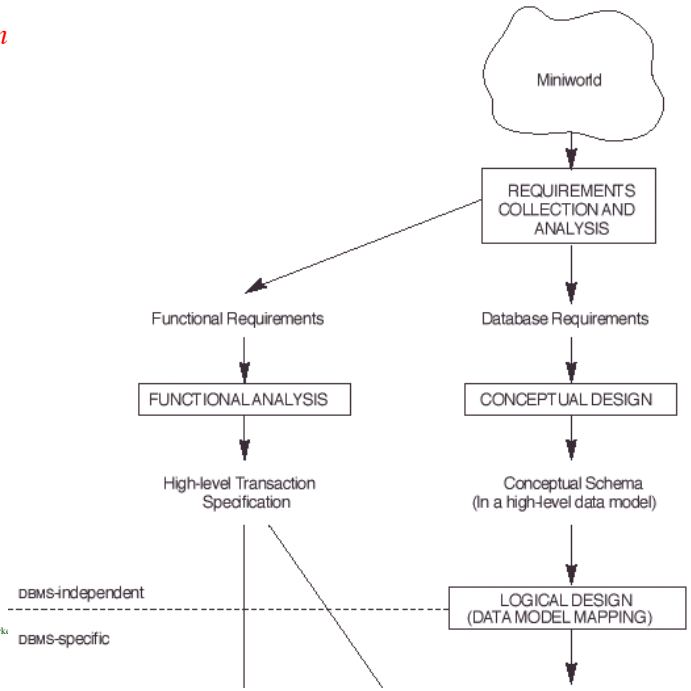
The Entity-Relationship Model

Relational Database Design

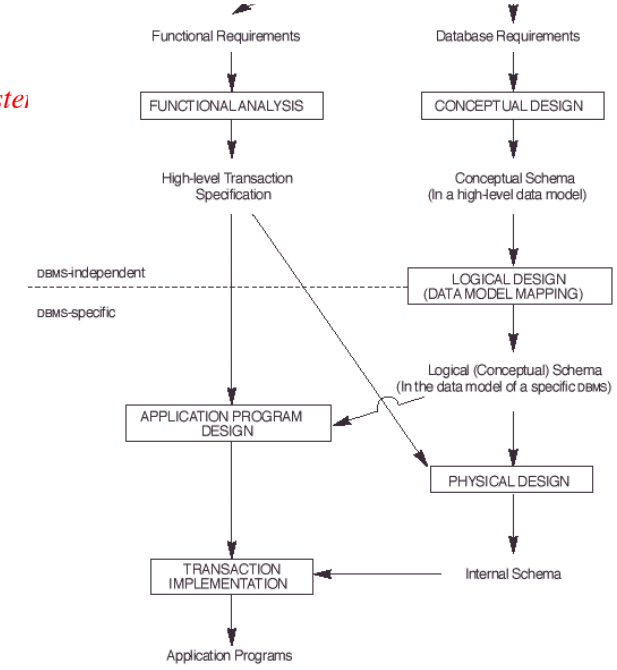
- Need to decide
  - **What are relations**
  - **What are columns/attributes in relations**
  - **What *integrity constraints* or *business rules* hold?**

How do we go about doing this?

Phases of Information design



Phases of Information Syste design



Approach: conceptual modeling

Principle: Information system manages a *MODEL* of some part of the world (Universe of Discourse). As with all models, (e.g., model airplanes), this makes it easier for users to get answers by inspecting the model, rather than going into the "real world".

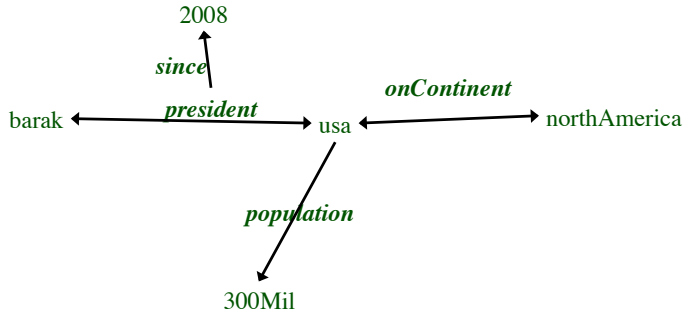
Need languages suitable for describing *human conceptual models*.

Relational databases/XML provide languages more suitable to describe information *structure in the computer*.

