E-R Design Issues

Winter 2006-2007

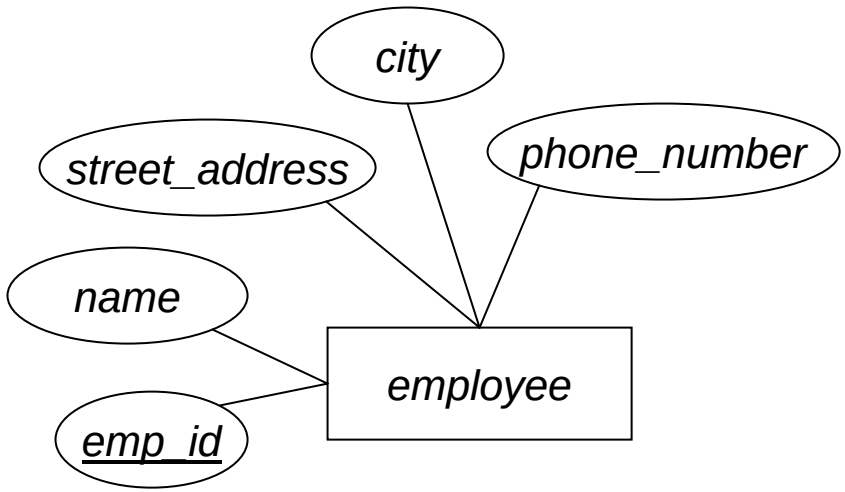
Lecture 16

E-R Model Schemas

- Can represent a particular design in many different ways
  - Different representations not always exactly equivalent
  - Can introduce subtle differences in capability
  - Can affect future extensibility
- Some uses of E-R model are just wrong
  - Abuses of the model

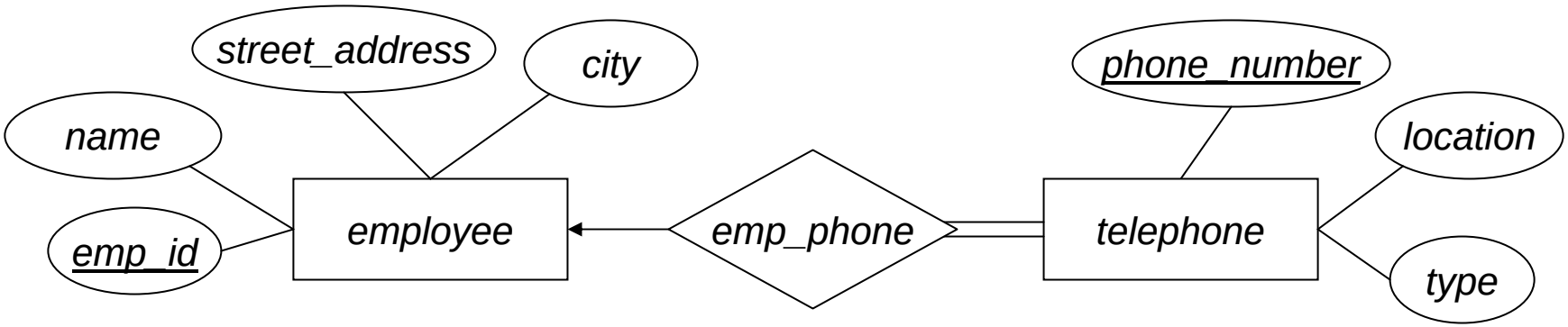
Attributes vs. Entity-Sets

- When to represent a value as an attribute?
- When to represent a value as a separate entity-set?
- Representing as a separate entity-set allows details to be added later
- Example:
  - Phone number is a good candidate for representing as a separate entity-set
  - Employee name definitely should stay an attribute



Attributes vs. Entity-Sets (2)

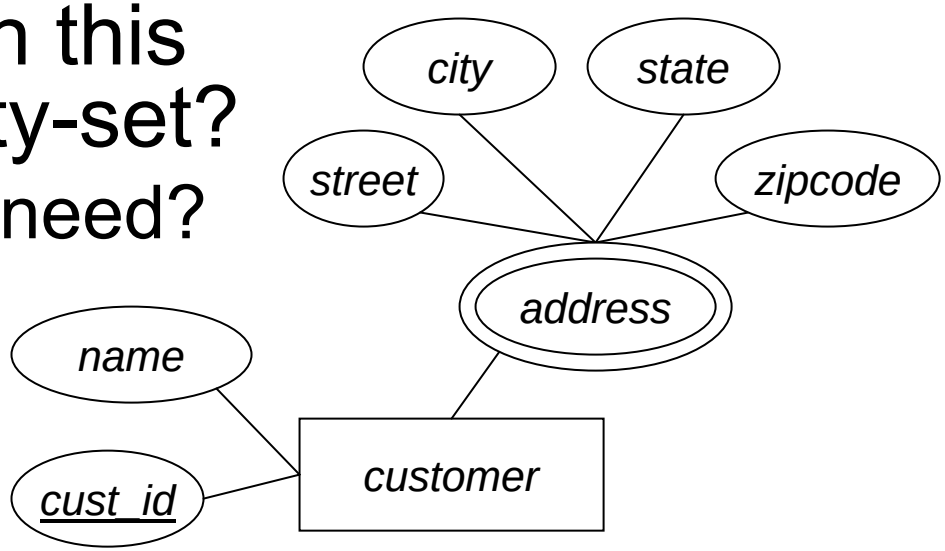
- Could move *phone_number* to another entity-set
  - Add *location*, *type* attributes as well



