The Entity-Relationship Model
ER Model - Part 3: Advanced Concepts

By Michael Hahsler

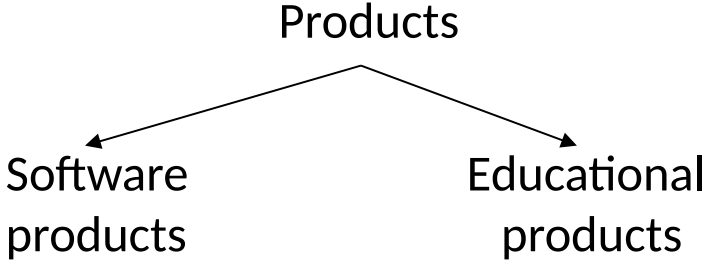
Based on slides for CS145 Introduction to Databases (Stanford)

What you will learn about in this section

1. Subclasses
2. Constraints
3. Weak entity sets
4. Normal Forms

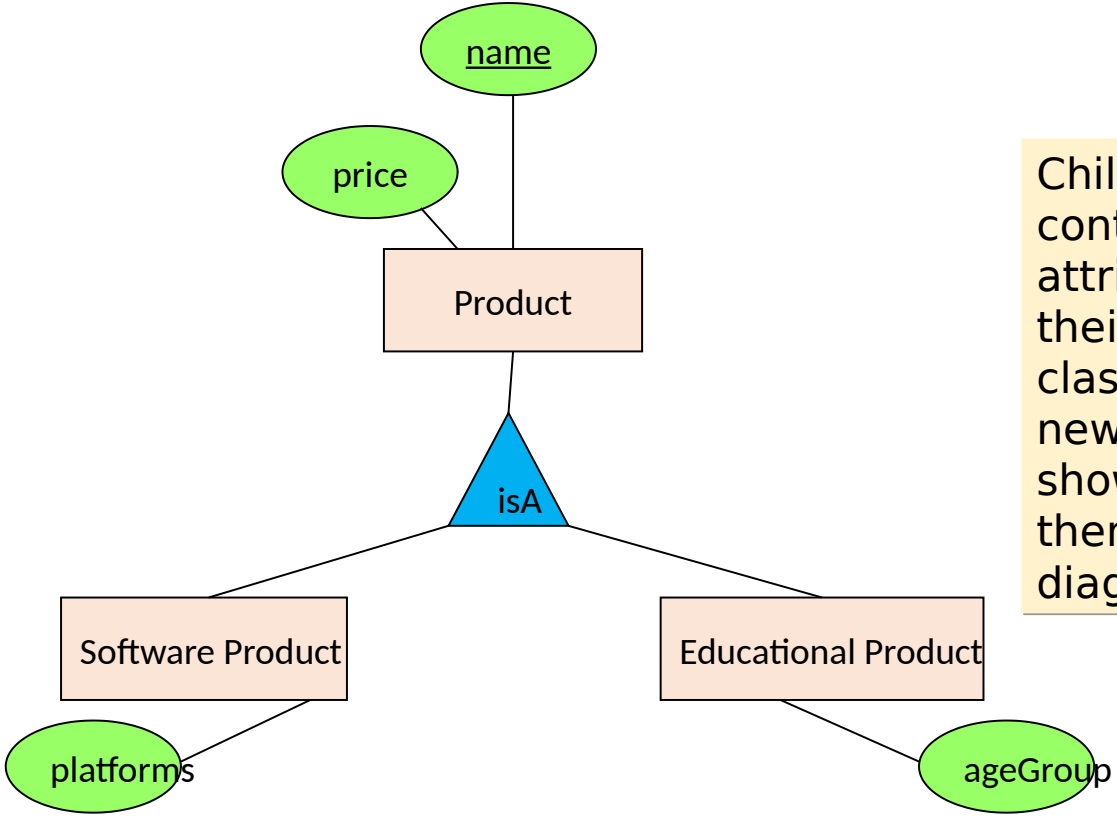
Modeling Subclasses

- Some objects in a class may be special, i.e. worthy of their own class
- Define a new class?
  - *But what if we want to maintain connection to current class?*
- Better: define a *subclass*
  - *Ex:*



We can define **subclasses** in ER!

