Updated 11/17/2022

Topic 2: Database design
L20: Normalization

Wolfgang Gatterbauer

CS3200 Database design (fa22)

https://northeastern-datalab.github.io/cs3200/fa22s3/

11/16/2022

Overview
Database normalization & Design Theory

Normalization: What you should take away

- Understand the **normalization process** and why a normalized data model is desirable (in short: we avoid redundancy)
  - Be able to **explain anomalies** and how to avoid them: Insertion, deletion, and modification
- Be able to explain and apply normal forms (NFs):
  - **3rd NF** and **Boyce-Codd NF**.
  - Be able to identify when a relational model is in NF
  - Actually apply normalization process

Normalization

- Organizing data to minimize redundancy (repeated data)
- This is good for two reasons
  - The database takes up less space
  - You have a lower chance of inconsistencies in your data (cp with keeping multiple calendars synched, say Piazza / Canvas / Website)
- If you want to make a change to a record, you only have to make it in one place
  - The relationships (via Foreign Keys) take care of the rest
- But you will usually need to link the separate tables together in order to retrieve information (that's why we have joins...)

First Normal Form (1NF)

- Database can only store "flat" tables (no "nested relations")
- A database schema is in First Normal Form (1NF) if all tables are flat.

Student

Name	GPA	Course
Alice	3.8	Math DB OS
Bob	3.7	DB OS
Carol	3.9	Math OS

How can we avoid "multi-valued attributes" ?

First Normal Form (1NF)

- Database can only store "flat" tables (no "nested relations")
- A database schema is in First Normal Form (1NF) if all tables are flat.

Student

Name	GPA	Course
Alice	3.8	Math DB OS
Bob	3.7	DB OS
Carol	3.9	Math OS

Student

Name	GPA	Course
Alice	3.8	Math
Alice	3.8	DB
Alice	3.8	OS
Bob	3.7	DB
Bob	3.7	OS
Carol	3.9	Math
Carol	3.9	OS

But now we have redundancies ☹

How can we avoid those? ?

First Normal Form (1NF): that is just the start

- Higher NFs avoid redundancies ☺

May need to add unambiguous keys

Student

Name	GPA	Course
Alice	3.8	Math DB OS
Bob	3.7	DB OS
Carol	3.9	Math OS

⇨

Student

Name	GPA	Course
Alice	3.8	Math
Alice	3.8	DB
Alice	3.8	OS
Bob	3.7	DB
Bob	3.7	OS
Carol	3.9	Math
Carol	3.9	OS

⇨

Student

Name	GPA
Alice	3.8
Bob	3.7
Carol	3.9

Takes

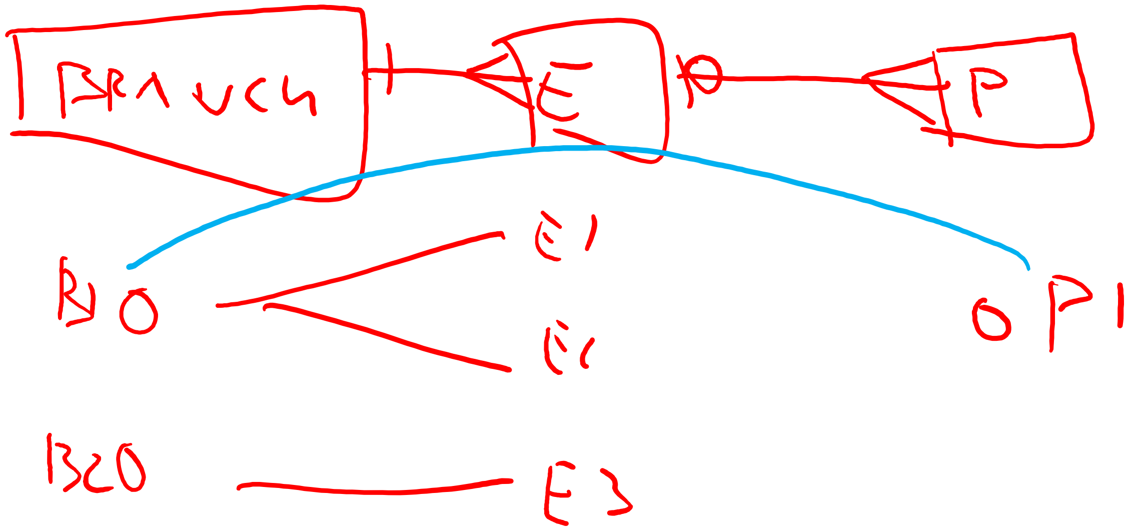
Student	Course
Alice	Math
Carol	Math
Alice	DB
Bob	DB
Alice	OS
Carol	OS

Course

Course
Math
DB
OS

Data Anomalies

- When a database is poorly designed we get anomalies (those are bad) resulting from redundancies:
  - Update anomalies: need to change in several places
  - Insert anomalies: need to repeat data for new inserts
  - Deletion anomalies: may lose data when we don't want (remember the chasm trap!)



Relational Schema Design

Recall multivalued (set) attributes (persons with several phones):

Employee

Name	SSN	PhoneNumber	City
Alice	123-45-6789	617-555-1234	Boston
Alice	123-45-6789	617-555-6543	Boston
Bob	987-65-4321	908-555-2121	Cambridge

A person may have multiple phones, but lives in only one city. PK is thus (SSN, PhoneNumber)

- **Update anomaly**
- **Insert anomaly**
- **Deletion anomaly**

Do you see any anomalies? ?

