

Relational Database Design: E/R-Relational Translation

Introduction to Databases

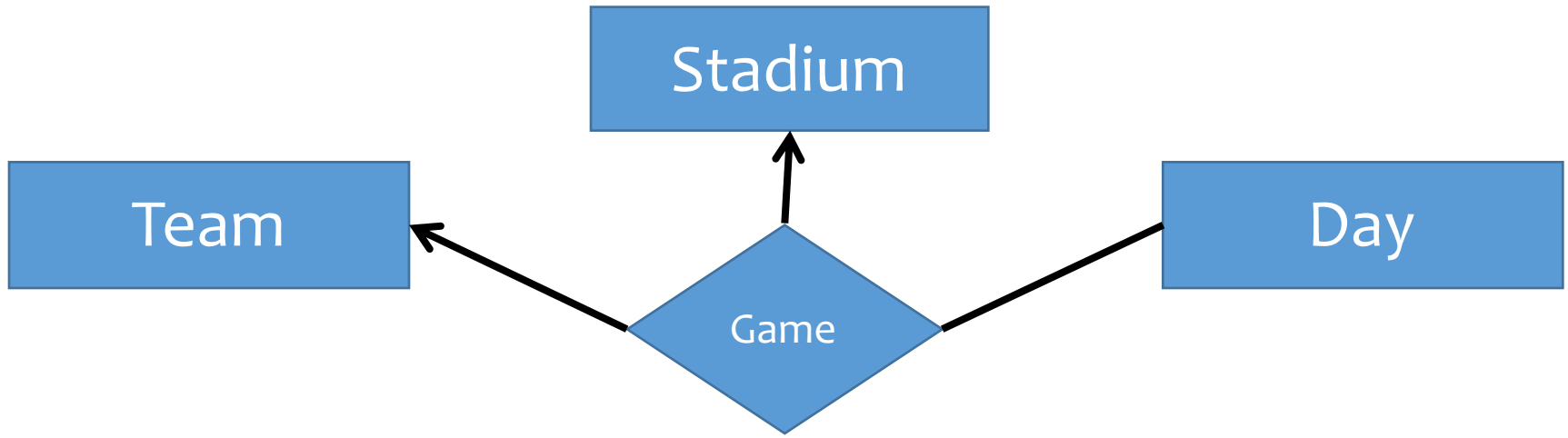
CompSci 316 Fall 2020



DUKE
COMPUTER SCIENCE

Announcements (Tue. Sep. 8)

- HW2 (written + gradiance) due tonight (11:59 pm)
 - No late days for gradiance
 - See late policy for written HWs
- Your team members due tonight
 - Then we will form remaining groups 😊



(A)

Team	Stadium	Day
T1	S1	D1
T1	S2	D1
T2	S3	D2

(B)

Team	Stadium	Day
T1	S1	D1
T2	S2	D2
T3	S3	D3

Team	Stadium	Day
T1	S1	D1
T1	S1	D2
T2	S1	D3

(C)

Team	Stadium	Day
T1	S1	D1
T2	S1	D2
T3	S1	D1

(D)

Stat...

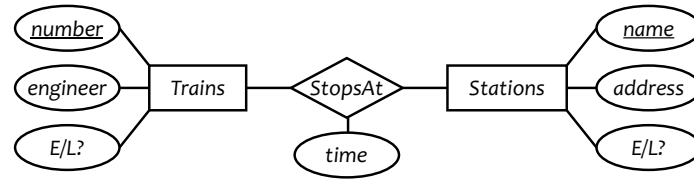
Quick clarifications: RA questions

- 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.

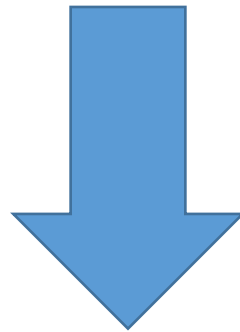
Database design steps: review

- 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 diagram



Translate it to a Relational Database

Train (*number*, *engineer*, *type*)

Station (*name*, *address*, *type*)