Entity Relationship Data Model [Chen 76]: a (weak) *graphical notation* for describing conceptual models; virtue: has easy mapping to relational schemas. ("UML" is a better notation, which is widely used in Object Oriented Software Development.)

What is our conceptual world made of?

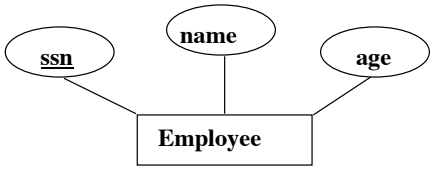
- Objects/entities and values (which have an intrinsic identity)
- Related to each other by relationships
- Described by properties/attributes



Example: Company

- The company is organized into departments. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager.
- Each department *controls* a number of projects. Each project has a name, number and a budget.
- For each employee we keep a social security number, name, age, address, salary, sex, and birthdate. Each employee *works in* one department but may *work on* several projects. We keep track of the time when the employee started working for the department. We also keep track of the *direct supervisor* of each employee.
- Each employee may *have* a number of dependents. For each dependent, we keep track of their name, sex, birthdate, and relationship to employee.

ER Model Basics



- *"Entity Set"*: A collection of similar individuals. E.g., all employees. (Like a *class* in ObjOriented programming language)
- *"Attribute"*: a simple value associated with the entity, usually describing it
  - **All entities in an entity set have the same set of attributes.**
- Each attribute has a "*domain* "
  - **base (value) type: int, float,... string, date,**
  - ***<< some versions of ER (not us) allow composite attributes: name[first,last] address[phone[area_code,local_phone],...]***
  - ***<< some versions of ER (not us) allow multi-valued attributes (e.g., color for Car***

ER Model Basics : Relationship (set)

- *Relationship set*: Set of specific relationships/**tuples** between individuals in the entity sets linked to it. Can think of the above as having one possible instance described as

- a mathematical relation with tuples of individuals

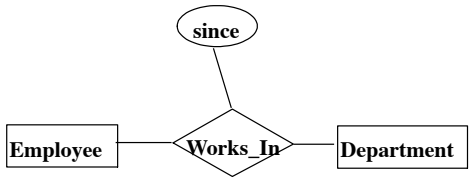
  WorksIn={ (emp1,dep3), (emp1,dep4), (emp7,dep3) }

- a table

WorksIn

emp1	dep3
emp1	dep4
emp7	dep3

ER Model Basics (Contd.)

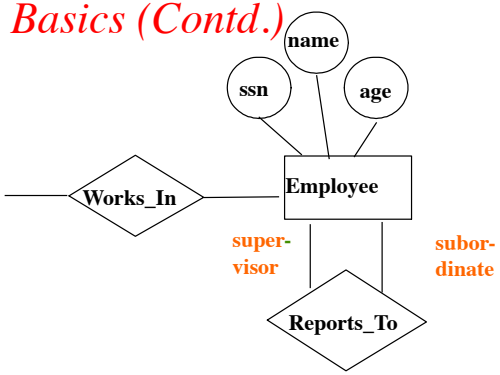


- *Relationship set attributes*: qualify nature of relationship (e.g. time)

  ***since*(12/3/2002,*worksIn*(mary,sales))**

  but remember that attributes have to have "atomic values" (int, string date,...); cannot be entities

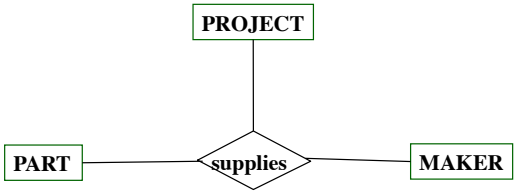
ER Model Basics (Contd.)



- Same entity set could participate in different relationship set, or in multiple ways in the same relationship set.
  - **To distinguish the participants, use "role names" as in *Reports_To***

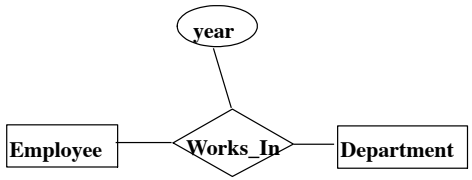
n-ary Relationships

"Makers supply parts to projects"
For this, we need a *ternary* relationship set.



Again, an instance of this can be viewed as a ternary relationship, one with 3-tuples
supplies = { (part1,proj45,Ford) , (part5,proj45,GE) }

BEWARE!

