_Id'ye bağlı olduğunu söyleyebiliriz.

Bir ilişki için işlevsel bağımlılıklar nasıl bulunur?

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajasthan	India	18
4	SURESH		Punjab	India	21

Table 1

Bir ilişkideki Fonksiyonel Bağımlılıklar, ilişkinin alanına bağlıdır. Tablo 1'de verilen ÖĞRENCİ ilişkisini düşünün.

- STUD_NO'nun her öğrenci için benzersiz olduğunu biliyoruz. Yani STUD_NO-> STUD_NAME, STUD_NO-> STUD_PHONE, STUD_NO-> STUD_STATE, STUD_NO-> STUD_COUNTRY ve STUD_NO-> STUD_AGE hepsi doğru olacak.
- Benzer şekilde, STUD_STATE-> STUD_COUNTRY, sanki iki kayıt aynı STUD_STATE'e sahipse, bunlar da aynı STUD_COUNTRY'ye sahip olacaklar.
- STUDENT_COURSE ilişkisi için, COURSE_NO-> COURSE_NAME doğru olacaktır çünkü aynı COURSE_NO'ya sahip iki kayıt aynı COURSE_NAME'e sahip olacaktır.



Bir ilişki için işlevsel bağımlılıklar nasıl bulunur?

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajasthan	India	18
4	SURESH		Punjab	India	21

Table 1

İşlevsel Bağımlılık Kümesi: İşlevsel Bağımlılık kümesi veya bir ilişkinin FD kümesi, ilişkide bulunan tüm FD'lerin kümesidir. Örneğin, tablo 1'de gösterilen ÖĞRENCİ ilişkisi için FD kümesi şu şekildedir:

```
{STUD_NO-> STUD_NAME, STUD_NO-> STUD_PHONE, STUD_NO-> STUD_STATE, STUD_NO-> STUD_COUNTRY,  
STUD_NO -> STUD_AGE, STUD_STATE-> STUD_COUNTRY}
```

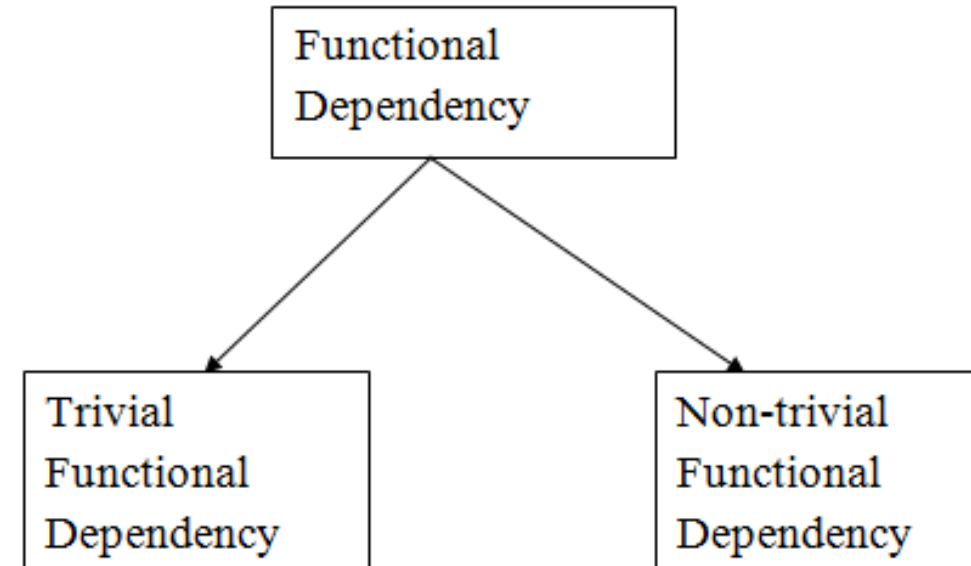
Armstrong Aksiyomları

F, bir dizi işlevsel bağımlılık ise, $F +$ olarak belirtilen kapanış, F'nin mantıksal olarak ima ettiği tüm işlevsel bağımlılıklar kümesidir. .

