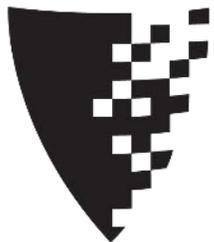


# Relational Database Design: Part I

Introduction to Databases

CompSci 316 Fall 2017



**DUKE**  
COMPUTER SCIENCE

# Announcements (Thu. Sep. 7)

- Homework #1 due in 12 days
  - Get started early!
  - Please set up VM now!
- Office hours have been posted
- More details on the course project available next week

# Relational model: review

- A database is a collection of **relations** (or **tables**)
- Each relation has a set of **attributes** (or **columns**)
- Each attribute has a name and a **domain** (or **type**)
- Each relation contains a set of **tuples** (or **rows**)

# Keys

- A set of attributes  $K$  is a **key** for a relation  $R$  if
  - In no instance of  $R$  will two different tuples agree on all attributes of  $K$ 
    - That is,  $K$  can serve as a “**tuple identifier**”
  - No proper subset of  $K$  satisfies the above condition
    - That is,  $K$  is **minimal**
- Example: *User* (*uid*, *name*, *age*, *pop*)
  - *uid* is a key of *User*
  - *age* is not a key (not an identifier)
  - {*uid*, *name*} is not a key (not minimal)

# Schema vs. instance

<i>uid</i>	<i>name</i>	<i>age</i>	<i>pop</i>
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3

- Is *name* a key of *User*?
  - Yes? Seems reasonable for this instance
  - No! User names are not unique **in general**
- Key declarations are part of the schema

# More examples of keys

- *Member* (*uid*, *gid*)

- {*uid*, *gid*}

☞ A key can contain multiple attributes

- *Address* (*street\_address*, *city*, *state*, *zip*)

- {*street\_address*, *city*, *state*}

- {*street\_address*, *zip*}

☞ A relation can have multiple keys!

- We typically pick one as the “primary” key, and underline all its attributes, e.g., *Address* (*street\_address*, *city*, *state*, *zip*)

# Use of keys

- More constraints on data, fewer mistakes
- Look up a row by its key value
  - Many selection conditions are “key = value”
- “Pointers” to other rows (often across tables)
  - Example: *Member (uid, gid)*
    - *uid* is a key of *User*
    - *gid* is a key of *Group*
    - A *Member* row “links” a *User* row with a *Group* row
  - Many join conditions are “key = key value stored in another table”

# Database design

- Understand the real-world domain being modeled
- Specify it using a database **design model**
  - More intuitive and convenient for schema design
  - But not necessarily implemented by DBMS
  - A few popular ones:
    - **Entity/Relationship (E/R) model**
    - **Object Definition Language (ODL)**
    - **UML (Unified Modeling Language)**
- Translate specification to the data model of DBMS
  - Relational, XML, object-oriented, etc.
- Create DBMS schema

# But what about ORM?

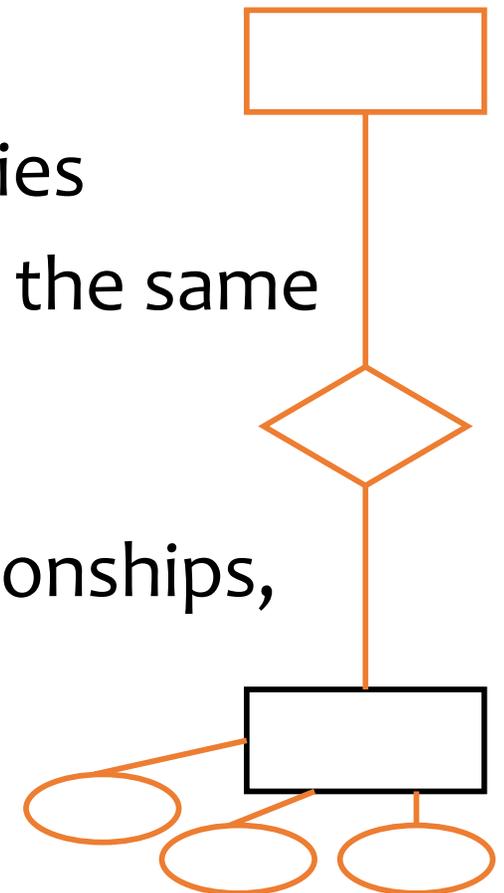
- Automatic **object-relational mappers** are made popular by rapid Web development frameworks
  - For example, with Python SQLAlchemy:
    - You declare Python classes and their relationships
    - It automatically converts them into database tables
    - If you want, you can just work with Python objects, and never need to be aware of the database schema or write SQL
- But you still need designer discretion in all but simple cases
- Each language/library has its own syntax for creating schema and for querying/modifying data
  - Quirks and limitations cause portability problems
  - They are not necessarily easier to learn than SQL

# Entity-relationship (E/R) model

- Historically and still very popular
- Concepts applicable to other design models as well
- Can think of as a “watered-down” object-oriented design model
- Primarily a design model—not directly implemented by DBMS
- Designs represented by E/R diagrams
  - We use the style of E/R diagram covered by the GMUW book; there are other styles/extensions
  - Very similar to UML diagrams

