# Relational Database Design: E/R-Relational Translation

Introduction to Databases

CompSci 316 Spring 2020

**DUKE**
COMPUTER SCIENCE

# Announcements (Thu. Jan. 30)

- HW2/Lab1 due tonight (Thurs, Jan 30, 11:59 pm)
- HW3 Q1-Q2 posted
  - Q3-Q5 to be posted next week after the material is covered in class
  - Many parts, start early!
- Please form your groups by next Thursday Feb 6!
  - So that we can help you find a group if needed well before MS1 is due
  - Project formation spreadsheet shared
  - 5 members for standard projects please! (otherwise we may have to shuffling later, better if you do it yourself)
  - If you want to do an open project, let me know asap

# Announcements – contd. (Thu. Jan. 30)

- HW extension requests (See the course policy)
  - We cannot accommodate requests for "I need more time" to be fair to all
  - For unforeseen situations not in our control like medical reasons, you **must submit an incapacitation form and copy your academic dean** while requesting an extension and mention the extra time you need (typically 1-2 days).
  - Make sure that you have an email from me accepting the extension request and specifying the deadline.
  - That deadline is final for you and late submissions with penalty do not apply
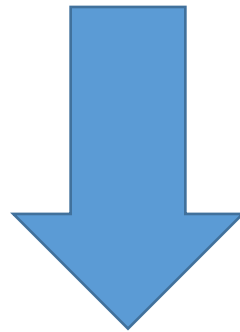  - Be careful as the next hw would be posted

# Database design steps: review

- Understand the real-world domain being modeled
- Specify it using a database design model (e.g., E/R)
- Translate specification to the data model of DBMS (e.g., relational)
- Create DBMS schema

# Today
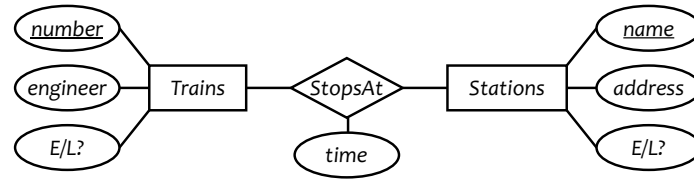


You designed an ER digram



Translate it to a Relational Database

*Train (number, engineer, type)*
*Station (name, address, type)*
*TrainStop (train_number, station_name, time)*

# E/R model: review

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

# Translating entity sets

- An entity set translates directly to a table
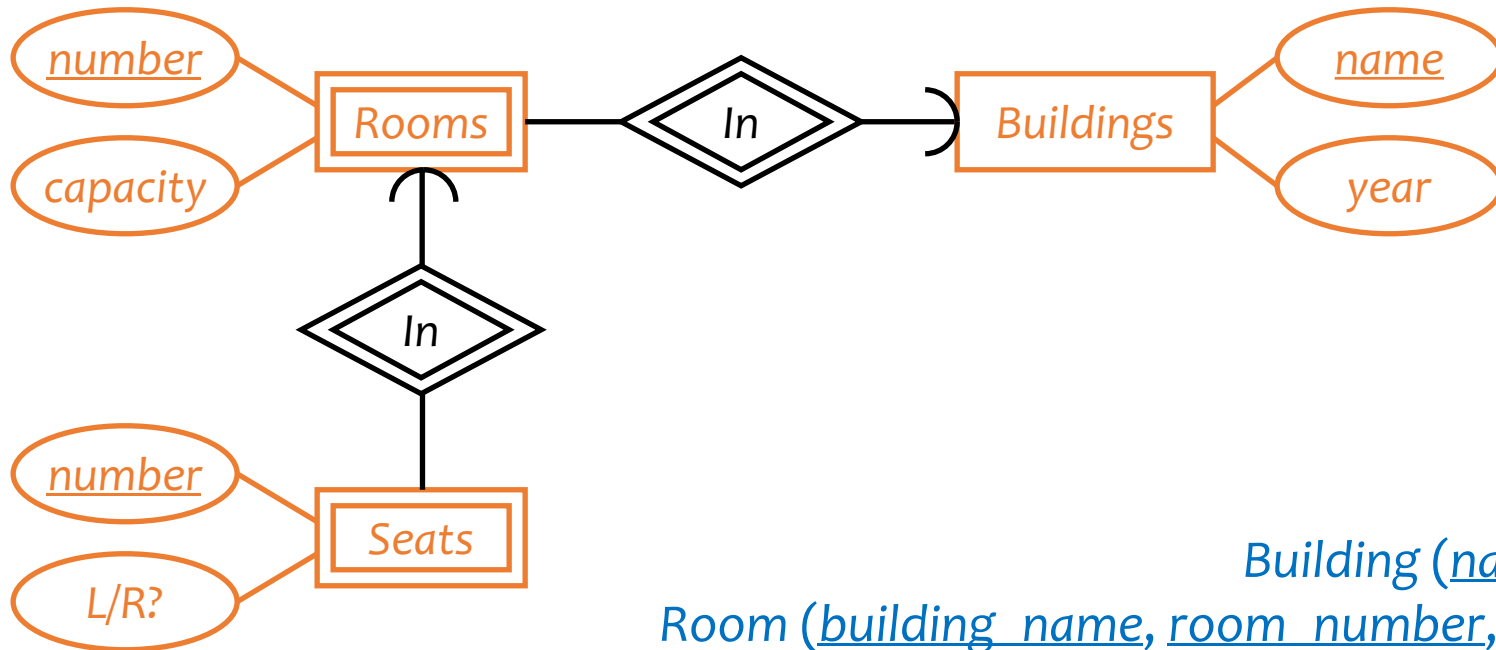  - Attributes → columns
  - Key attributes → key columns
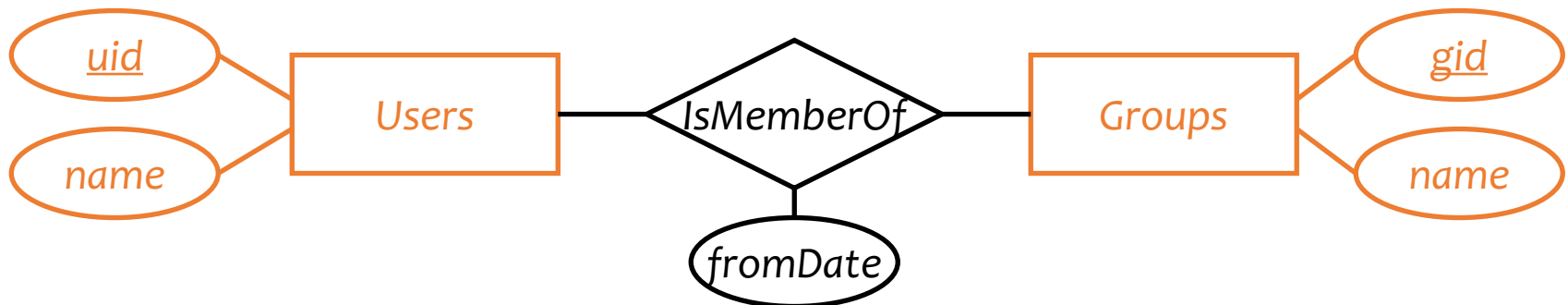