- **Dönüştürme kuralı** - Alfa bir öznelik kümesi ve beta, alfa'nın alt kümesi ise, alfa beta'yı tutar.
- **Arttırma kuralı** - Eğer $a \rightarrow b$ tutulursa ve y öznelik ayarlanmışsa, o zaman $ay \rightarrow by$ de tutar. Yani bağımlılıklara öznelikler eklemek temel bağımlılıkları değiştirmez.
- **Geçiş kuralı** - Cebirdeki geçiş kuralı ile aynı, eğer $a \rightarrow b$ tutarsa ve $b \rightarrow c$ tutarsa, o zaman $a \rightarrow c$ de geçerlidir. $a \rightarrow b$, b'yi belirleyen işlevsel olarak adlandırılır.

İşlevsel bağımlılık türleri

- Önemsiz işlevsel bağımlılık
- önemsiz olmayan işlevsel bağımlılık
- Birden çok değerli bağımlılık
- Geçişli bağımlılık



DBMS'de önemsiz işlevsel bağımlılık

- Bir özniteliğin bir öznitelik kümesi üzerindeki bağımlılığı, öznitelik kümesi bu özniteliği içeriyorsa önemsiz işlevsel bağımlılık olarak bilinir.
- Sembolik olarak : B, A'nın bir alt kümesiye, $A \rightarrow B$ önemsiz işlevsel bağımlılıktır.
- Aşağıdaki bağımlılıklar da önemsizdir: $A \rightarrow A$ & $B \rightarrow B$

Önemsiz - Y'nin X'in bir alt kümesi olduğu fonksiyonel bir bağımlılık (FD) $X \rightarrow Y$ tutarsa, buna önemsiz bir FD denir. Önemsiz FD'ler her zaman geçerlidir.

Önemsiz - Y'nin X'in bir alt kümesi olmadığı bir FD $X \rightarrow Y$ tutarsa, buna önemsiz olmayan FD denir.

Tamamen önemsiz olmayan - X'in $Y = \Phi$ ile kesiştiği bir FD $X \rightarrow Y$ tutarsa, bunun tamamen önemsiz olmayan bir FD olduğu söylenir.



DBMS'de önemsiz işlevsel bağımlılık

Örneğin : Student_id ve Student_Name olmak üzere iki sütun içeren bir tablo düşünün.

- $\{ \text{Student_Id}, \text{Student_Name} \} \rightarrow \text{Student_Id}$, $\{ \text{Student_Id}, \text{Student_Name} \}$ 'nin bir alt kümesi olduğu için Student_Id önemsiz bir işlevsel bağımlılıktır. Bu mantıklıdır çünkü Öğrenci_Kimliği ve Öğrenci_Adı değerlerini bilirsek Öğrenci_Kimliği'nin değeri benzersiz bir şekilde belirlenebilir.
- Ayrıca, $\text{Student_Id} \rightarrow \text{Student_Id} \ \& \ \text{Student_Name} \rightarrow \text{Student_Name}$, önemsiz bağımlılıklardır.

DBMS'de Önemsiz olmayan işlevsel bağımlılık

B, A'nın bir alt kümesi değilse, $A \rightarrow B$ 'nin önemsiz olmayan bir işlevsel bağımlılığı vardır.

A kesişimi B NULL olduğunda, $A \rightarrow B$ tam önemsiz değil olarak adlandırılır.

Örneğin :

Üç özniteliğe sahip bir çalışan tablosu: emp_id, emp_name, emp_address.

Aşağıdaki işlevsel bağımlılıklar önemsiz değildir:

emp_id \rightarrow emp_name (emp_name, emp_id'nin bir alt kümesi değildir)

emp_id \rightarrow emp_address (emp_address, emp_id'nin bir alt kümesi değildir)

Öte yandan, aşağıdaki bağımlılıklar önemsizdir:

{emp_id, emp_name} \rightarrow emp_name [emp_name, {emp_id, emp_name} 'nin bir alt kümesidir]

DBMS'de birden çok değerli bağımlılık

Bir tabloda birden fazla bağımsız birden çok değerli öznitelik olduğunda birden çok değerli bağımlılık oluşur .