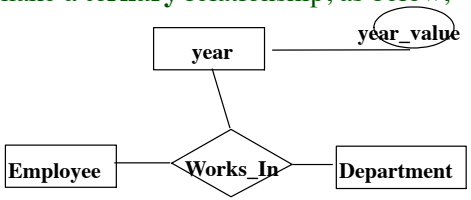


- This **binary** relation only allows one pair (emp1,dep2) because it is a set – ie., an employee can only work in a department once
- If you wanted to have an employee work in the same department in different years, you need to make a **ternary** relationship, as below; now you can have *both*

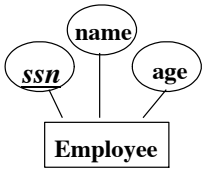
(emp1,dep2,2014)

and

(emp1,dep2,1985)



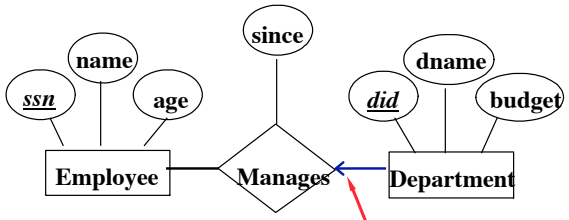
Constraints *(1): "key" attribute(s) for entities*



- An entity key is a minimal set of attributes that identifies every individual in that entity set
- e.g., ***ssn*** uniquely identifies an employee
- if you needed both *ssn* and *name* to uniquely identify an employee, they would form a *composite* key
- Note that the set of all attributes must always uniquely identify the entity but it may not be minimal, hence not a key
- ***notation: underscore the attribute name(s) forming the key***

Constraints (2): "functional participation"

- suppose each department has at most one manager So a department can participate in at most Manages pair, as second argument.



Manages

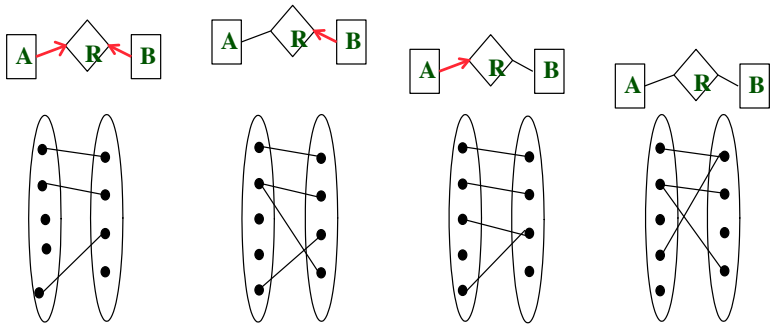
emp1	dep3
emp1	dep4
emp7	dep3

this tuple would not be allowed!

Notation: arrow head here

(According to this diagram can an Employee manage more than one Dept? Yes)

Possible "functional constraints" on binary relationship in pictures, with individuals



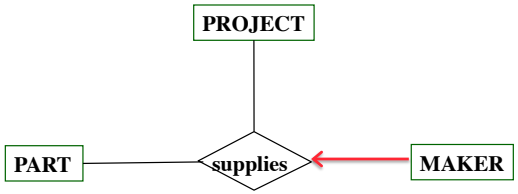
Each A and B instance appears at most once in an R relation/edge

Each B instance appears at most once in an R relation/edge

Each A instance appears at most once in an R relation/edge

No constraint

n-ary relationship with functional constraints



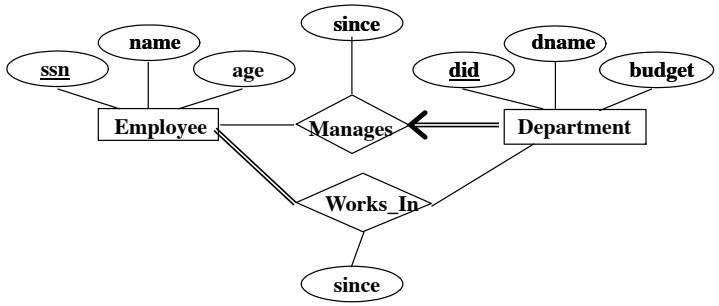
This says that each maker appears in only one (proj,part,maker) tuple. So it can only supply one part, and that is supplied to one project only.

The book notation does not allow a way of saying that a maker supplies only one (possibly different) part to each project. This needs to be stated in English.

Constraints (3): participation

Does every employee work somewhere? (Is every Employee entity involved in some Works_In relationship?)