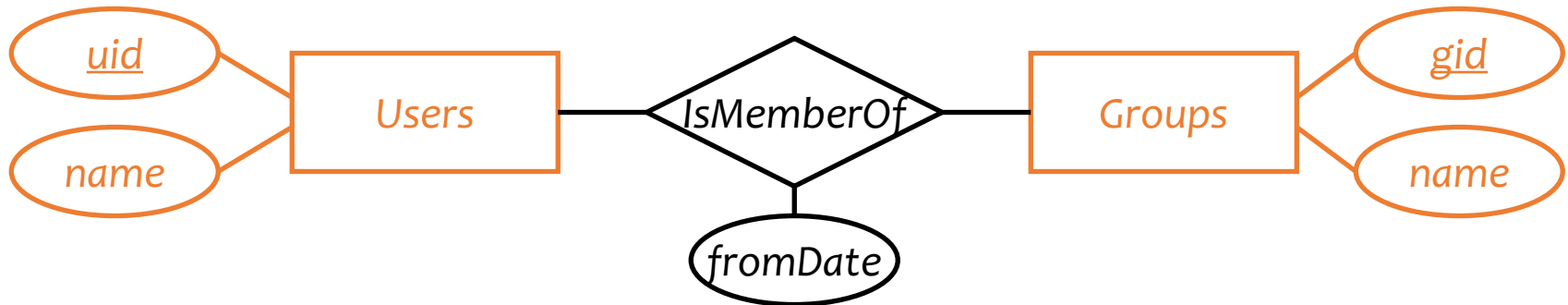
TrainStop (*train_number*, *station_name*, *time*)

E/R model: concepts

- 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

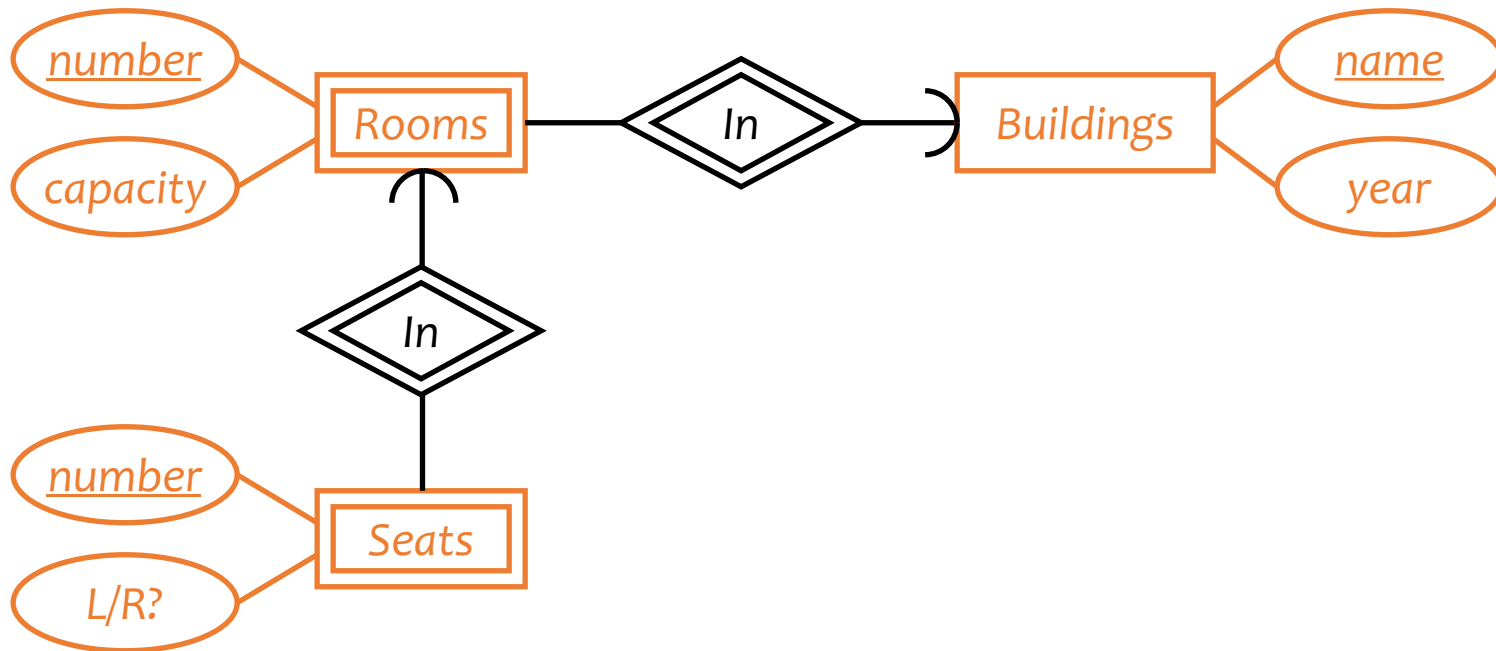


User (uid, name)

Group (gid, name)