$User\ (\underline{uid},\ name)$        $Group\ (\underline{gid},\ name)$

# Translating weak entity sets

- Remember the "borrowed" key attributes
- Watch out for attribute name conflicts



Building (*name*, *year*)
Room (*building_name*, *room_number*, *capacity*)
Seat (*building_name*, *room_number*, *seat_number*, *left_or_right*)
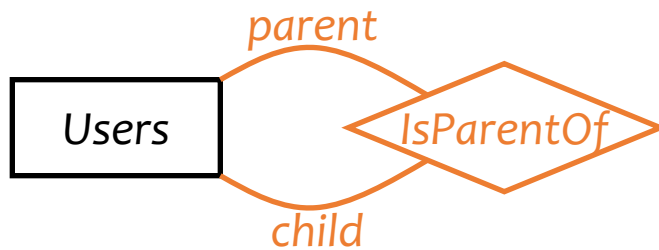
# Translating relationship sets

- A relationship set translates to a table
  - Keys of connected entity sets → columns
  - Attributes of the relationship set (if any) → columns
  - Multiplicity of the relationship set determines the key of the table
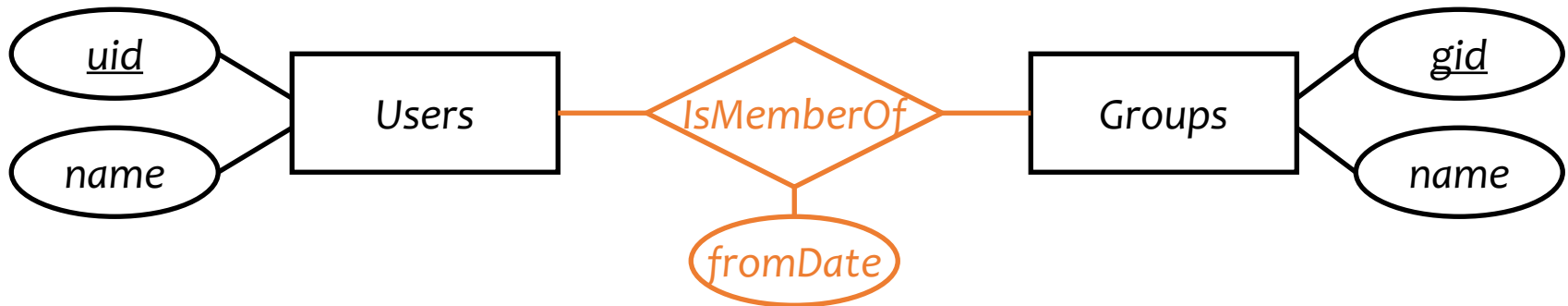


