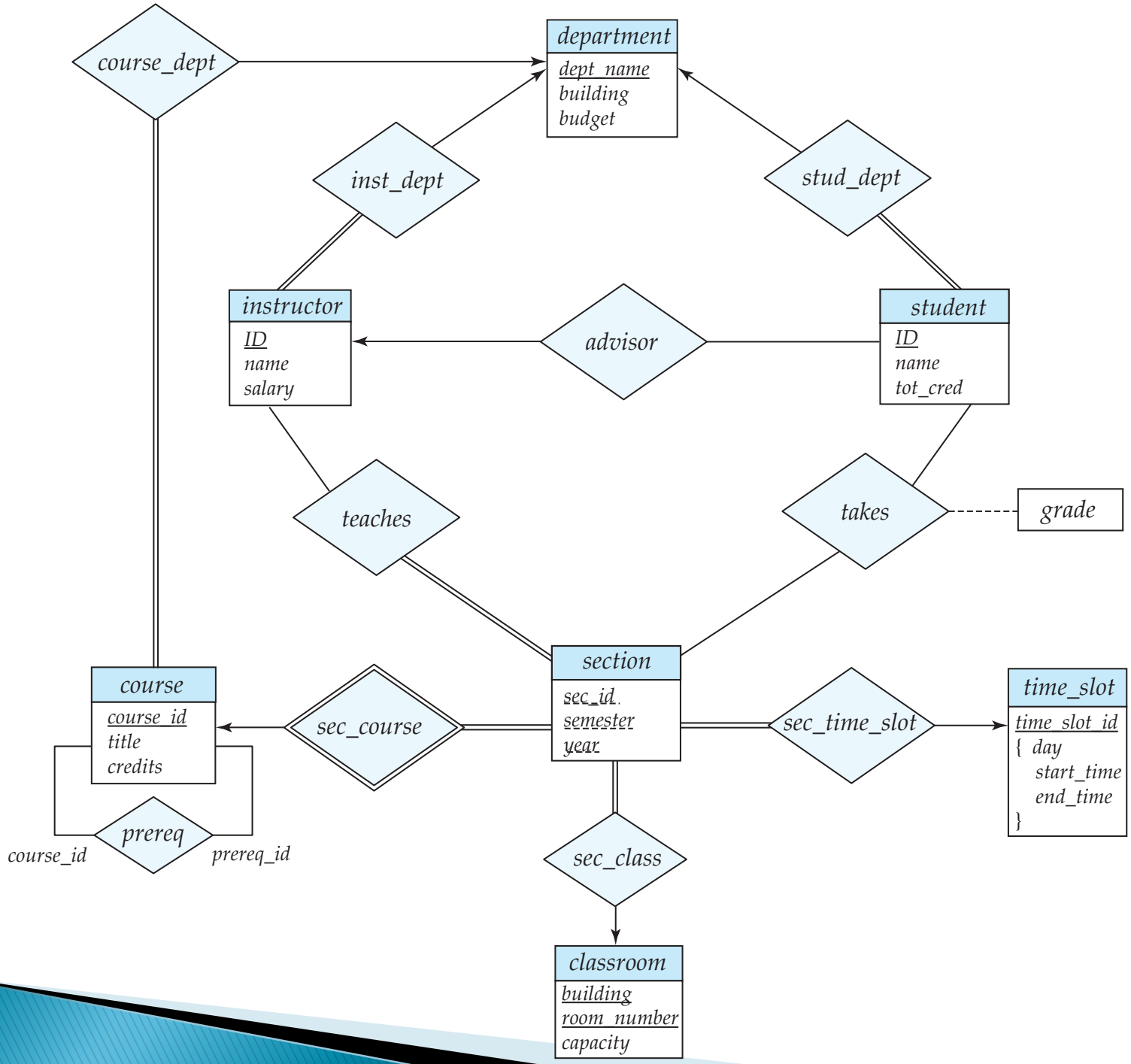
CMSC424: Database Design
E/R; Normalization

February 24, 2020

Instructor: Amol Deshpande
amol@cs.umd.edu

Today's Plan

- Entity-Relationship Model Review
- Converting from E/R Model to Relational Schema
- Normalization


course_dept
department
dept_name
building
budget
inst_dept
stud_dept
instructor
ID
name
salary
advisor
student
ID
name
tot_cred
teaches
takes
grade
section
sec_id
semester
year
course
course_id
title
credits
sec_course
sec_time_slot
time_slot
time_slot_id
{ day
start_time
end_time
}
prereq
course_id
prereq_id
sec_class
classroom
building
room_number
capacity