- Differences:
  - Attribute was single-valued before…
  - Now employees can have multiple phone numbers
  - Need to choose participation, cardinality constraints!
  - Could also just use a multi-valued composite attribute

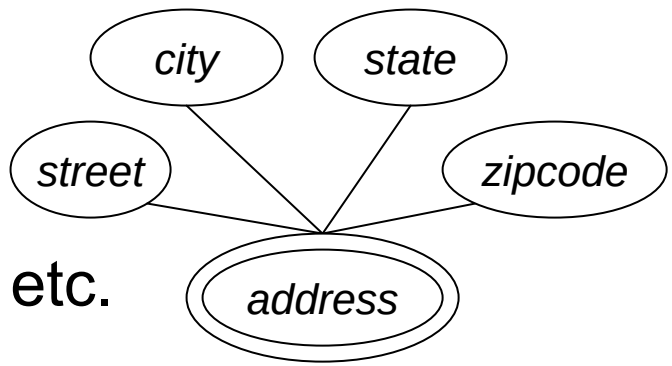
Composite Attributes

- Composite attributes have a big overlap with entity-sets
- Example:
  - Customers can have multiple addresses
  - Make address a composite attribute
- Differences between this and a separate entity-set?
  - What do entity-sets need?



Composite Attributes (2)

- Entity-sets need a primary key, or at least a partial key
- Does composite attribute have a key?
  - If so, should probably migrate to a separate entity-set
  - With this example, entire address is composite key. Not ideal for a separate entity-set.
- If no key attributes, could use a weak entity-set
  - Need to specify a discriminator
  - For customer addresses, could give addresses a name, like “home” or “work”
  - Set up an identifying relationship, etc.

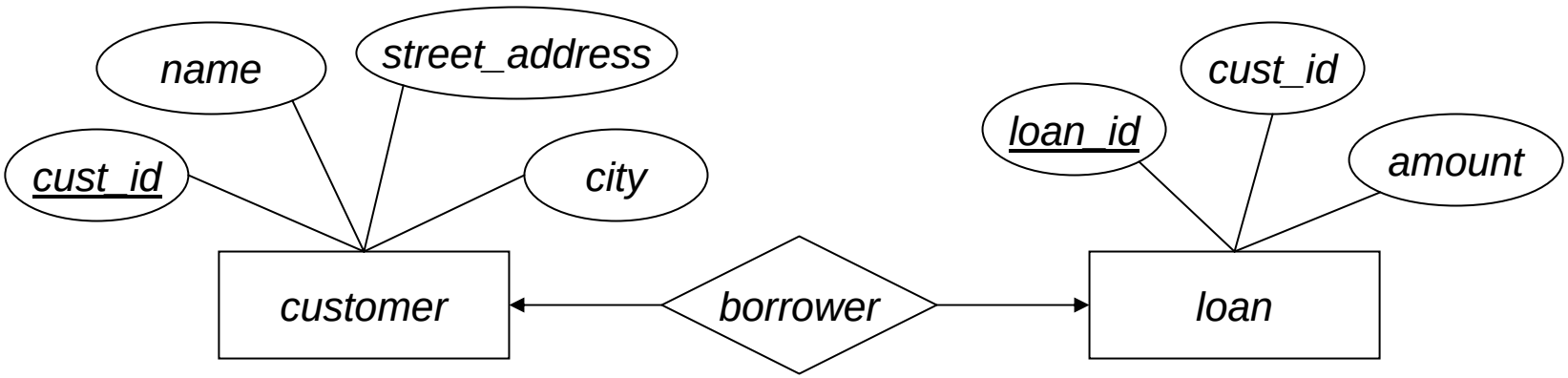


Composite Attributes (3)

- Mapping multi-valued composite attributes to relational schema very similar to mapping for weak entity-sets
- Recommendation:
  - Prefer weak entity-sets over multi-valued composite attributes
  - Most databases don’t support user-defined composite types
  - Using weak entity-sets will more closely model implementation schema

Common Attribute Mistakes

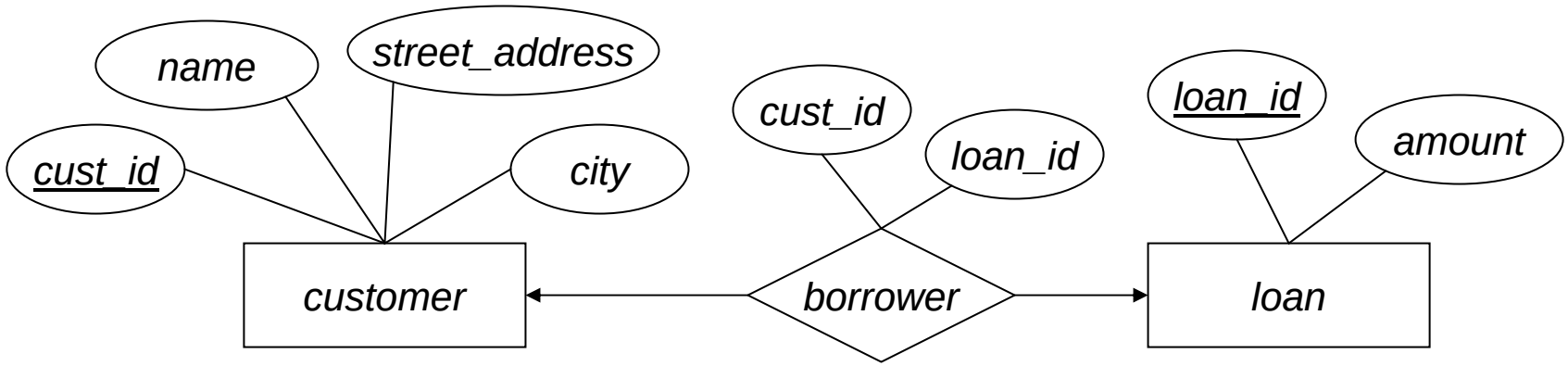
- Don’t include entity-set primary key attributes on other entity-sets
  - e.g. customers and loans