Relational Schema Design

Recall multivalued (set) attributes (persons with several phones):

Employee

Name	SSN	PhoneNumber	City
Alice	123-45-6789	617-555-1234	Boston
Alice	123-45-6789	617-555-6543	Boston
Bob	987-65-4321	908-555-2121	Cambridge

A person may have multiple phones, but lives in only one city. PK is thus (SSN, PhoneNumber)

- **Update anomaly** What if Alice moves to "New York"?
- **Insert anomaly** What if Alice gets a 3rd telephone number?
- **Deletion anomaly** What if Bob deletes his phone number? (or Joe has no phone number; recall chasm trap)

So what do we do ?

Relation Decomposition

Employee

Name	SSN	PhoneNumber	City
Alice	123-45-6789	617-555-1234	Boston
Alice	123-45-6789	617-555-6543	Boston
Bob	987-65-4321	908-555-2121	Cambridge

Break the single relation
into two relations!
Hint: "separation of concerns"

Employee

Name	SSN	City
Alice	123-45-6789	Boston
Bob	987-65-4321	Cambridge

Phone

SSN	PhoneNumber
123-45-6789	617-555-1234
123-45-6789	617-555-6543
987-65-4321	908-555-2121

Now Anomalies have gone ☺

- No more repeated data
- Easy update for Alice to move to "New York" (how ?)
- Deleting Bob's single phone number (how ?) has no side-effects

Good News / Bad News

- The good news: when you start with solid ER modeling and follow the steps described to create relations then your relations will usually be pretty well normalized

- The bad news: you often don't have the benefit of starting from a well-designed model.

- The good news (part 2): the steps we will cover in class will help you convert poorly normalized tables into highly normalized tables ("mechanical translation")

1. Normal forms and Functional Dependencies

Database design & Normal forms

- Normalization (and database design) is about how to represent your data to avoid anomalies.

- It is a mostly mechanical process
  - Tools can carry out routine portions

- We have a Python notebook from the Stanford group that implements and illustrates the algorithms!
  - (If there is strong demand, I will post it again and you can play with it. In the past, it created lots of confusion because students did not know Python)

Data Normalization

- Data normalization is the process of decomposing relations with anomalies to produce smaller, well-structured relations

- Goals of normalization include:
  - Minimize data redundancy
  - Simplifying the enforcement of referential integrity constraints
  - Simplify data maintenance (inserts, updates, deletes)
  - Improve representation model to match "the real world"

Well-Structured Relations

- A <u>well-structured relation</u> contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies

- <u>Anomalies</u> are errors or inconsistencies that may result when a user attempts to update a table that contains redundant data.

- Three types of anomalies:
  - <u>Insertion Anomaly</u> – adding new rows forces user to create duplicate data
  - <u>Deletion Anomaly</u> – deleting rows may cause a loss of data that would be needed for other future rows
  - <u>Modification Anomaly</u> – changing data in a row forces changes to other rows because of duplication

- General rule of thumb: a table should not pertain to more than one entity type

Normal Forms

Normal Form: a state of a relation that results from applying simple rules regarding FDs ("Functional Dependencies") to that relation

- 1st Normal Form (1NF) = All tables are flat
- 2nd Normal Form = not used anymore
  - no more "partial FDs" (those are part of the "bad" FDs)
- **3rd Normal Form (3NF)**
  - **no more transitive FDs (also "bad")**
- **Boyce-Codd Normal Form (BCNF)**
  - **every determinant is a candidate key**

DB designs based on FDs (*functional dependencies*), intended to prevent data ***anomalies***

Our focus next

- 4th: any multivalued dependencies have been removed (we will give intuition)
- 5th: any remaining anomalies have been removed (not covered)

1st Normal Form (1NF)

Student	**Courses**
Mary	{CS3200, CS4240}
Joe	{CS3200, CS4240}
…	…

Violates 1NF.

1st Normal Form (1NF)

Student	Courses
Mary	{CS3200, CS4240}
Joe	{CS3200, CS4240}
…	…

Violates 1NF.

Student	Course
Mary	CS3200
Mary	CS4240
Joe	CS3200
Joe	CS4240

In 1st NF

1NF Constraint: Types must be atomic!

Constraints Prevent (some) Anomalies in the Data

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

Student	**Course**	**Room**
Mary	CS3200	WVF20
Joe	CS3200	WVF20
Sam	CS3200	WVF20
..	..	..

If every course is in only one room, contains ***redundant*** information!

Constraints Prevent (some) Anomalies in the Data

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

Student	**Course**	**Room**
Mary	CS3200	WVF20
Joe	CS3200	B12
Sam	CS3200	WVF20
..	..	..

If we update the room number for one tuple, we get inconsistent data = an ***update* anomaly**

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes *anomalies*:

Student	Course	Room
..	..	..

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

Constraints Prevent (some) Anomalies in the Data

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

...	CS4240	B12

Student	Course	Room
Mary	CS3200	WVF20
Joe	CS3200	WVF20
Sam	CS3200	WVF20
..	..	..

Similarly, we can’t reserve a room without students = a variant of an ***insert* anomaly**

Constraints Prevent (some) Anomalies in the Data

Student	Course
Mary	CS3200
Joe	CS3200
Sam	CS3200
..	..