Thoughts…

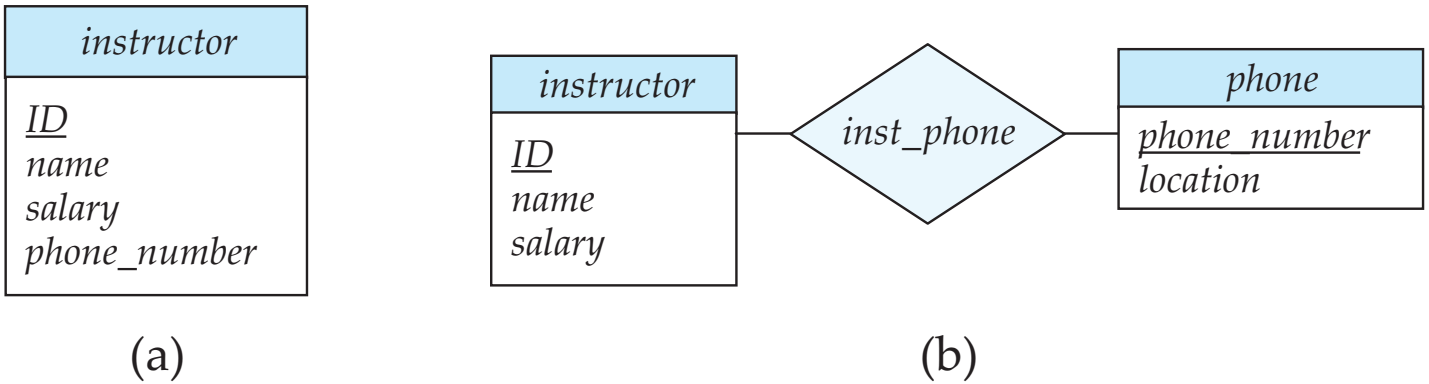
- Nothing about actual data
  - How is it stored ?
- No talk about the query languages
  - How do we access the data ?
- Semantic vs Syntactic Data Models
  - Remember: E/R Model is used for conceptual modeling
  - Many conceptual models have the same properties
- They are much more about representing the knowledge than about database storage/querying

Design Issues

- Entity sets vs attributes
  - Depends on the semantics of the application
  - Consider *telephone*
- Entity sets vs Relationsihp sets
  - Consider *takes*
- N-ary vs binary relationships
  - Possible to avoid n-ary relationships, but there are some cases where it is advantageous to use them
- It is not an exact science !!

Design Issues

- Entity sets vs attributes
  - Depends on the semantics of the application
  - Consider *telephone*