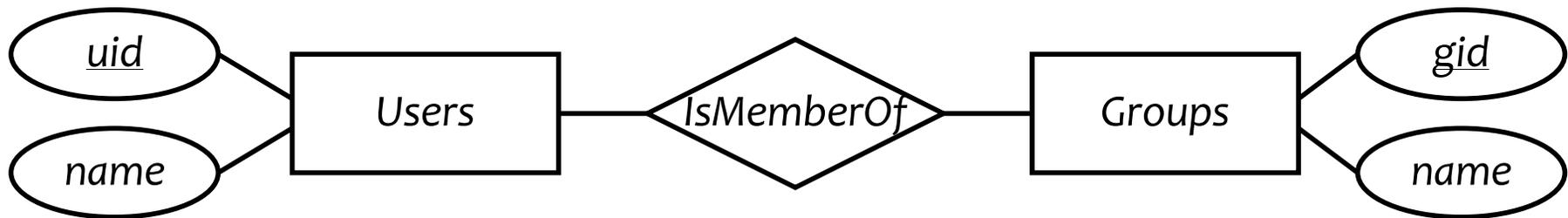
# E/R basics

- **Entity**: a “thing,” like an object
- **Entity set**: a collection of things of the same type, like a relation of tuples or a class of objects
  - Represented as a rectangle
- **Relationship**: an association among entities
- **Relationship set**: a set of relationships of the same type (among same entity sets)
  - Represented as a diamond
- **Attributes**: properties of entities or relationships, like attributes of tuples or objects
  - Represented as ovals



# An example E/R diagram

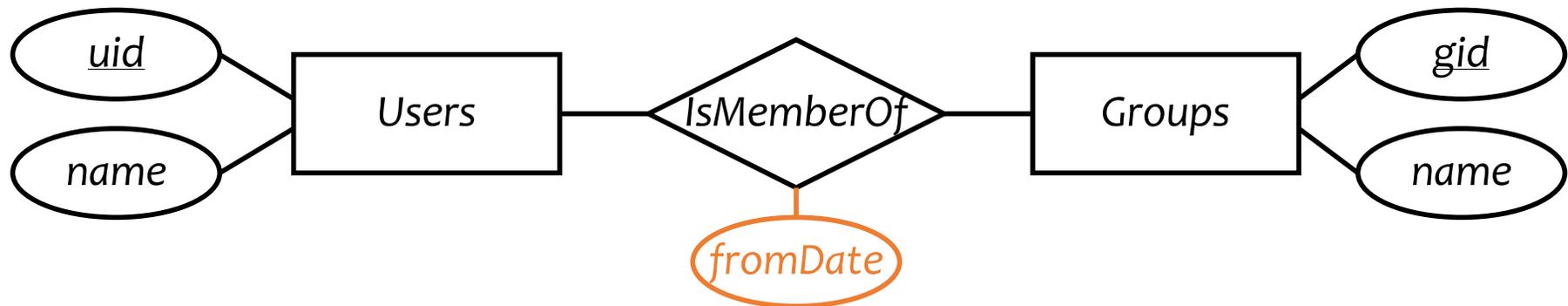
- Users are members of groups



- A key of an entity set is represented by underlining all attributes in the key
  - A key is a set of attributes whose values can belong to at most one entity in an entity set—like a key of a relation

# Attributes of relationships

- Example: a user belongs to a group since a particular date



- Where do the dates go?
  - With *Users*?
    - But a user can join multiple groups on different dates
  - With *Groups*?
    - But different users can join the same group on different dates
  - With *IsMemberOf*!

# More on relationships

- There could be multiple relationship sets between the same entity sets
  - Example: *Users IsMemberOf Groups*; *Users Likes Groups*
- In a relationship set, each relationship is uniquely identified by the entities it connects
  - Example: Between Bart and “Dead Putting Society”, there can be at most one *IsMemberOf* relationship and at most one *Likes* relationship
  - ☞ What if Bart joins DPS, leaves, and rejoins? How can we modify the design to capture historical membership information?
    - ☞ Make an entity set of *MembershipRecords*

# Multiplicity of relationships

- $E$  and  $F$ : entity sets
- **Many-many**: Each entity in  $E$  is related to 0 or more entities in  $F$  and vice versa

- Example:



- **Many-one**: Each entity in  $E$  is related to 0 or 1 entity in  $F$ , but each entity in  $F$  is related to 0 or more in  $E$

- Example:



- **One-one**: Each entity in  $E$  is related to 0 or 1 entity in  $F$  and vice versa

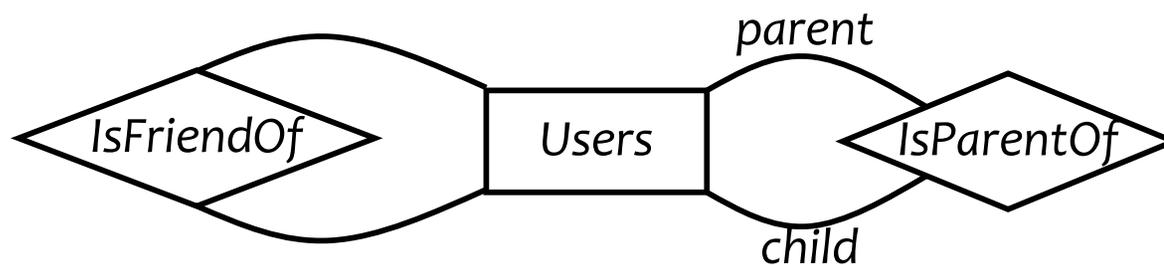
- Example:



- “One” (0 or 1) is represented by an arrow  $\longrightarrow$
- “Exactly one” is represented by a rounded arrow  $\longrightarrow$

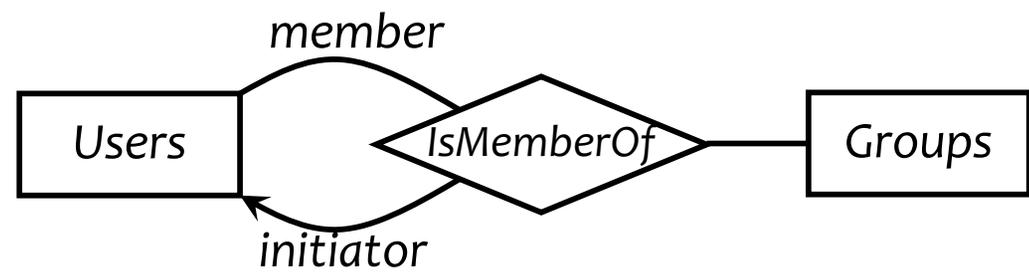
# Roles in relationships

- An entity set may participate more than once in a relationship set
- ☞ May need to label edges to distinguish **roles**
- Examples
  - Users may be parents of others; label needed
  - Users may be friends of each other; label not needed



# *n*-ary relationships

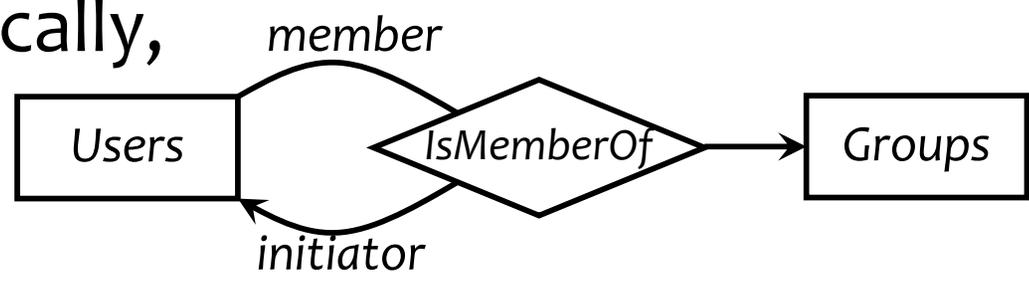
- Example: a user must have an initiator in order to join a group



Rule for interpreting an arrow into entity set *E* in an *n*-ary relationship:

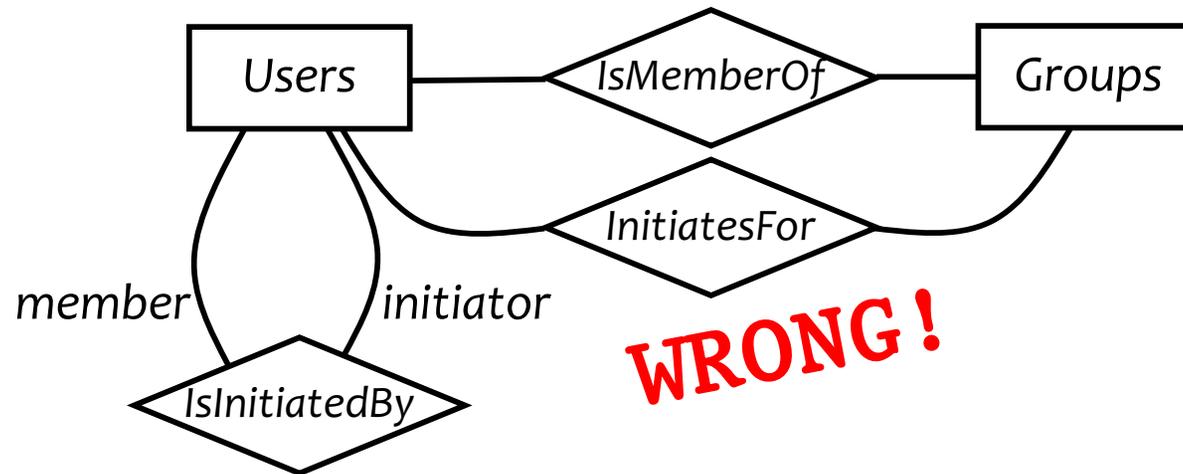
- Pick one entity from each of the other entity sets; together they can be related to at most one entity in *E*

- Exercise: hypothetically, what do these arrows imply?



# $n$ -ary versus binary relationships

- Can we model  $n$ -ary relationships using just binary relationships?