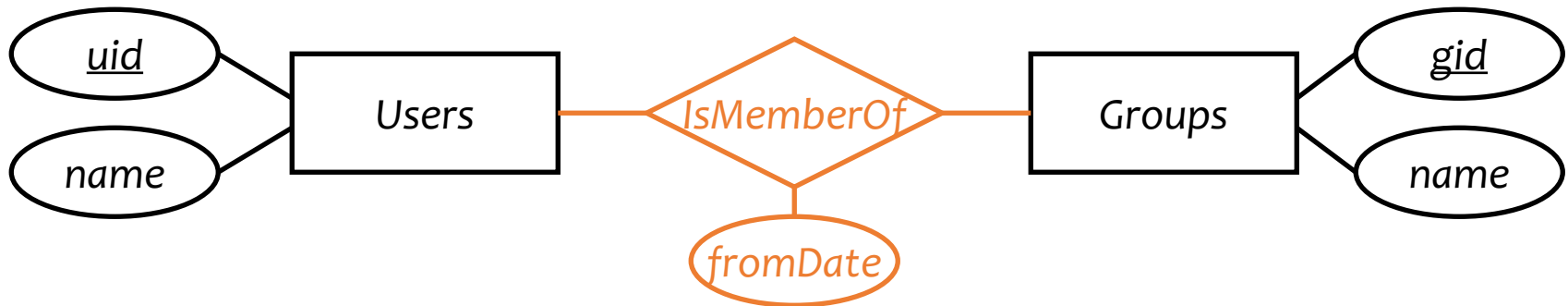
Translating weak entity sets

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



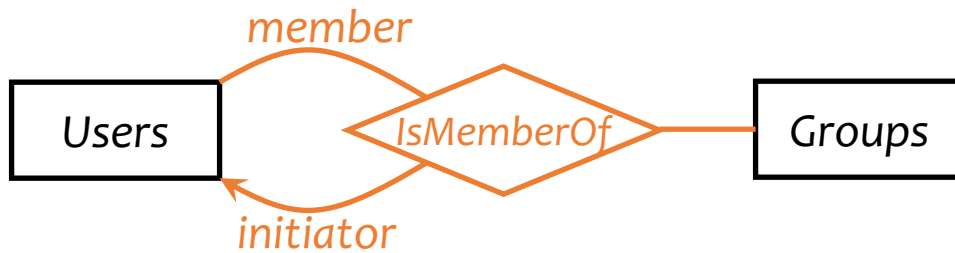
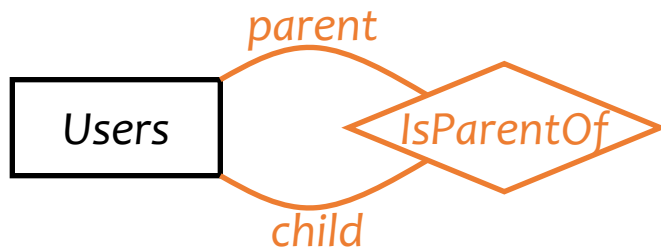
Translating relationship sets

