DATABASE DESIGN THEORY

THE COMPLETE BOOK by Jeff Ullman

Chapter 3: Design Theory for Relational Databases

Fundamental of Database Systems by Elmasri

Chapter 15-16

Relational Database Designs

Can we have **many different** database **"designs"** (schemas) to store the mini-world information?

Can one schema be much better than another?

Data Anomalies & Constraints

Data Anomalies & Constraints

Student	Course	Room
111	CS145	B01
123	CS145	B01
145	CS145	B01
..	..	..

All classes of a course are held in the same room

Contains ***redundant*** information!

Data Anomalies & Constraints

A poorly designed database causes ***anomalies***

Student	Course	Room
111	CS145	B01
123	CS145	C12
145	CS145	B01
..	..	..

If we **update** the room number for one tuple,

we get inconsistent data => an ***update* anomaly**

Data Anomalies & Constraints

A poorly designed database causes ***anomalies***:

Student	Course	Room
..	..	..

If everyone drops the class, we lose what room the class is in! => a <u>***delete*** **anomaly**</u>

Data Anomalies & Constraints

A poorly designed database causes ***anomalies***:

…	CS229	C12

Student	Course	Room
111	CS145	B01
123	CS145	B01
145	CS145	B01
..	..	..

Similarly, we can't reserve a room without students => an ***insert anomaly***

Data Anomalies & Constraints

<u>Student</u>	<u>Course</u>
111	CS145
123	CS145
145	CS145
..	..

<u>Course</u>	Room
CS145	B01
CS229	C12

Is this form better?

- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

Today: develop theory to understand why this design may be better **and** how to find this *decomposition*…

Informal Design Guidelines

Each tuple in a relation should represent one entity or relationship instance

Student	**Course**
111	CS145
123	CS145
145	CS145
..	..

Course	**Room**
CS145	B01
CS229	C12

- Attributes of different entities should not be mixed in the same relation
- Only foreign keys should be used to refer to other entities

Null Values in Tuples

- Relations should have few NULL values
- Attributes that are **NULL frequently** could be placed in separate relations (with the primary key)

UID	Name	..	**Job**	**OfficeNo**
111	A		NULL	NULL
123	B		NULL	NULL
145	C		intern	345628
..		..	..	

- Reasons for nulls:
  - *Attribute not applicable or invalid*
  - *Attribute value unknown (may exist)*
  - *Value known to exist, but unavailable*

Design Theory

Design theory is about how to represent your data to avoid ***anomalies***.

It is a mostly a mechanical process

Tools can carry out routine portions

EXAMPLE

Shipment of Parts

Key ?
S#, P#, date

S#	sname	city	P#	pname	colour	weight	qty	date
S1	Smith	London	P1	Nut	Red	12	200	980620
S1	Smith	London	P1	Nut	Red	12	700	980625
S2	Jones	Paris	P3	Screw	Blue	17	400	980620
S2	Jones	Paris	P5	Bolt	Green	17	300	980620
S2	Jones	Paris	P2	Screw	Red	14	200	980621
S3	Clark	Rome	P1	Nut	Red	12	300	980612
S3	Clark	Rome	P6	Cog	Red	19	600	980612
S4	Blake	Athens	P4	Cam	Blue	12	200	980619
S4	Blake	Athens	P1	Nut	Red	12	300	980619
S4	Blake	Athens	P3	Screw	Blue	17	100	980620
S5	Alex	Paris	P1	Nut	Red	12	250	980626

https://meet.google.com/jam-kyxi-qsg

Above table contains information regarding shipments of parts from suppliers.

- A supplier belongs to a particular city.
- A part has a name, color, quantity and weight and is shipped on particular date

Problems ??

Design Anomalies

EXAMPLE

Shipment of Parts

S#	sname	city	P#	pname	colour	weight	qty	date
S1	Smith	London	P1	Nut	Red	12	200	980620
S1	Smith	London	P1	Nut	Red	12	700	980625
S2	Jones	Paris	P3	Screw	Blue	17	400	980620
S2	Jones	Paris	P5	Bolt	Green	17	300	980620
S2	Jones	Paris	P2	Screw	Red	14	200	980621
S3	Clark	Rome	P1	Nut	Red	12	300	980612
S3	Clark	Rome	P6	Cog	Red	19	600	980612
S4	Blake	Athens	P4	Cam	Blue	12	200	980619
S4	Blake	Athens	P1	Nut	Red	12	300	980619
S4	Blake	Athens	P3	Screw	Blue	17	100	980620
S5	Alex	Paris	P1	Nut	Red	12	250	980626

Key ?