Burada, üretim yılı ve renk sütunları birbirinden bağımsızdır ve bisiklet modeline bağlıdır. Bu durumda, bu iki sütunun bike_model'e bağlı olarak birden çok değerli olduğu söylenir. Bu bağımlılıklar şu şekilde temsil edilebilir:

```
bike_model - >> manuf_year  
bike_model - >> renk
```

bisiklet_modeli	manuf_year	renk
M1001	2007	Siyah
M1001	2007	Kırmızı
M2012	2008	Siyah
M2012	2008	Kırmızı
M2222	2009	Siyah
M2222	2009	Kırmızı

DBMS'de geçişli bağımlılık

Dolaylı olarak iki işlevsel bağımlılık tarafından oluşturulmuşsa, işlevsel bir bağımlılığın geçişli olduğu söylenir. Örneğin

$X \rightarrow Z$, aşağıdaki üç işlevsel bağımlılık doğruysa, geçişli bir bağımlılıktır:

$X \rightarrow Y$

$Y \text{ değil } \rightarrow X$

$Y \rightarrow Z$

Not: Geçişli bir bağımlılık yalnızca üç veya daha fazla öznitelik ilişkisinde ortaya çıkabilir.

Bu bağımlılık, veritabanını 3NF'de (3. Normal Form) normalleştirmemize yardımcı olur .

DBMS'de geçişli bağımlılık

{Kitap} \rightarrow {Yazar} (kitabı biliyorsak, yazar adını da biliyoruz)

{Author} bunu yapmaz \rightarrow {Kitap}

{Yazar} \rightarrow {Yazar_age}

Bu nedenle, geçişli bağımlılık kuralına göre : {Kitap} \rightarrow {Yazar_age} geçerli olmalıdır, bu mantıklıdır, çünkü kitabın adını bilirsek yazarın yaşını da bilebiliriz.

Kitap	Yazar	Author_age
Game of Thrones	George RR Martin	66
Harry Potter	JK Rowling	49
Işığın Ölmesi	George RR Martin	66

DBMS'de Normalleştirme: 1NF, 2NF, 3NF ve BCNF

Normalleştirme , veri fazlalığını, ekleme anormalliğini, güncelleme anormalliğini ve silme anormalliğini önlemek için verileri veritabanında düzenleme işlemidir. Önce anormallikleri tartışalım, sonra normal formları örneklerle tartışalım.

DBMS'deki anormallikler

Veritabanı normalleştirilmediğinde ortaya çıkan üç tür anormallik vardır. Bunlar - Ekleme, güncelleme ve silme anormalliği. Bunu anlamak için bir örnek alalım.

Örnek : Bir üretim şirketinin, çalışan ayrıntılarını dört öz niteliğe sahip çalışan adlı bir tabloda depoladığını varsayalım: çalışanın kimliğini saklamak için emp_id, çalışanın adını saklamak için emp_name, çalışanın adresini saklamak için emp_address ve çalışanın çalıştığı departman ayrıntılarını depolamak için emp_dept. Bir noktada tablo şöyle görünür:

DBMS'deki anormallikler

emp_id	emp_name	emp_addresses	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

DBMS'deki anormallikler

Güncelleme Anormalliği : Yukarıdaki tabloda, Rick çalışanı şirketin iki departmanına ait olduğu için iki satırımız var. Rick'in adresini güncellemek istiyorsak, aynı şeyi iki satırda güncellememiz gerekir, aksi takdirde veriler tutarsız hale gelir. Eğer bir şekilde doğru adres bir departmanda güncellenir ancak diğerinde güncellenmezse, o zaman veritabanına göre Rick iki farklı adrese sahip olur ki bu doğru değildir ve tutarsız verilere yol açar.

DBMS'deki anormallikler

Ekleme Anormalliği: Eğitim altında olan ve şu anda herhangi bir departmana atanmamış yeni bir çalışanın şirkete katıldığını varsayalım, emp_dept alanı boş değerlere izin vermiyorsa verileri tabloya ekleyemeyiz.