Design Issues

- Entity sets vs Relationsihp sets
  - Consider *takes*

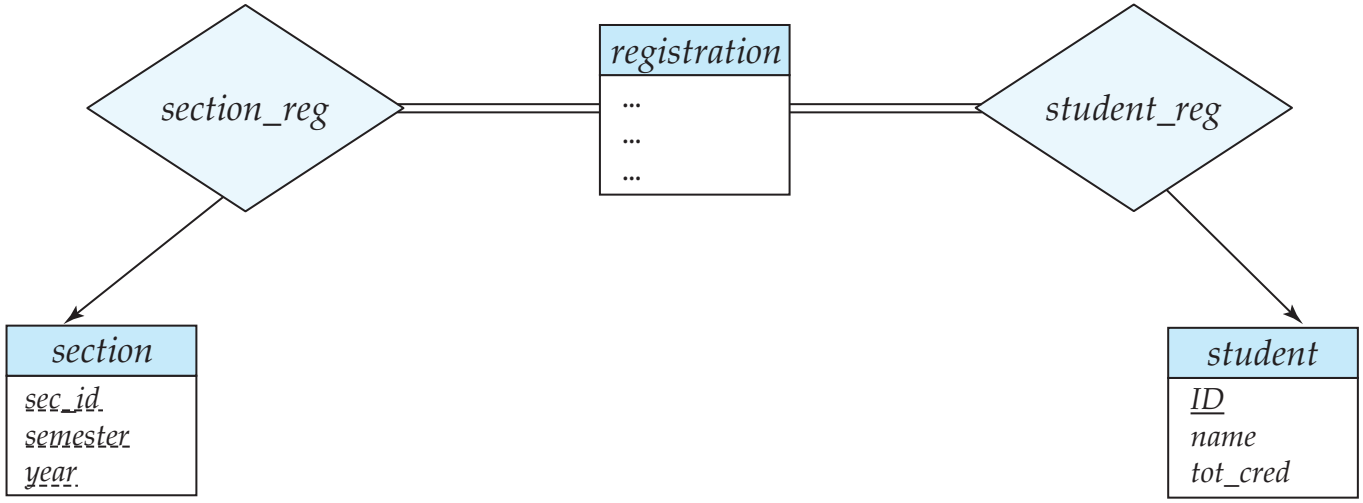


Figure 7.18 Replacement of *takes* by *registration* and two relationship sets

Thoughts…

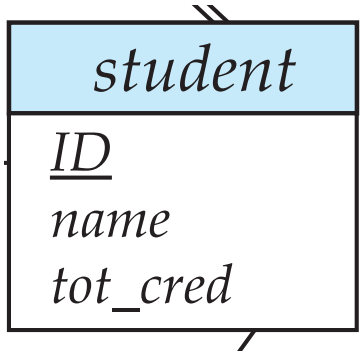
- Basic design principles
  - Faithful
    - Must make sense
  - Satisfies the application requirements
  - Models the requisite domain knowledge
    - If not modeled, lost afterwards
  - Avoid redundancy
    - Potential for inconsistencies
  - Go for simplicity
- Typically an iterative process that goes back and forth

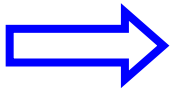
Today's Plan

- Entity-Relationship Model Review
- Converting from E/R Model to Relational Schema
- Normalization

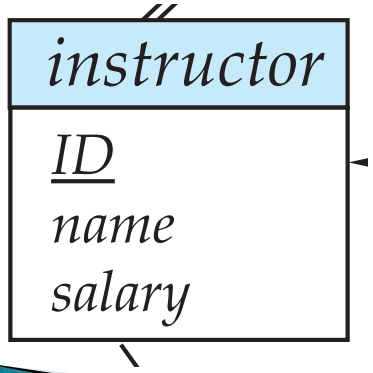
E/R Diagrams → Relations

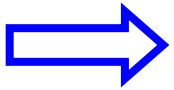
- Convert entity sets into a relational schema with the same set of attributes





Student (ID, name, tot_cred)

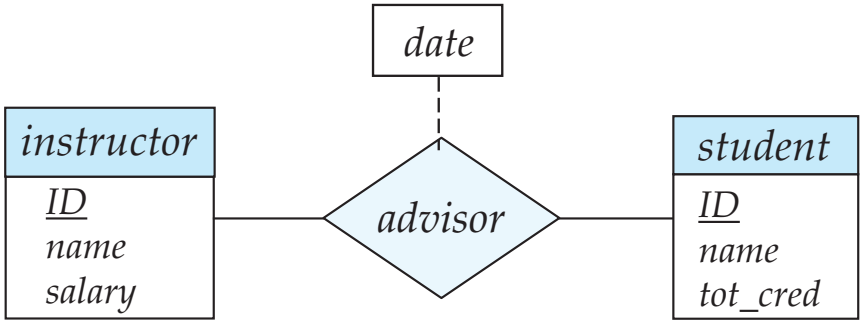




Instructor(ID, name, salary)

E/R Diagrams → Relations

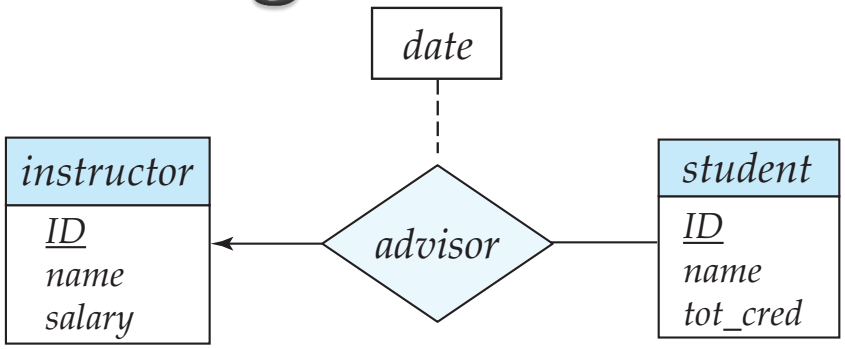
- Convert relationship sets *also* into a relational schema
- Remember: A relationship is completely described by primary keys of associate entities and its own attributes