- 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



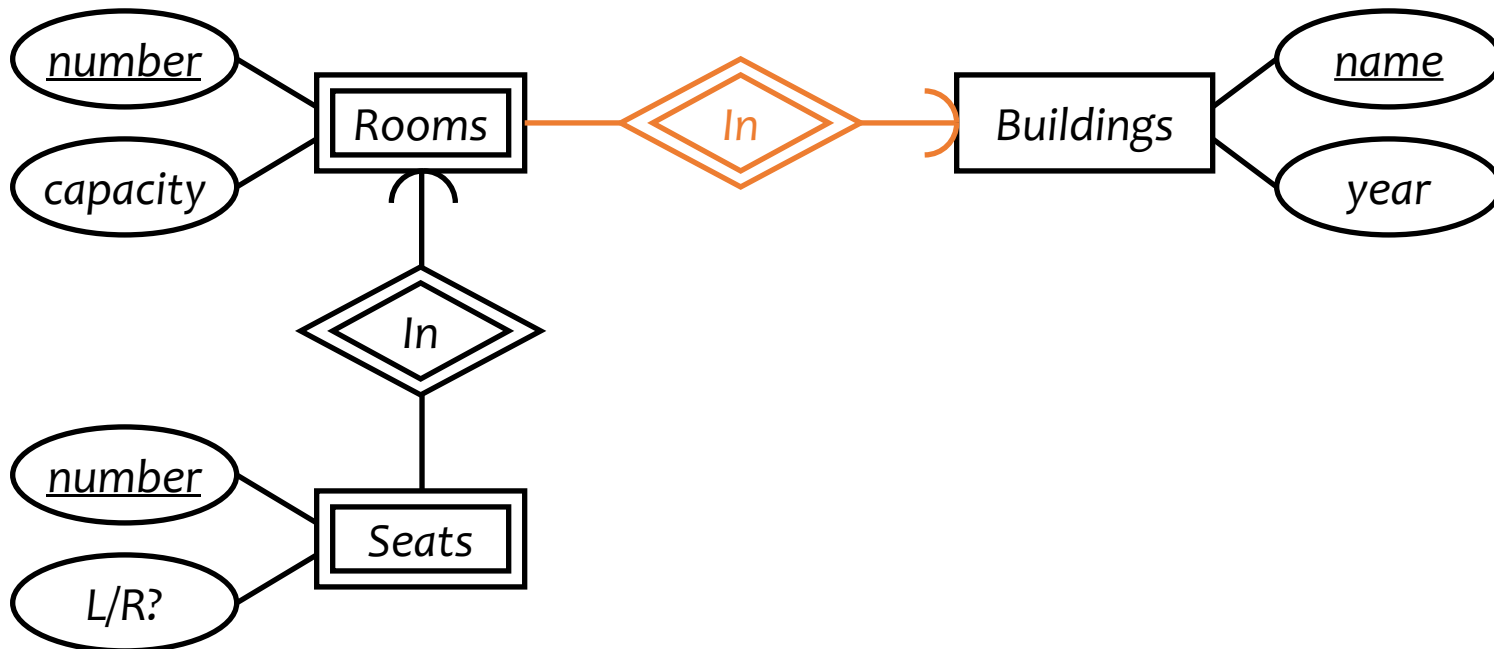
How do the keys change if you have arrow to Users, Groups, or both?

More examples

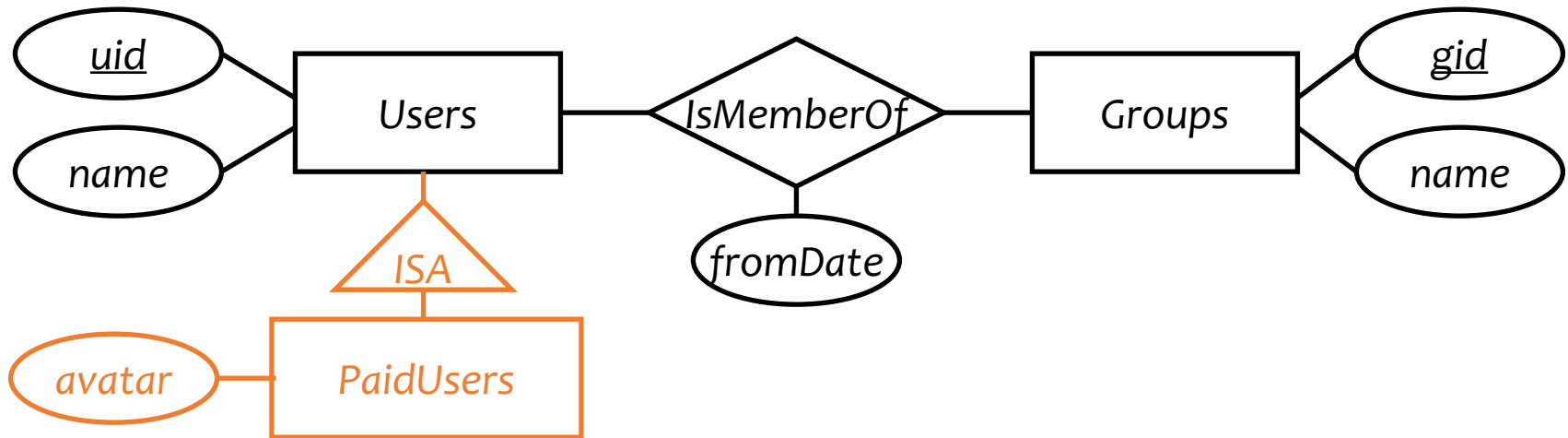


Translating double diamonds?

