SFU

Database Systems I

CMPT 354 Summer 2024

Zhengjie Miao

24 May 2024

Announcement

- RA mini tutorial
  - Be careful with rename
  - All queries are transient except view definition

There is an another radb syntax sheet

- Will release Assignment 2 May 26 midnight
  - Due Jun 7
  - Slightly reduced the workload
  - Only need to setup PostgreSQL
  - Web interface will also be available

SQL lectures

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

Outline

- E/R Diagram Design Cases
- E/R – Relational Model Translation

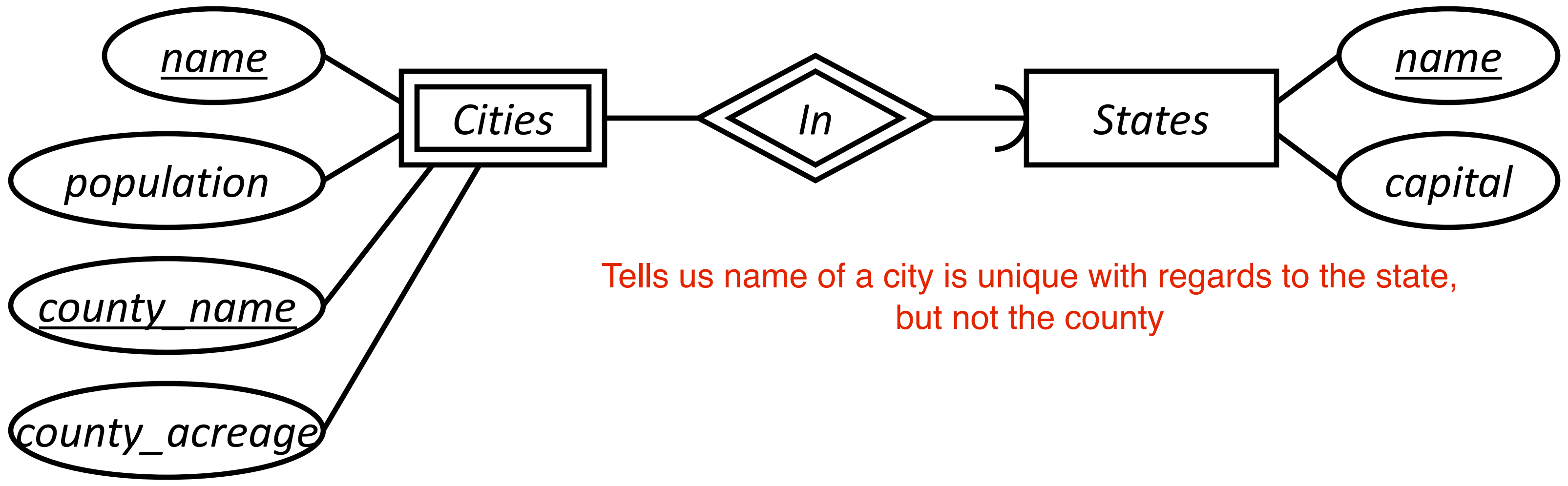
Case study 1

- Design a database representing cities, counties, and states
  - For each state, record name and capital (city)
  - For each county, record name, acreage, and state it’s in
  - For each city, record name, population, and county/state it belongs to
- 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 in a single county

    Multiple cities in a county?
  - A county is always in a single state

City = capitol city

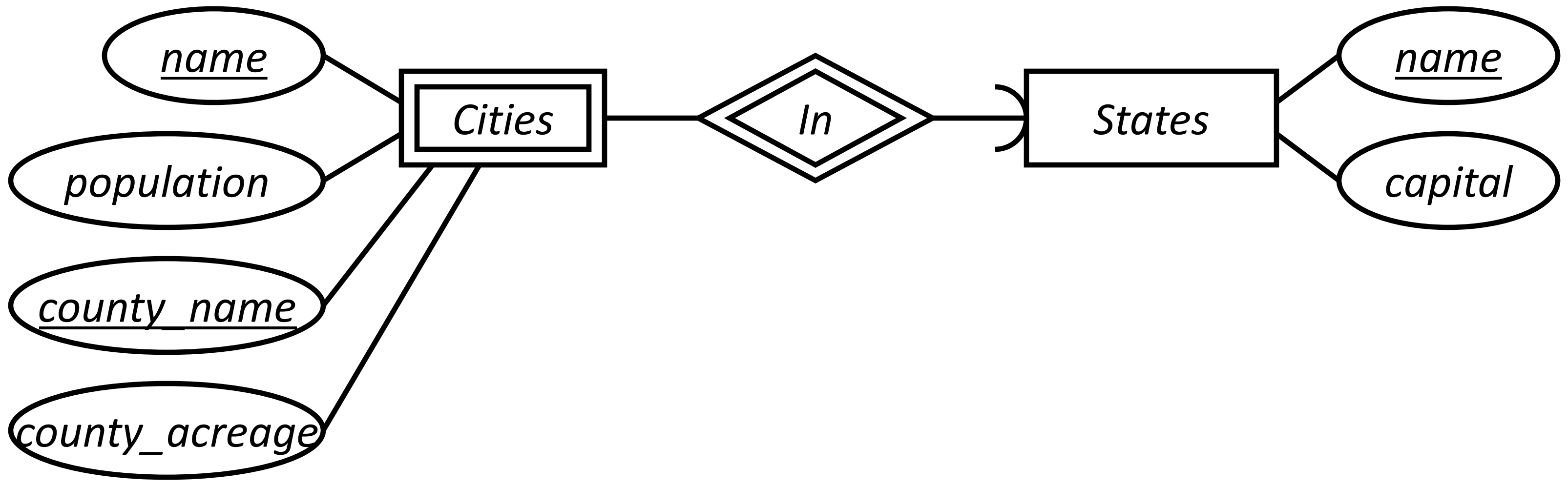
Case study 1: first design



Tells us name of a city is unique with regards to the state, but not the county

Since we can have multiple cities in one county, we are storing the acreage multiple times