- Even if loans are owned by only one customer, this is still wrong
  - Association is contained by the relationship, so specifying foreign key attributes is redundant

Common Attribute Mistakes (2)

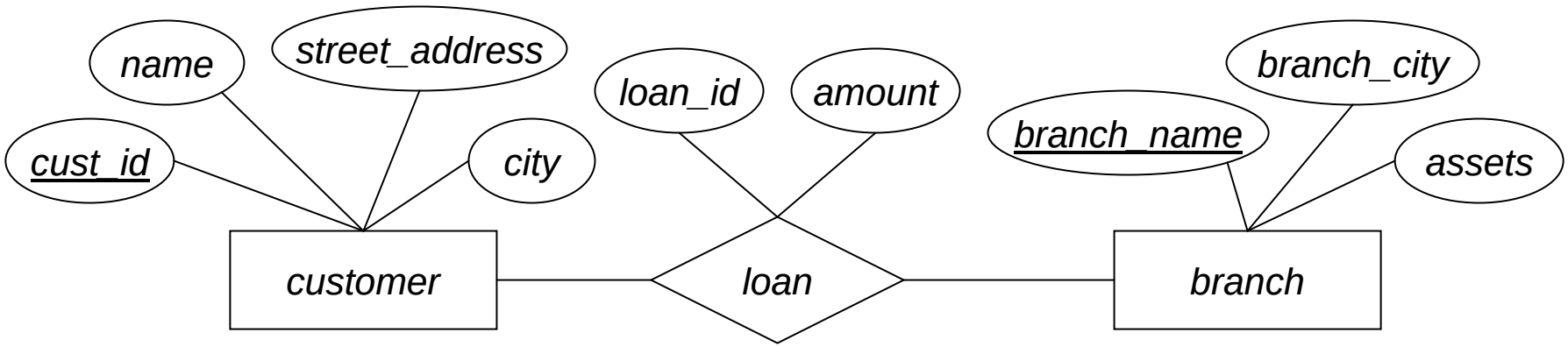
- Don't include primary key attributes as descriptive attributes on relationship-set
- Example: *customer* and *loan* relations again
  - IDs used as descriptive attributes on *borrower*



- Again, this is implicit in the relationship

Entities vs. Relationships

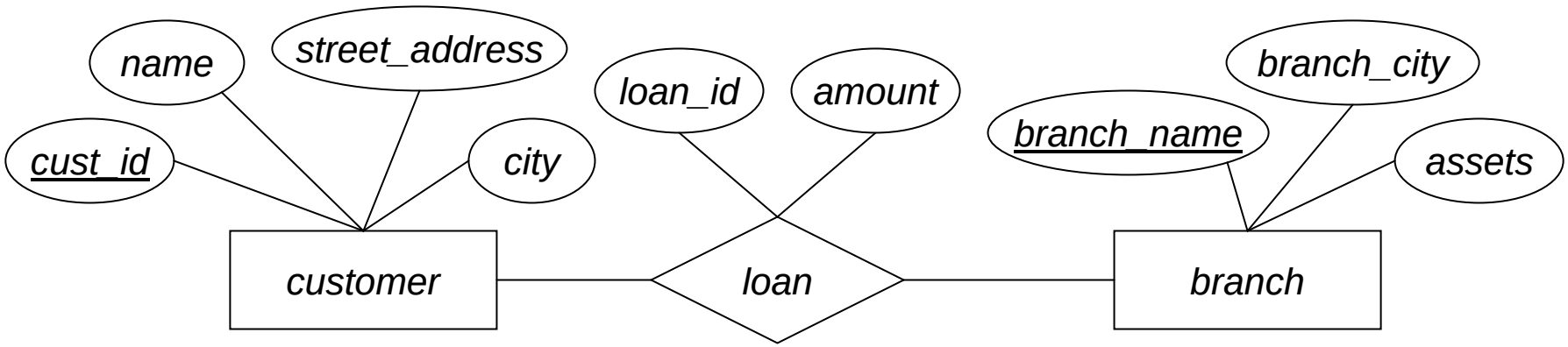
- Want to represent loans given to exactly one customer
  - Each loan is given at a particular bank branch
- What about this:



  - Represent loans as relationships between customers and branches
  - Loan number and amount are descriptive attributes

Entities vs. Relationships (2)