S#, P#, date

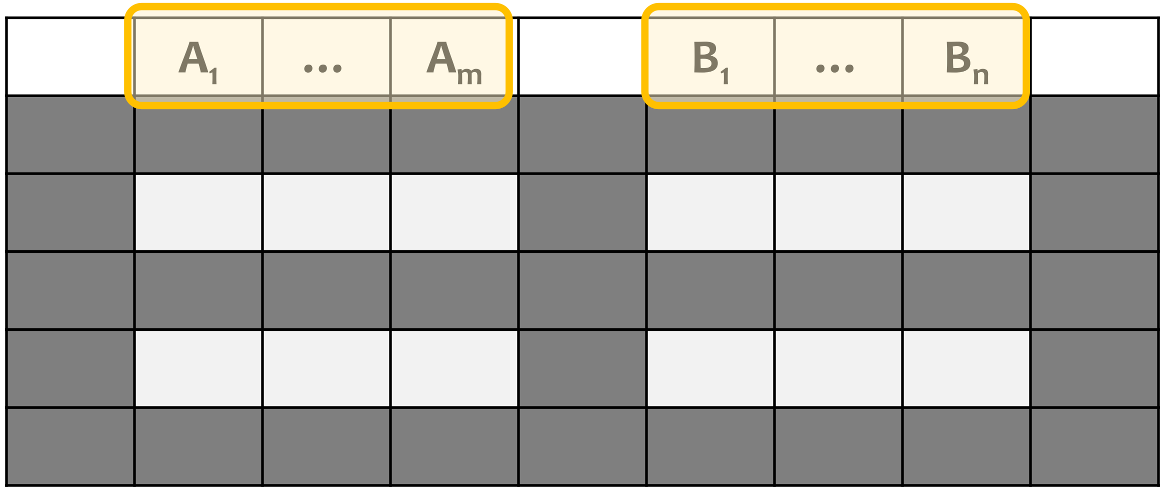
S#	P#	qty	date
S1	P1	200	980620
S1	P1	700	980625
S2	P3	400	980620
S2	P5	300	980620
S2	P2	200	980621
S3	P1	300	980612
S3	P6	600	980612
S4	P4	200	980619
S4	P1	300	980619
S4	P3	100	980620
S5	P1	250	980626

S#	sname	city
S1	Smith	London
S2	Jones	Paris
S3	Clark	Rome
S4	Blake	Athens
S5	Alex	Paris

P#	pname	colour	weight
P1	Nut	Red	12
P2	Screw	Red	14
P3	Screw	Blue	17
P4	Cam	Blue	12
P5	Bolt	Green	17
P6	Cog	Red	19

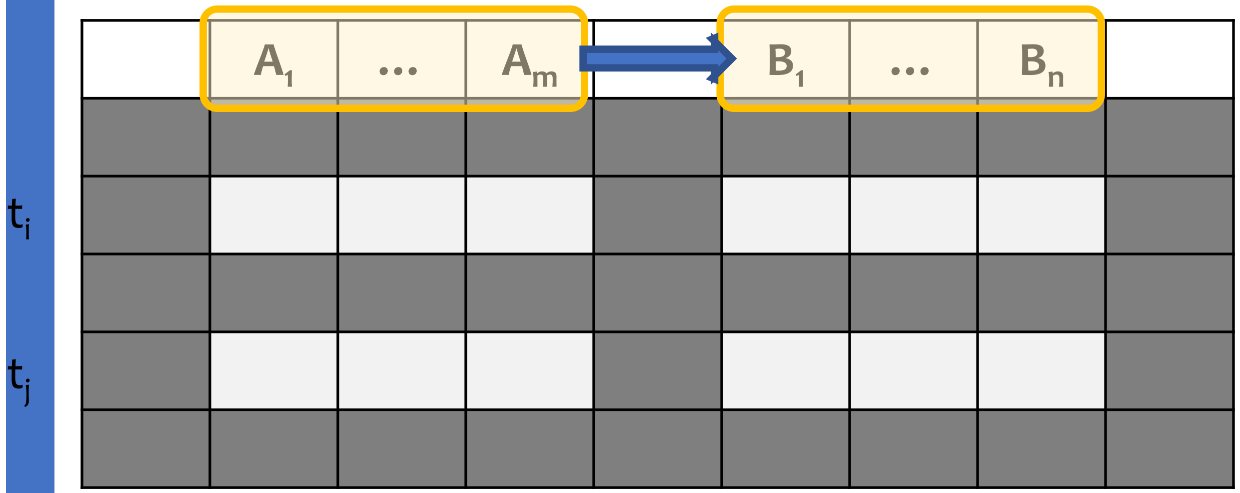
Functional Dependencies

A Picture Of FDs



Def (again):
Given attribute sets $\mathbf{A}=\{\mathbf{A}_1,\dots,\mathbf{A}_m\}$ and $\mathbf{B} = \{\mathbf{B}_1,\dots \mathbf{B}_n\}$ in **R,**