Silme Anormalliği : Farz edin ki, eğer şirket bir noktada D890 departmanını kapatırsa, emp_dept olan satırların D890 olarak silinmesi, Maggie çalışanın sadece bu departmana atandığı için bilgilerini de silecektir.

Bu anormalliklerin üstesinden gelmek için verileri normalleştirmemiz gerekiyor.



DBMS Normalleştirme

Normalleştirme , bir ilişkiden veya bir dizi ilişkiden fazlalığı en aza indirme sürecidir . İlişkili fazlalık, ekleme, silme ve güncelleme anormalliklerine neden olabilir. Böylece ilişkilerdeki fazlalığı en aza indirmeye yardımcı olur. Normal formlar , veritabanı tablolarındaki fazlalığı ortadan kaldırmak veya azaltmak için kullanılır.

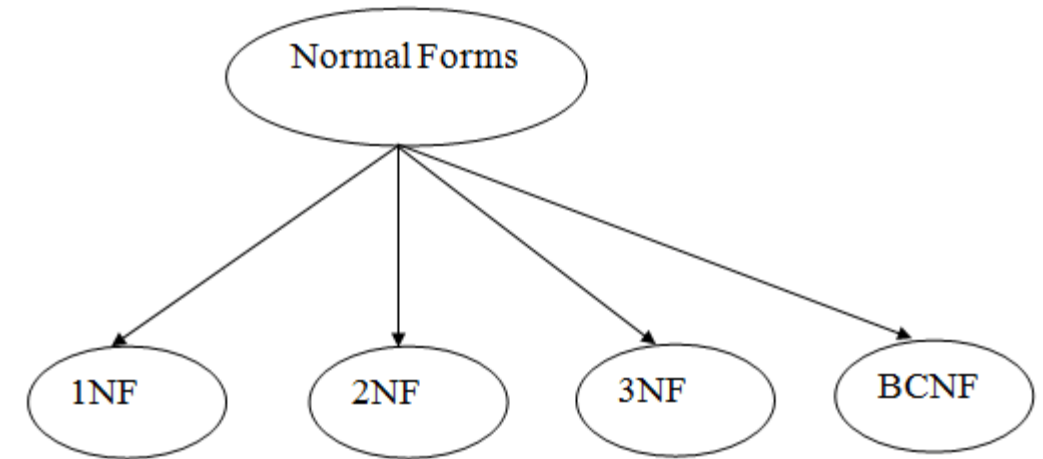
- Normalleştirme, veri tabanındaki verileri düzenleme işlemidir.
- Normalleştirme, bir ilişkiden veya bir dizi ilişkiden fazlalığı en aza indirmek için kullanılır. Ayrıca Ekleme, Güncelleme ve Silme Anomalileri gibi istenmeyen özellikleri ortadan kaldırmak için kullanılır.
- Normalleştirme, büyük tabloyu daha küçük tabloya böler ve ilişkiyi kullanarak bunları birbirine bağlar.
- Normal form, veritabanı tablosundaki fazlalığı azaltmak için kullanılır.



Normal Formlar

Basitçe tanımlamak gerekirse, normal formlar normalizasyon seviyeleridir. Bu seviyeler gereksiz veri tekrarlarını ne derecede engellediği ve tutarlılığı ne kadar sağladığına bağlı olarak derecelendirilir. Seviye yükseldikçe veri tutarlılığı artar, veri tekrarı düşer.

Normalizasyon seviyeleri 1NF (Birinci Normal Form), 2NF, 3NF, BCNF (Boyce-Codd Normal Form, 3.5NF'de denir), 4NF şeklinde adlandırılır ve yukarı doğru devam eder. Ancak daha yukarı normalizasyon seviyeleri çok nadiren kullanılır çünkü çoğu zaman uygulanması mümkün olmayabilir.