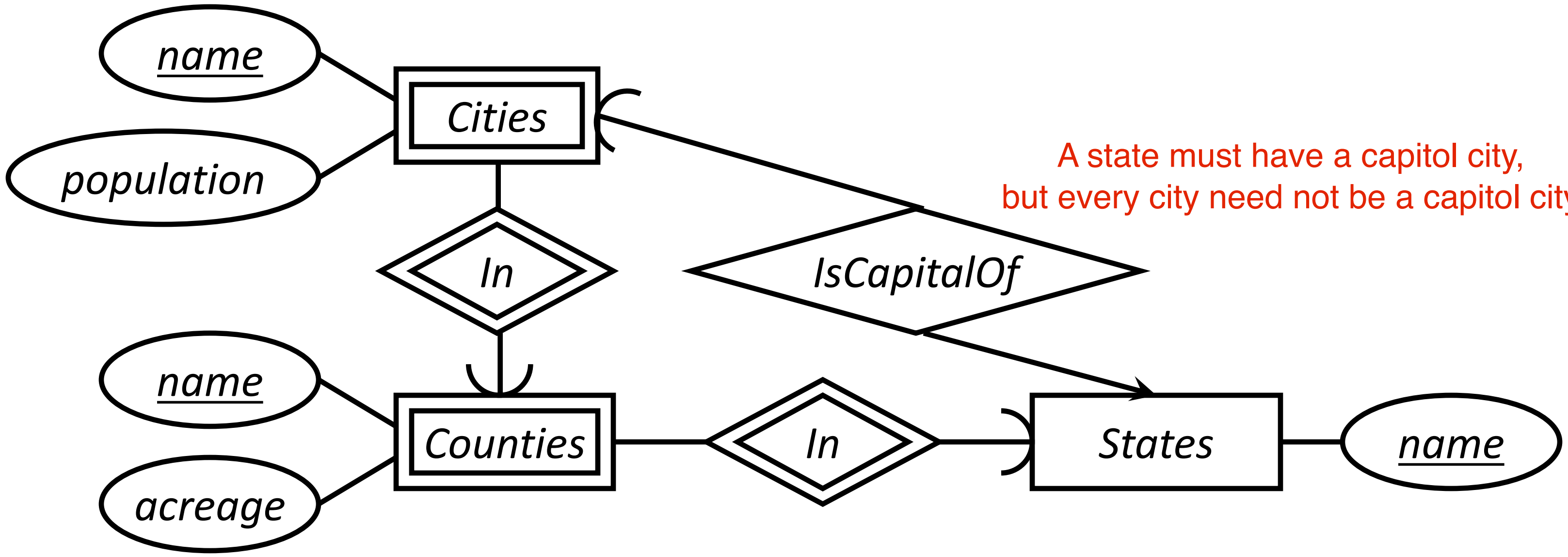
Case study 1: first design



- County acreage 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

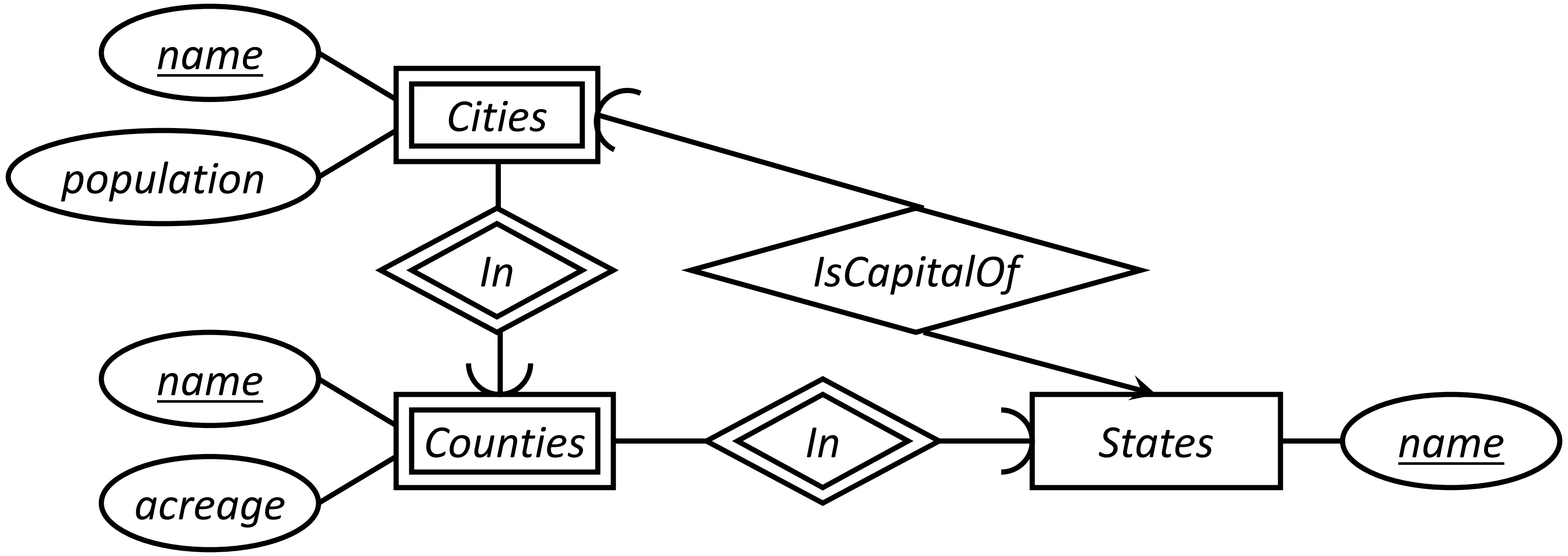
If you do not make the updates to the county information consistent amongst all records with this information, you could be working with incorrect data

Case study 1: second design



A state must have a capitol city,
but every city need not be a capitol city

Case study 1: second design



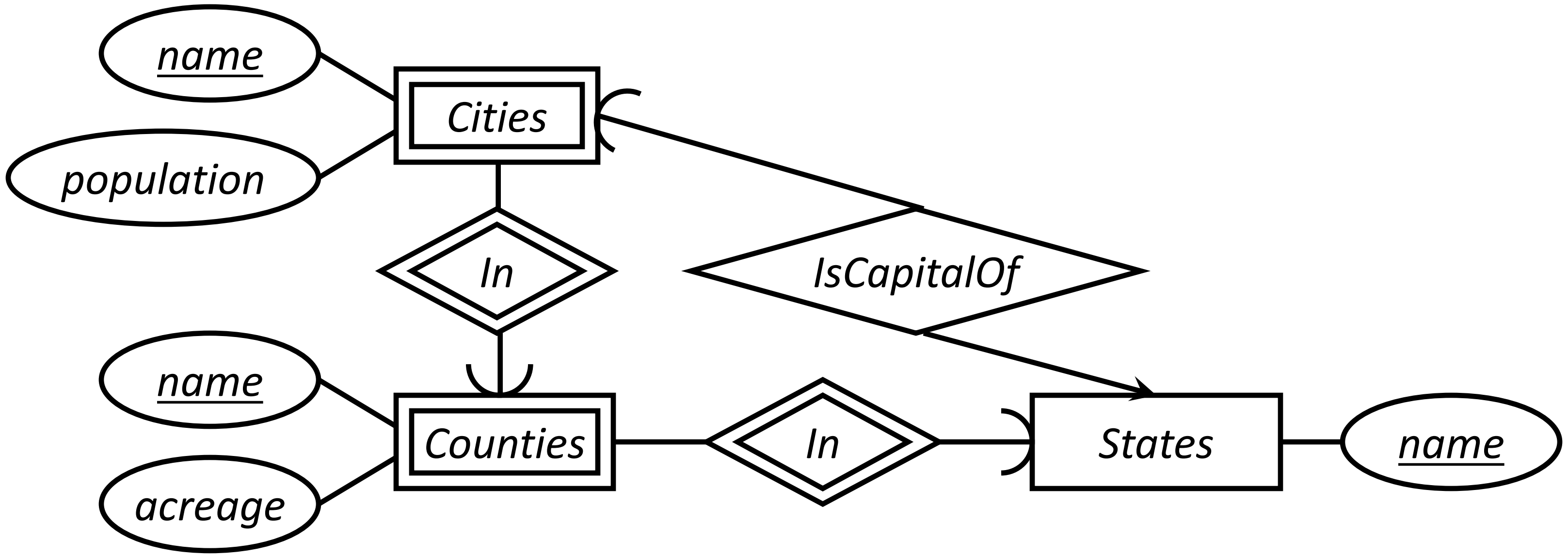
- Technically, nothing in this design prevents a city in state X from being the capital of another state Y, but oh well…

Ask professor to clarify how this design can allow it

General steps

1. Recognize entity sets
2. Recognize relationship sets and participating entity sets
3. Recognize attributes of entity and relationship sets
4. Define relationship types and existence dependencies
5. Define general cardinality constraints and keys
6. Draw diagram