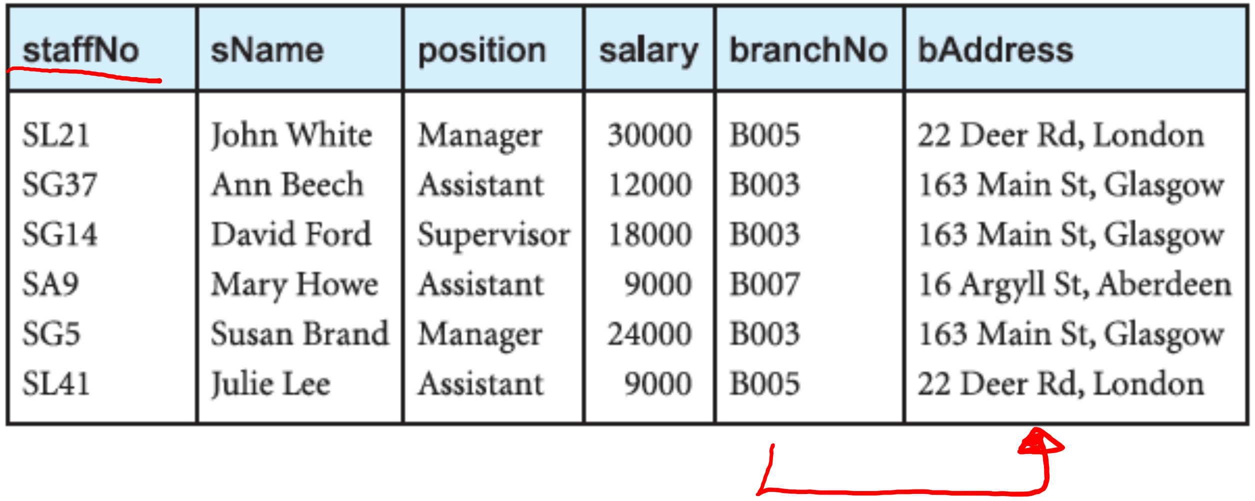
Course	Room
CS3200	WVF20
CS4240	B12

Is this form better?

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

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

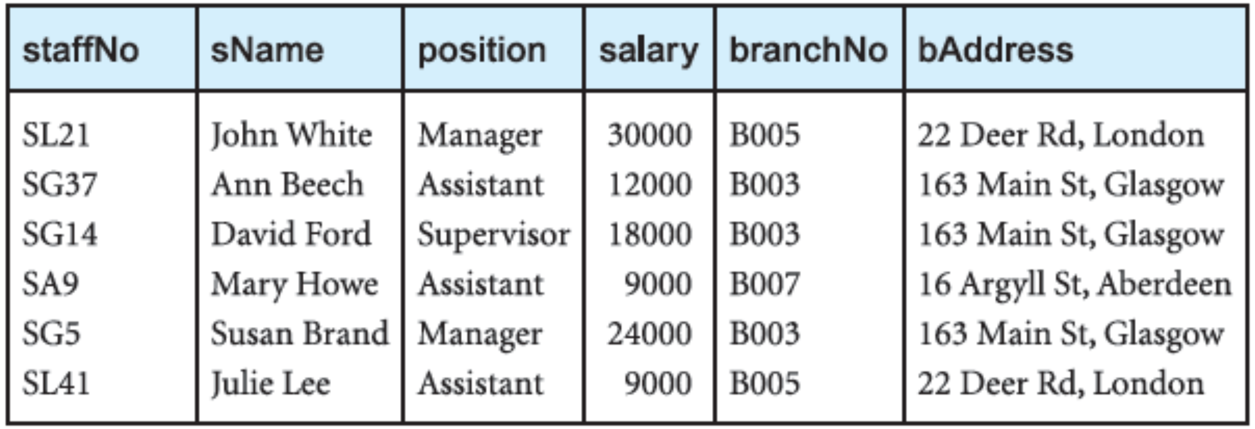
StaffBranch



staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

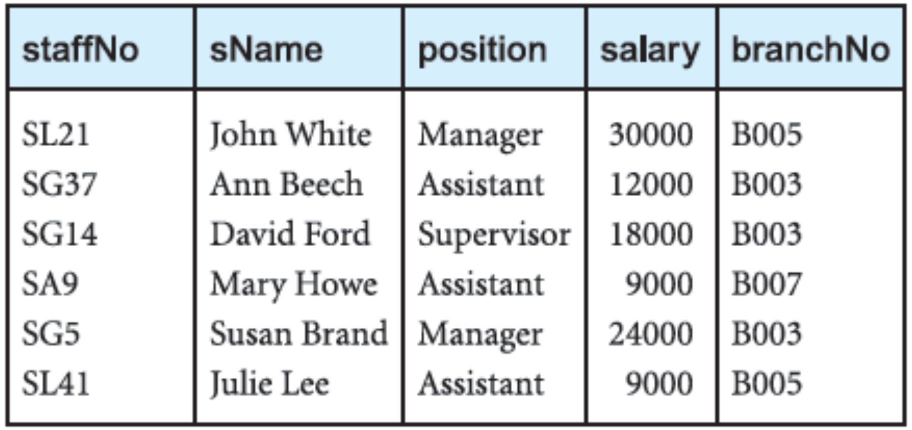
Source: Connolly, Begg: Database systems, 4th ed, p. 423, 2005.