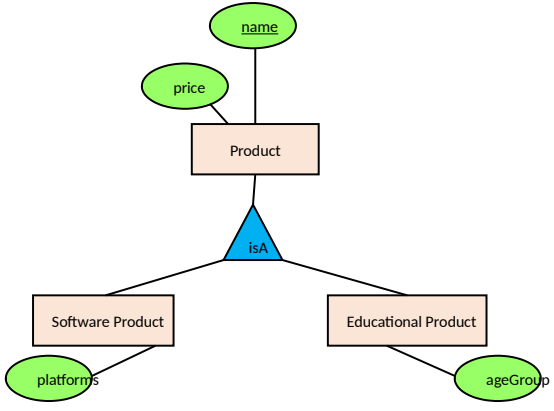
Modeling Subclasses



Child subclasses contain all the attributes of *all* of their parent classes **plus** the new attributes shown attached to them in the ER diagram

Understanding Subclasses

- Think in terms of records; ex:



- Product

name
price

- SoftwareProduct

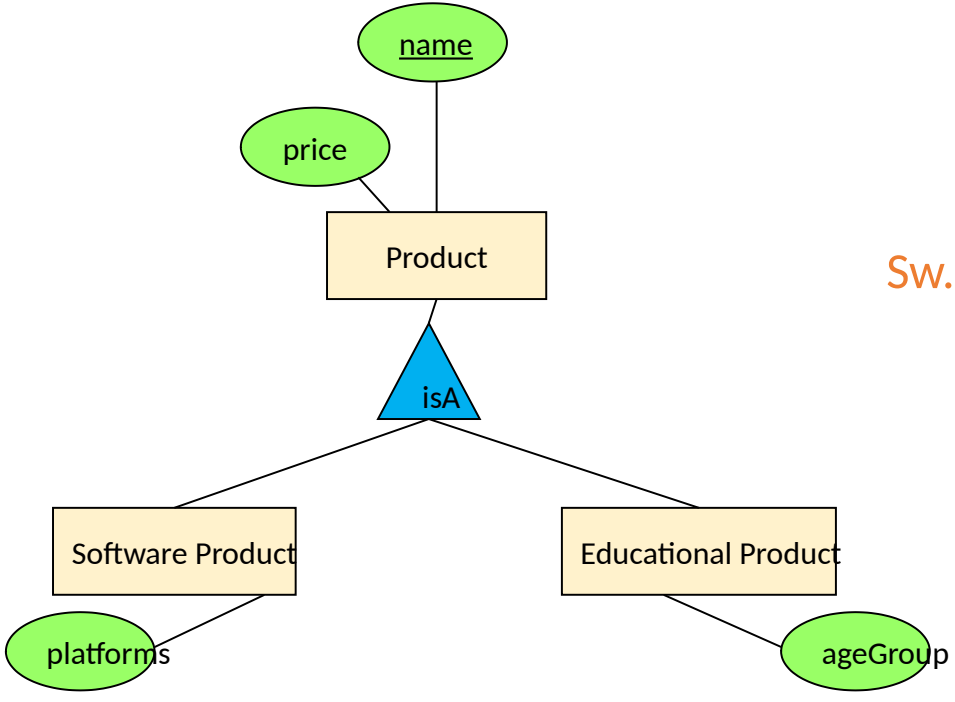
name
price
platforms

- EducationalProduct

name
price
ageGroup

Child subclasses contain all the attributes of *all* of their parent classes **plus** the new attributes shown attached to them in the ER diagram

Think like tables...



Product

name	price	category
Gizmo	99	gadget
Camera	49	photo
Toy	39	gadget

Sw.Product

name	platforms
Gizmo	unix

Ed.Product

name	ageGroup
Gizmo	todler
Toy	retired

IsA Review

- If we declare ***A IsA B*** then every **A** is a **B**
- We use IsA to
  - Add descriptive attributes to a subclass
  - To identify entities that participate in a relationship
- **No need for multiple inheritance**