Case study 1: second design



Make sure design does not have arbitrary constraints

Design a database representing cities, counties, and states

- For each state, record name and capital (city)
- For each county, record name, acreage, and state it's in
- For each city, record name, population, and county/state it belongs to

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 in a single county
- A county is always in a single state

Case study 2

Station

Express Local

Train

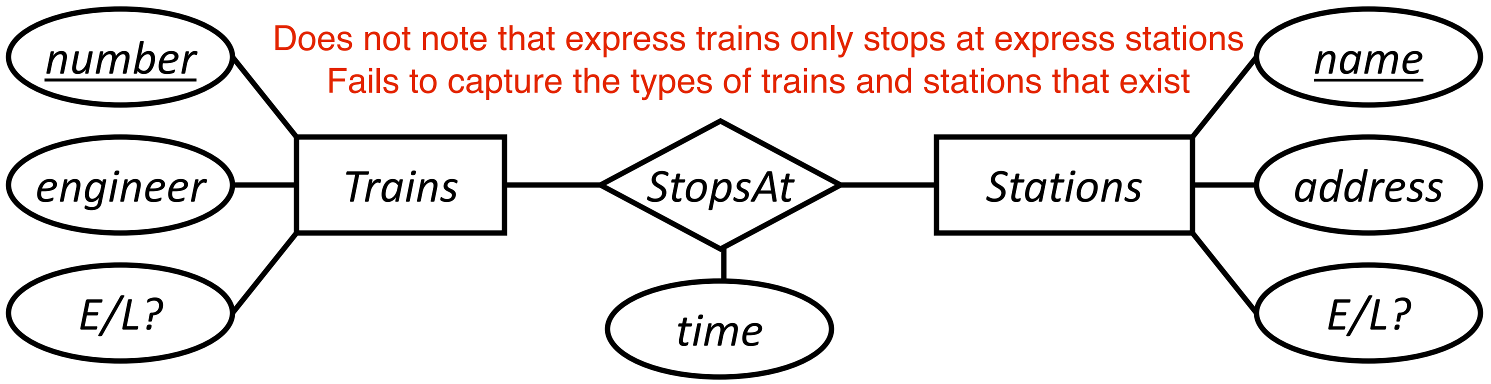
Express Local

- 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 every day

Case study 2

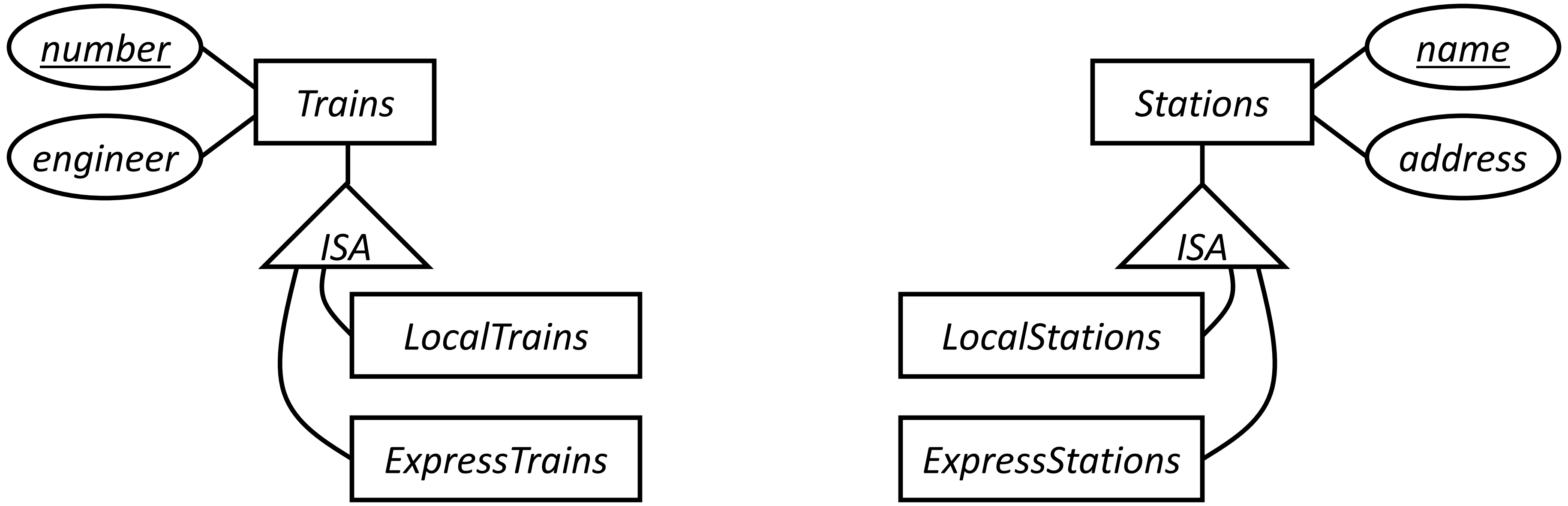
- Design a database consistent with the following:
  - A station has a unique name and an address, and is either an express station or a local station
  - A train has a unique number and an engineer, and is either an express train or a local train
  - A local train can stop at any station
  - An express 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 every day

Case study 2: first design


number
engineer
E/L?
Trains
StopsAt
time
Stations
name
address
E/L?
Does not note that express trains only stops at express stations
Fails to capture the types of trains and stations that exist

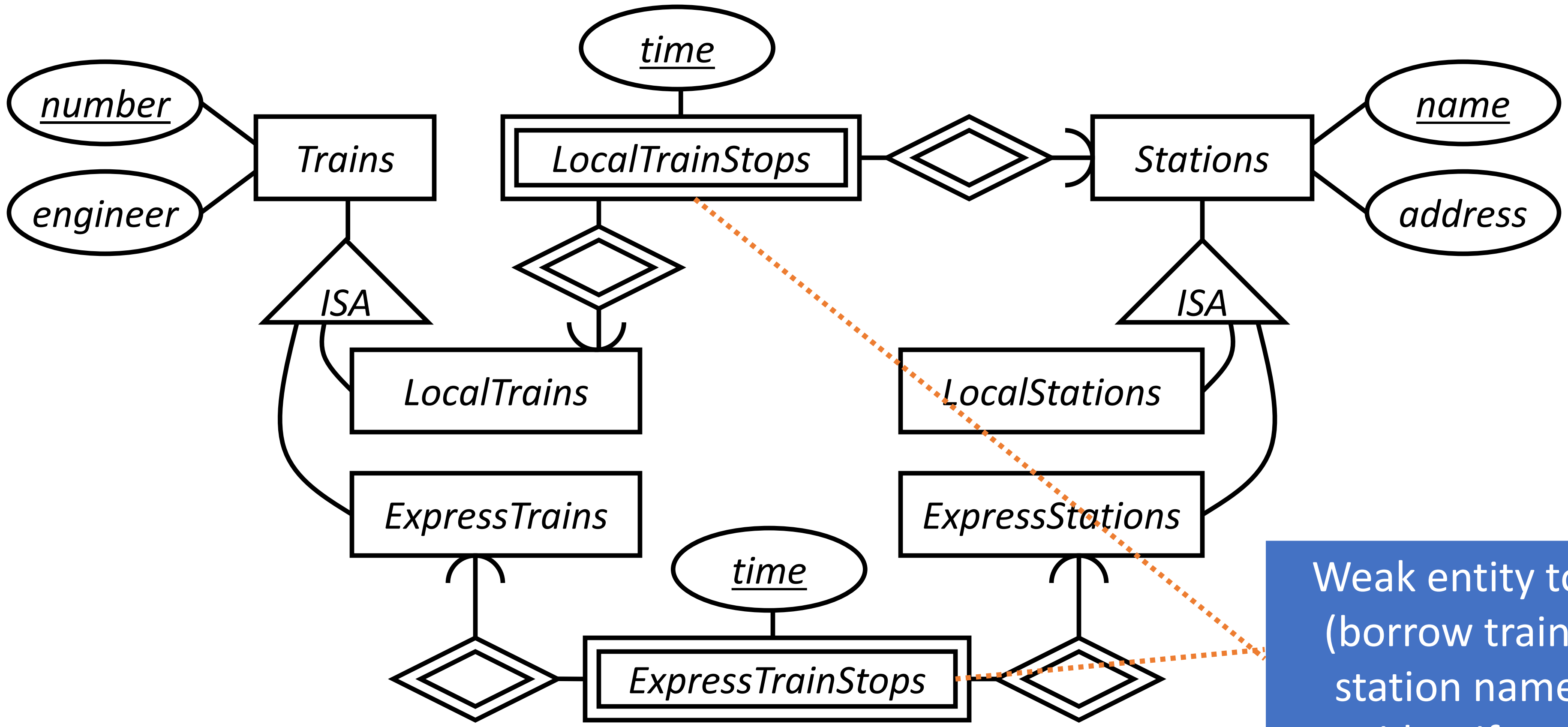
- Nothing in this design prevents express trains from stopping at local stations
  - ☞We should capture as many constraints as possible
- A train can stop at a station only once during a day
  - ☞We should not introduce unintended constraints