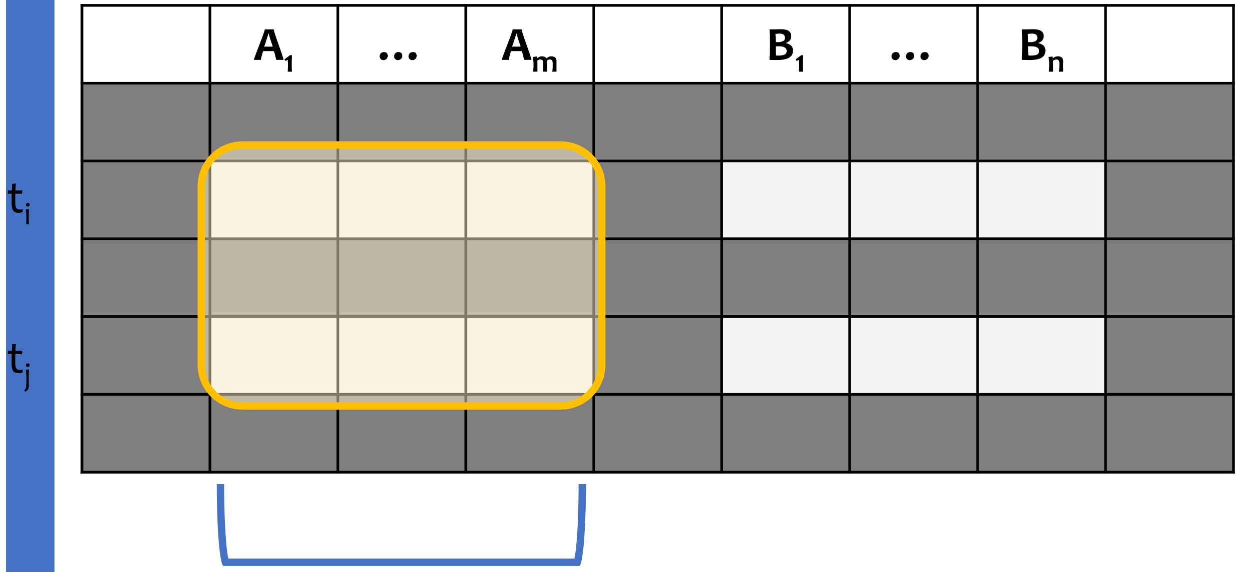
A Picture Of FDs



<u>Def (again):</u>
Given attribute sets $\mathbf{A=\{A_1,\ldots,A_m\}}$ and $\mathbf{B = \{B_1,\ldots B_n\}}$ in **R,**

The ***functional dependency*** $\mathbf{A \rightarrow B}$ **on R** holds if for ***any*** t_i, t_j in R:

A Picture Of FDs



If t1,t2 agree here..