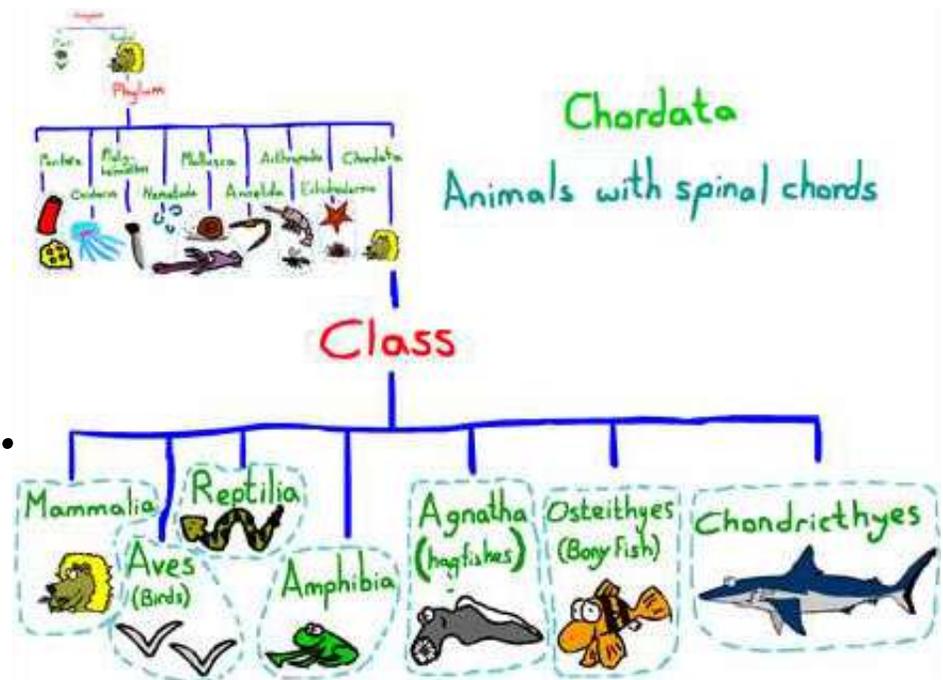


- No; for example:
  - Ralph is in both abc and gov
  - Lisa has served as initiator in both abc and gov
  - Ralph was initiated by Lisa in abc, but not by her in gov

# Next: two special relationships



... is part of/belongs to ...

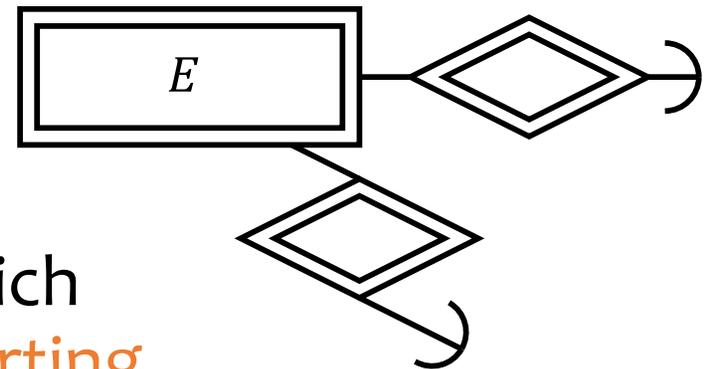


... is a kind of ...

# Weak entity sets

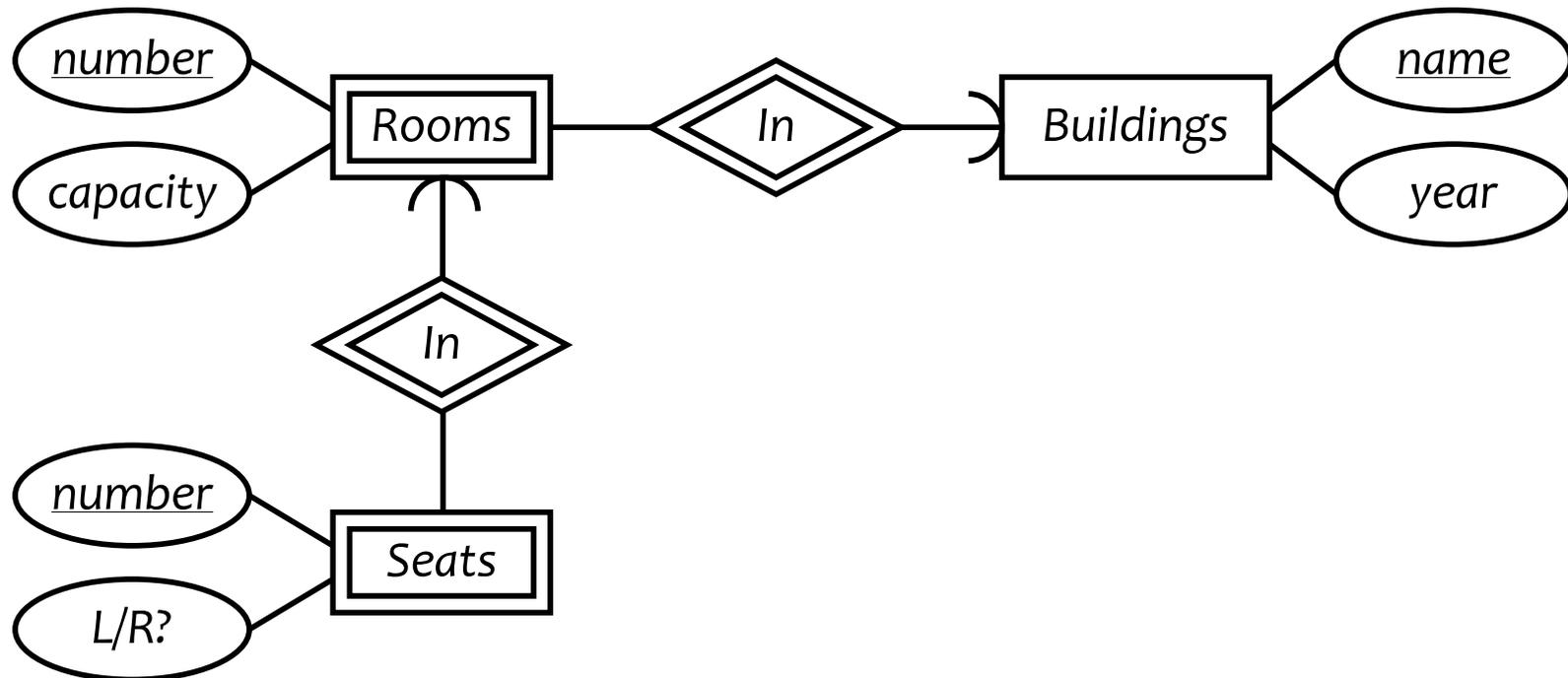
Sometimes, an entity's identity depends on some others'