Case study 2: second design



- A station is either an express station or a local station
- A train is either an express train or a local tarin
- How to model stop schedule?

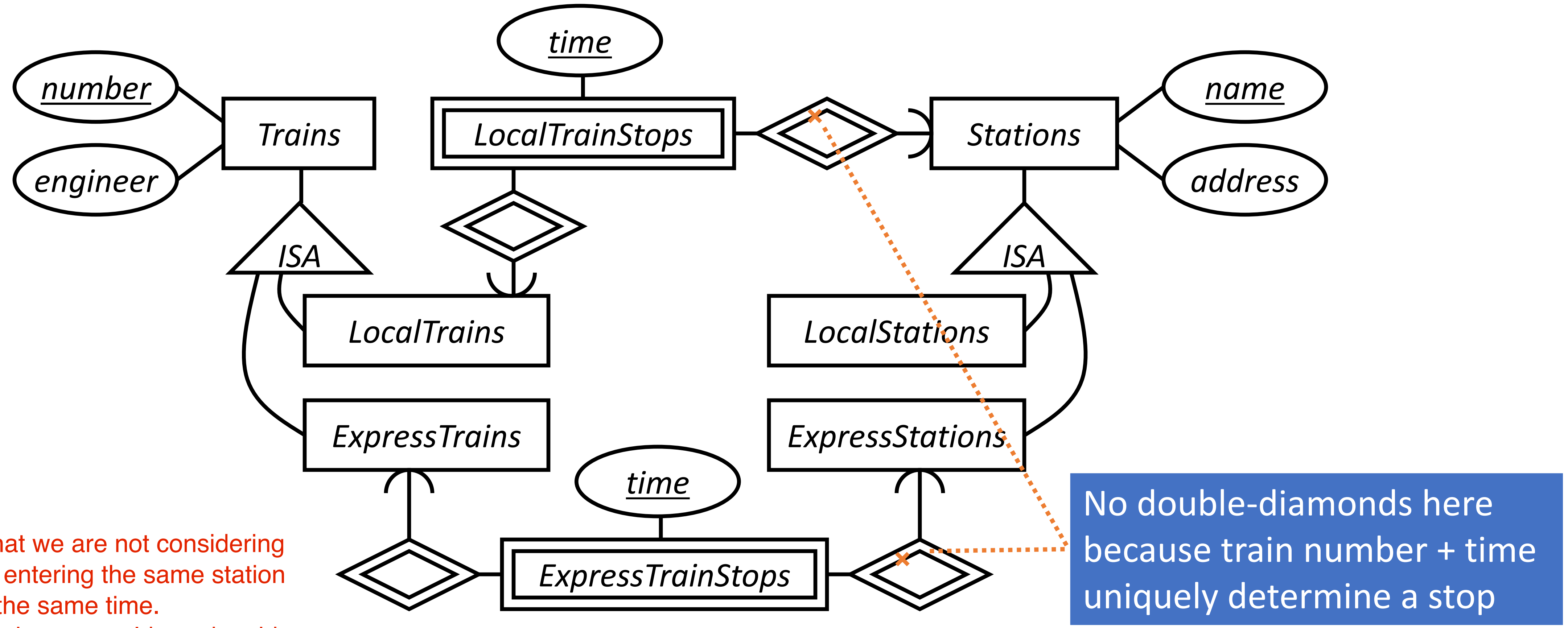
Case study 2: second design



Weak entity to model stops (borrow train number and station name to uniquely identify a train stop)