If so, this is a *participation constraint*: the participation of Employee in *Works_In* (vs. Employee in *Manages*) is said to be *total* (vs. *partial*). **[*double edge* notation in my notes; thick edge notation in book].** Arrow can be dble, to indicate both total and functional



More general participation cardinality constraints

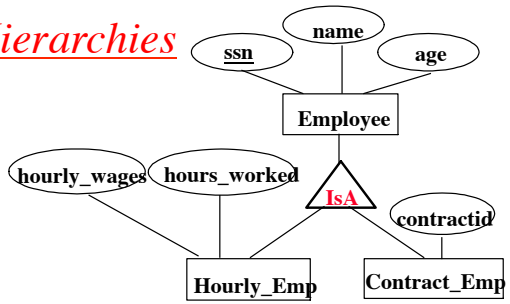
• "Every department participates in at least 5 and at most 40 **worksIn** relationships -- (emp, dept) pairs

• "Every employee participates in at least 1 and at most 1 **worksIn** relationship pair."

Summary of relationship constraint notation

- In a diagram for relationship R, each entity E is considered separately, ignoring the other entities participating in the relationship. We record what constraints there are on any *instance* of E, in terms of how many specific relationship tuples it can participate in.
- The choices are
  - **no constraints**
    *(notation: **simple edge**; 0..* in cardinality notation)*
  - **at least one relationship for each instance**
    *(notation: **thick edge**; 1..* in cardinality notation)*
  - **at most one relationship for each instance**
    *(notation: **edge with arrow**; 0..1 in cardinality notation)*
  - **exactly one relationship for each instance**
    *(notation: **thick edge with arrow**; 1..1 in cardinality notation)*

IsA (subclass) Hierarchies

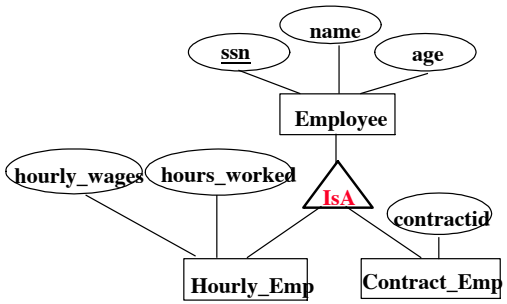


❖D **IsA** C *test*: every D entity set instance is also a C entity set instance. (Therefore the number of D instances must be less than or equal to the number of C instances)

❖ Attributes are *inherited*. (So are relationships)

❖ Superclass specifies the key, which is inherited

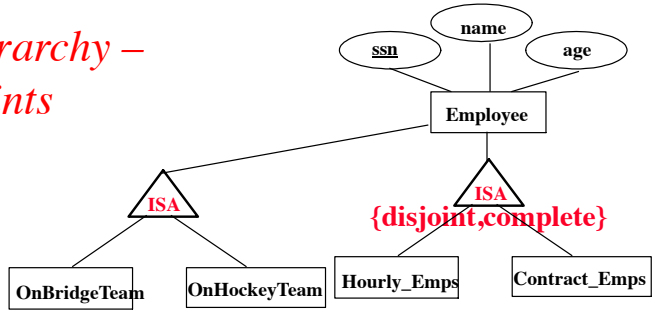
IsA Hierarchies



❖ When to use use **IsA:**

1. **To factor out commonalities and avoid repetition:** eg, *age*
2. **To more precisely identify subset of entities that are domains/co-domains of a relationship** (*Contract_Emps* participate in *Manages* reln, but *Hourly_Emps* do not participate at all) **or have specific attributes** (*contractId*)
3. **To organize large models and the process of creating/ modifying models** (*inheritance of changes*)

ISA Hierarchy – constraints

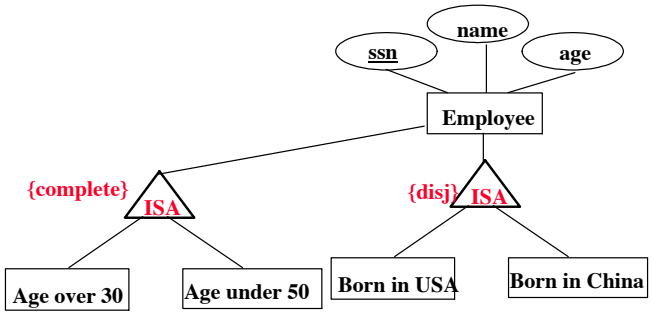


Constraints on IS-A hierarchies:

❖ *Overlap constraint*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? ***disjoint*** *(vs. overlapping)*

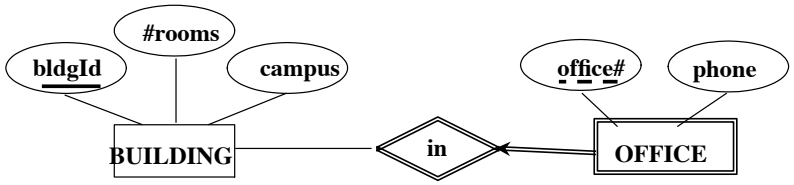
❖ *Covering constraint:* : Does every Employee entity also have to be an Hourly_Emps or a Contract_Emps entity? ***complete*** *(vs. partial)*

ISA Hierarchy constraints (cont'd)



Weak Entities

- A *weak entity* does not have enough attributes specified to identify it uniquely by their values. It must therefore be identified through its relationship to another (*owner*) entity. *(think of an office with an office number in a building)*
  - **Owner entity set and weak entity set must participate in a *identifying* relationship**
  - **Weak entity set must have total participation in this relationship set, which must be functional (only one owner)**



Notation: weak entity and identif. reln are thick lines

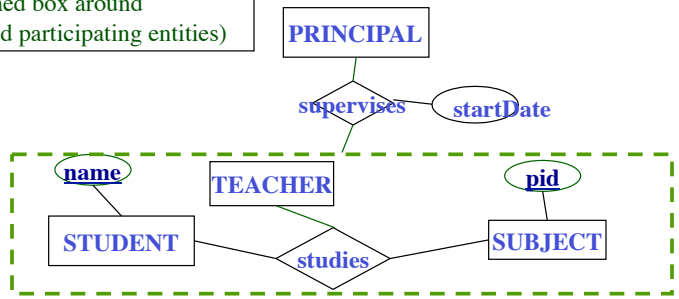
Describing Relationships

- Relationships may need to be "described" by more complex things than simple attributes
  - ***teaches(Teacher,Student,Subject)* is *supervisedBy Principal***
  - do not like the idea of writing a single 4-place relationships ***teachingSupevisedBy(Tchr,Std,Subj,Principal)*** because this is composite!
  - another eg. '**"Anna likes Vronya" was said by Tony**'
- Might want to represent the fact that a relationship is specialized (but ER does not have IS-A for relationship sets)
  - ***sonOf* is a specialization of *childOf* is a specialization of *relativeOf***

Describing Relationships in ER: "Aggregation"

- Relationships may need to be "described" by more complex things than simple attributes
  - **`teaches(Teacher,Student,Subject)` is `supervisedBy Principal`**

Aggregation allows us to treat a relationship set as an entity set. (*notation*: dashed box around relationship and participating entities)



Subtleties in EER modeling

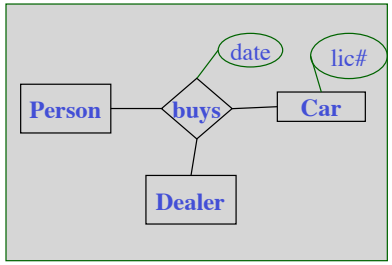
- These are commonly occurring modeling situations for which there is no special notation in EER. Just alternative ways of creating entities and relationships.

1. Reifiying ("making real") relationships

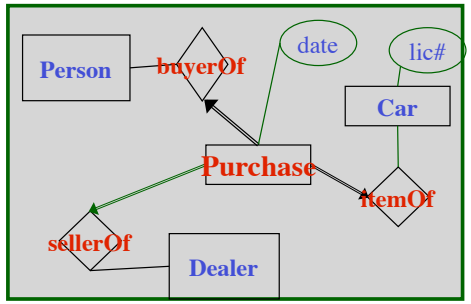
- take a relationship R and think of it as an entity, linked by *simple functional binary relationships* to the *participants* in R.
- Use the role names to label these binary relationships

*e.g., **1** is reified to **2** below. BUT, beware that **2**. allows duplicate purchases not allowed by **1**.* (purchA and purchB both linked to the same person, car and dealer instance.)