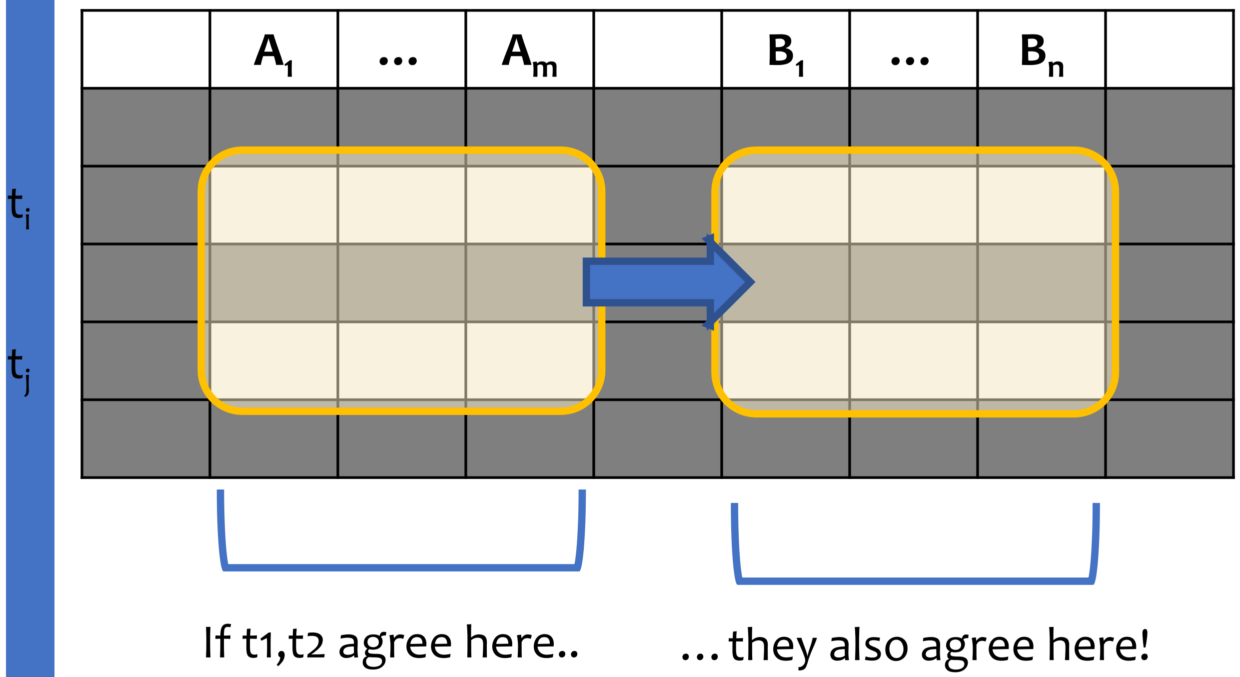
<u>Def (again):</u>

Given attribute sets $\mathbf{A=\{A_1,\ldots,A_m\}}$ and $\mathbf{B = \{B_1,\ldots B_n\}}$ in **R,**

The ***functional dependency*** **A→ B on R** holds if for ***any*** t_i,t_j in R:

$t_i[A_1] = t_j[A_1]$ AND $t_i[A_2]=t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

A Picture Of FDs



<u>Def (again):</u>
Given attribute sets $\mathbf{A=\{A_1,\ldots,A_m\}}$ and $\mathbf{B = \{B_1,\ldots B_n\}}$ in **R,**

The ***functional dependency*** **A→ B on R** holds if for ***any*** t_i,t_j in R:

<u>**if**</u> $t_i[A_1] = t_j[A_1]$ AND $t_i[A_2]=t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

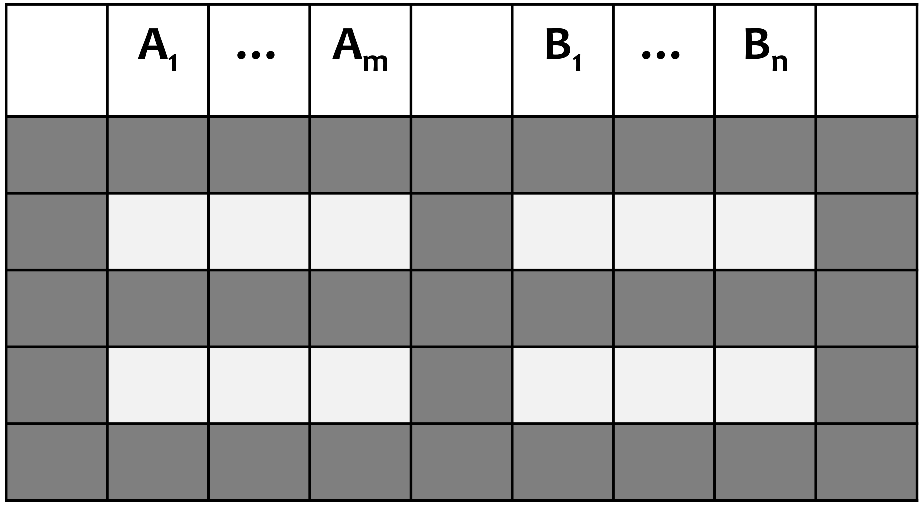
<u>**then**</u> $t_i[B_1] = t_j[B_1]$ AND $t_i[B_2]=t_j[B_2]$ AND … AND $t_i[B_n] = t_j[B_n]$

Functional Dependency

Def: Let A,B be *sets* of attributes
We write A → B or say A ***functionally determines*** B if, for any tuples t_1 and t_2:

$$t_1[A] = t_2[A] \text{ implies } t_1[B] = t_2[B]$$

and we call A → B a **functional dependency**



A->B means that
"whenever two tuples agree on A then they agree on B."