StaffBranch



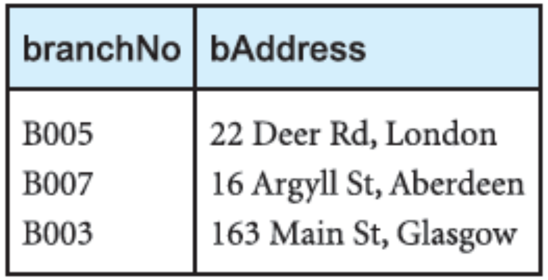
staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Staff



staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

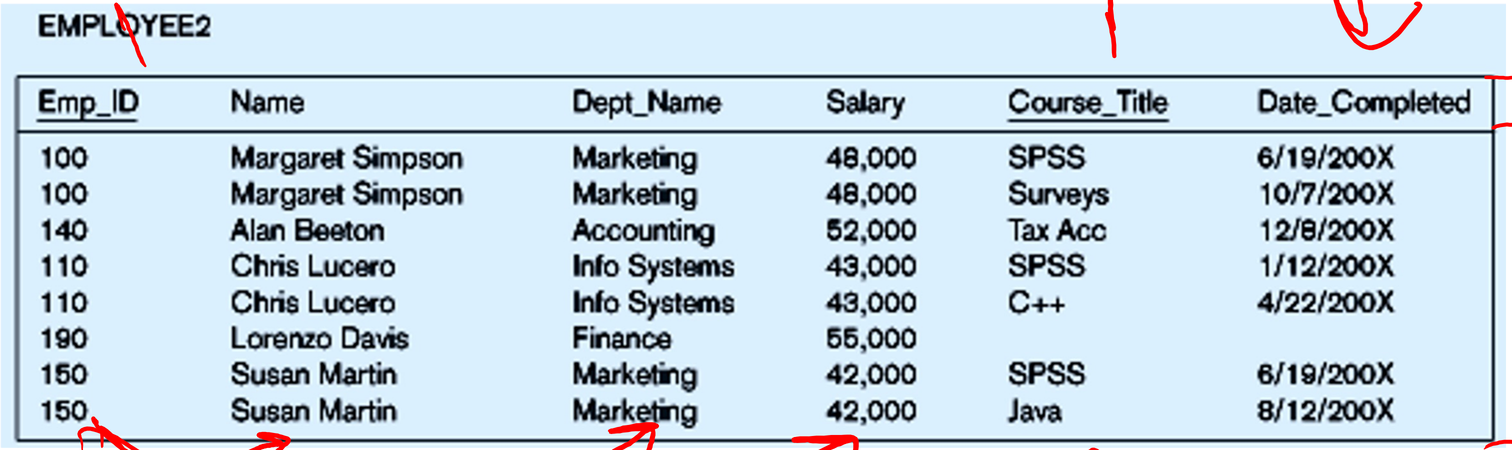
Branch



branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Source: Connolly, Begg: Database systems, 4th ed, p. 423, 2005.

Is This Table Well Structured?



EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	SPSS	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

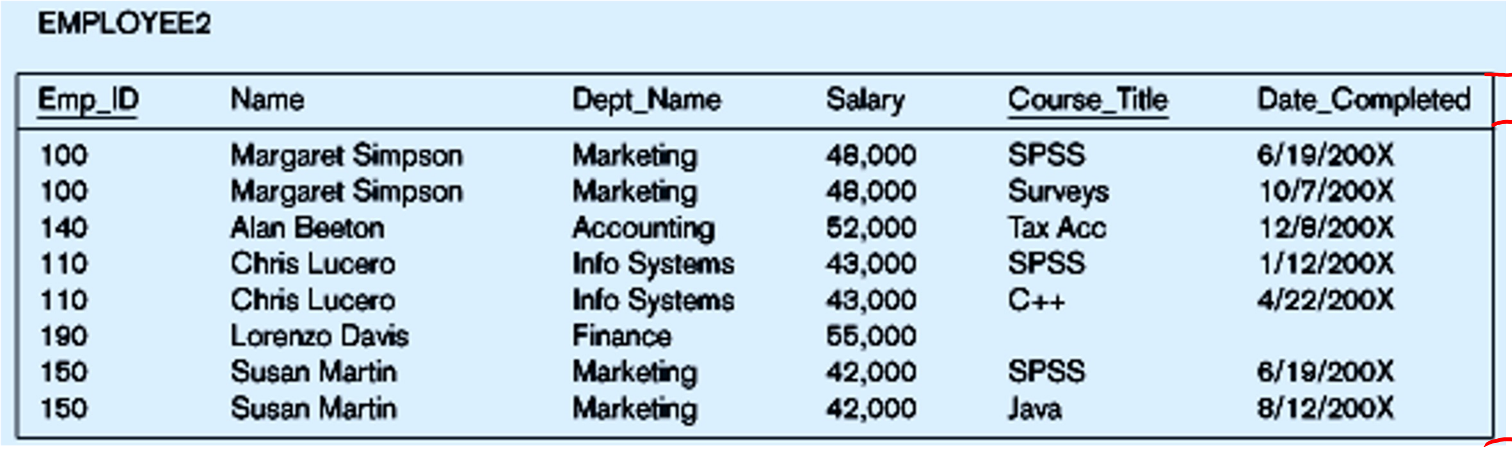
DIFFICULTY

1
2
3
1
2
1

- Does it contain anomalies?

Figure Source: Hoffer, Ramesh, Topi, "Modern database management," 10th ed, 2010.
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

Is This Table Well Structured?



EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	SPSS	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

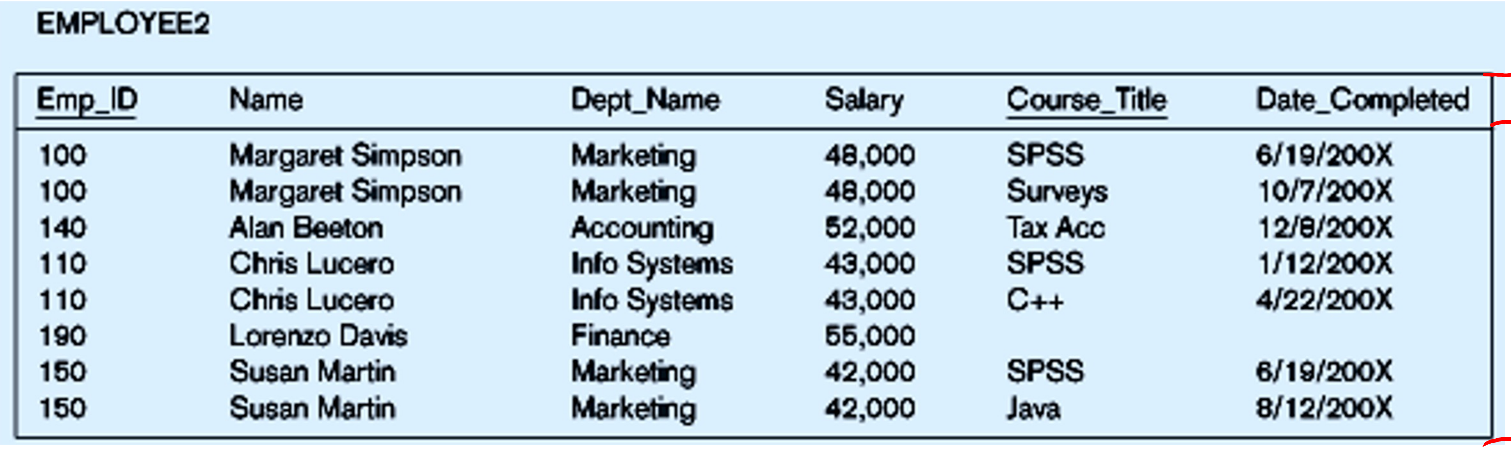
DIFFICULTY

1
2
3
1
2
1

- Does it contain anomalies?
  - Insertion: if an employee takes a new class we need to add duplicate data (Name, Dept_Name, Salary)
  - Deletion: If we remove employee 140, we lose information about the existence of a Tax Acc class
  - Modification: Giving a salary increase to employee 100 forces us to update multiple records
- Why do these anomalies exist?

Figure Source: Hoffer, Ramesh, Topi, "Modern database management," 10th ed, 2010.

Is This Table Well Structured?



EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	SPSS	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

DIFFICULTY

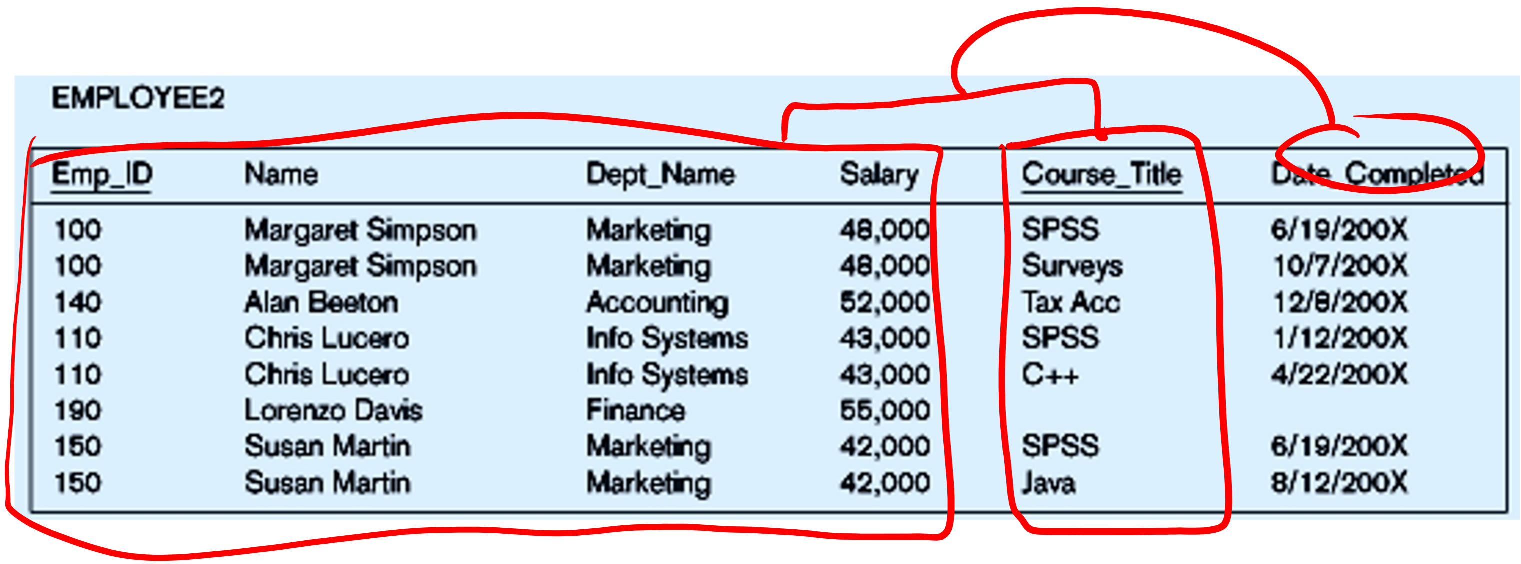
1
2
3
1
2
1

- Does it contain anomalies?
  - Insertion: if an employee takes a new class we need to add duplicate data (Name, Dept_Name, Salary)
  - Deletion: If we remove employee 140, we lose information about the existence of a Tax Acc class
  - Modification: Giving a salary increase to employee 100 forces us to update multiple records

- Why do these anomalies exist?
  - Because there are two themes (entity types) in one relation. This results in duplication, and an unnecessary dependency between the entities

Figure Source: Hoffer, Ramesh, Topi, "Modern database management," 10th ed, 2010.