1



2

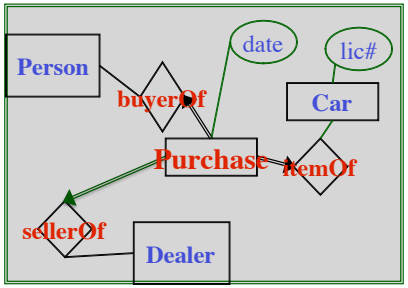


Reification (cont'd)

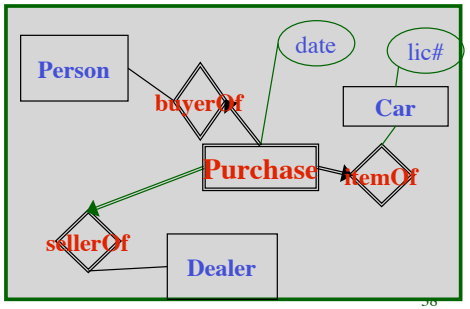
- Problem A: (2) allows duplicate purchases, as noted on the previous page
- Problem B: entity Purchase does not have a key – illegal in ER! So you would have to introduce a key attribute for Purchase, e.g., *purchase#*

Solution to both: make Purchase in 2 be a *weak entity* in 3

2

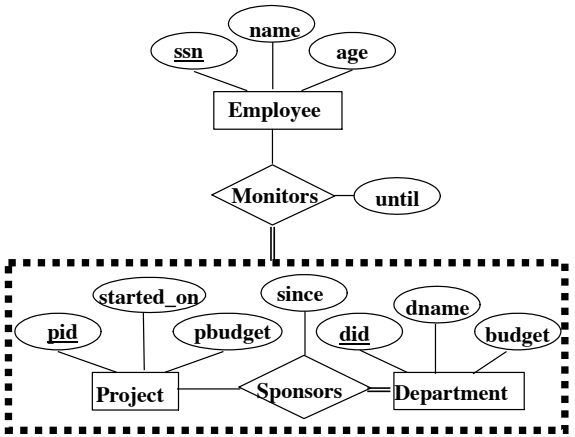


3



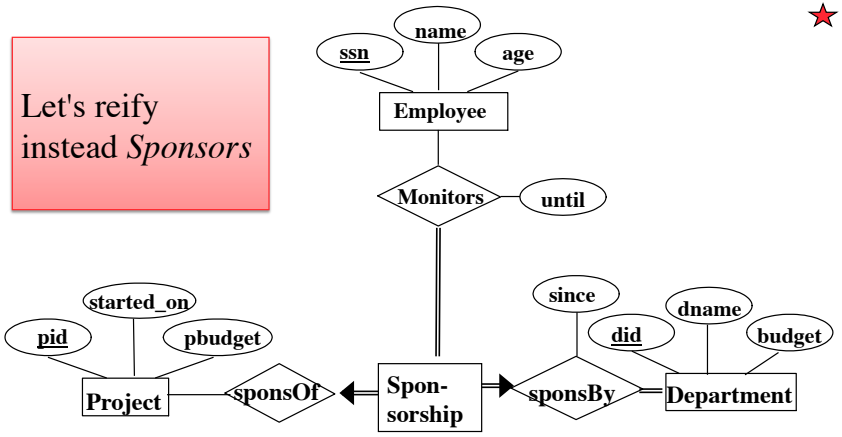
Let's redo the whole process of reification with another example, in this case replacing aggregation by a reification of the relationship.

Start with aggreagation:



Monitors represented using aggregation (as before)

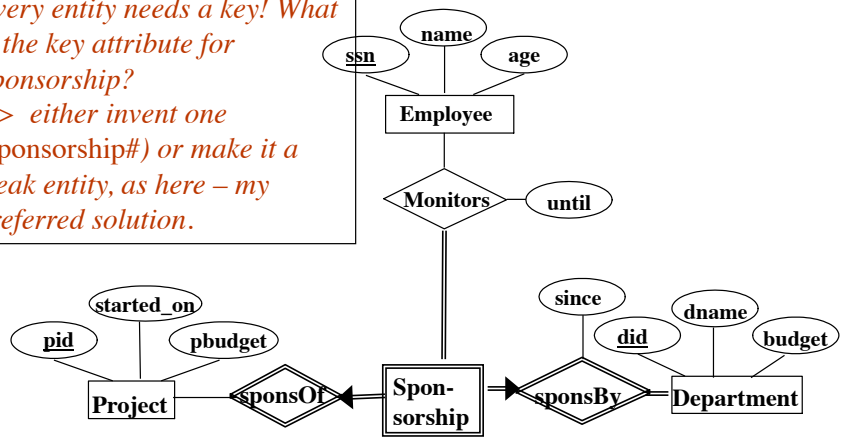
Let's reify instead *Sponsors*



*Step1: make relation "**Sponsors**" (a verb) into an entity "**Sponsorship**" (a noun), and connect **Monitors** to **Sponsorship** directly.*

Step 2 of reification

Every entity needs a key! What is the key attribute for Sponsorship?
>> either invent one (sponsorship#) or make it a weak entity, as here – my preferred solution.