- The key of a **weak entity set**  $E$  comes not completely from its own attributes, but from the keys of one or more other entity sets
  - $E$  must link to them via many-one or one-one relationship sets
- Example: *Rooms* inside *Buildings* are partly identified by *Buildings'* name
- A weak entity set is drawn as a double rectangle
- The relationship sets through which it obtains its key are called **supporting relationship sets**, drawn as double diamonds



# Weak entity set examples

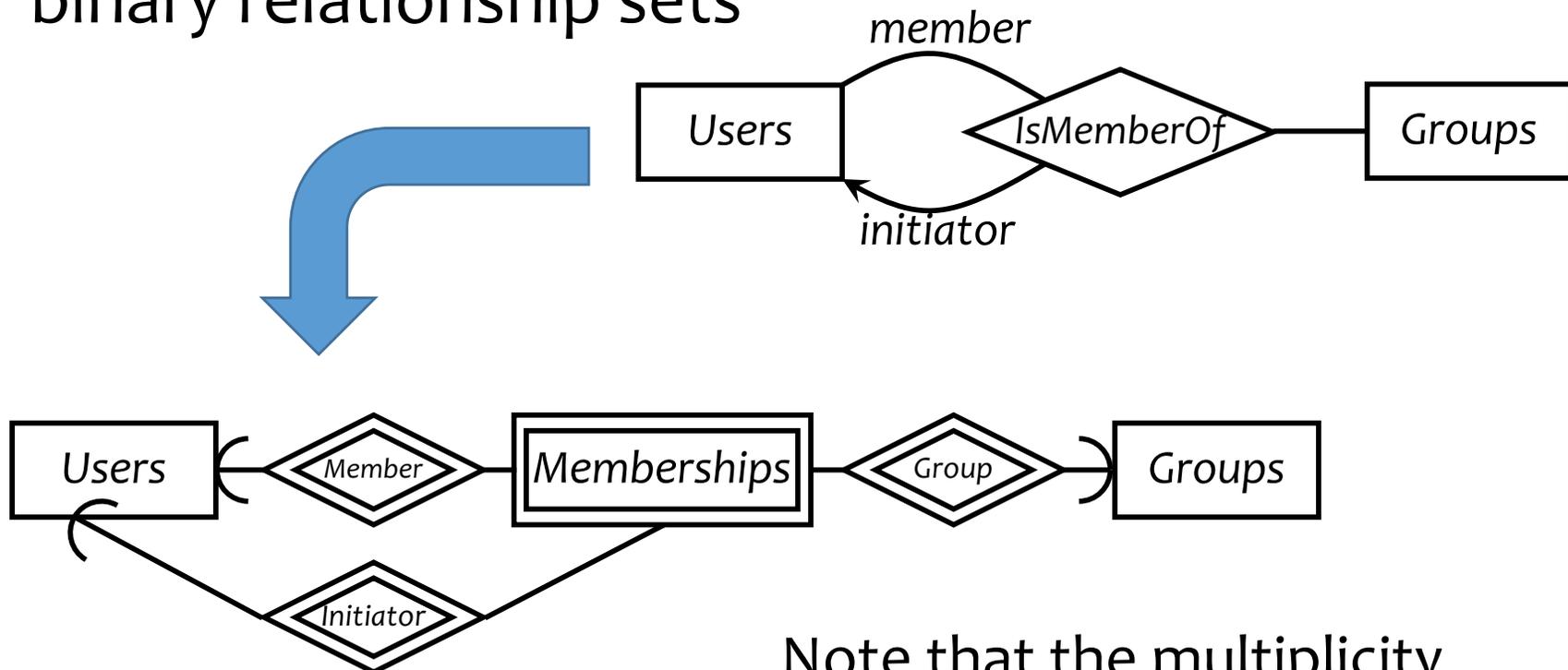
- Seats in rooms in building



- Why must double diamonds be many-one/one-one?
  - With many-many, we would not know which entity provides the key value!