Can we represent every relation as a weak entity set?
no

Case study 2: second design

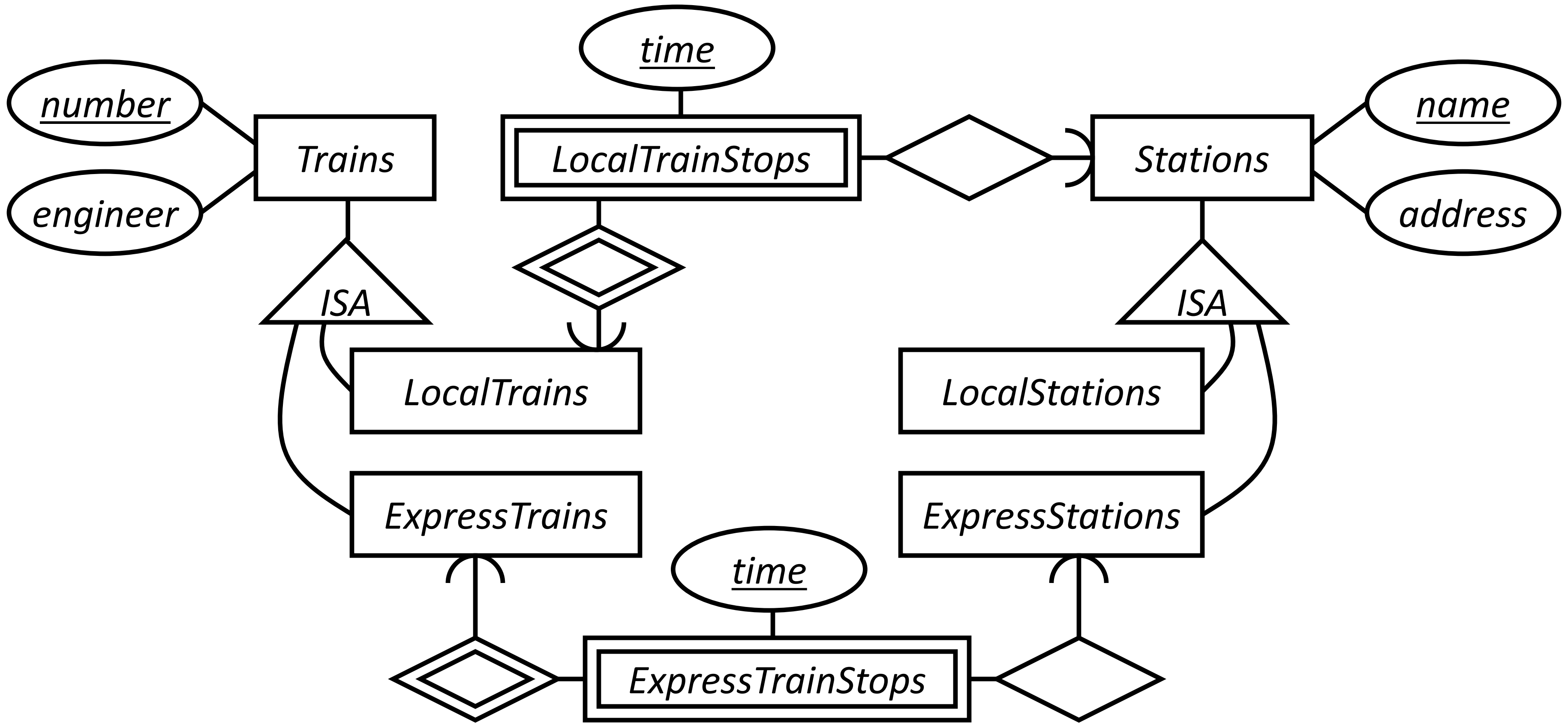


No double-diamonds here because train number + time uniquely determine a stop

The issue is that we are not considering multiple trains entering the same station at the same time.
For this scenario, we would need an id for each train in order to uniquely identify them

Is the extra complexity worth it?

Case study 2: second design (final draft)



Is there a software which can take your design and analyze it?

Outline

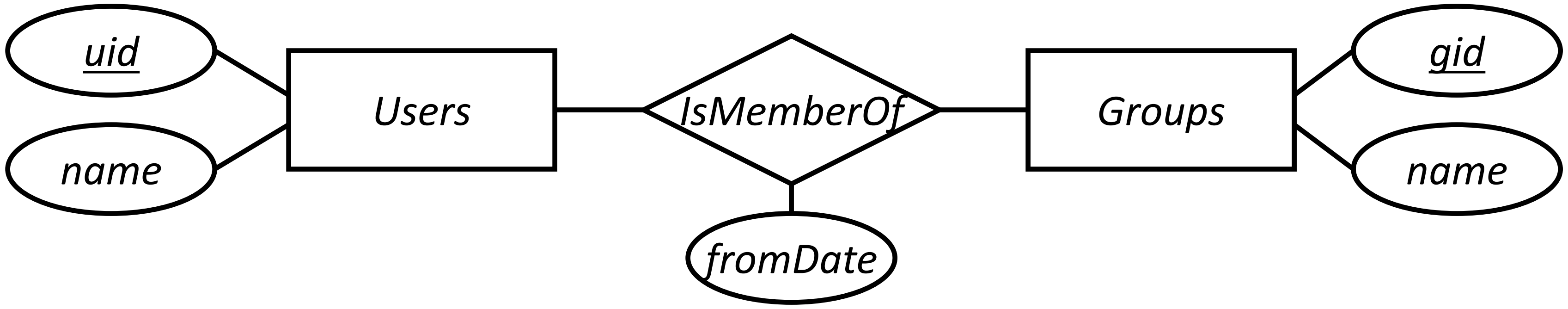
- E/R Diagram Design Cases
- **E/R – Relational Model Translation**

Database design steps

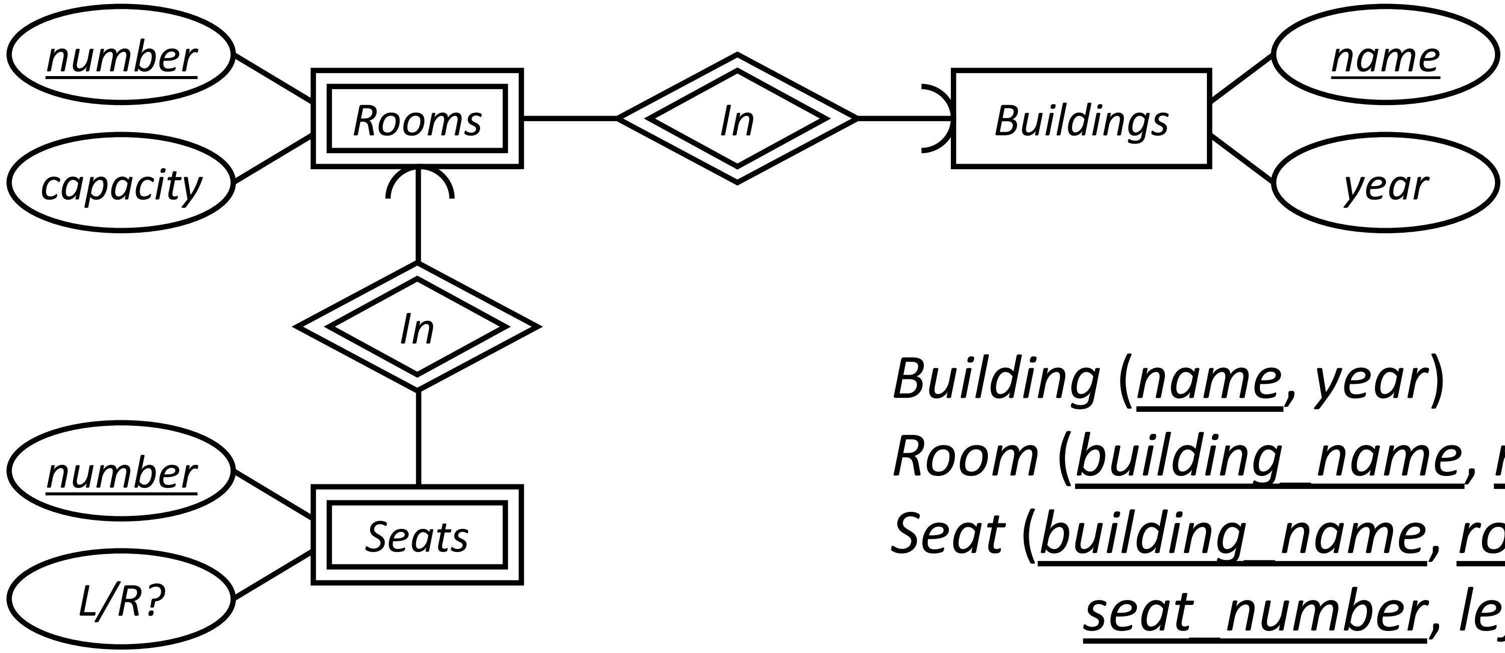
- Understand the real-world domain being modeled
- Specify it using a database design model (e.g., E/R)
- Translate specification to the data model of DBMS (e.g., relational)
- Create DBMS schema