Modeling UnionTypes With Subclasses

Person

FurniturePiece

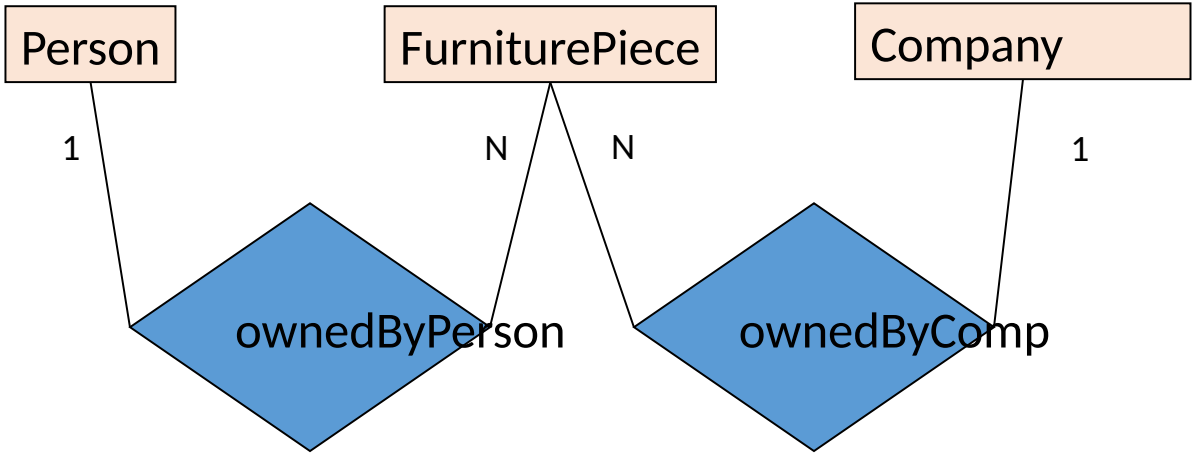
Company

Suppose each piece of furniture is owned either by a person, or by a company. *How do we represent this?*

Modeling Union Types with Subclasses

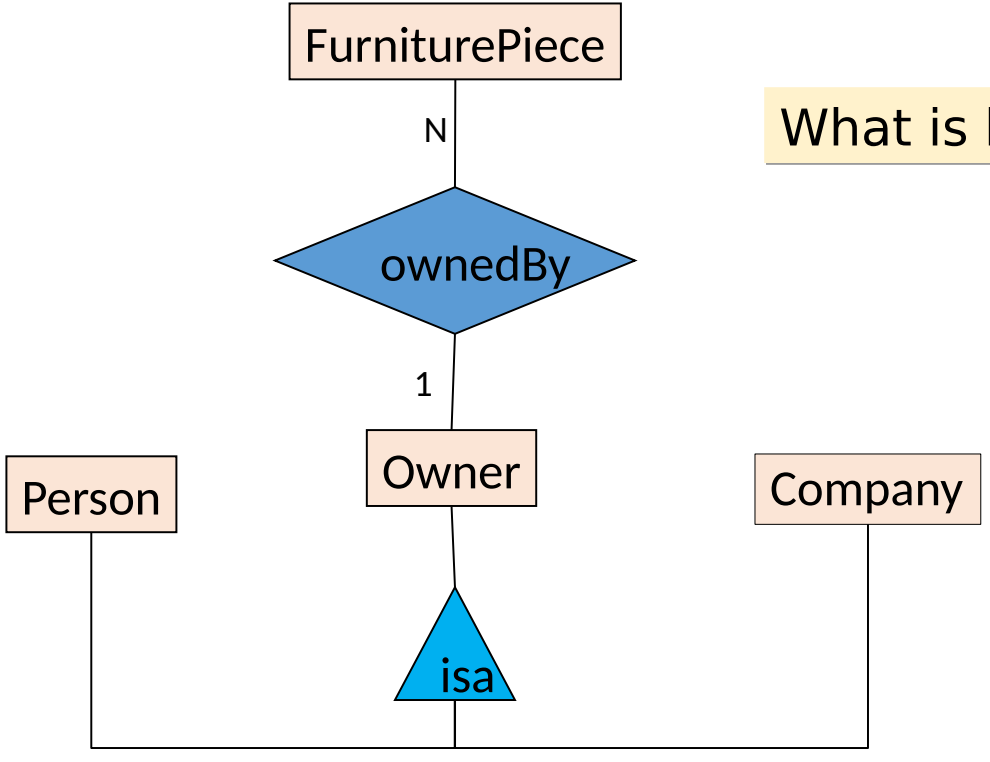
Say: each piece of furniture is owned either by a person, or by a company

Solution 1. Acceptable, but imperfect (What's wrong ?)