# Remodeling $n$ -ary relationships

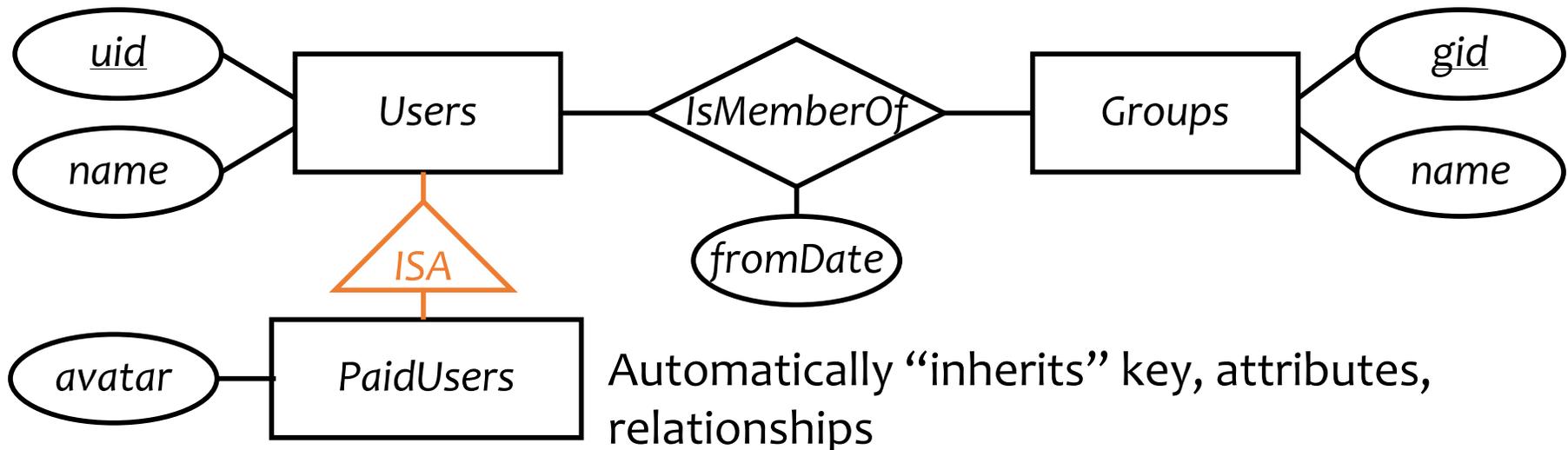
- An  $n$ -ary relationship set can be replaced by a weak entity set (called a **connecting entity set**) and  $n$  binary relationship sets



Note that the multiplicity constraint for *IsMemberOf* is lost

# ISA relationships

- Similar to the idea of subclasses in object-oriented programming: subclass = special case, fewer entities, and possibly more properties
  - Represented as a triangle (direction is important)
- Example: paid users are users, but they also get avatars (yay!)



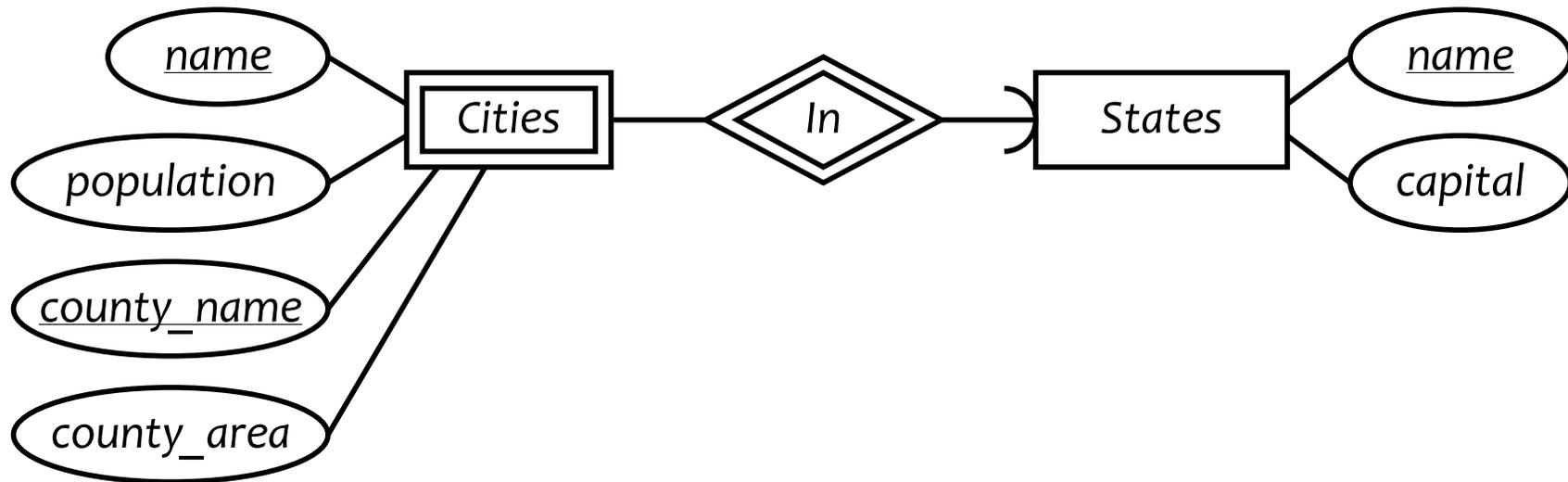
# Summary of E/R concepts

- Entity sets
  - Keys
  - Weak entity sets
- Relationship sets
  - Attributes of relationships
  - Multiplicity
  - Roles
  - Binary versus  $n$ -ary relationships
    - Modeling  $n$ -ary relationships with weak entity sets and binary relationships
  - ISA relationships