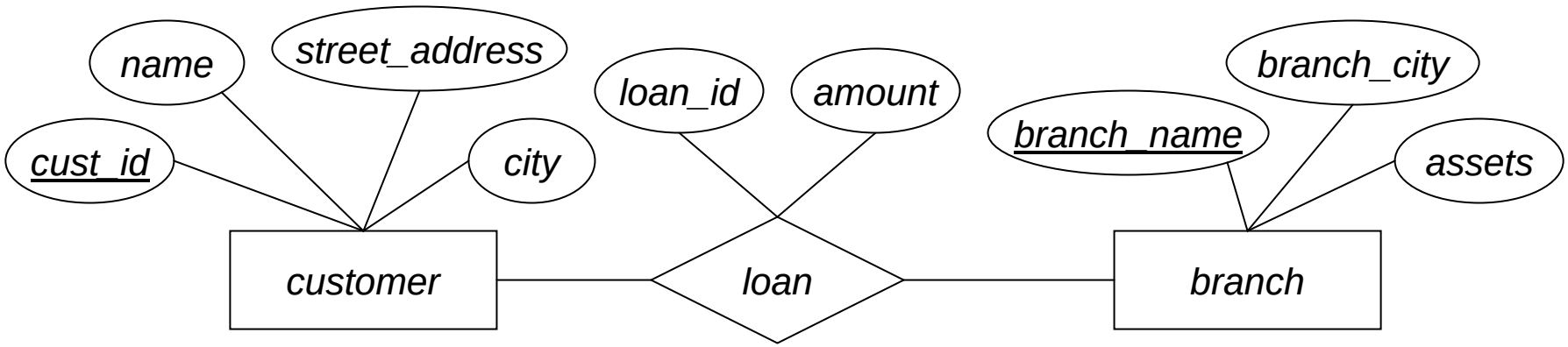
- Would definitely work…



- Problem:
  - How to enforce uniqueness of loan IDs?
  - Can't make keys from descriptive attributes on relationships
  - Using this approach, can't constrain values. Not good

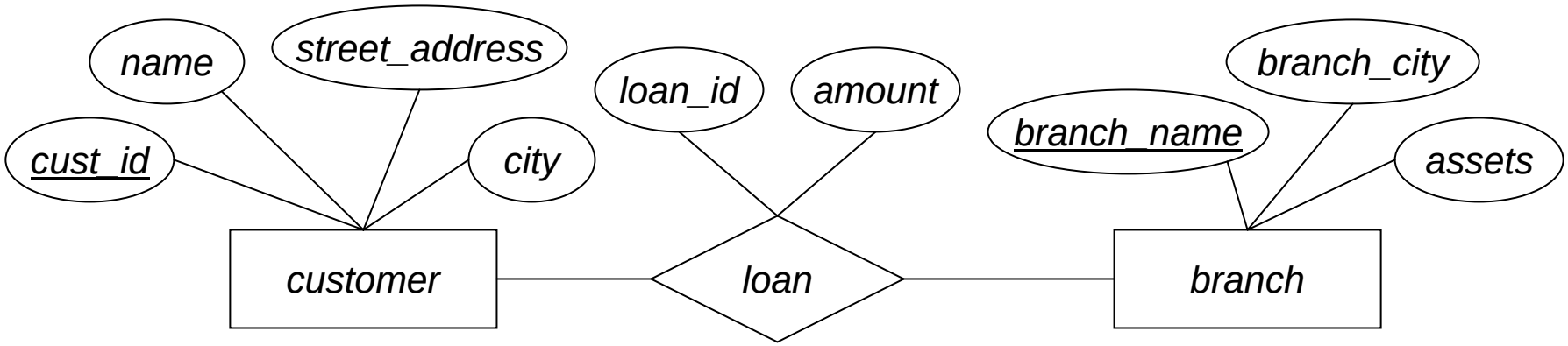
Entities vs. Relationships (3)

- Also constrains us to an unrealistic scenario
  - Normally want to allow joint ownership of loans
- *Could* we model joint ownership with this?



  - *Could* have multiple relationships with same loan ID…

Entities vs. Relationships (4)



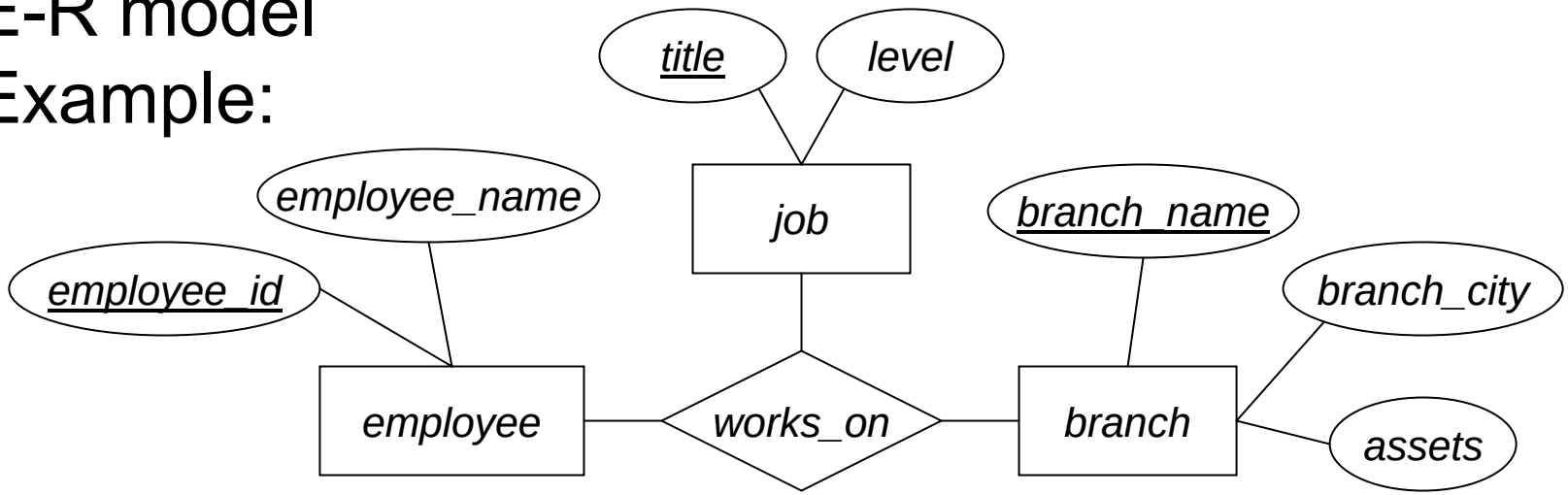
- Modeling multiple-owner loans with this schema causes big problems
  - Same loan number and amount must appear in multiple relationships
  - Wastes storage space
  - Presents serious consistency issues!
- Want to avoid necessary redundancy!