Is This Table Well Structured?



EMPLOYEE2

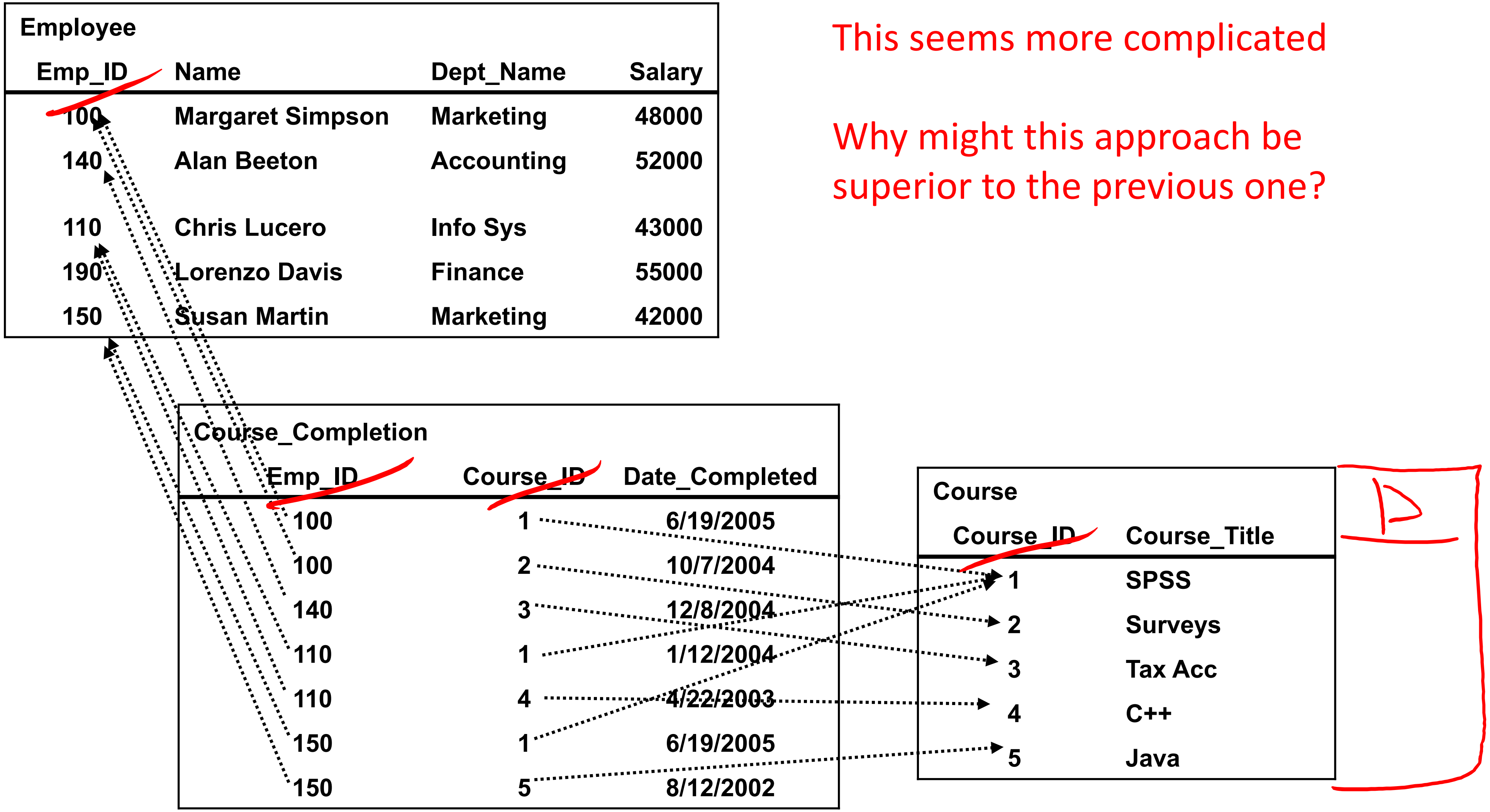
Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	SPSS	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

- Does it contain anomalies?
  - Insertion: if an employee takes a new class we need to add duplicate data (Name, Dept_Name, Salary)
  - Deletion: If we remove employee 140, we lose information about the existence of a Tax Acc class
  - Modification: Giving a salary increase to employee 100 forces us to update multiple records

- Why do these anomalies exist?
  - Because there are <u>two themes (entity types) in one relation</u>. This results in duplication, and an unnecessary dependency between the entities

Figure Source: Hoffer, Ramesh, Topi, "Modern database management," 10th ed, 2010.

Normalizing Previous Employee/Class Table



Employee

Emp_ID	Name	Dept_Name	Salary
100	Margaret Simpson	Marketing	48000
140	Alan Beeton	Accounting	52000
110	Chris Lucero	Info Sys	43000
190	Lorenzo Davis	Finance	55000
150	Susan Martin	Marketing	42000

Course_Completion

Emp_ID	Course_ID	Date_Completed
100	1	6/19/2005
100	2	10/7/2004
140	3	12/8/2004
110	1	1/12/2004
110	4	4/22/2003
150	1	6/19/2005
150	5	8/12/2002

Course

Course_ID	Course_Title
1	SPSS
2	Surveys
3	Tax Acc
4	C++
5	Java

This seems more complicated

Why might this approach be superior to the previous one?