Advisor (student_ID, instructor_ID, date)

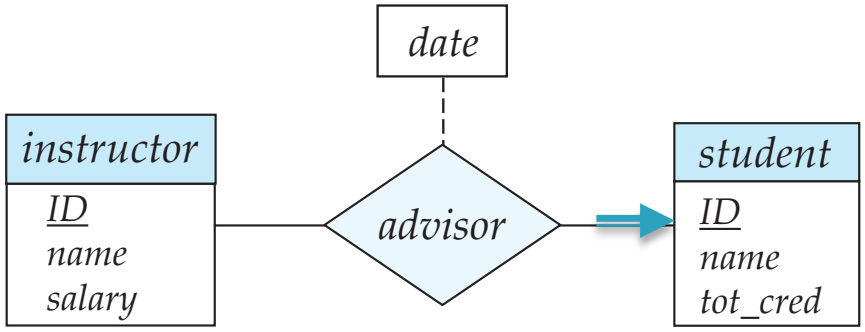
We can do better for many-to-one or one-to-one

E/R Diagrams → Relations



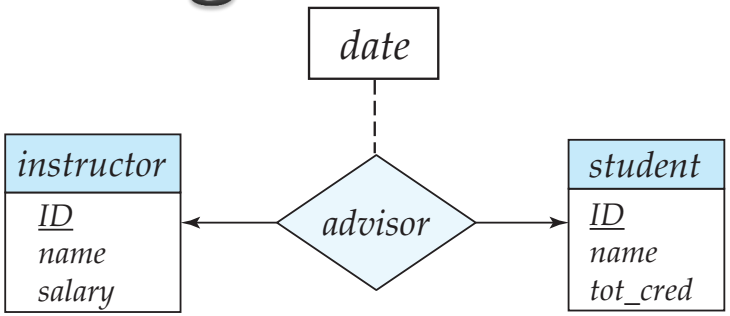
Foreign key into Instructor relation

Fold into Student:
Student(ID, name, tot_credits, advisor_ID)



Fold into Instructor:
Instructor(ID, name, salary, advisee_ID)

E/R Diagrams → Relations



Fold into Student:
Student(ID, name, tot_credits, advisor_ID)

OR

Fold into Instructor:
Instructor(ID, name, salary, advisee_ID)

Weak Entity Sets

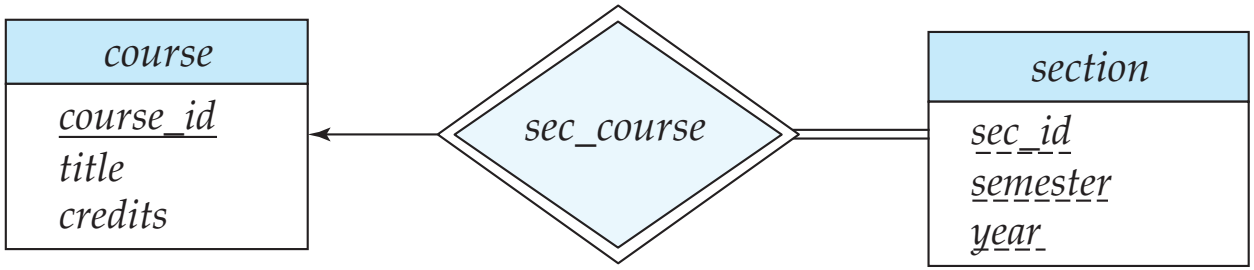


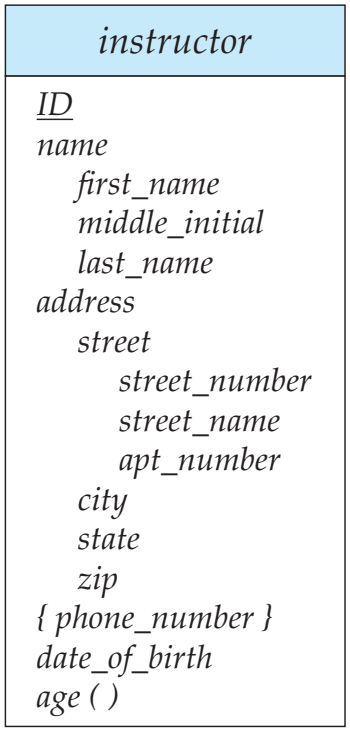
Figure 7.14 E-R diagram with a weak entity set.