Normal Formlar

Normal Form	Açıklama
<u>1NF</u>	Bir atomik değer içeriyorsa, bir ilişki 1NF'dedir.
<u>2NF</u>	Bir ilişki 1NF içindeyse ve anahtar olmayan tüm öznitelikler birincil anahtara bağlı olarak tamamen işlevselse 2NF'de olacaktır.
<u>3NF</u>	Bir ilişki 2NF'de ise ve geçiş bağımlılığı yoksa 3NF'de olacaktır.
<u>4NF</u>	Boyce Codd normal formunda ve çok değerli bağımlılığı yoksa ilişki 4NF'de olacaktır.
<u>5NF</u>	Bir ilişki 4NF içindeyse ve herhangi bir birleştirme bağımlılığı içermiyorsa 5NF içindedir ve birleştirme kayıpsız olmalıdır.

Normal Formlar

Konuyu detaylandırabilmek için bir veri tabanı oluşturalım ve normalizasyonunu yapalım. Tabloda bir teknik destek firmasının çalışanları, servis araçları, servis şöförleri ve servis verilen semtler bulunsun. Her bir şöför tek araç ile semt bazında servis yapmaktadır.

Calisan	Soyad	Sofor	Arac	Semt
Orçun	Yılmaz	Ahmet	Toyota	Levent, Etiler, Ulus
Metin	Seyyar	Mehmet	Honda	Bakırköy, Ataköy, Yeşilköy
Metin	Seyyar	Tolga	Ford	Kandilli, Beylerbeyi, Kuzguncuk

1NF (1. Normal Form)

Bir veri tabanının 1NF olabilmesi için aşağıdaki özellikleri karşılayabilmesi gerekir:

- Aynı tablo içinde tekrarlayan kolonlar bulunamaz,
- Her kolonda yalnızca bir değer bulunabilir (bkz. "Semt" kolonu)
- Her satır bir eşsiz anahtarla tanımlanmalıdır (Unique Key - Primary Key)
- Atomik bir değer içeriyorsa ilişki 1NF olacaktır.
- Bir tablonun bir özniteliğinin birden fazla değeri tutamayacağını belirtir. Yalnızca tek değerli özniteliği taşımalıdır.
- İlk normal biçim, çok değerli özniteliğe, bileşik özniteliğe ve bunların kombinasyonlarına izin vermez.

1NF (1. Normal Form)

Calisan	Soyad	Sofor	Arac	Semt 1	Semt 2	Semt 3
Orçun	Yılmaz	Ahmet	Toyota	Levent	Etiler	Ulus
Metin	Seyyar	Mehmet	Honda	Bakırköy	Ataköy	Yeşilköy
Metin	Seyyar	Tolga	Ford	Kandilli	Beylerbeyi	Kuzguncuk

Calisan	Soyad	Sofor	Arac	Semt
Orçun	Yılmaz	Ahmet	Toyota	Levent
Orçun	Yılmaz	Ahmet	Toyota	Etiler
Orçun	Yılmaz	Ahmet	Toyota	Ulus
Metin	Seyyar	Mehmet	Honda	Bakırköy
Metin	Seyyar	Mehmet	Honda	Ataköy
Metin	Seyyar	Mehmet	Honda	Yeşilköy
Metin	Seyyar	Tolga	Ford	Kandilli
Metin	Seyyar	Tolga	Ford	Beylerbeyi
Metin	Seyyar	Tolga	Ford	Kuzguncuk

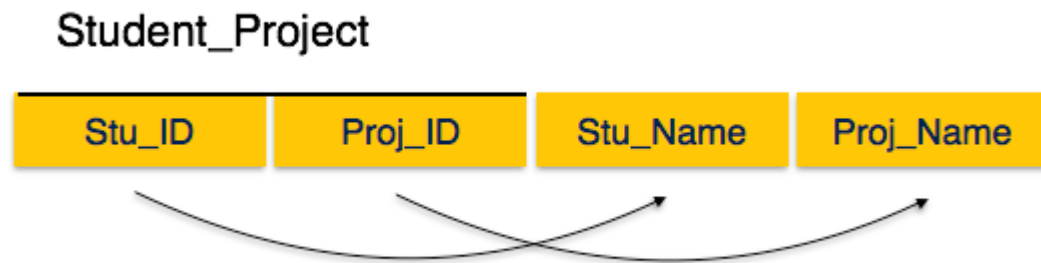
2NF (2. Normal Form)

İkinci normal formu öğrenmeden önce, aşağıdakileri anlamamız gerekir -

Asal öznitelik - Aday anahtarın bir parçası olan öznitelik, ana öznitelik olarak bilinir.