Modeling Union Types with Subclasses

Solution 2: better (though more laborious)



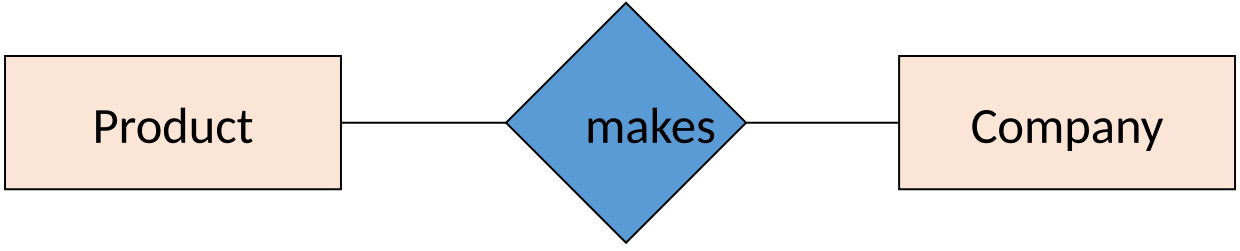
What is happening here?

Constraints in ER Diagrams

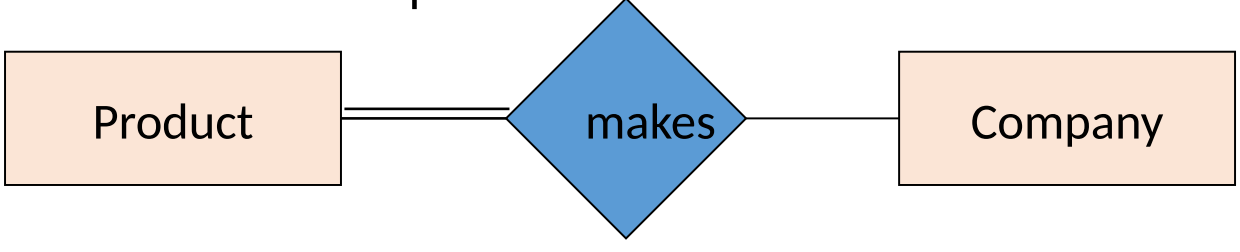
- Finding constraints is part of the ER modeling process. Commonly used constraints are:
  - Keys: Implicit constraints on uniqueness of entities
    - *Ex: An SSN uniquely identifies a person*
  - Single-value constraints:
    - *Ex: a person can have only one father*
  - Referential integrity constraints: Referenced entities must exist
    - *Ex: if you work for a company, it must exist in the database*
  - Other constraints:
    - *Ex: peoples' ages are between 0 and 150*

Recall FOREIGN KEYs!

Participation Constraints: Partial v. Total

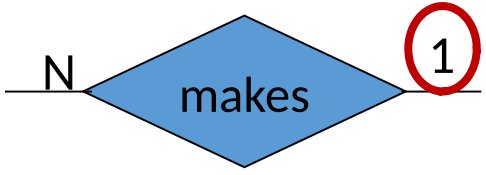


Are there products made by no company? Companies that don't make a product?

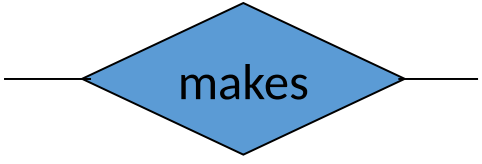


Double line indicates *total participation* (i.e. here: all products are made by a company)