- 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

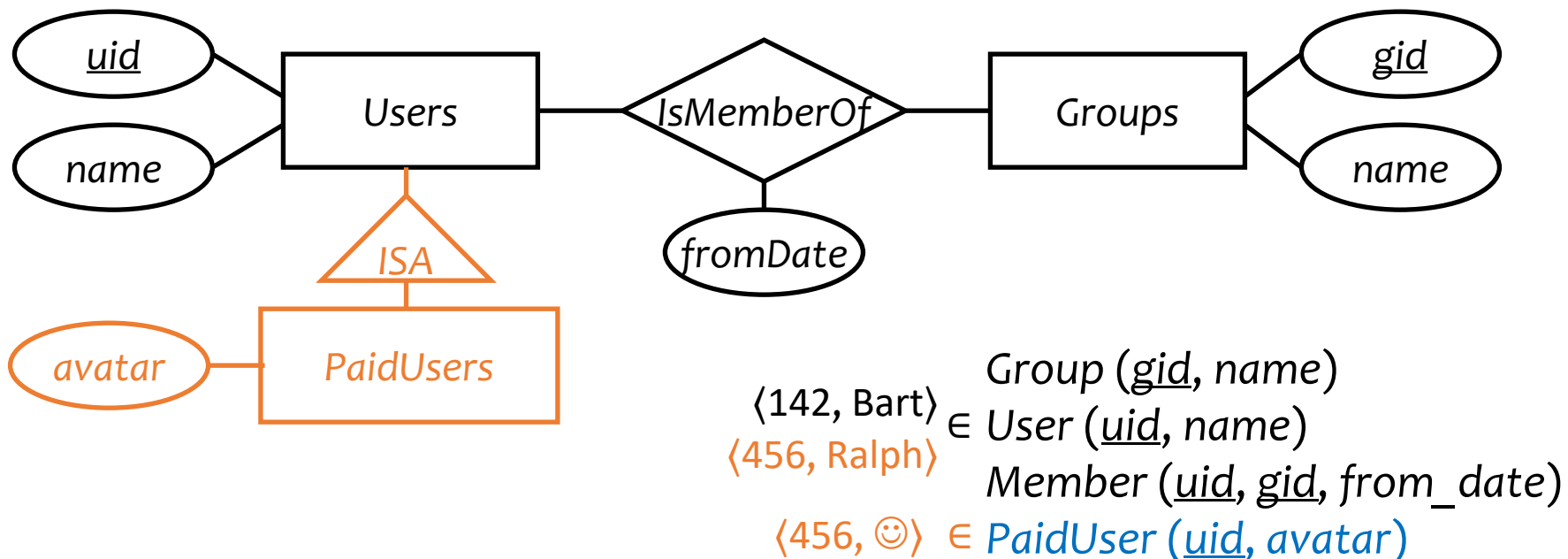


Translating subclasses & ISA



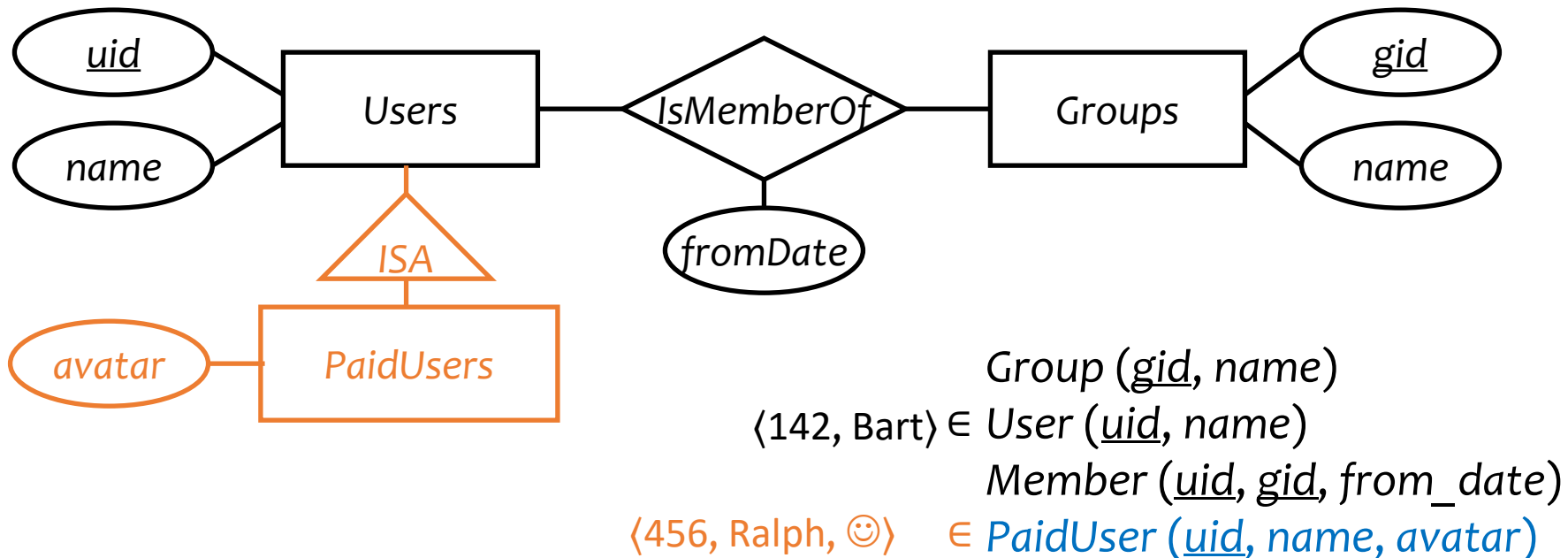
Translating subclasses & ISA: approach 1