Asal olmayan öznitelik - Ana anahtarın bir parçası olmayan bir özneliğin asal olmayan bir öznitelik olduğu söylenir.

İkinci normal biçimi izlersek, o zaman asal olmayan her öznitelik tamamen işlevsel olarak birincil anahtar özneliğine bağlı olmalıdır. Yani, eğer $X \rightarrow A$ tutarsa, o zaman X 'in $Y \rightarrow A$ 'nın da doğru olduğu herhangi bir uygun Y alt kümesi olmamalıdır.



Student

Stu_ID	Stu_Name	Proj_ID
--------	----------	---------

Project

Proj_ID	Proj_Name
---------	-----------

2NF (2. Normal Form)

Bir veri tabanının 2NF olabilmesi için aşağıdaki özellikleri karşılayabilmesi gerekir:

1. Tablo 1NF olmalıdır,
2. Anahtar olmayan değerler ile kompozit (bileşik) anahtarlar arasında kısmi (partial) bağımlılık durumu oluşmamalıdır. Kısmi bağımlılık durumu, anahtar olmayan herhangi bir değer kompozit bir anahtarın yalnızca bir kısmına bağlı ise oluşur. (Evet farkındayım çok karmaşık görünüyor, örnekte net bir şekilde anlayacaksınız. Söz...)
3. Herhangi bir veri alt kümesi birden çok satırda tekrarlanmamalıdır. Bu tür veri alt kümeleri için yeni tablolar oluşturulmalıdır.
4. Ana tablo ile yeni tablolar arasında, dış anahtarlar (foreign key) kullanılarak ilişkiler tanımlanmalıdır.

2NF (2. Normal Form)

Aslında üç ve dördüncü maddeler ikinci maddenin sonuçlarıdır. Eğer anahtar olmayan bir kolonla herhangi bir komposit anahtar arasında kısmi bağımlılık varsa her zaman tekrarlayan veri alt kümeleri oluşur. Bu durumu düzeltmek için bahis konusu alt kümeleri farklı bir tablo haline getirmeli ve elde ettiğimiz tablolar ile ana tablomuz arasındaki ilişkiyi tanımlamalıyız.

Calisan	Soyad	Sofor	Arac	Semt
Orçun	Yılmaz	Ahmet	Toyota	Levent
Orçun	Yılmaz	Ahmet	Toyota	Etiler
Orçun	Yılmaz	Ahmet	Toyota	Ulus
Metin	Seyyar	Mehmet	Honda	Bakırköy
Metin	Seyyar	Mehmet	Honda	Ataköy
Metin	Seyyar	Mehmet	Honda	Yeşilköy
Metin	Seyyar	Tolga	Ford	Kandilli
Metin	Seyyar	Tolga	Ford	Beylerbeyi
Metin	Seyyar	Tolga	Ford	Kuzguncuk

2NF (2. Normal Form)

Ana Tablo

Id	Calisan	Soyad
1	Orçun	Yılmaz
2	Metin	Seyyar

Servis Tablosu

Cid	Sofor	Arac	Semt
1	Ahmet	Toyota	Levent
1	Ahmet	Toyota	Etiler
1	Ahmet	Toyota	Ulus
2	Mehmet	Honda	Bakırköy
2	Mehmet	Honda	Ataköy
2	Mehmet	Honda	Yeşilköy
2	Tolga	Ford	Kandilli
2	Tolga	Ford	Beylerbeyi
2	Tolga	Ford	Kuzguncuk

Yeni tablomuz ile ana tablomuzu ilişkilendirmek için "Cid" (Çalışan ID) isimli bir kolon yarattık. Dikkat ederseniz bu kolonun aldığı değer ana tablomuzdaki eşsiz anahtarı işaret ediyor.