Need to copy the primary key from the strong entity set:

Section(course_id, sec_id, semester, year)

Primary key for section = Primary key for course + Discriminator Attributes

Multi-valued Attributes



instructor (*ID*, *first_name*, *middle_name*, *last_name*, *street_number*, *street_name*, *apt_number*, *city*, *state*, *zip_code*, *date_of_birth*)

BUT

Phone_number needs to be split out into a separate table

Instructor_Phone(Instructor_ID, phone_number)

Specialization and Generalization

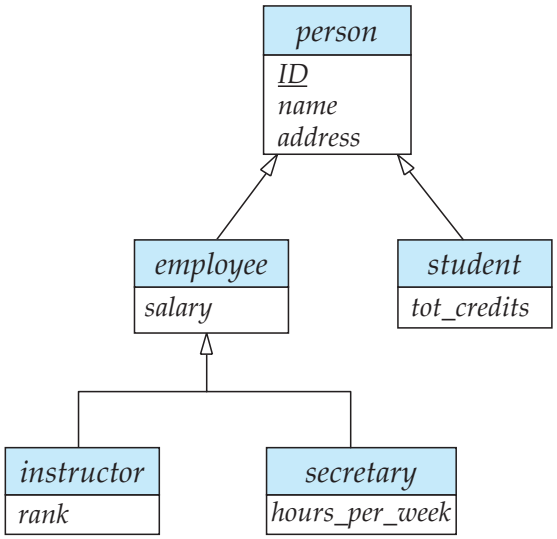


Figure 7.21 Specialization and generalization.

A few different ways to handle it

1. Common table for common information and separate tables for additional information

   person (<u>*ID*</u>, *name*, *street*, *city*)
   employee (<u>*ID*</u>, *salary*)
   student (<u>*ID*</u>, *tot_cred*)

2. Separate tables altogether – good idea if an employee can't be a student also – querying becomes harder (have to do unions for queries across all "persons")

   employee (<u>*ID*</u>, *name*, *street*, *city*, *salary*)
   student (<u>*ID*</u>, *name*, *street*, *city*, *tot_cred*)

Today's Plan

- Entity-Relationship Model Review
- Converting from E/R Model to Relational Schema
- Normalization

Relational Database Design

- Where did we come up with the schema that we used ?
  - E.g. why not store the actor names with movies ?
- If from an E-R diagram, then:
  - Did we make the right decisions with the E-R diagram ?
- Goals:
  - Formal definition of what it means to be a “good” schema.
  - How to achieve it.

Movies Database Schema

Movie(*title, year*, length, inColor, studioName, producerC#)

StarsIn(movieTitle, movieYear, starName)

MovieStar(*name*, address, gender, birthdate)

MovieExec(name, address, *cert#,* netWorth)

Studio(*name*, address, presC#)

Changed to:

Movie(*title, year*, length, inColor, studioName, producerC#, starName)

<StarsIn merged into above>

MovieStar(*name*, address, gender, birthdate)

MovieExec(name, address, *cert#,* netWorth)

Studio(*name*, address, presC#)

Is this a good schema ???

Movie(*title, year*, length, inColor, studioName, producerC#, starName)

Title	Year	Length	inColor	StudioName	prodC#	StarName
Star wars	1977	121	Yes	Fox	128	Hamill
Star wars	1977	121	Yes	Fox	128	Fisher
Star wars	1977	121	Yes	Fox	128	H. Ford
King Kong	2005	187	Yes	Universal	150	Watts
King Kong	1933	100	no	RKO	20	Fay

Issues:

1. Redundancy ➔ higher storage, inconsistencies ("anomalies")

   update anomalies, insertion anamolies

2. Need nulls

   Unable to represent some information without using nulls

   How to store movies w/o actors (pre-productions etc) ?

Movie(*title, year*, length, inColor, studioName, producerC#, starNames)

Title	Year	Length	inColor	StudioName	prodC#	StarNames
Star wars	1977	121	Yes	Fox	128	{Hamill, Fisher, H. ford}
King Kong	2005	187	Yes	Universal	150	Watts
King Kong	1933	100	no	RKO	20	Fay

Issues:

3. Avoid sets

- Hard to represent
- Hard to query

Smaller schemas always good ????