*Member (uid, gid, fromDate)*
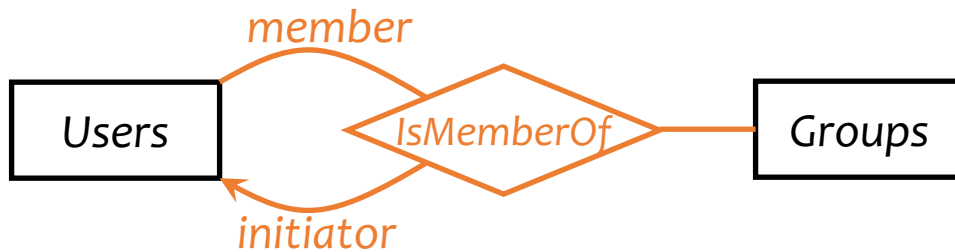
How do the keys change if you have arrow to Users, Groups, or both?

# More examples



parent

Users — IsParentOf

child

Parent (parent_uid, child_uid)

member

Users — IsMemberOf — Groups

initiator

Member (uid, initiator_uid, gid)

# Translating double diamonds?

- Recall that a double-diamond (supporting) relationship set connects a weak entity set to another entity set

- No need to translate because the relationship is implicit in the weak entity set's translation

*number*  *Rooms*  *In*  *Buildings*  *name*

*capacity*  *year*

*In*

*number*  *Seats*

*L/R?*

*RoomInBuilding*
(*room_building_name*, *room_number*, *building_name*)
is subsumed by
*Room* (*building_name*, *room_number*, *capacity*)
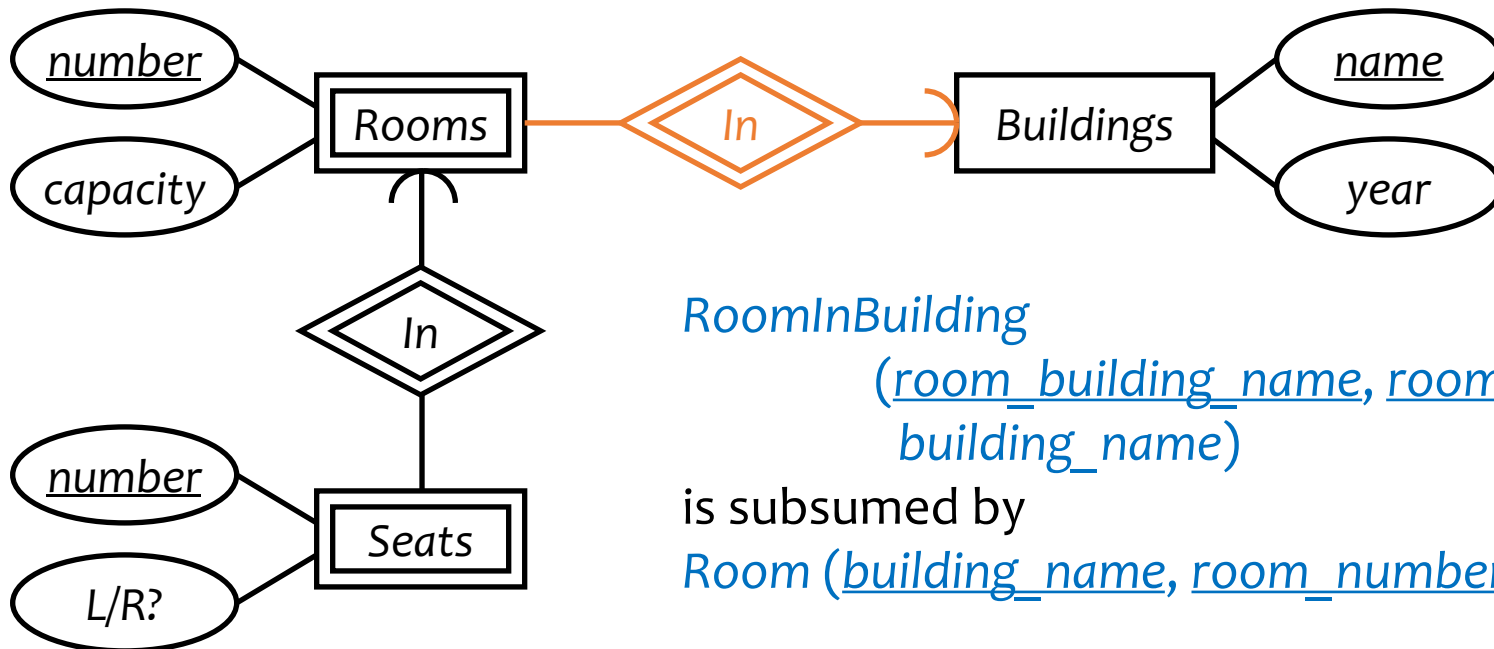
# Translating subclasses & ISA
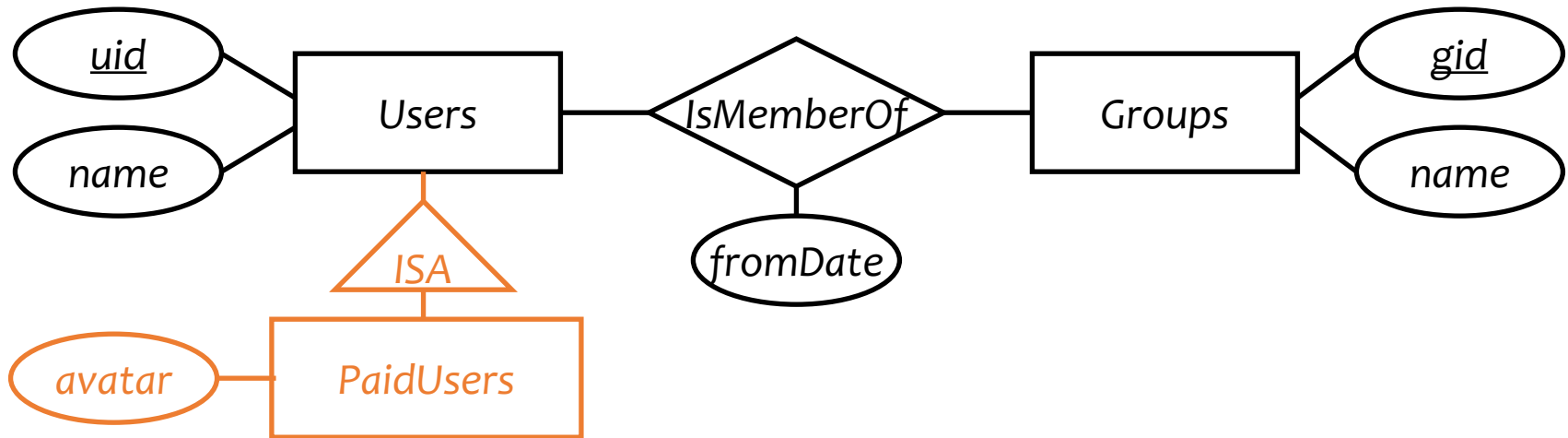
# Translating subclasses & ISA: approach 1