# Case study 1

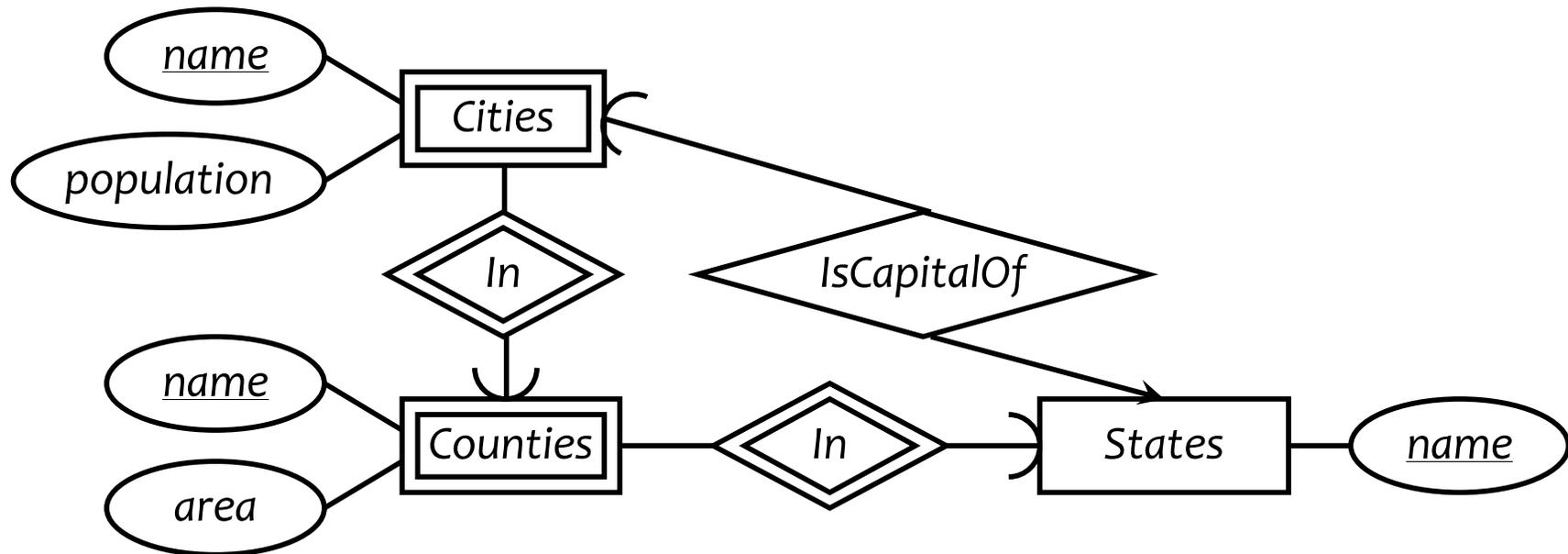
- Design a database representing cities, counties, and states
  - For states, record name and capital (city)
  - For counties, record name, area, and location (state)
  - For cities, record name, population, and location (county and state)
- Assume the following:
  - Names of states are unique
  - Names of counties are only unique within a state
  - Names of cities are only unique within a county
  - A city is always located in a single county
  - A county is always located in a single state

# Case study 1: first design



- County area information is repeated for every city in the county
  - ☞ Redundancy is bad (why?)
- State capital should really be a city
  - ☞ Should “reference” entities through explicit relationships