Split Studio(*name*, address, presC#) into:

Studio1 (name, presC#)

Name	presC#
Fox	101
Studio2	101
Universial	102

Studio2(name, address)???

Name	Address
Fox	Address1
Studio2	Address1
Universial	Address2

This process is also called *"decomposition"*

Issues:

4. Requires more joins (w/o any obvious benefits)

5. Hard to check for some dependencies

What if the "address" is actually the presC#'s address ?

No easy way to ensure that constraint (w/o a join).

Smaller schemas always good ????

Decompose StarsIn(movieTitle, movieYear, starName) into:

StarsIn1(movieTitle, movieYear)

movieTitle	movieYear
Star wars	1977
King Kong	1933
King Kong	2005

StarsIn2(movieTitle, starName) ???

movieTitle	starName
Star Wars	Hamill
King Kong	Watts
King Kong	Faye

Issues:

6. "joining" them back results in more tuples than what we started with

(King Kong, 1933, Watts) & (King Kong, 2005, Faye)

This is a "lossy" decomposition

We lost some constraints/information

The previous example was a "lossless" decomposition.

Desiderata

- No sets
- Correct and faithful to the original design
  - Avoid lossy decompositions
- As little redundancy as possible
  - To avoid potential anomalies
- No “inability to represent information”
  - Nulls shouldn’t be required to store information
- Dependency preservation
  - Should be possible to check for constraints

Not always possible.

We sometimes relax these for:

simpler schemas, and *fewer joins during queries.*

Some of Your Questions

- Atomicity
  - It depends primarily on how you use it
  - A String is not really atomic (can be split into letters), but do you want to query the letters directly? Or would your queries operate on the strings?
- Which NF to use?
  - Your choice – Normalization theory is a tool to help you understand the tradeoffs
- Normal forms higher than 3NF?
  - Actually we always use 4NF – we will discuss later
- Trivial FDs
  - Just means that: RHS is contained in LHS – that's all

Approach

1. We will encode and list all our knowledge about the schema
   - Functional dependencies (FDs)

     SSN → name (means: SSN "implies" length)
   - If two tuples have the same "SSN", they must have the same "name"

     movietitle → length ???? Not true.
   - But, (movietitle, movieYear) → length --- True.
2. We will define a set of rules that the schema must follow to be considered good
   - "Normal forms": 1NF, 2NF, 3NF, BCNF, 4NF, …
   - A normal form specifies constraints on the schemas and FDs
3. If not in a "normal form", we modify the schema

FDs: Example 1

Title	Year	Length	StarName	Birthdate	producerC#	Producer-address	Prdocuer-name	netWorth
Plane Crazy	1927	6	NULL	NULL	WD100	Mickey Rd	Walt Disney	100000
Star Wars	1977	121	H. Ford	7/13/42	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	M. Hamill	9/25/51	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	C. Fisher	10/21/56	GL102	Tatooine	George Lucas	10^9
King Kong	1933	100	F. Wray	9/15/07	MC100	…	…	…
King Kong	2005	187	N. Watts	9/28/68	PJ100	Middle Earth	Peter Jackson	10^8

FDs: Example 2

State Name	State Code	State Population	County Name	County Population	Senator Name	Senator Elected	Senator Born	Senator Affiliation
Alabama	AL	4779736	Autauga	54571	Jeff Sessions	1997	1946	'R'
Alabama	AL	4779736	Baldwin	182265	Jeff Sessions	1997	1946	'R'
Alabama	AL	4779736	Barbour	27457	Jeff Sessions	1997	1946	'R'
Alabama	AL	4779736	Autauga	54571	Richard Shelby	1987	1934	'R'
Alabama	AL	4779736	Baldwin	182265	Richard Shelby	1987	1934	'R'
Alabama	AL	4779736	Barbour	27457	Richard Shelby	1987	1934	'R'

FDs: Example 3

Course ID	Course Name	Dept Name	Credits	Section ID	Semester	Year	Building	Room No.	Capacity	Time Slot ID

Functional dependencies

course_id → title, dept_name, credits
building, room_number → capacity
course_id, section_id, semester, year → building, room_number, time_slot_id

Examples from Quiz

- advisor(s_id, i_id, s_name, s_dept_name, i_name, i_dept_name)

- friends(userid1, userid2, name1, name2, birthdate1, birthdate2)