- Entity-in-all-superclasses approach ("E/R style")
  - An entity is represented in the table for each subclass to which it belongs
  - A table includes only the attributes directly attached to the corresponding entity set, plus the inherited key



$\langle 142, \text{Bart} \rangle$
$\langle 456, \text{Ralph} \rangle$ $\in$ 

Group (*gid*, *name*)
User (*uid*, *name*)
Member (*uid*, *gid*, *from_date*)

$\langle 456, \text{☺} \rangle$ $\in$ *PaidUser* (*uid*, *avatar*)
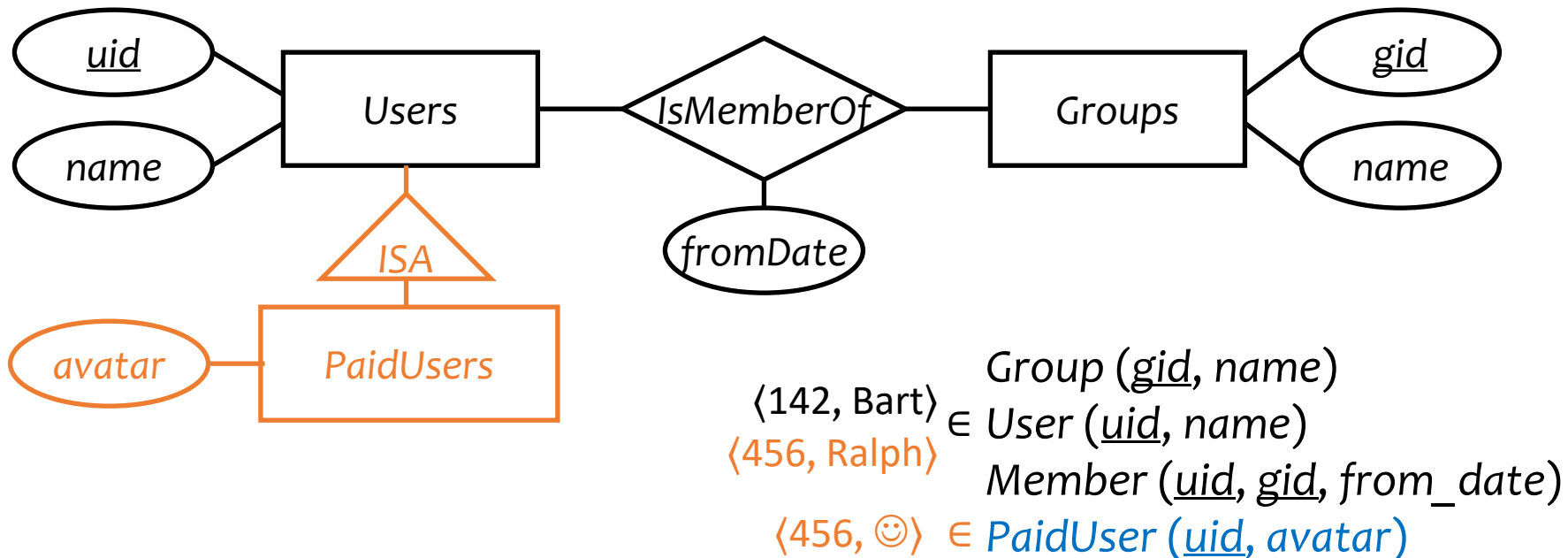
# Translating subclasses & ISA: approach 2