Single Value Constraints



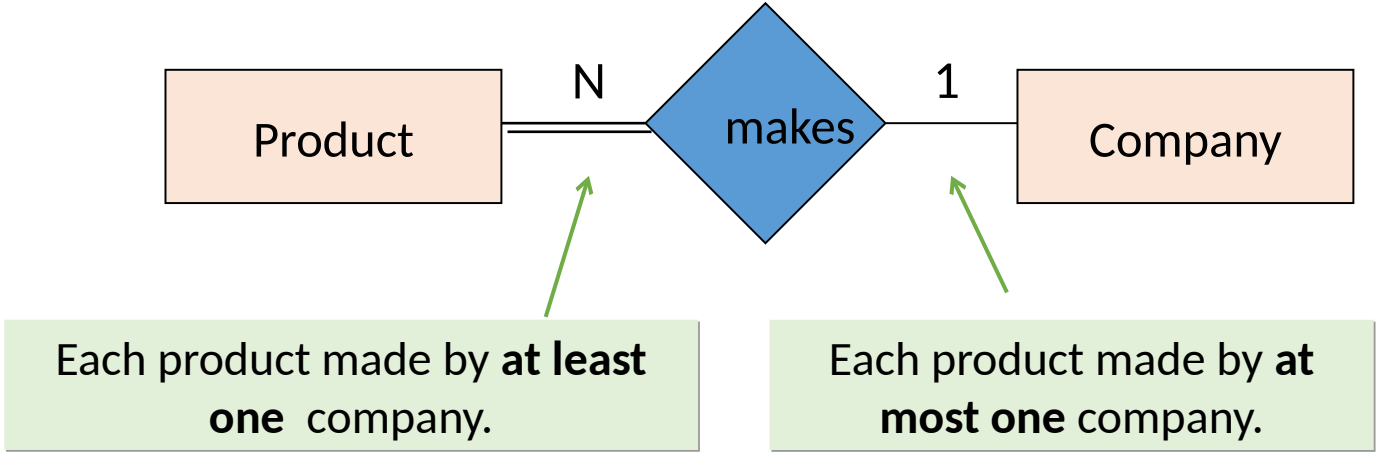
vs.



Referential Integrity

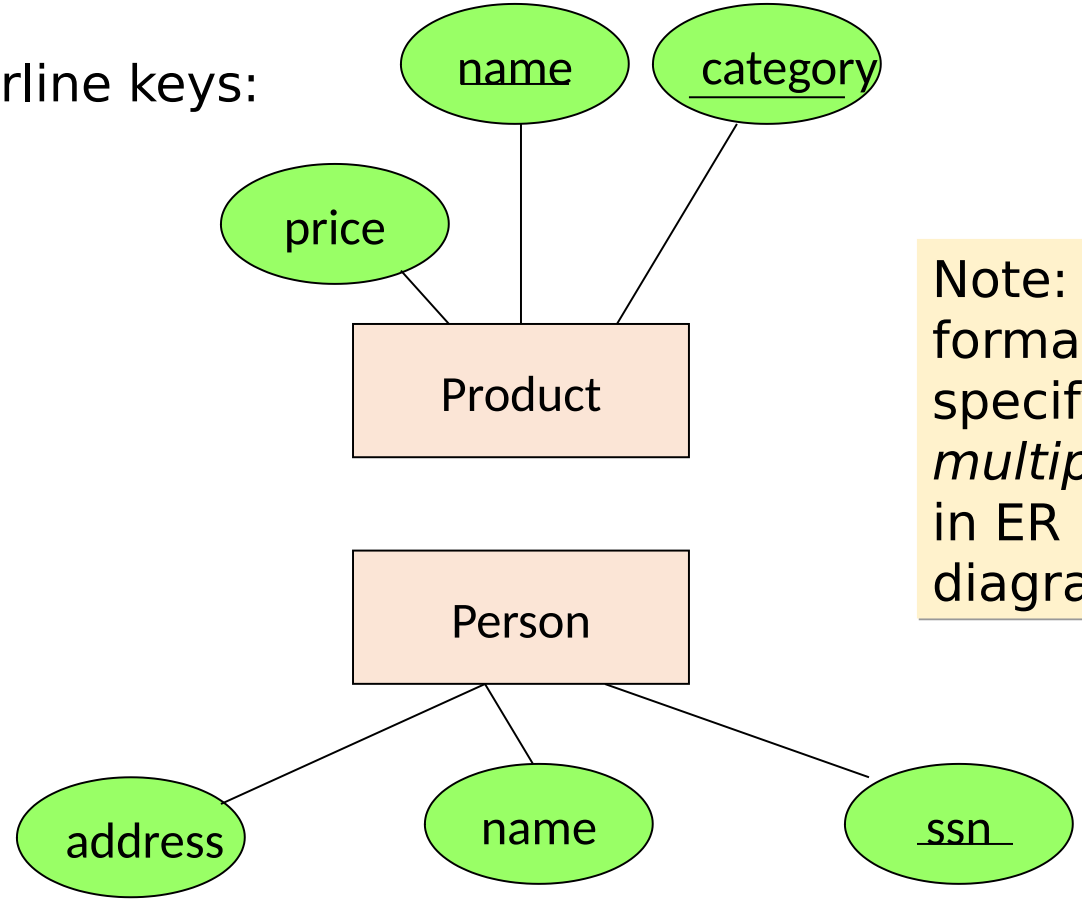
Express that each product is made by exactly one company:

Total Participation + Multiplicity



Keys in ER Diagrams

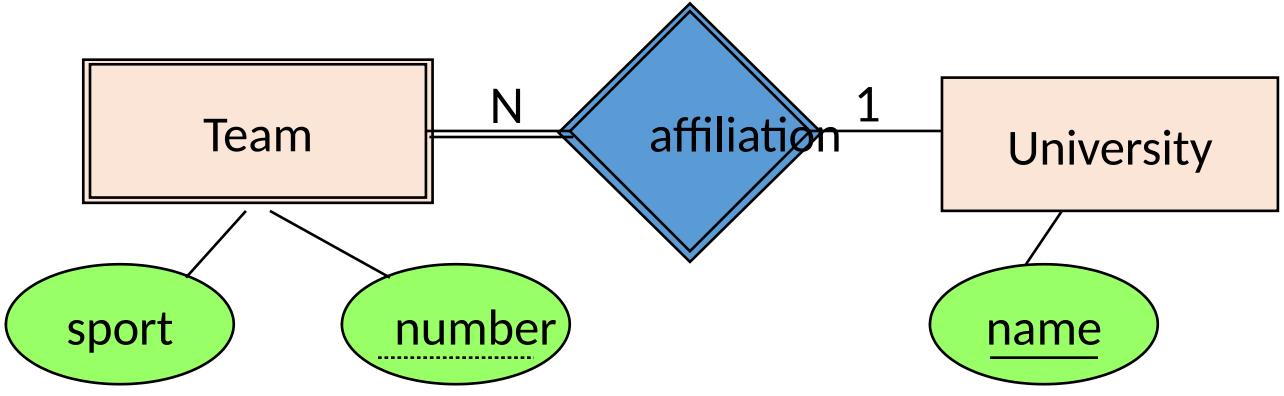
Underline keys:



Note: no formal way to specify *multiple* keys in ER diagrams…

Weak Entity Sets

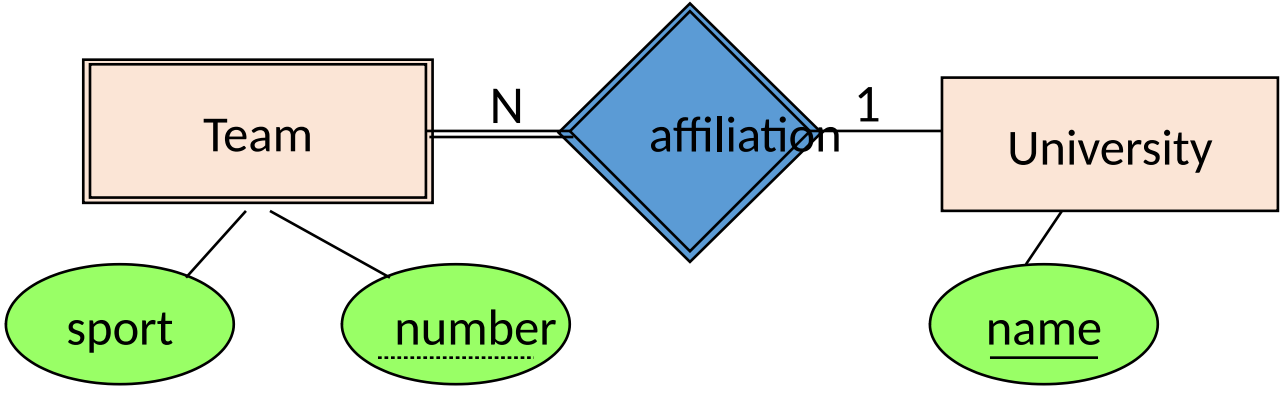
Entity sets are *weak* when their key comes from other classes to which they are related.



"Football team" v. "***The Stanford*** Football team"
(E.g., Berkeley has a football team too, sort of)

Weak Entity Sets

Entity sets are *weak* when their key comes from other classes to which they are related.



- number is a *partial key*. (denote with dashed underline).
- University is called the *identifying owner*.
- Participation in affiliation must be total. Why?