Bu ilişkilendirmeye ***Foreign Key*** diyoruz.

3NF (3. Normal Form)

- Bir ilişki, 2NF'de ise ve herhangi bir geçişli kısmi bağımlılık içermiyorsa 3NF'de olacaktır.
- Veri çoğaltmayı azaltmak için 3NF kullanılır. Veri bütünlüğünü sağlamak için de kullanılır.
- Asal olmayan nitelikler için geçişli bağımlılık yoksa, bu durumda ilişki üçüncü normal formda olmalıdır.

Her önemsiz olmayan fonksiyon bağımlılığı $X \rightarrow Y$ için aşağıdaki koşullardan en az birini barındıran bir ilişki üçüncü normal formdadır.

- X bir süper anahtardır.
- Y asal bir nitelik, yani Y'nin her bir ögesi bazı aday anahtarların bir parçasıdır.

3NF (3. Normal Form)

Bir veri tabanının 3NF olabilmesi için aşağıdaki özellikleri karşılayabilmesi gerekir:

- Veri tabanı 2NF olmalıdır,
- Anahtar olmayan hiç bir kolon bir diğerine (anahtar olmayan başka bir kolona) bağlı olmamalı ya da geçişken fonksiyonel bir bağımlılığı (transitional functional dependency) olmamalıdır. Başka bir deyişle her kolon eşsiz anahtara tam bağımlı olmak zorundadır.

Veri tabanımızı 3NF şartlarına uydurabilmek için anahtar olmayan ve eşsiz anahtara tam bağımlı olmayan tüm kolonları kaldırmalıyız. Dikkat ederseniz bizim tablomuzda "Araç" kolonu eşsiz anahtarımıza değil "Şoför" kolonuna bağımlı. Birbirine bağlı olan bu iki kolonu (Şoför - Araç) ayrı bir tabloya ayırmamız ve tablomuzla aralarında bir ilişki yaratmamız gerekiyor.

3NF (3. Normal Form)

Ana Tablo

Id	Calisan	Soyad
1	Orçun	Yılmaz
2	Metin	Seyyar

Servis Tablosu

Cid	Sid	Semt
1	1	Levent
1	1	Etiler
1	1	Ulus
2	2	Bakırköy
2	2	Ataköy
2	2	Yeşilköy
2	3	Kandilli
2	3	Beylerbeyi
2	3	Kuzguncuk

3NF (3. Normal Form)

Şoför Tablosu

Sid	Sofor	Arac
1	Ahmet	Toyota
2	Mehmet	Honda
3	Tolga	Ford

Öncelikle şoför tablosu adında yeni bir tablo oluşturduk. Bu tabloda Sid (Şoför ID) adıyla bir eşsiz anahtar yarattık ve Servis tablomuzdaki Sid kolonundan bu eşsiz anahtara referans vererek foreign key oluşturduk.

3NF (3. Normal Form)

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	YUKARI	Noida
333	Stephan	02228	BİZE	Boston
444	Lan	60007	BİZE	Chicago
555	Katharine	06389	İngiltere	Norwich
666	John	462007	MP	Bhopal

Yukarıdaki tablodaki süper anahtar:

{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}

Aday anahtar: {EMP_ID}

Asal olmayan öznitelikler: Verilen tabloda, EMP_ID dışındaki tüm öznitelikler asal değildir.

3NF (3. Normal Form)

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	YUKARI	Noida
333	Stephan	02228	BİZE	Boston
444	Lan	60007	BİZE	Chicago
555	Katharine	06389	İngiltere	Norwich
666	John	462007	MP	Bhopal

Burada, EMP_STATE & EMP_CITY, EMP_ZIP'ye bağlıdır ve EMP_ZIP, EMP_ID'ye bağlıdır. Asal olmayan özellikler (EMP_STATE, EMP_CITY) geçişli olarak süper anahtara (EMP_ID) bağlıdır. Üçüncü normal biçim kuralını ihlal ediyor.