FDs describe the relationships between attributes
FDs are derived from the real-world constraints on the attributes

EXAMPLE

FDs

Shipment of Parts

Key: <u>S#</u>, <u>P#</u>, <u>date</u>

S#	sname	city	P#	pname	colour	weight	qty
S1	Smith	London	P1	Nut	Red	12	200
S1	Smith	London	P1	Nut	Red	12	700
S2	Jones	Paris	P3	Screw	Blue	17	400
S2	Jones	Paris	P5	Bolt	Green	17	300
S2	Jones	Paris	P2	Screw	Red	14	200
S3	Clark	Rome	P1	Nut	Red	12	300
S3	Clark	Rome	P6	Cog	Red	19	600

FDs

S# -> sname, city

P# -> pname, colour, weight

FDs helps to determine Anomalies

FDs for Relational Schema Design

- High-level idea: **why do we care about FDs?**

1. *Start with some relational schema*

2. *Find out its functional dependencies (FDs)*

3. *Use these to design a better schema*
    - One which minimizes the possibility of anomalies

Functional Dependencies as Constraints

A **functional dependency** is a form of **constraint**

- Part of the schema, helps define a *valid* instance
- *We* can check whether there are violations of FDs

*Recall: an **instance** of a schema is a multiset of tuples conforming to that schema, **i.e. a table***

Student	Course	Room
111	CS145	B01
123	CS145	B01
145	CS145	B01
..	..	..

The FD {Course} -> {Room} ***holds on this instance***

Functional Dependencies as Constraints

Note that:

- You can check if an FD **does not hold** by examining a single instance
- However, you **cannot prove** that an FD is part of the schema by examining a single instance.
  - *This would require checking every valid instance*

Student	Course	Room
111	CS145	B01
123	CS145	B01
145	CS145	B01
..	..	..

You cannot *prove* that the FD {Course} -> {Room} is ***part of the schema***

More Examples

An FD is a constraint which holds, or does not hold on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

More Examples

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876 ←	Salesrep
E1111	Smith	9876 ←	Salesrep
E9999	Mary	1234	Lawyer

{Position} → {Phone}

More Examples

EmpID	Name	Phone	Position
E0045	Smith	1234 →	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234 →	Lawyer

but *not* {Phone} → {Position}

ACTIVITY 1: Which FDs hold ?

A	B	C	D	E
1	2	4	3	6
3	2	5	1	8
1	4	4	5	7
1	2	4	3	6
3	2	5	1	8

Find at least *three* FDs which ***does not hold*** on this instance

{ } → { }
{ } → { }
{ } → { }

What about these FDs?
{ D } → { C }
{ A B } → { C }
{ A B } → { D }

ACTIVITY 2: Which FDs hold ?

Certain FD's can be ruled out based on a given state of the database

TEACH

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

- Text -> course
  - possible FD
- Teacher -> course
  - ruled out

Activity 2: Finding FDs

FDs are derived from the real-world constraints on the attributes.

- Must be define by someone who knows the semantic of attributes.
- FD is a property of relational schema, in real world if we define an FD's, ***we wants it to hold at all times***

***StudentGrade* (*rollNo*, *name*, *email*, *courseID*, *grade*)**

- Each student has a unique rollNo and email assigned by the university
  - *rollNo → name, email*
  - *email → rollNo*
  - *rollNo, courseID → grade*
- Is it a good design
  - *Not a good design. WHY ??*

Activity 3: Finding FDs

***Address* (*street_address, city, state, zip*)**

Use your knowledge and intuition to determine FDs

- *street_address, city, state* $\longrightarrow$ *zip*
- *zip* $\longrightarrow$ *city*
- zip $\longrightarrow$ *state*
- *zip, state* $\longrightarrow$ *zip*?
  - *This is a trivial FD*
  - *Trivial FD: LHS $\supseteq$ RHS*

Activity 4: Convert Business statements into FDs

DISK_DRIVE (Serial_number, Manufacturer, Model, Batch, Capacity, Retailer)

- **Example:**
  - ***Disk_drive ('1978619', 'WesternDigital', 'A2235X', '765234', 500, 'CompUSA')***

- Write each of the following dependencies as an FD:

  a) *The manufacturer and serial number uniquely identifies the drive.*

  b) *A model number is registered by a manufacturer and therefore can't be used by another manufacturer.*

  c) *All disk drives in a particular batch are the same model.*

  d) *All disk drives of a certain model of a particular manufacturer have exactly the same capacity.*