ER Summary

- ER diagrams are a visual syntax that allows technical and non-technical people to talk
  - For conceptual design

- Basic constructs: **entity**, **relationship**, and **attributes**

- A good design is faithful to the constraints of the application, but not overzealous

Design Theory

- Design theory 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 notebook implementing all algorithms!*
  - *We'll play with it in the activities!*

Normal Forms

- 1st Normal Form (1NF) = All tables are flat

- *2nd Normal Form = disused*

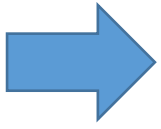
- **Boyce-Codd Normal Form (BCNF)**
- **3rd Normal Form (3NF)**

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

- *4th and 5th Normal Forms*

1st Normal Form (1NF)

Student	Courses
Mary	{CS145,CS229}
Joe	{CS145,CS106}
…	…



Student	Courses
Mary	CS145
Mary	CS229
Joe	CS145
Joe	CS106

In 1st NF

1NF: Entries have to be single values not a list!

Issues with 1NF

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

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

Issues with 1NF

Student	Course	Room
Mary	CS145	B01
Joe	CS145	C12
Sam	CS145	B01
..	..	..

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

Issues with 1NF

Student	Course	Room
..	..	..

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

Issues with 1NF

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

... CS229 C12

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

Issues with 1NF: Split Table

Student	Course
Mary	CS145
Joe	CS145
Sam	CS145
..	..

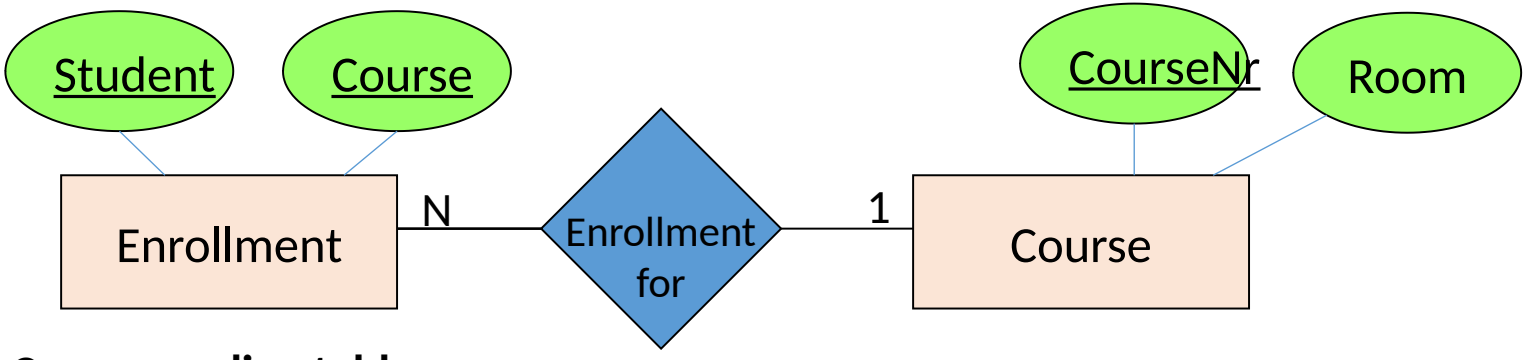
Course	Room
CS145	B01
CS229	C12

Is this form better?

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

How do we decide how to split the table?

ER Model (if done right) already creates 3NF



Corresponding tables:

Enrollment

Student	Course
Mary	CS145
Joe	CS145
Sam	CS145
..	..

Course

CourseNr	Room
CS145	B01
CS229	C12

3NF

- Each attribute needs to rely on the complete primary key (= 2NF).
- All attributes only directly depend on the primary key (no transitive dependencies).

Student	Course	Room	Grade
Mary	CS145	B01	NULL
Joe	CS145	B01	A
Sam	CS145	B01	NULL
..	..	..	

Student	Course	Grade
Mary	CS145	NULL
Joe	CS145	A
Sam	CS145	NULL
..	..	

CourseNr	Room
CS145	B01
CS229	C12

In short: The data depends on the key [1NF], the whole key [2NF] and nothing but the key [3NF]

Conclusion

- Databases should be in 3. NF to avoid anomalies.
- Some databases are poorly designed leading to bad data quality as a result of anomalies.
- Sometimes databases are on purpose not designed in 3. NF. E.g., for performance or other reasons. In this case, the application/user needs to prevent anomalies (which does not always work).
- For analytics purposes we often need a single table with all the data. Essentially this table is in 1. NF