# Case study 1: second design

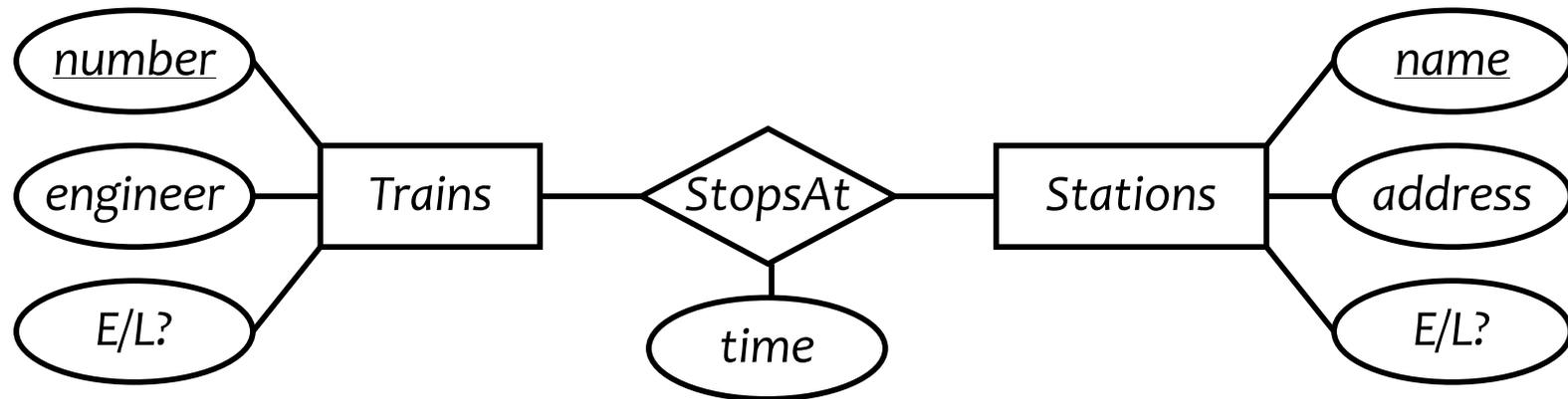


- Technically, nothing in this design prevents a city in state  $X$  from being the capital of another state  $Y$ , but oh well...

# Case study 2

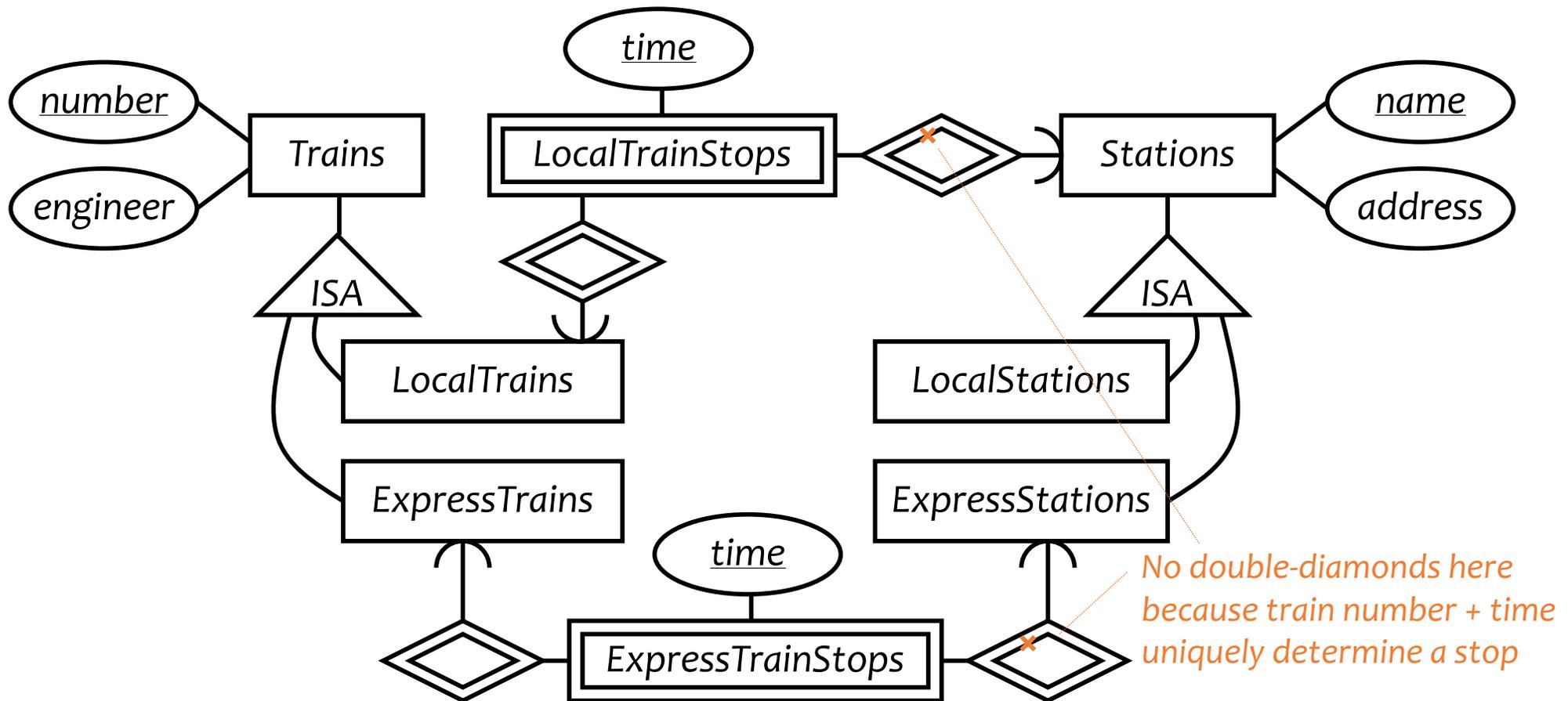
- Design a database consistent with the following:
  - A station has a unique name and an address, and is either an express station or a local station
  - A train has a unique number and an engineer, and is either an express train or a local train
  - A local train can stop at any station
  - An express train only stops at express stations
  - A train can stop at a station for any number of times during a day
  - Train schedules are the same everyday

# Case study 2: first design



- Nothing in this design prevents express trains from stopping at local stations
  - ☞ We should capture as many constraints as possible
- A train can stop at a station only once during a day
  - ☞ We should not introduce unintended constraints

# Case study 2: second design



Is the extra complexity worth it?