- **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



Translating subclasses & ISA: approach 2

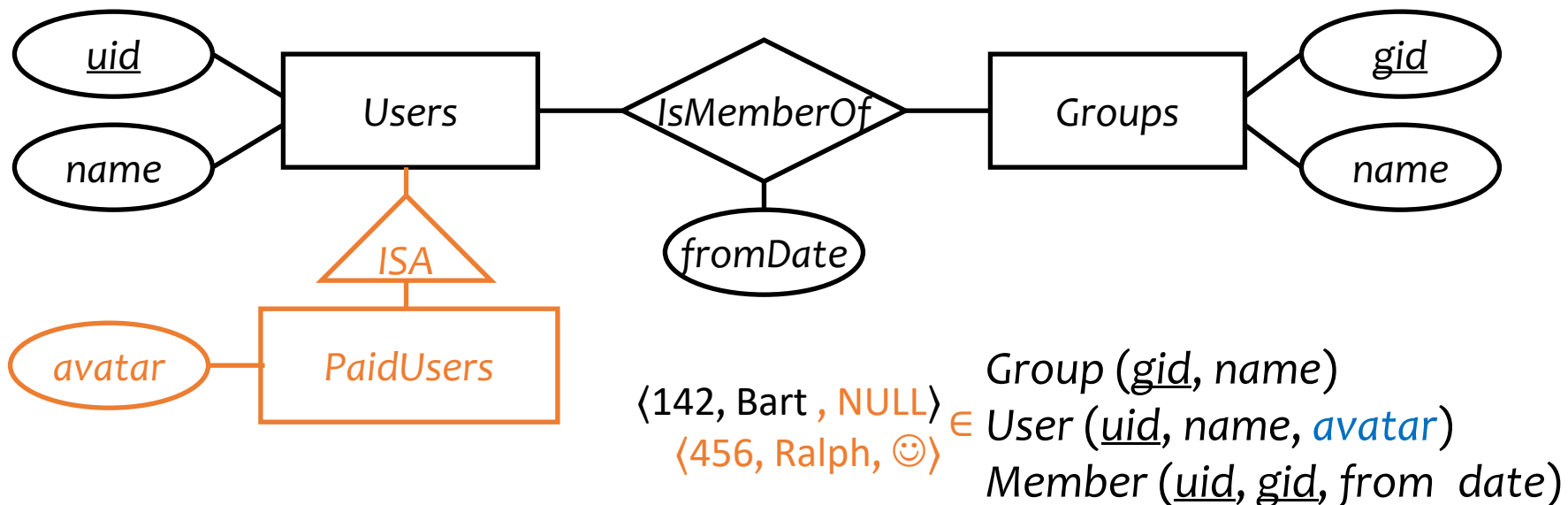
- **Entity-in-most-specific-class** approach (“OO style”)
 - An entity is only represented in one table (the most specific entity set to which the entity belongs)
 - A table includes the attributes attached to the corresponding entity set, plus all inherited attributes