Entities vs. Relationships (5)

- Simple guideline for choosing entities or relationships
  - An entity is a “thing”
  - A relationship is an “action” involving entities
- In general, evaluate schema against various scenarios
  - Watch out for unnecessary redundancy, or potential for consistency issues

N-ary Relationships

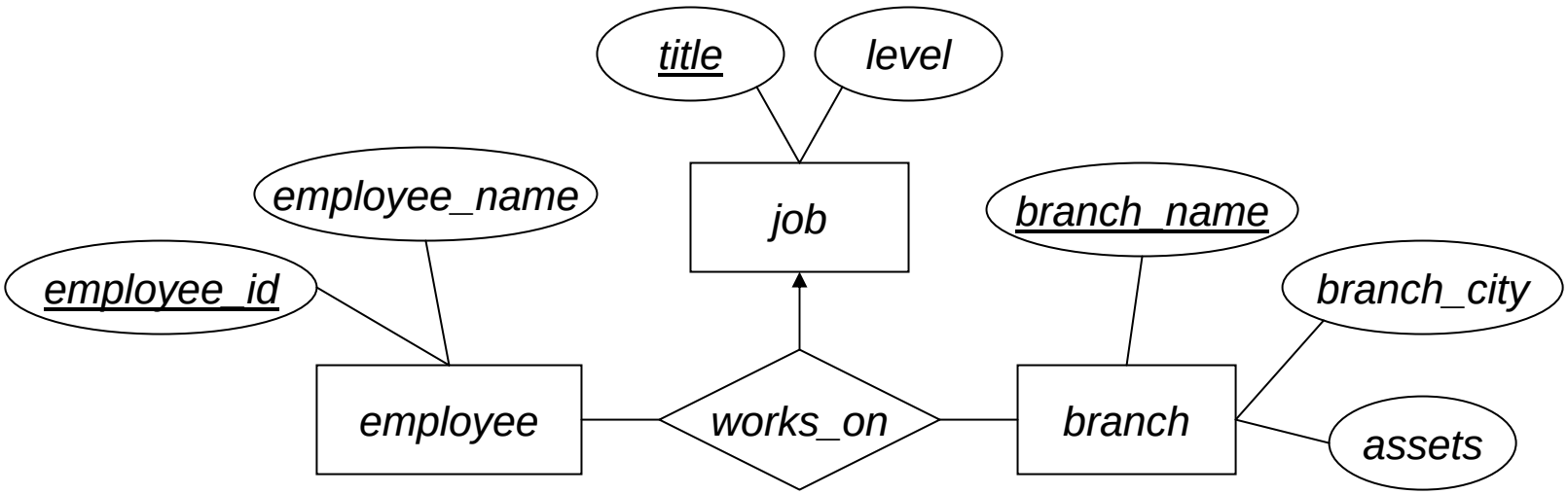
- Can specify relationships of degree > 2 in E-R model
- Example:



  - Employees are assigned to jobs at various branches
  - Many-to-many mapping: any combination of employee, job, and branch is allowed
  - An employee can have several jobs at one branch

N-ary Mapping Cardinalities

- Can specify *some* mapping cardinalities on relationships with degree > 2
- Constrain employees to one job per branch:



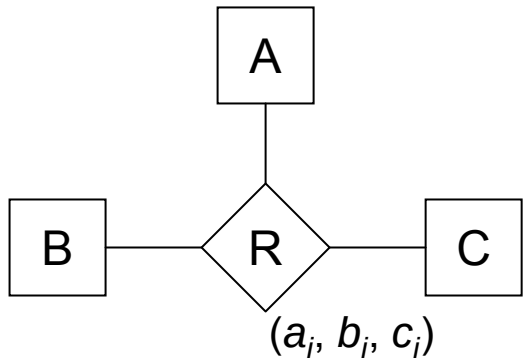
  - Each combination of employee and branch can only be associated with one job

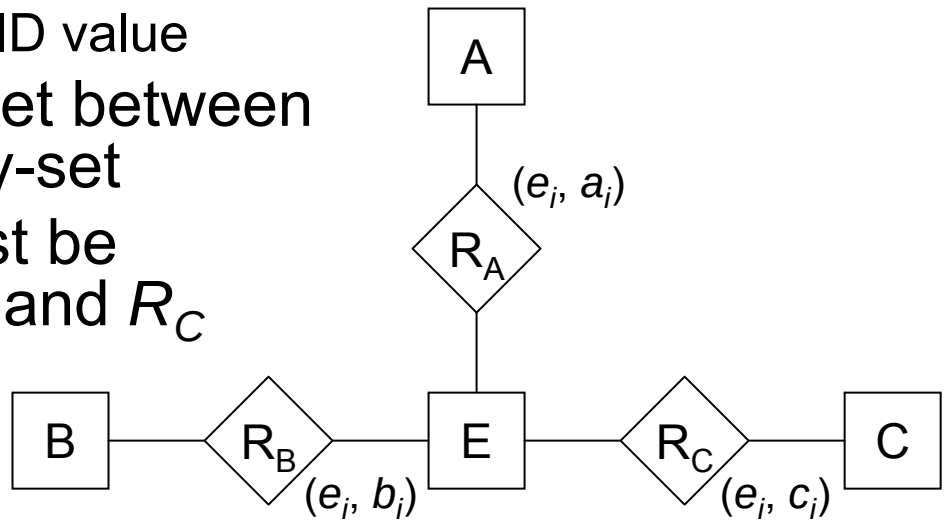
N-ary Mapping Cardinalities (2)