*The "key" of **Sponsorship** is (pid,did) – the combination of the keys of the entities the weak entity depends on for identification.*

Another example of Reifying Relationships

Instead of Lend(Library,Material,ToPatron,...)

Loan **with relationships**

--<lentTo>-- Patron
--<onLoan>-- Material
--<lentOn>-- Date
--<dueOn>-- Date

BENEFITS: This allows you to declare *specialized relations like "lending for short time":*

ShortermLoan IsA Loan

(How would you model hasParent, hasMother with reification?)

2 Manifestation

Another special kind of relationship, which often causes problems, even in textbooks. (No special notation for it in ER!)

A case of one entity being *more abstract* than another

- a **book** is a more abstract notion than a **book copy** that you can hold in your hand (there are multiple copies of a book)
- a **play** ("Hamlet") is a more abstract notion than a **staging of the play (by a director),** which in turn is more abstract than a **particular performance of it (on March 25)**
- a **movie** ("LOTR3") is a more abstract notion than a particular **showing of it** (on 2/6/2016 at 8pm, at Hadfield Cinema)
- a **course** (336 described in the catalogue) is more abstract than my particular **offering** of it in Spring 2017
- a **car model** (Tesla I) is more abstract than a **particular vehicle** of that model, which is parked in my garage (I wish)

Manifestation (cont'd)

But in all cases the more abstract kinds of things share ***some*** properties with more concrete ones:

- a **book** is a more abstract notion than a **book copy** that you can hold in your hand (there are multiple copies of a book) — author, title
- a **play** ("Hamlet") is a more abstract notion than a **staging of the play (by a director),** which in turn is more abstract than a **particular performance of it (on March 25)** — author, title
- a **movie** ("LOTR3") is a more abstract notion than a particular **showing of it** (on 2/6/2016 at 8pm, at Hadfield Cinema) — title, stars, director
- a **course** (e.g. 336), described in the catalogue, is more abstract than my particular **offering** of it in Spring 2017 — crs name, csr# #credits
- a **car model** (Tesla I) is more abstract than a **particular vehicle** of that model, which is parked in my garage (I wish) — maker, name, year

A Manifestation example in detail:

Book vs BookEdition vs BookCopy

You cannot buy the first two kinds! They have some common but also different attributes!

/* Books have information about authors, etc. */
Book hasAuthors
hasTitle
/* Editions of books are related to the book ("described by it") but have their own properties/relationships too */
BookEdition: *describedBy* Book ... AND
publishedBy PublishingCompany
hasIsbnNr Number
hasFormat { 'printed , 'audio}
/* Book copies are related to book editions, and in turn have their own properties */
BookCopy: *describedBy* BookEdition
hasCallNr Number
hasCopyNr Number
locatedAtBranch LibraryBranch

A Manifestation example in detail (cont'd):

Book vs BookEdition vs BookCopy

Each may have attributes that do not make sense for the other:

/* Books have information about the manuscript and circumstances of writing
Book
#_of_pages_of_manuscript
written_in_cities
/* Editions of books
BookEdition: *describedBy* Book ... AND
#_copies_printed
/* Book copies
BookCopy: *describedBy* BookEdition
torn_page

Manifestation: *How NOT to represent it in ER*

Even though you might want BookEdition to "inherit" properties of Book (e.g., title) you should almost never make