- Entity-in-most-specific-class approach ("OO style")
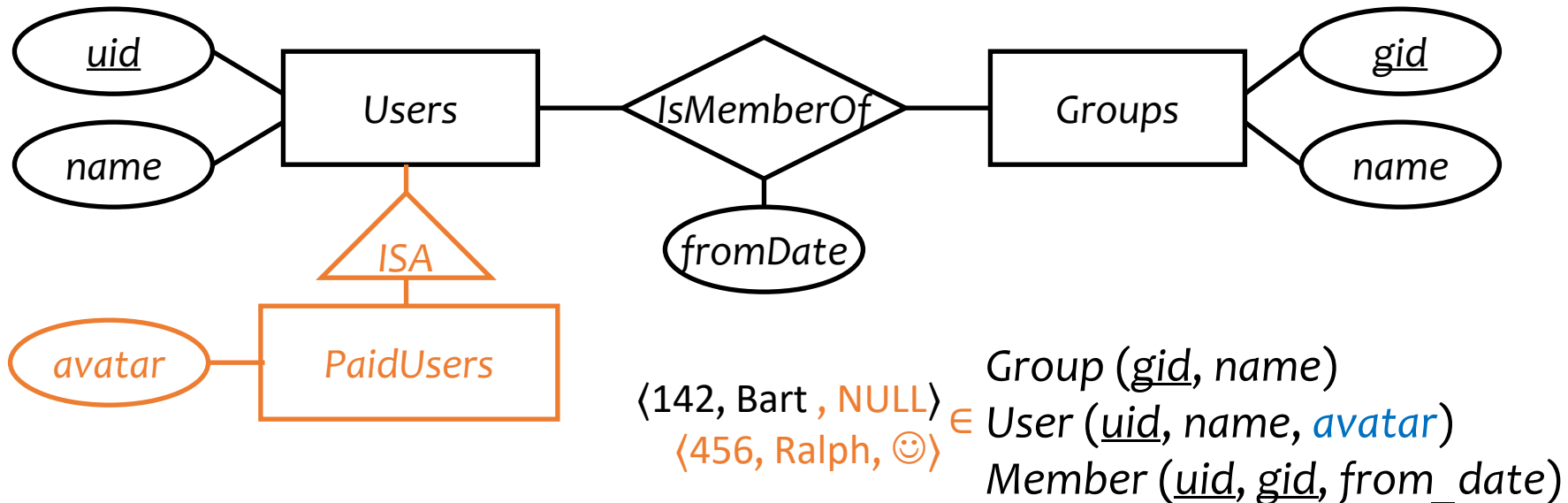  - An entity is only represented in one table (the most specific entity set to which the entity belongs)
  - A table includes the attributes attached to the corresponding entity set, plus all inherited attributes



Group (*gid*, *name*)

⟨142, Bart⟩ ∈ *User* (*uid*, *name*)

*Member* (*uid*, *gid*, *from_date*)