- For degree > 2 relationships, allow at most one arrow
- Multiple arrows on N-ary relationship-set is ambiguous
- Relationship-set R associating entity-sets $A_1, A_2, \ldots, A_n$
  - Arrows are on edges to $A_{i+1}, \ldots, A_n$
- Meaning 1: A combination of entities in $A_1, \ldots, A_i$ can be associated with at most one set of entities in $A_{i+1}, \ldots, A_n$
- Meaning 2: For each entity-set A_k ($i < k \leq n$), a combination of entities in $A_1, \ldots, A_i$ can be associated with at most one entity in A_k
- Both interpretations have been used historically…
  - But, if only one edge has an arrow, ambiguity disappears!

Binary vs. N-ary Relationships

- Normally have only binary relationships in database schemas
- For degree > 2 relationships, can replace with binary relationships
  - Replace N-ary relationship-set with a new entity-set *E*
    - Create an identifying attribute for *E*
    - e.g. an auto-generated ID value
  - Create a relationship-set between *E* and each other entity-set
  - Relationships in *R* must be represented in R_A, R_B, and R_C



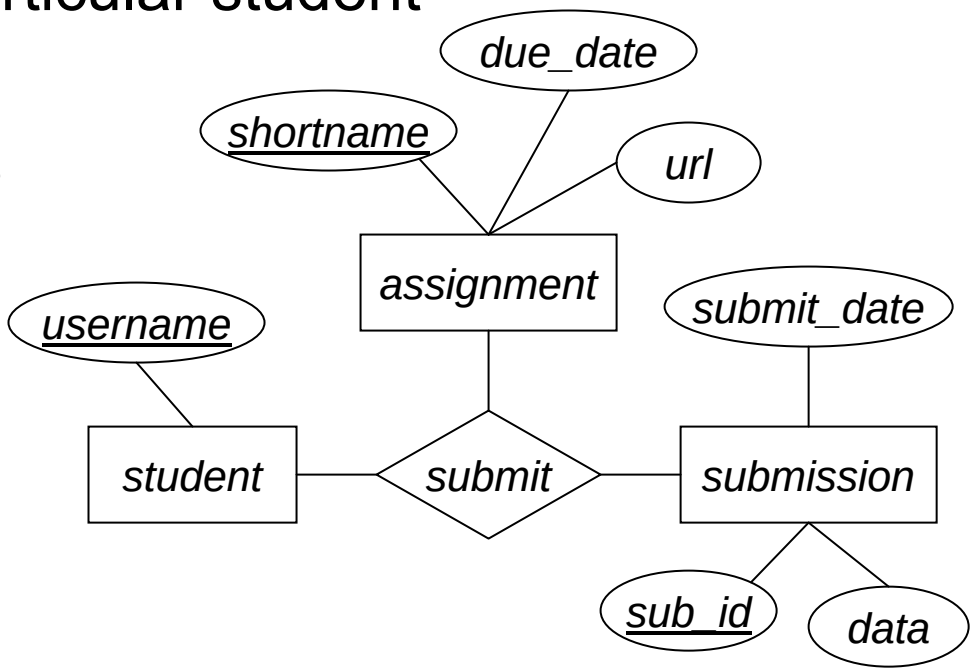


Binary vs. N-ary Relationships (2)

- Using binary relationships is sometimes more intuitive for particular designs
- Example: office-equipment inventory database
  - Ternary relationship-set *inventory*, associating *department*, *machine*, and *vendor* entity-sets
  - For ternary relationship, must use *null* values to represent missing vendor details
  - With binary relationships, can simply not have a relationship between *machine* and *vendor*
- For cases like these, use binary relationships
  - If it makes sense to model as separate binary relationships, do it that way!

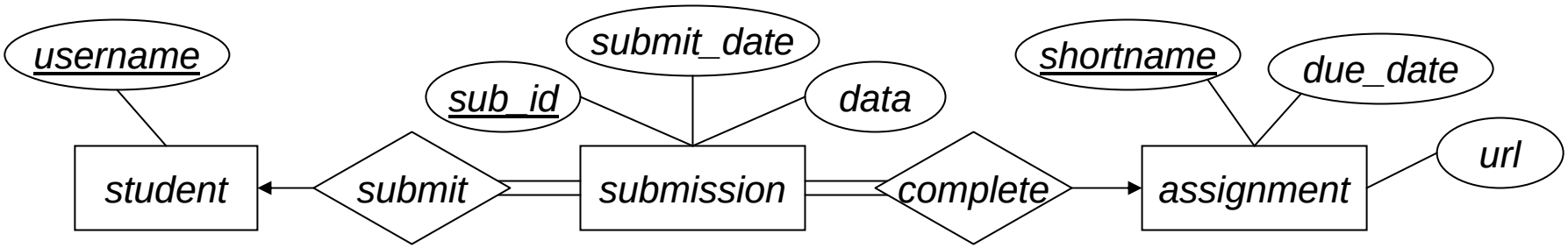
Course Database Example

- What about this case:
  - Ternary relationship between *student*, *assignment*, and *submission*
  - Need to allow multiple submissions for a particular assignment, from a particular student
- In this case, it makes sense to represent as a ternary relationship
  - Doesn't make sense to have only two of these three entities in a relationship



Course Database Example (2)