☞Next: translating E/R design to relational schema

E/R Model to relational schema



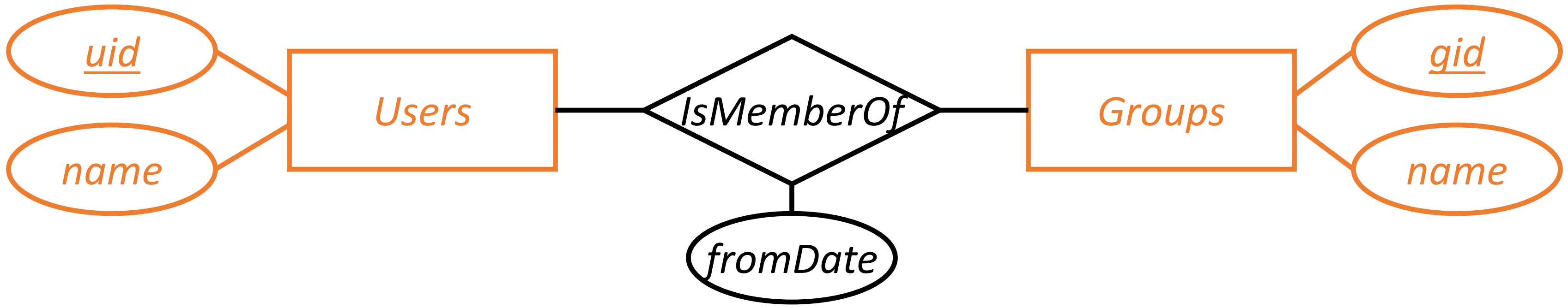
User (*<u>uid</u>*, *name*)
Group (*<u>gid</u>*, *name*)
Member (*<u>uid</u>*, *<u>gid</u>*, *fromDate*)



Building (*<u>name</u>*, *year*)
Room (*<u>building_name</u>*, *<u>room_number</u>*, *capacity*)
Seat (*<u>building_name</u>*, *<u>room_number</u>*,
<u>seat_number</u>, *left_or_right*)

Translating entity sets

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



User (<u>uid</u>, name)

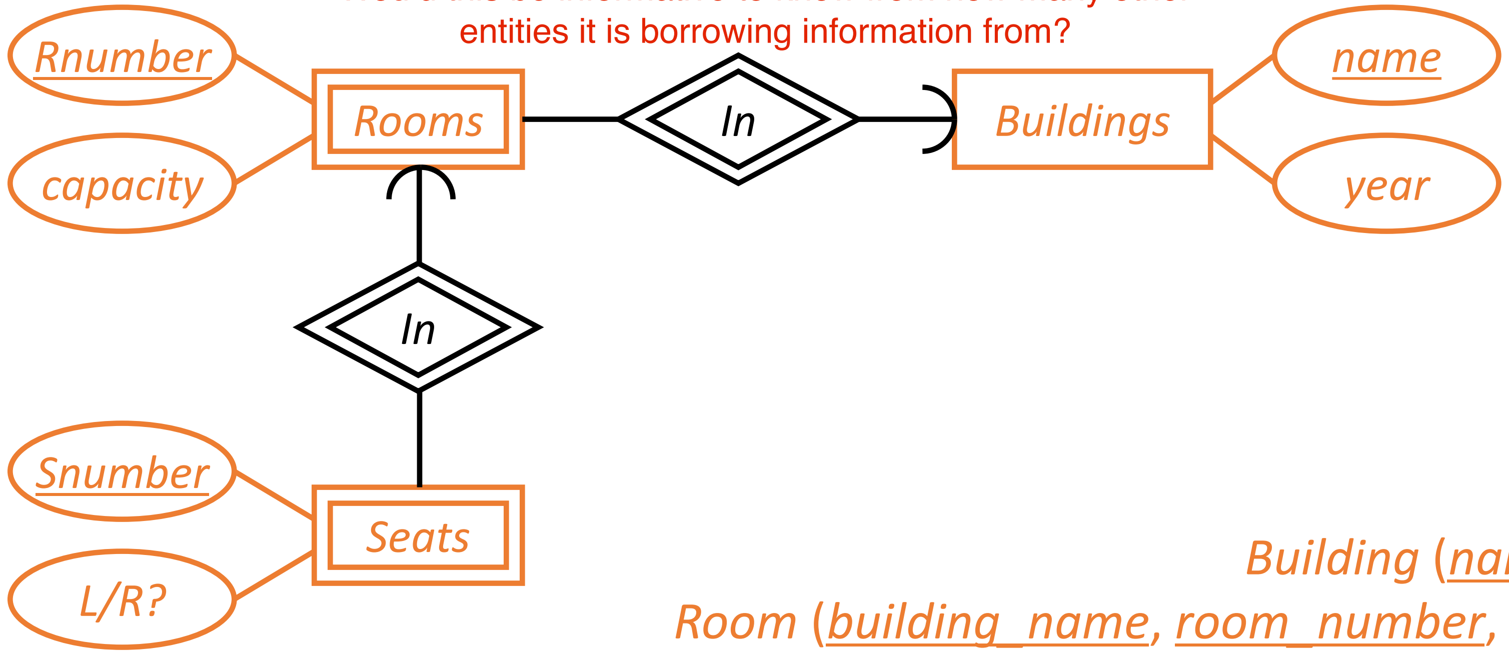
Group (<u>gid</u>, name)

When you convert an ER diagram to a concrete design, you must consider the type of the attribute

Translating weak entity sets

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

Do we have a way to measure how "weak" an entity is?
Would this be informative to know from how many other entities it is borrowing information from?



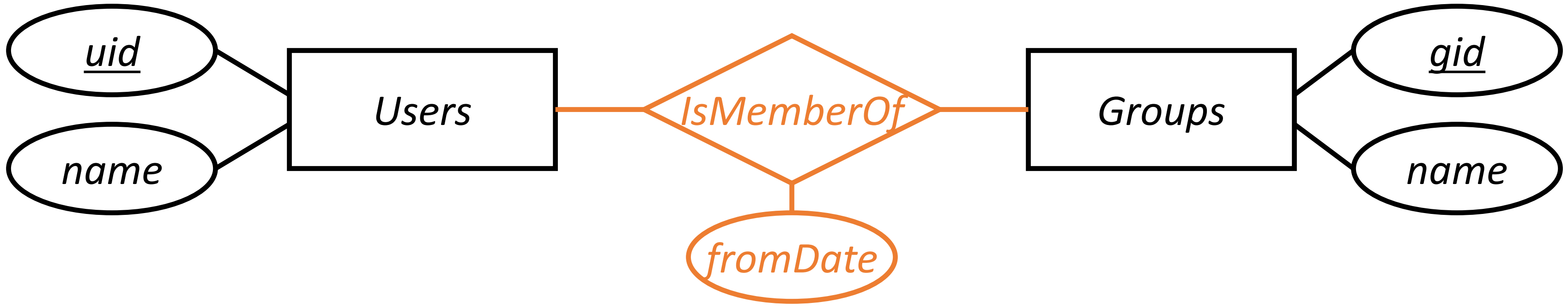
Building (<u>name</u>, year)

Room (<u>building_name</u>, <u>room_number</u>, capacity)

Seat (<u>building_name</u>, <u>room_number</u>, <u>seat_number</u>, 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