⟨456, Ralph, ☺⟩    ∈ *PaidUser* (*uid*, *name*, *avatar*)
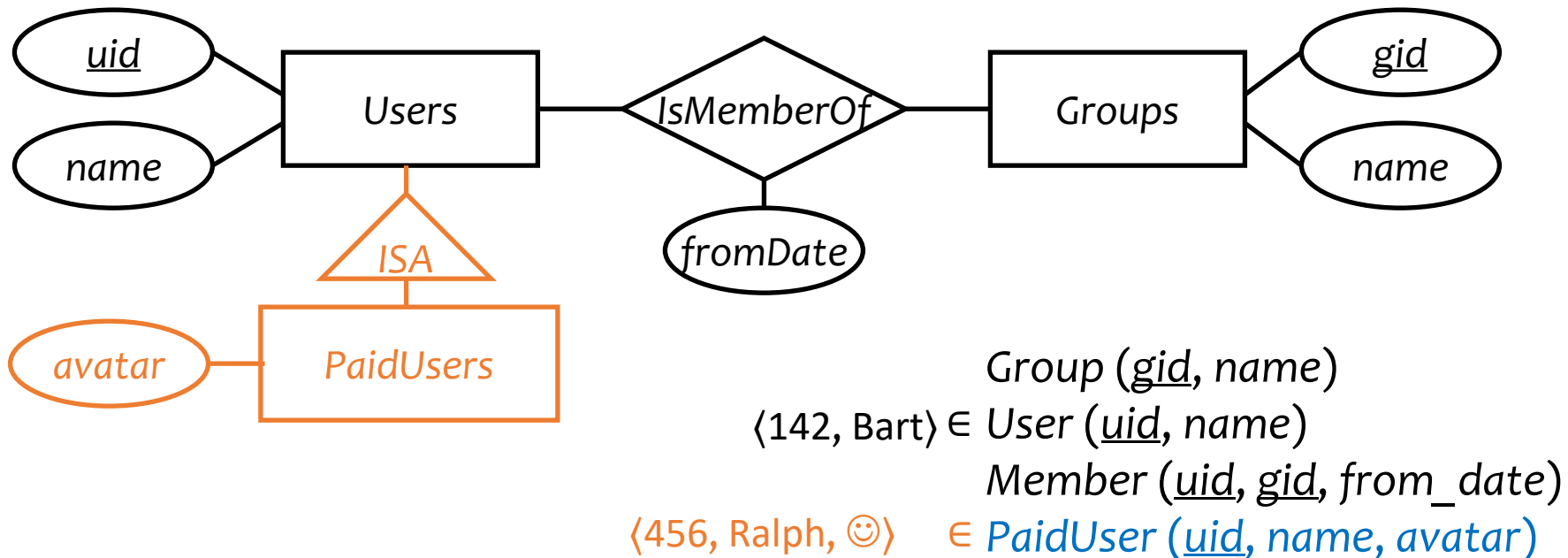
# Translating subclasses & ISA: approach 3

- ## All-entities-in-one-table approach ("NULL style")
  - One relation for the root entity set, with all attributes found in the network of subclasses (plus a "type" attribute when needed)
  - Use a special NULL value in columns that are not relevant for a particular entity



⟨142, Bart , NULL⟩
⟨456, Ralph, ☺⟩ ∈

Group (*gid, name*)
User (*uid, name, avatar*)
Member (*uid, gid, from_date*)

# Comparison of three approaches

- Entity-in-all-superclasses