Functional Dependencies ("FDs")

Definition:

If two tuples agree on the attributes

$A_1, A_2, \ldots, A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots, B_m$

Formally:

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

Functional Dependencies ("FDs")

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:

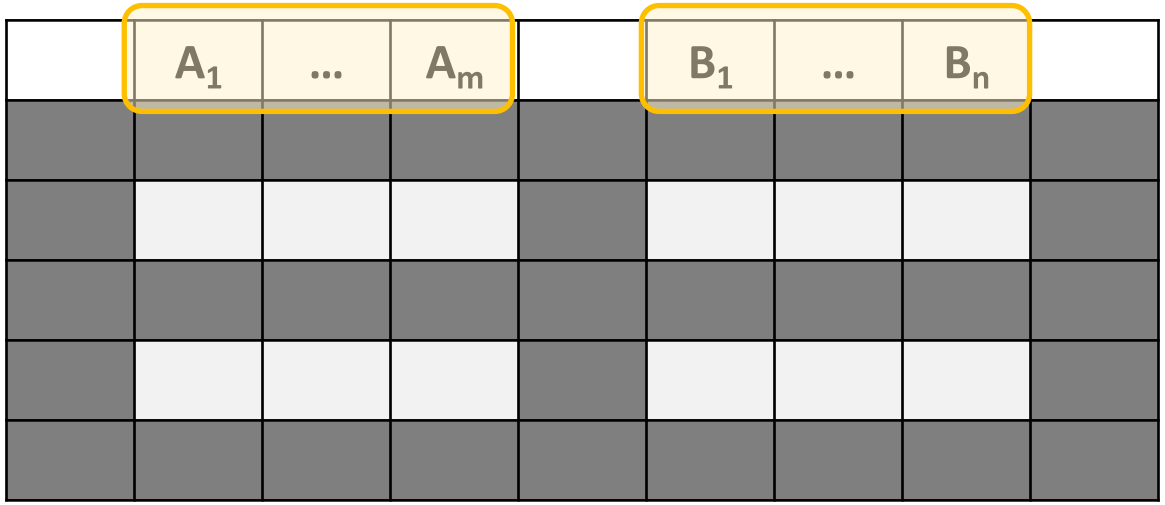
$$t_1[A] = t_2[A] \text{ implies } t_1[B] = t_2[B]$$

and we call A → B a <u>functional dependency</u>

A (<u>determinant</u>) → B (<u>dependent</u>)

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

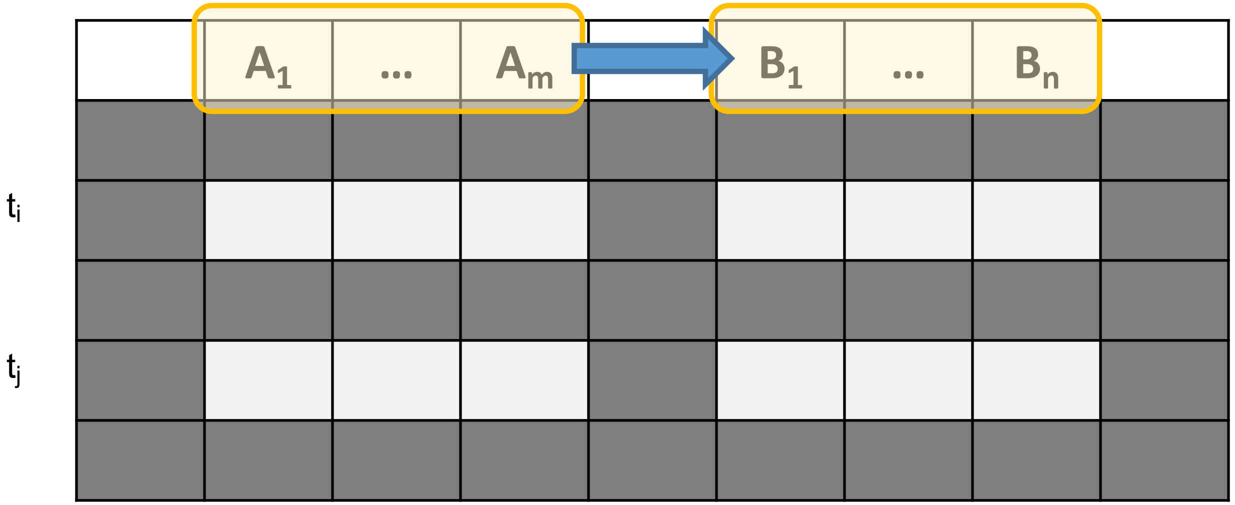
A Picture Of FDs



Defn (again):