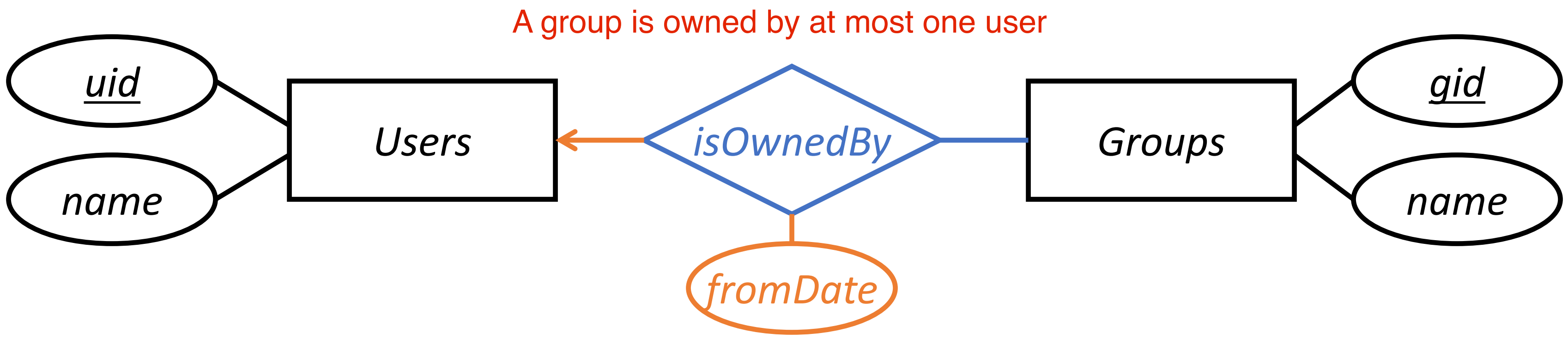


Member (<u>uid</u>, <u>gid</u>, fromDate)

- If we can deduce the general cardinality constraint for a connected entity set E, then take the primary key attributes for E
- Otherwise, choose primary key attributes of each component entity

Translating relationship sets

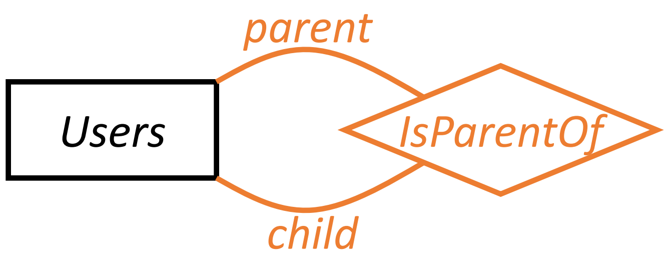
- A relationship set translates to a table
  - Keys of connected entity sets → columns
  - Attributes of the relationship set (if any) → columns
  - Multiplicity of the relationship set determines the key of the table



Owner (uid, <u>gid</u>, fromDate)

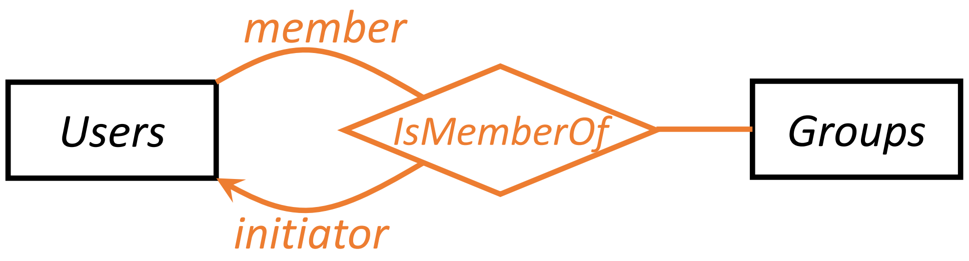
- If we can deduce the general cardinality constraint for a connected entity set E, then take the primary key attributes for E
- Otherwise, choose primary key attributes of each component entity

More examples



Parent (parent_uid, child_uid)

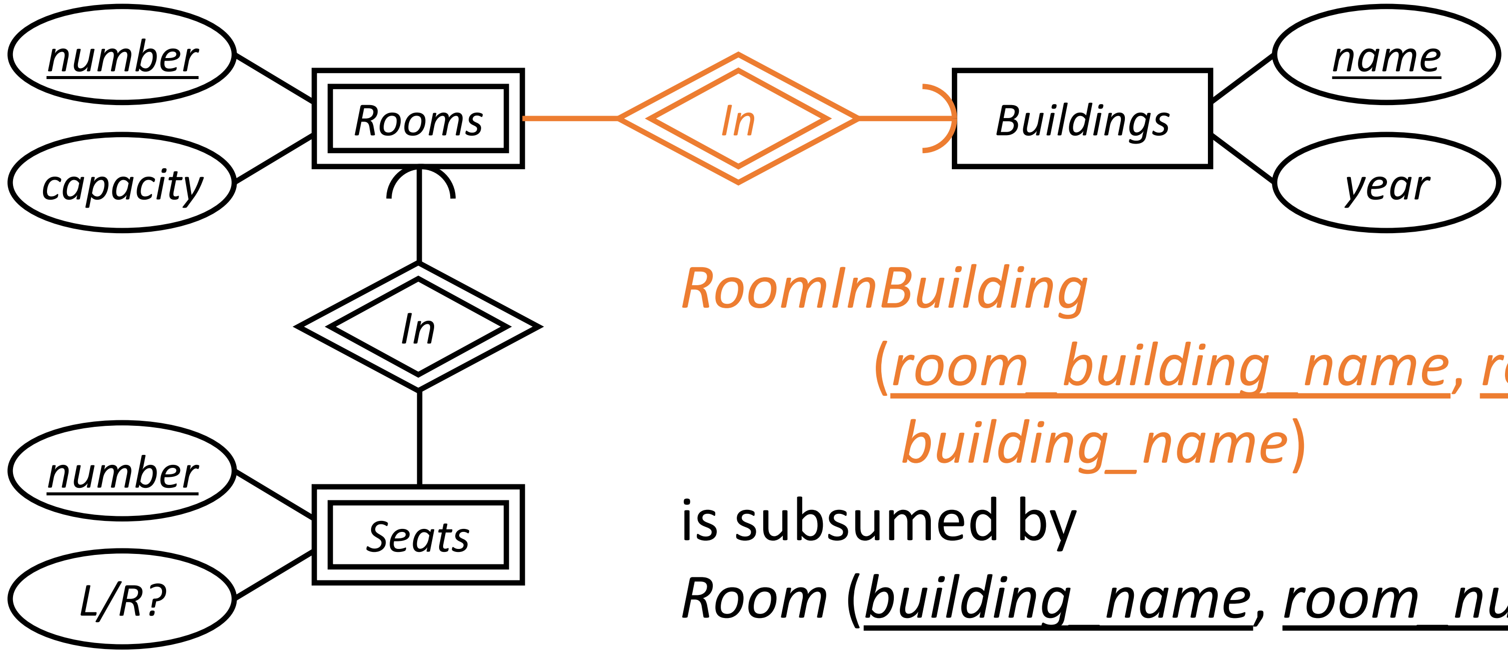
Text



Member (uid, initiator_uid, gid)

Translating double diamonds?