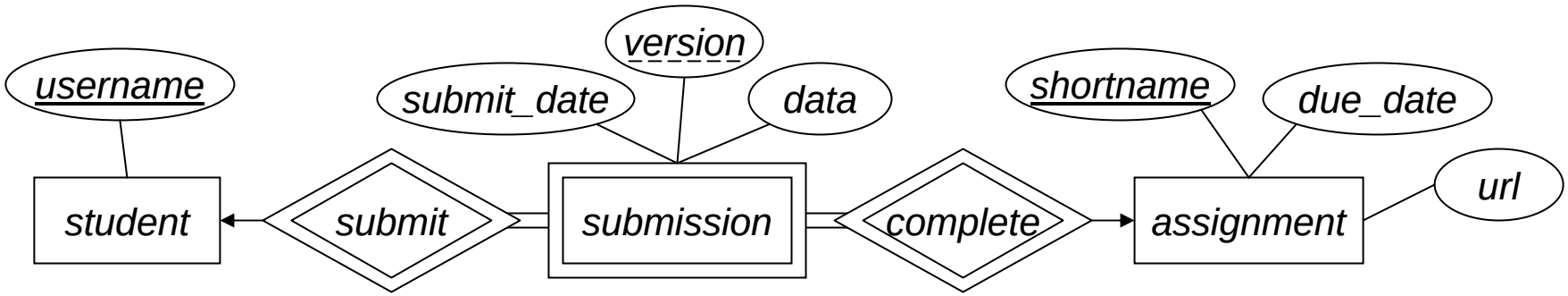
- Other ways to represent students, assignments and submissions?
- Can also represent as two binary relationships



- Note total participation constraints!
  - Required to ensure that every *submission* has an associated *student*, and an associated *assignment*
  - Also, two one-to-many constraints

Course Database Example (3)

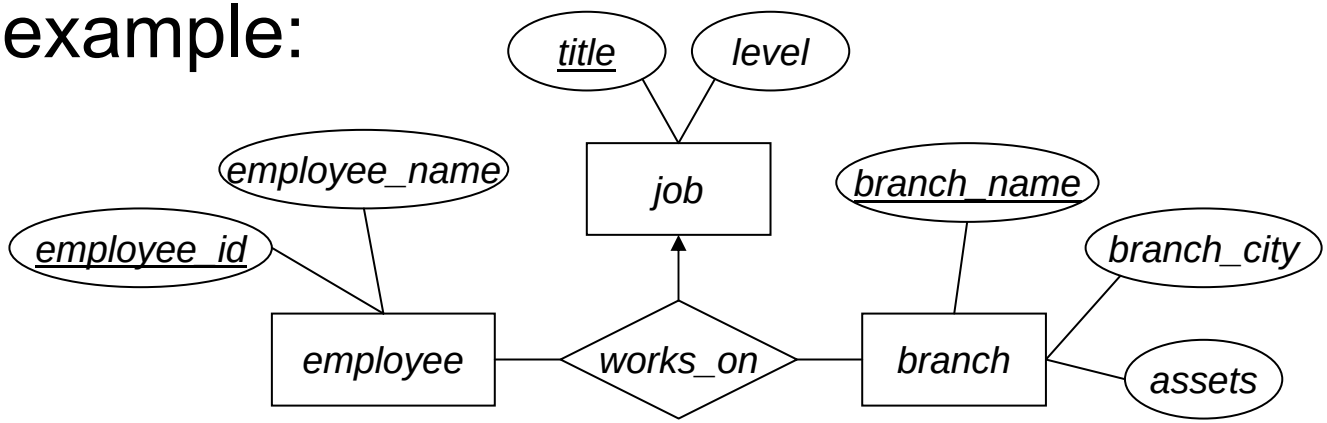
- Can also make *submission* a weak entity-set
  - Both *student* and *assignment* are identifying entities!



- Discriminator for *submission* is version number
- Primary key for *submission* ?
  - Union of primary keys from all owner entity-sets, plus discriminator
  - (*username*, *shortname*, *version*)

Binary vs. N-ary Relationships

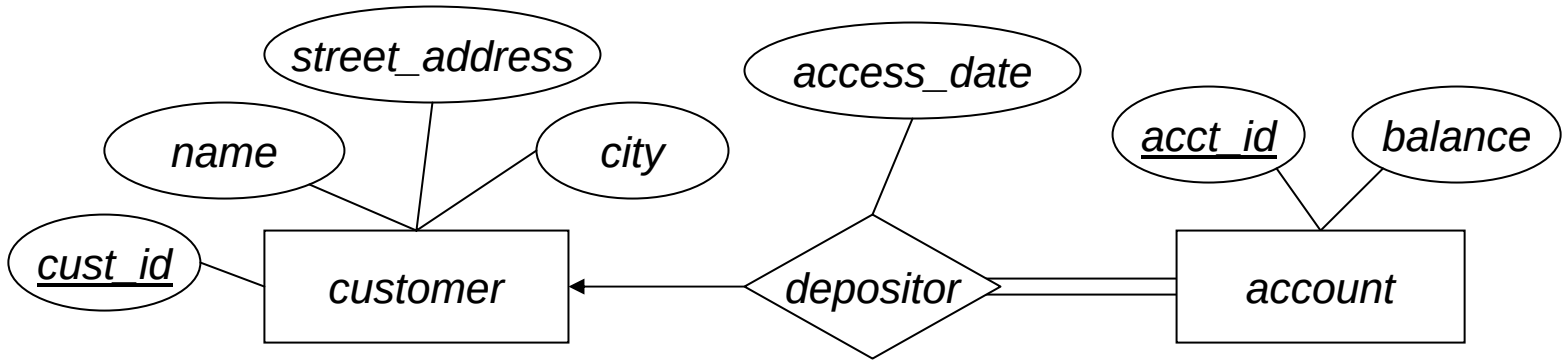
- Sometimes ternary relationships are best
  - Clearly indicates all entities involved in relationship
  - Only way to represent certain constraints!
- Bank jobs example:



  - Each (*employee*, *branch*) pair can have only one job
  - Simply cannot construct same constraint using only binary relationships

Placement of Attributes

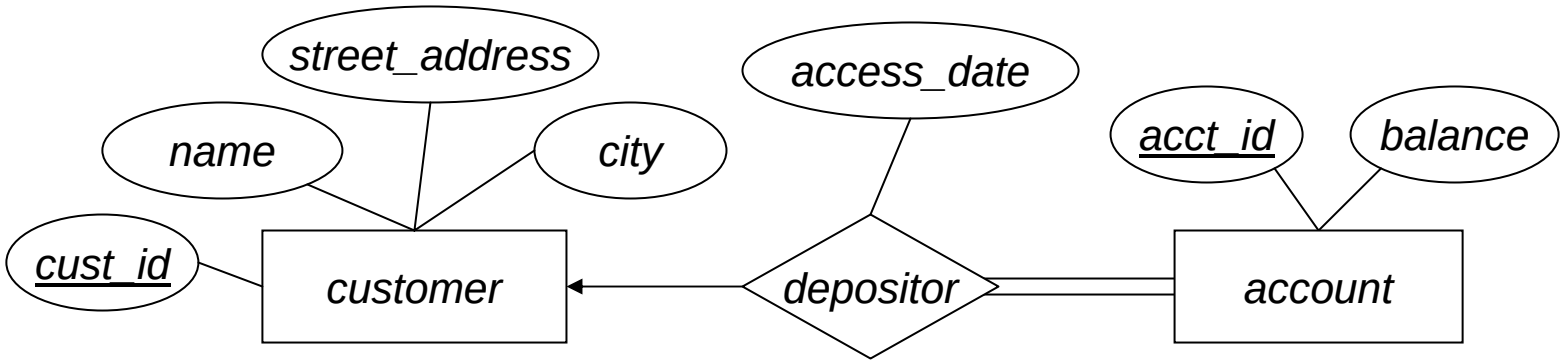
- Relationship-sets can have descriptive attributes
- For one-to-one or one-to-many relationship-sets:
  - Can associate descriptive attributes with one of the participating entity-sets, instead of the relationship-set
- Example: *customer*, *account*, and *depositor*



  - Would be identical to have *access_date* on *account*, since each account associated with just one customer

Placement of Attributes (2)

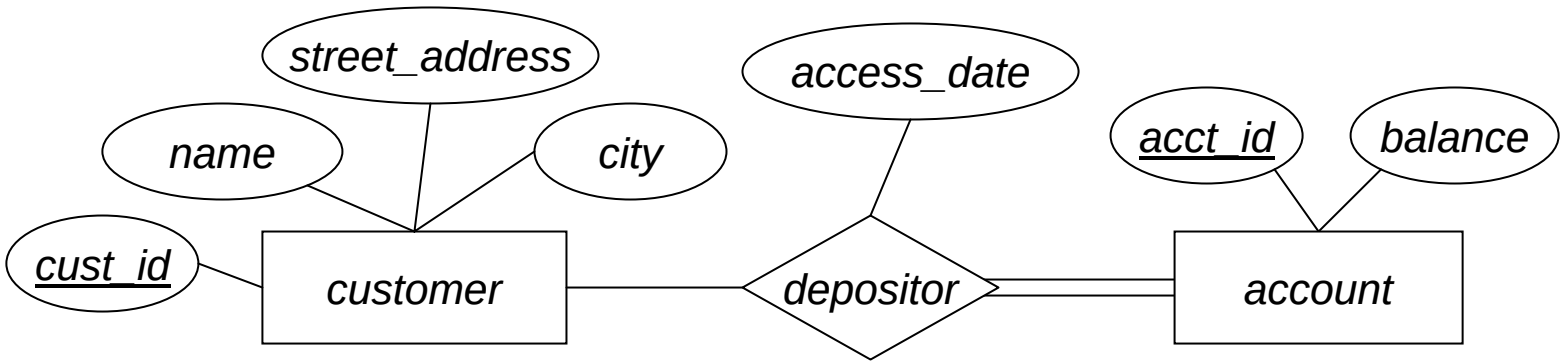
- Choose attribute placement that best indicates characteristics of enterprise being modeled



  - If *depositor* is only relationship between customers and accounts, leave *access_date* on *depositor*
  - If other relationships exist between customers and accounts, move *access_date* to *account* entity-set

Placement of Attributes (3)

- If relationship-set is many-to-many, different placements have different implications!
- With banking example again:



  - Now an account can have multiple owners
- What if *access_date* were on *account* entity-set?
  - Couldn't tell *which customer* made last access!
  - Actually affects what schema can represent.

Placement of Attributes (4)

- For one-to-one and one-to-many relationships:
  - Flexibility in where descriptive attributes are place
  - Can place descriptive attribute on “many” side of relationship-set
  - Choose option that best models the enterprise
- For many-to-many relationships:
  - Different options have different implications
  - If attribute value should be associated with the combination of entities, must appear on relationship
  - If attribute value relates only to a particular entity, move it to that entity-set

Review

- Entity-relationship model is inexact
  - Can represent a particular design in several different ways
- Covered variations on themes
  - Some options are easier/harder to constrain
  - Some options mean slightly different things
  - Some options are just bad design
- Next time:
  - How to map E-R schemas to relational model