  - *User (<u>uid</u>, name), PaidUser (<u>uid</u>, avatar)*
  - Pro: All users are found in one table
  - Con: Attributes of paid users are scattered in different tables
- Entity-in-most-specific-class
  - *User (<u>uid</u>, name), PaidUser (<u>uid</u>, name, avatar)*
  - Pro: All attributes of paid users are found in one table
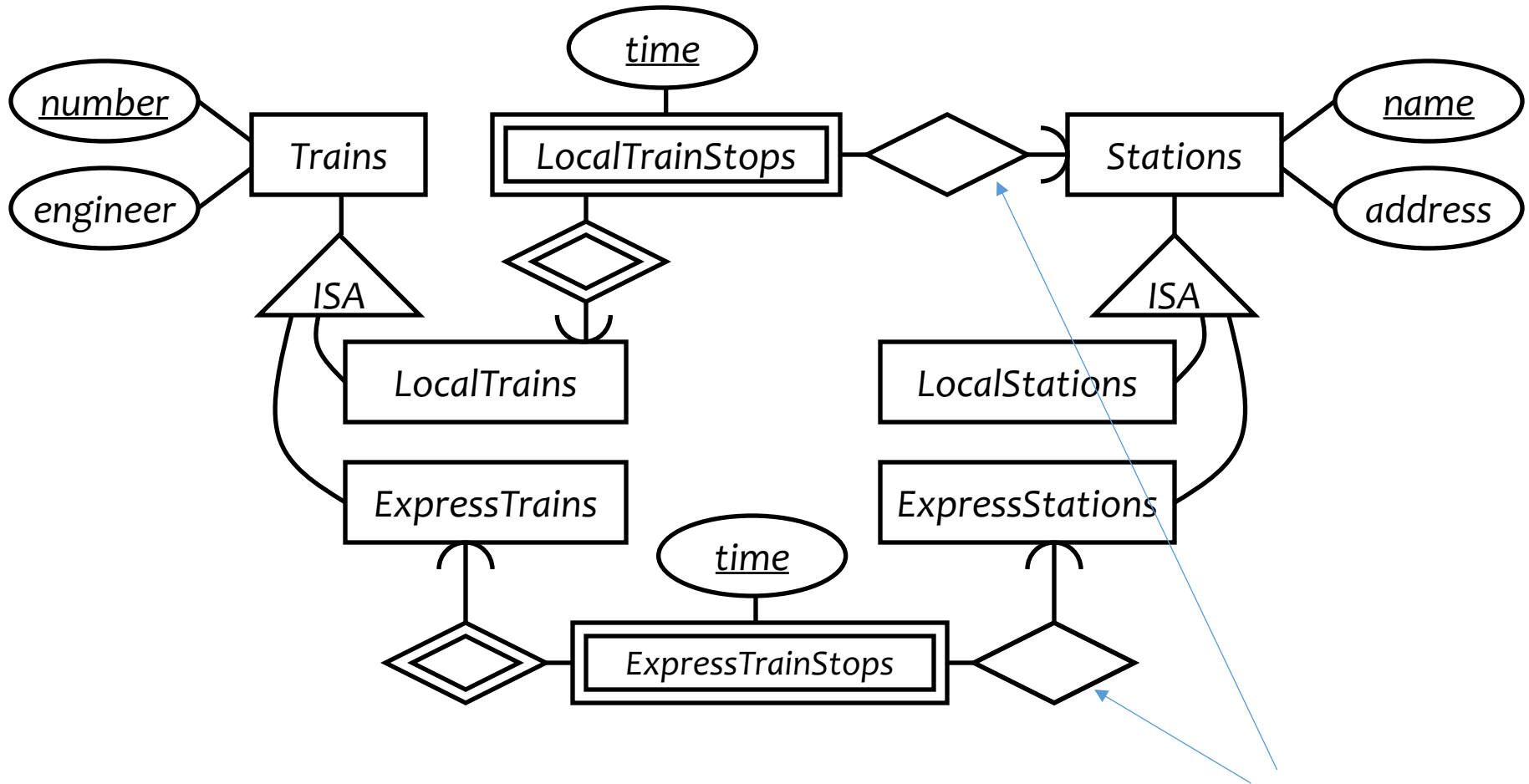  - Con: Users are scattered in different tables
- All-entities-in-one-table
  - *User (<u>uid</u>, [type, ]name, avatar)*
  - Pro: Everything is in one table
  - Con: Lots of NULL's; complicated if class hierarchy is complex