Bu nedenle, EMP_CITY ve EMP_STATE'i, Birincil anahtar olarak EMP_ZIP ile yeni <EMPLOYEE_ZIP> tablosuna taşımamız gerekiyor.



3NF (3. Normal Form)

ÇALIŞAN tablosu:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP tablosu:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	YUKARI	Noida
02228	BİZE	Boston
60007	BİZE	Chicago
06389	İngiltere	Norwich
462007	MP	Bhopal

BCNF / 3.5NF (Boyce-Codd Normal Form)

Bir veri tabanının 3.5NF olabilmesi için aşağıdaki özellikleri karşılayabilmesi gerekir:

- Veri Tabanı 3NF olmalıdır,
- Her determinant (belirleyici kolon) aynı zamanda bir aday anahtar olmalıdır.

Determinant: Aynı satırdaki diğer kolon değerlerini belirlemek için kullanılan kolon kümesi determinant olarak adlandırılır.

Servis tablomuza dikkatle baktığımızda iki tane determinant olduğunu görebiliriz. Semt kolonu, Cid - Sid kombinasyonunun; Sid ise Cid kolonunun determinantıdır.

Bu noktada Semt kolonunun hali hazırda bir aday anahtar olduğunu görebiliyoruz çünkü her bir değer tekrar oluşturmaksızın tüm kayıt satırını tanımlayabilmekte. Öte yandan Sid için aynı şeyi söylemek mümkün değil çünkü tekrarlanıyor.

BCNF / 3.5NF (Boyce-Codd Normal Form)

- BCNF, 3NF'nin gelişmiş sürümüdür. 3NF'den daha katıdır.
- Her fonksiyonel bağımlılık $X \rightarrow Y$, X , tablonun süper anahtarı ise bir tablo BCNF'dedir.
- BCNF için tablo 3NF'de olmalıdır ve her FD için LHS süper anahtardır.

BCNF / 3.5NF (Boyce-Codd Normal Form)

Ana Tablo

Id	Calisan	Soyad
1	Orçun	Yılmaz
2	Metin	Seyyar

Servis Tablosu

Cid	Semt
1	Levent
1	Etiler
1	Ulus
2	Bakırköy
2	Ataköy
2	Yeşilköy
2	Kandilli
2	Beylerbeyi
2	Kuzguncuk

BCNF / 3.5NF (Boyce-Codd Normal Form)

Semt Tablosu

Semt	Sid
Levent	1
Etiler	1
Ulus	1
Bakırköy	2
Ataköy	2
Yeşilköy	2
Kandilli	3
Beylerbeyi	3
Kuzguncuk	3

BCNF / 3.5NF (Boyce-Codd Normal Form)

Şoför Tablosu

Sid	Sofor	Arac
1	Ahmet	Toyota
2	Mehmet	Honda
3	Tolga	Ford

Görülebileceği gibi artık tablolarımızın hiçbirinde aday anahtar olmayan determinant yok. Bu nedenle veri tabanımız BCNF'tir diyebiliriz.

4NF (4. Normal Form)

Bir veri tabanının 4NF olabilmesi için aşağıdaki özellikleri karşılayabilmesi gerekir:

- Veri Tabanı 3NF olmalıdır,
- Çok-değerli bağımlılıkları (Multli-Valued dependency) olmamalıdır.

Multi-Valued Dependency: Bu durum bir ya da daha çok veri satırının var olması, aynı tabloda başka bir (ya da daha çok) veri satırının bulunmasını gerektirdiğinde ortaya çıkar. Örneğin, bir yazılım firması düşünelim. Geliştirdikleri yazılımların masaüstü bilgisayarlar için olanlarını tek-kullanıcılı ve çok-kullanıcılı olarak iki versiyonla piyasaya sunuyor olsunlar. Diyelim ki bu firmanın geliştirdiği tüm yazılımları barındıran bir veri tabanı oluşturuyoruz. Bu veri tabanında bir masaüstü yazılımın tek-kullanıcılı versiyonunu eklediysek mutlaka bir başka satırda aynı yazılımın çok-kullanıcılı versiyonu için de bir kayıt açılmış olmak durumundadır...