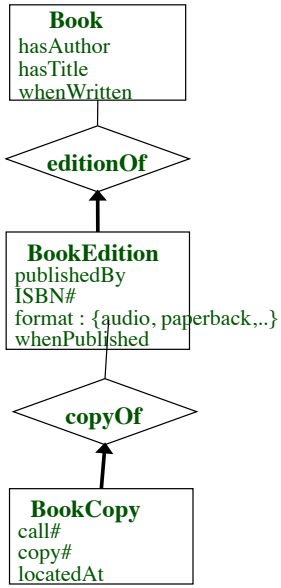
- **BookCopy** IsA **BookEdition** IsA **Book** **(a common mistake!)**
  - **Note that (BookEdition *subclass_of* Book) does not pass the IsA test: not every instance of BookEdition is an instance of Book (do a count test). Same for BookCopy and BookEdition**
- One should **NOT** make book copies be ***instanceOf* Book**
  - **when you sell the last copy of some book, you will have no record of the book itself ever having existed (never heard of something titled "Harry Poter and the Chamber of Secrets" by J.K.Rowling). So you won't be able to talk about the book being out of print (or not having any first editions of the book)**
- **nor** should you create open ended sets of classes
  - **LOTR1** isa **Movie,**
  - **HarryPotter&ChamberOfSecrets** isa **Movie**
  - **...**

Manifestation: ***Solutions***

- ***Approach 1*:*** if you truly don't care about Book, only editions and copies, "merge" the information about Book into BookEdition. (Potential problems: some attributes, such as "dateWhenPrinted" mean different things for Book and BookEdition.)
- ***Approach 2:*** relate the descriptor to the described with an explicitly named relationships **(editionOf , copyOf,descrOf);**

/* Book editions are manifestations of books */
BookEdition ---<editionOf >---- Book
/* Book copies are manifestations of book editions */
BookCopy ---<copyOf>--- BookEdition

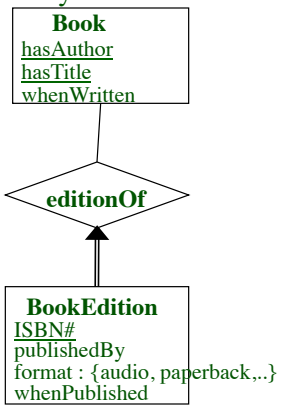
To "inherit" properties, state constraints
(hasTitle *of* **BookEdition** instance) *is the same as* **(hasTitle** *of* **isCopyOf** *of* **BookEdition** instance)

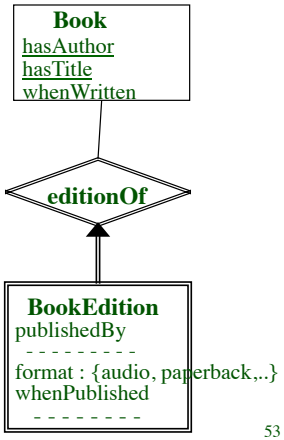


- Note that if this was not a library, just a book store, it would not make much sense to make a separate BookCopy entity, since they could not be distinguished.
- Better to merge BookEdition and Book, and add a *#ofCopiesInStock* attribute

Manifestation and keys

ER notation requires keys for every entity. In this case (left) we have no problem (key attribs underlined). But if there was no ISBN# then we would have to model it as on the right, as a weak entity:





ER methodology (1)

- Figure out what entities/objects are mentioned
  beware of synonyms and homonyms
  in general, new entity classes should be "useful" (have something to say about them)
- Figure out what relationships between the entities are useful, and their arity (binary, ternary)
  remember to label roles on relationships involving the same entity more than once
- Add attributes describing entities
- Abstract out commonalities into super-classes related by IsA relationship
  add disjoint and/or covers annotations on the set of subclasses
- Add attributes describing relationships
  remember that there can be only one relationship tuple with the same participants
- Specify participation constraints on relationships (*total*: lower bound > 0, *functional*: upper bound=1)

ER methodology (2)

- Find keys for entities
  - **if only partial key exists, see if there are identifying relationships so you can make it a weak entity**
  - **otherwise, you may need to generate internal identifiers (e.g., studentId)**
  - **remember that every IsA hierarchy needs a key at the top, which is inherited to all subclasses**
- If relationships need to participate in other relationships they need to be made aggregates or reified.
- Look out for "manifestations" (Course vs CourseOffering):
  - **did you conflate them, and if so is that ok?**
  - **did you by mistake relate them by IsA or InstanceOf?**
  - **you should relate them by some relationship (at the very least "*manifestationOf*")**
- * If you used partOf relationship
  - **was it transitive?**
  - **if the part was faulty would you say that the whole was faulty?**

ER methodology (3)

- Lookout for redundancy, especially among reltionships.
  Bad idea to model both <husbandOf> and <wifeOf>
- State any other constraints that cannot be expressed in the ER notation (e.g., managers earn more than their subordinates) in English.