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



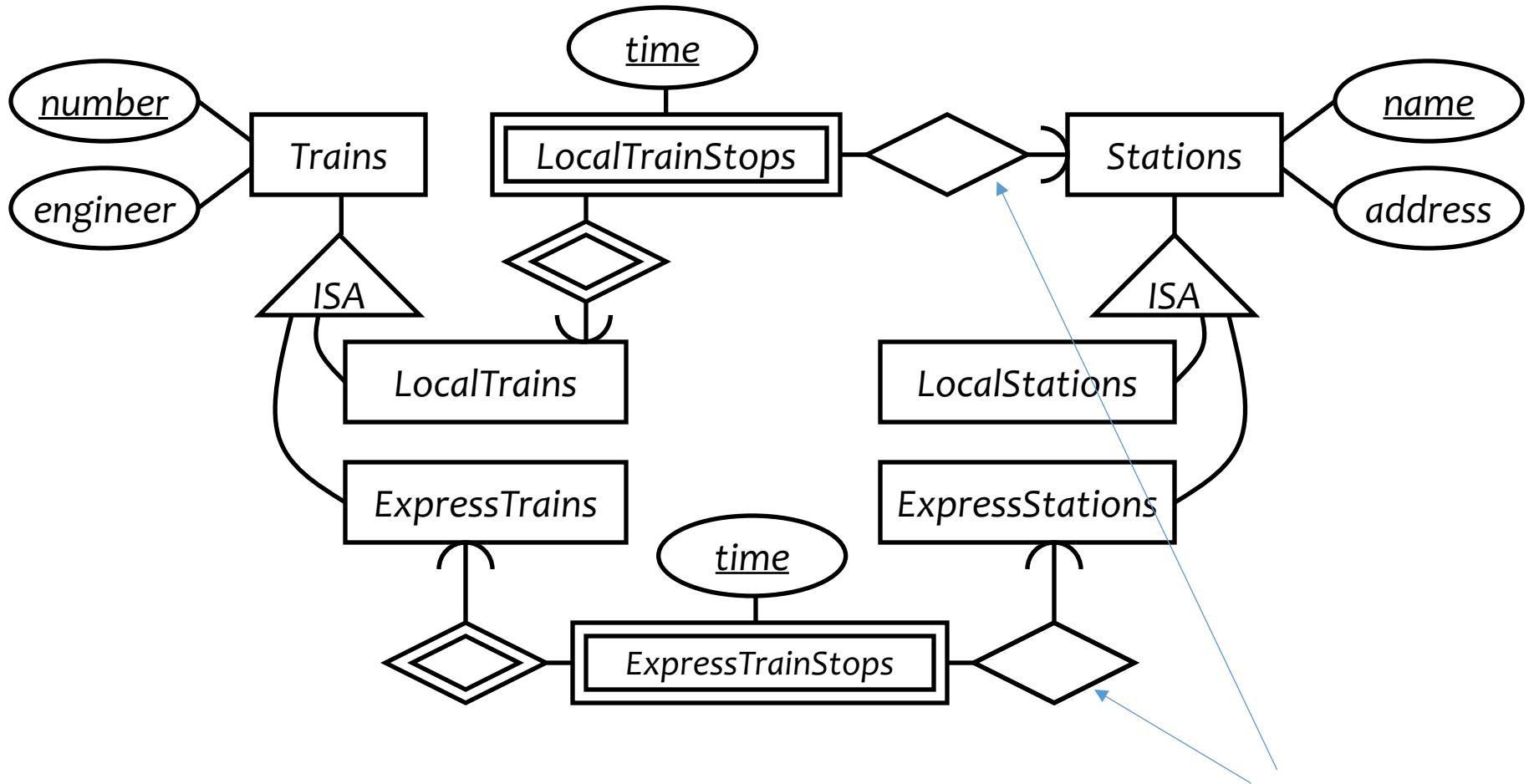
Comparison of three approaches

- Entity-in-all-superclasses
 - *User* (uid, name), *PaidUser* (uid, avatar)
 - Pro:
 - Con:
- Entity-in-most-specific-class
 - *User* (uid, name), *PaidUser* (uid, name, avatar)
 - Pro:
 - Con:
- All-entities-in-one-table
 - *User* (uid, [type,]name, avatar)
 - Pro:
 - Con:

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

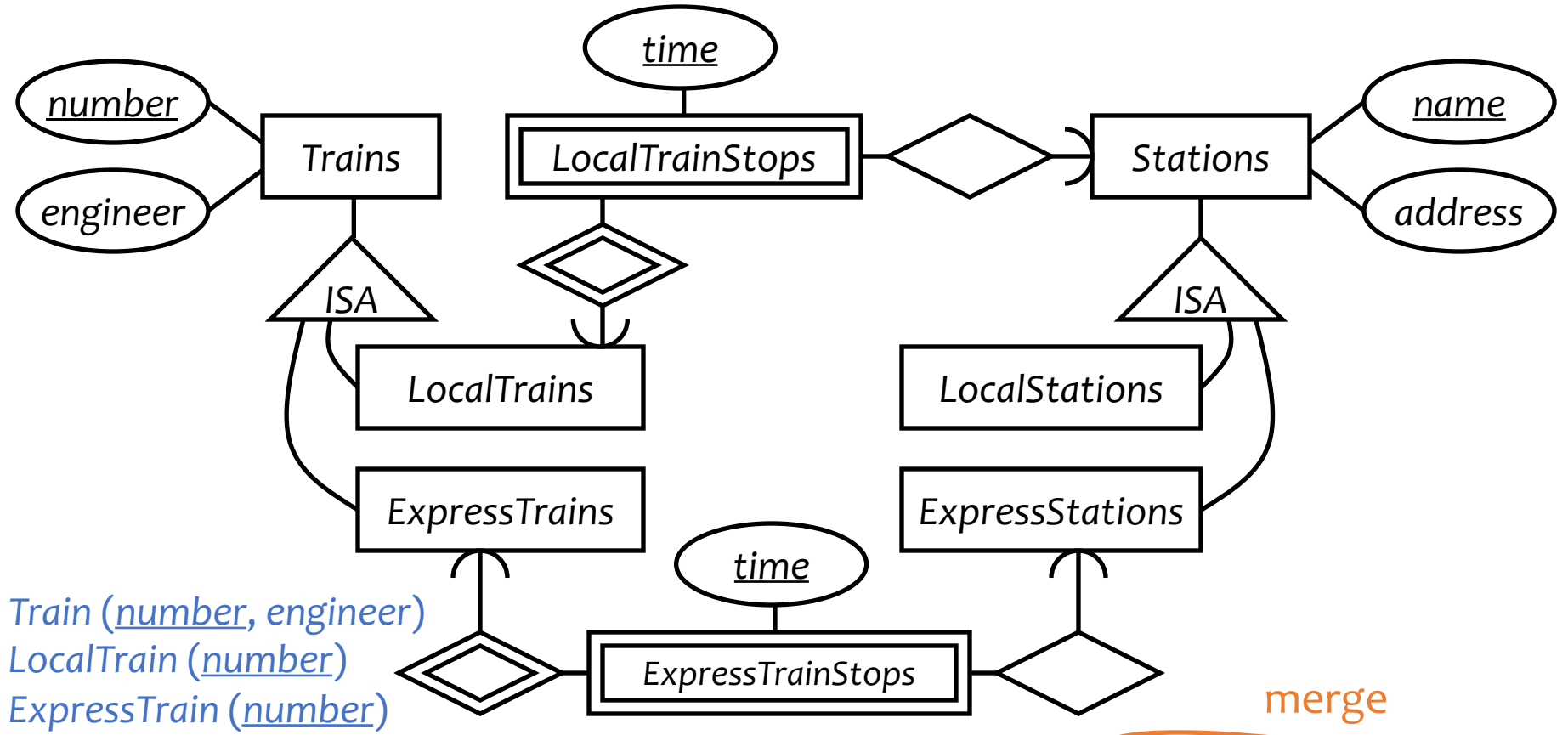


Why no double diamond?

Next, Relational Design!

Train no. and time uniquely determine the stop

A complete example: ER diagram



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

Station (name, address)
 LocalStation (name)
 ExpressStation (name)

LocalTrainStop (local_train_number, time)
 LocalTrainStopsAtStation (local_train_number, time, station_name)
 ExpressTrainStop (express_train_number, time)
 ExpressTrainStopsAtStation (express_train_number, time, express_station_name)

merge

merge

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 $\pi_{number}(Station) - ExpressStation$

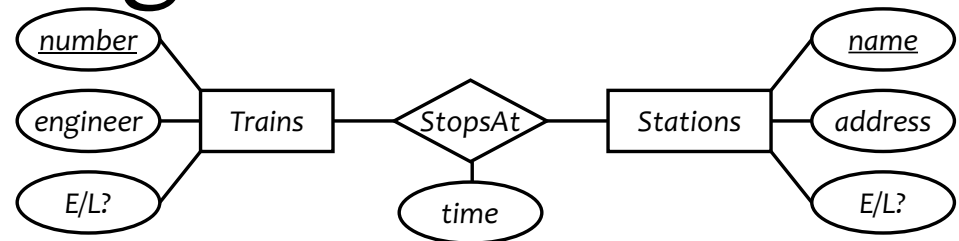
- 8 to 6 relations, still 6 (and more work for queries)!

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 tasks 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