# A complete example

- 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

- Draw the ER diagram and translate into relational model

# A complete example: ER diagram



time

number
engineer
Trains

LocalTrainStops

name
address
Stations

ISA

LocalTrains

ISA

LocalStations
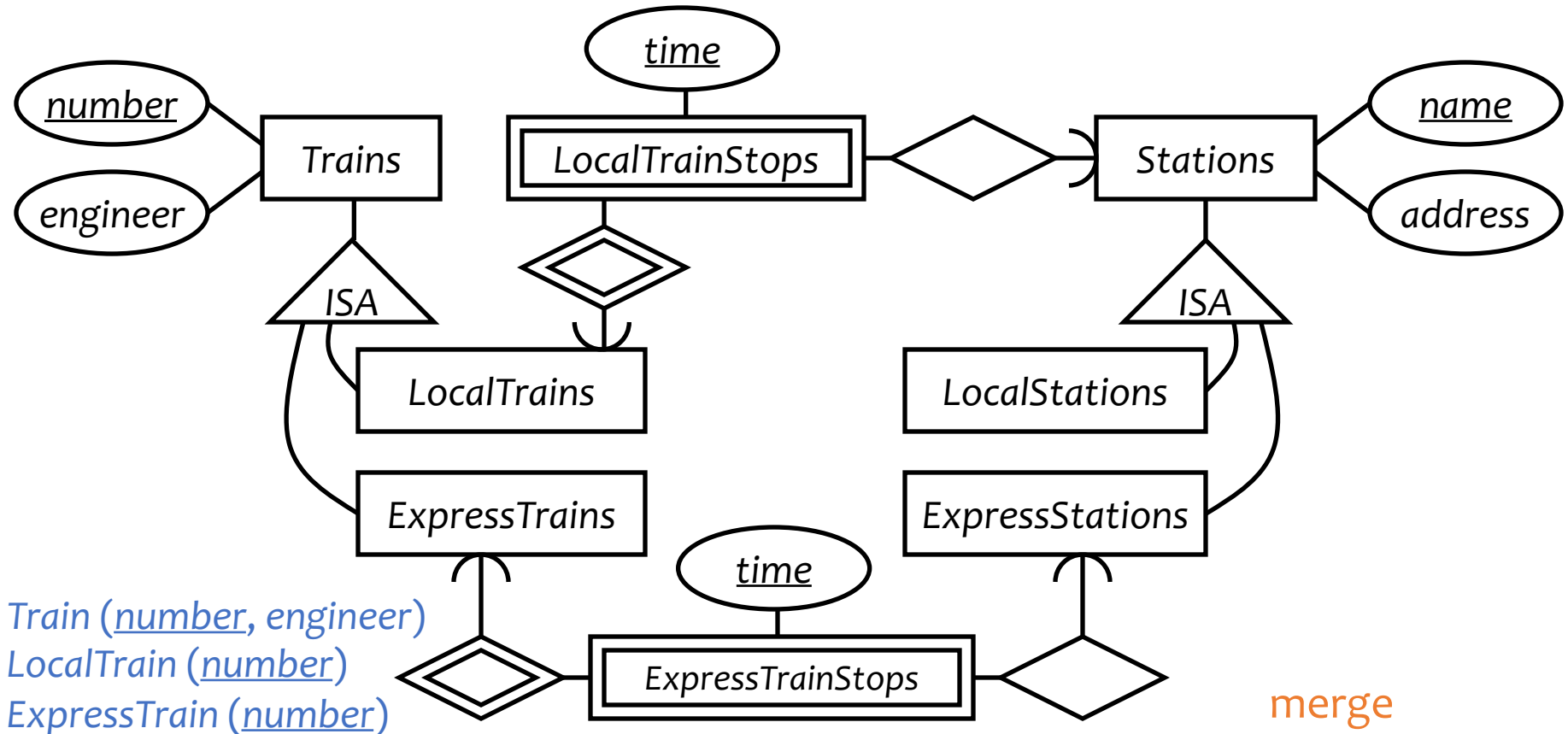
ExpressTrains

ExpressStations

time

ExpressTrainStops

Why no double diamond?

Next, Relational Design!

Train no. and time uniquely determine the stop

# A complete example: ER diagram

time

number

Trains

engineer

LocalTrainStops

name

Stations

address

ISA

LocalTrains

ISA

LocalStations

ExpressTrains

ExpressStations

time

ExpressTrainStops

*Train (number, engineer)*
*LocalTrain (number)*
*ExpressTrain (number)*

merge

*LocalTrainStop (local_train_number, time)*

*Station (name, address)*   *LocalTrainStopsAtStation (local_train_number, time, station_name)*
*LocalStation (name)*   *ExpressTrainStop (express_train_number, time)*
*ExpressStation (name)*   *ExpressTrainStopsAtStation (express_train_number, time,*

merge

*express_station_name)*

# Simplifications and refinements

- *10 to 8 relations*

*Train (number, engineer), LocalTrain (number), ExpressTrain (number)*
*Station (name, address), LocalStation (name), ExpressStation (name)*
*LocalTrainStop (local_train_number, station_name, time)*
*ExpressTrainStop (express_train_number, express_station_name, time)*

- Eliminate *LocalTrain* table
  - Redundant: can be computed as
    $$\pi_{number}(Train) - ExpressTrain$$
  - Slightly harder to check that *local_train_number* is indeed a local train number

- Eliminate *LocalStation* table
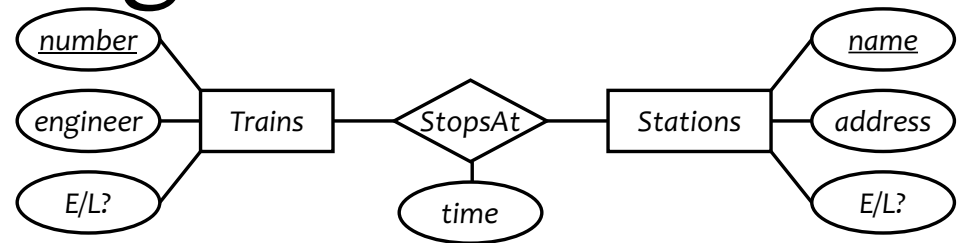  - It can be computed as $\pi_{number}(Station) - ExpressStation$

- 8 to 6 relations, still 6 (and more work for queries)!

# An alternative design

*Train (number, engineer, type)*

*Station (name, address, type)*

*TrainStop (train_number, station_name, time)*



Not a good ER diagram, but giving a simpler design, with more taks for DBMS/SQL

- Encode the type of train/station as a column rather than creating subclasses

- What about the following constraints?
  - Type must be either "local" or "express"
  - Express trains only stop at express stations
  - ☞They can be expressed/declared explicitly as database constraints in SQL (we will see soon)

- Arguably a better design because it is simpler!

**Warning:** mechanical translation procedures given in this lecture are no substitute for your own judgment!

What is a "Good" design often depends on your requirements, expected actions, and datasets

# Design principles

- KISS
  - Keep It Simple, Stupid
- Avoid redundancy
  - Redundancy wastes space, complicates modifications, promotes inconsistency
- Capture essential constraints, but don't introduce unnecessary restrictions
- Use your common sense

POOR DESIGN!

# Quick clarifications: RA questions

- What is "some"?
  - At least one
  - e.g., Drinkers frequent some bars that serve beer X
  - = Drinker D in the answer can frequent bar B1, B2, B3. At least one of them should serve X. It is okay if other bars serve X too.

- What is "only"?
  - E.g., Drinkers frequent only bars that serve beer X
  - = If drinker D in the answer frequents a bar B, then B serves X

- What is "every"?
  - E.g., Drinkers frequent every bars that serve beer X
  - = If bar B serves beer X, then drinker D in the answer frequents B.