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

A Picture Of FDs



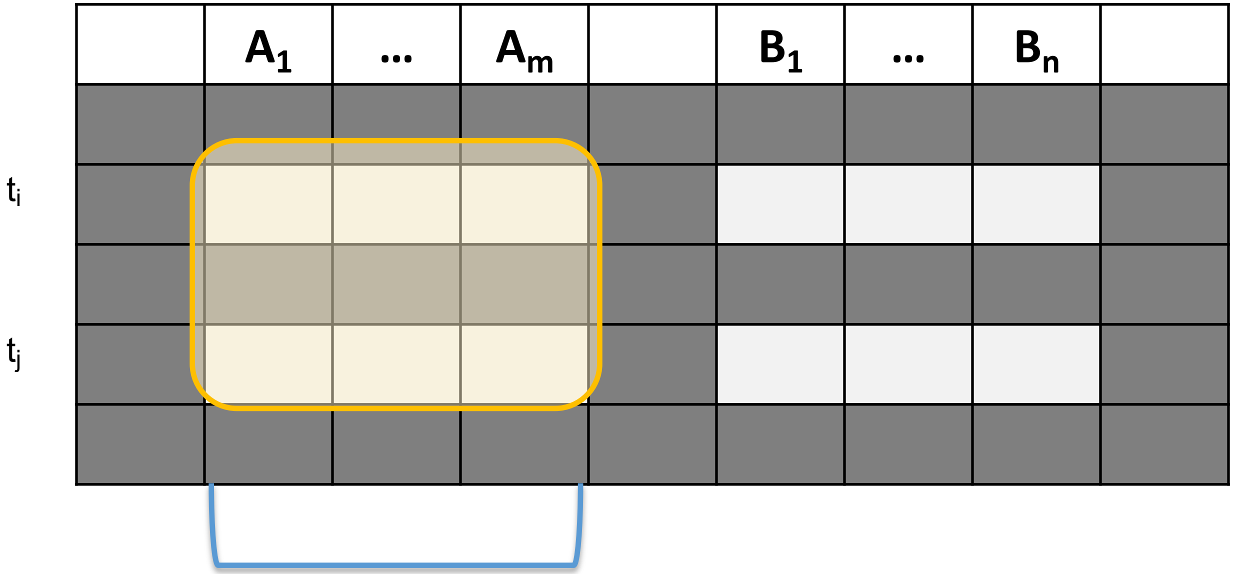
Defn (again):

Given attribute sets **A={A_1,…,A_m}** and **B = {B_1,…B_n}** in **R**,

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

Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

A Picture Of FDs



If t_i,t_j agree here..

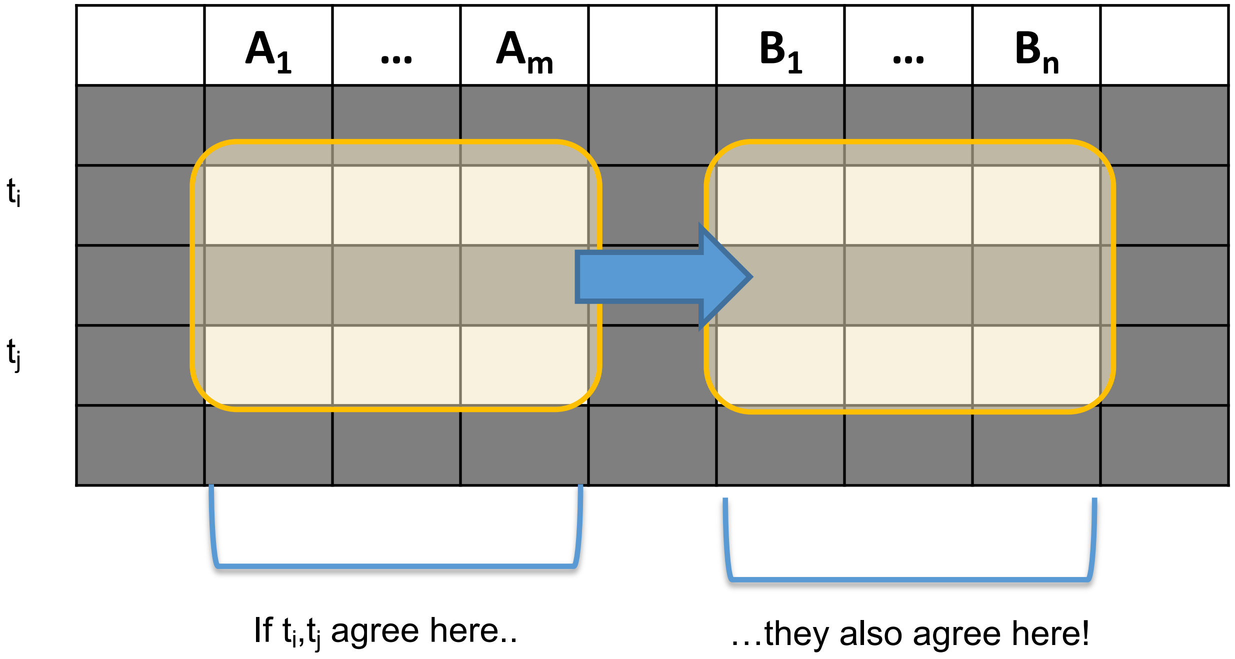
Defn (again):

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

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

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

A Picture Of FDs



Defn (again):

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

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

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

then $t_i[B_1] = t_j[B_1]$ AND $t_i[B_2]=t_j[B_2]$ AND … AND $t_i[B_n] = t_j[B_n]$

FDs for Relational Schema Design

- High-level idea: why do we care about FDs?
  - Start with some relational schema
  - Find out its functional dependencies (FDs)
  - 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**

- *Holds* on some instances (but not others) – can check whether there are violations
- Part of the schema, helps define a *valid* instance

Student	Course	Room
Mary	CS3200	WVF20
Joe	CS3200	WVF20
Sam	CS3200	WVF20
..	..	..

Note: The FD {Course} → {Room} ***holds on this instance***

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

Functional Dependencies as Constraints

Note that:

- You can check if an FD is **violated** 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**
Mary	CS3200	WVF20
Joe	CS3200	WVF20
Sam	CS3200	WVF20
..	..	..

However, 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}

Practice

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 are violated on this instance:

{ } → { }
{ } → { }
{ } → { }