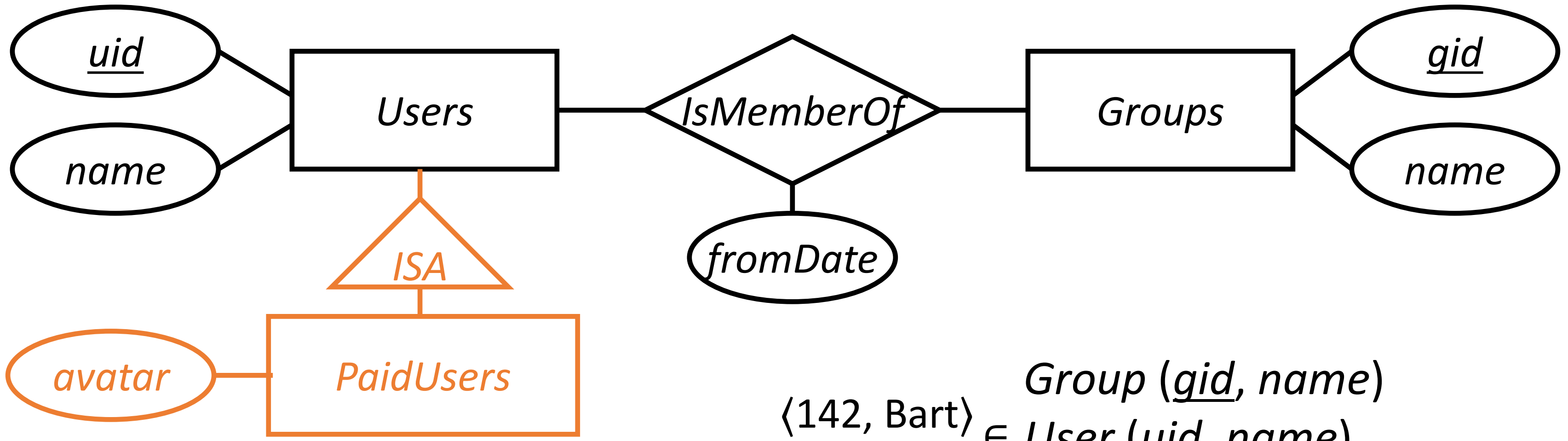
- Recall that a double-diamond (supporting) relationship set connects a weak entity set to another entity set
- No need to translate because the relationship is implicit in the weak entity set's translation



RoomInBuilding
(*room_building_name*, *room_number*, *building_name*)
is subsumed by
Room (*building_name*, *room_number*, *capacity*)

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



⟨142, Bart⟩ ⟨456, Ralph⟩ ∈ *User* (*uid*, *name*)

Group (*gid*, *name*)

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

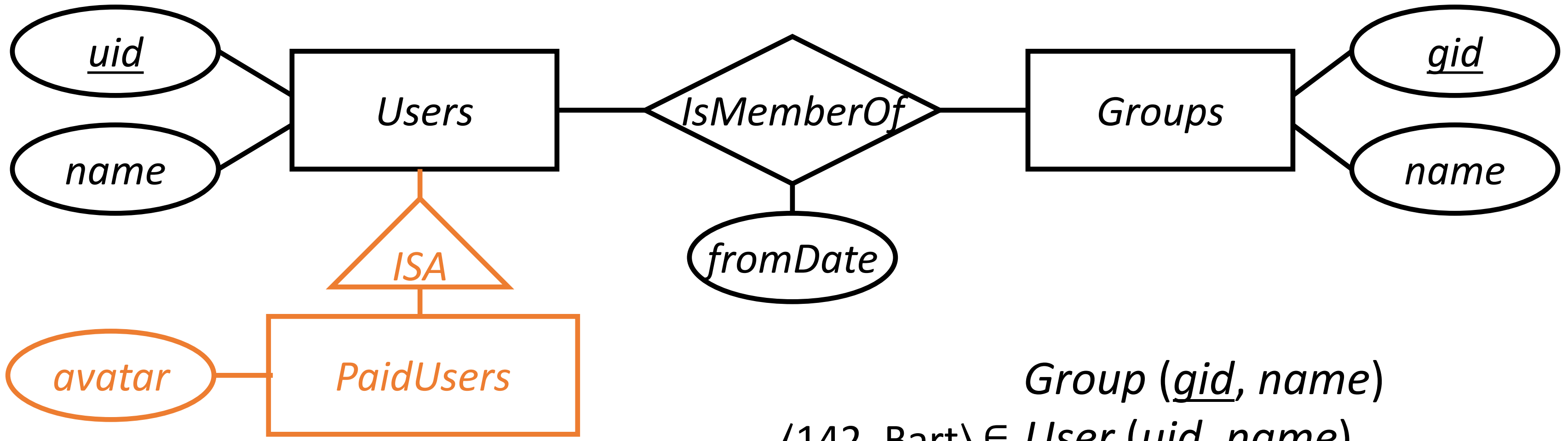
⟨456, ☺⟩ ∈ *PaidUser* (*uid*, *avatar*)

Translating subclasses & ISA: approach 2

Can query paid users more quickly

If the paid and free users were together, you would need to filter out paid users

- 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*)

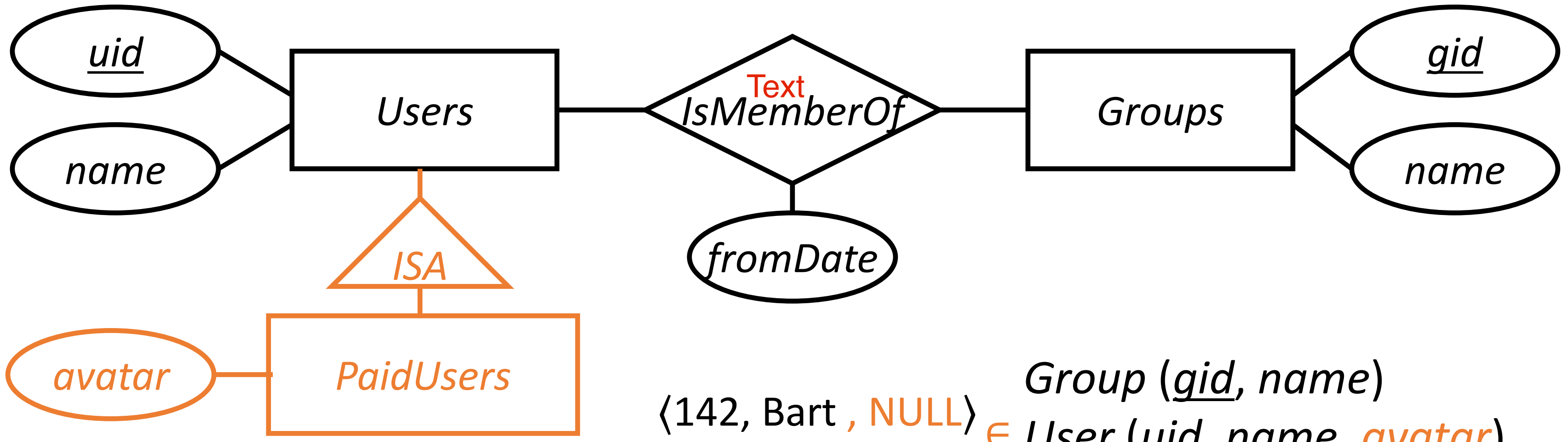
⟨142, Bart⟩ ∈ *User* (*uid*, *name*)

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 (*<u>gid</u>*, *name*)
User (*<u>uid</u>*, *name*, *avatar*)
Member (*<u>uid</u>*, *<u>gid</u>*, *from_date*)

Comparison of three approaches

- Entity-in-all-superclasses
  - *User (uid, name), PaidUser (uid, avatar)*
  - Pro: All users are found in one table
  - Con: Attributes of paid users are scattered in different tables

If you have more queries performed on a specific group, it will be better to separate the groups in different tables

- Entity-in-most-specific-class
  - *User (uid, name), PaidUser (uid, name, avatar)*
  - Pro: All attributes of paid users are found in one table
  - Con: Users are scattered in different tables

Running queries on all users requires you to query both tables

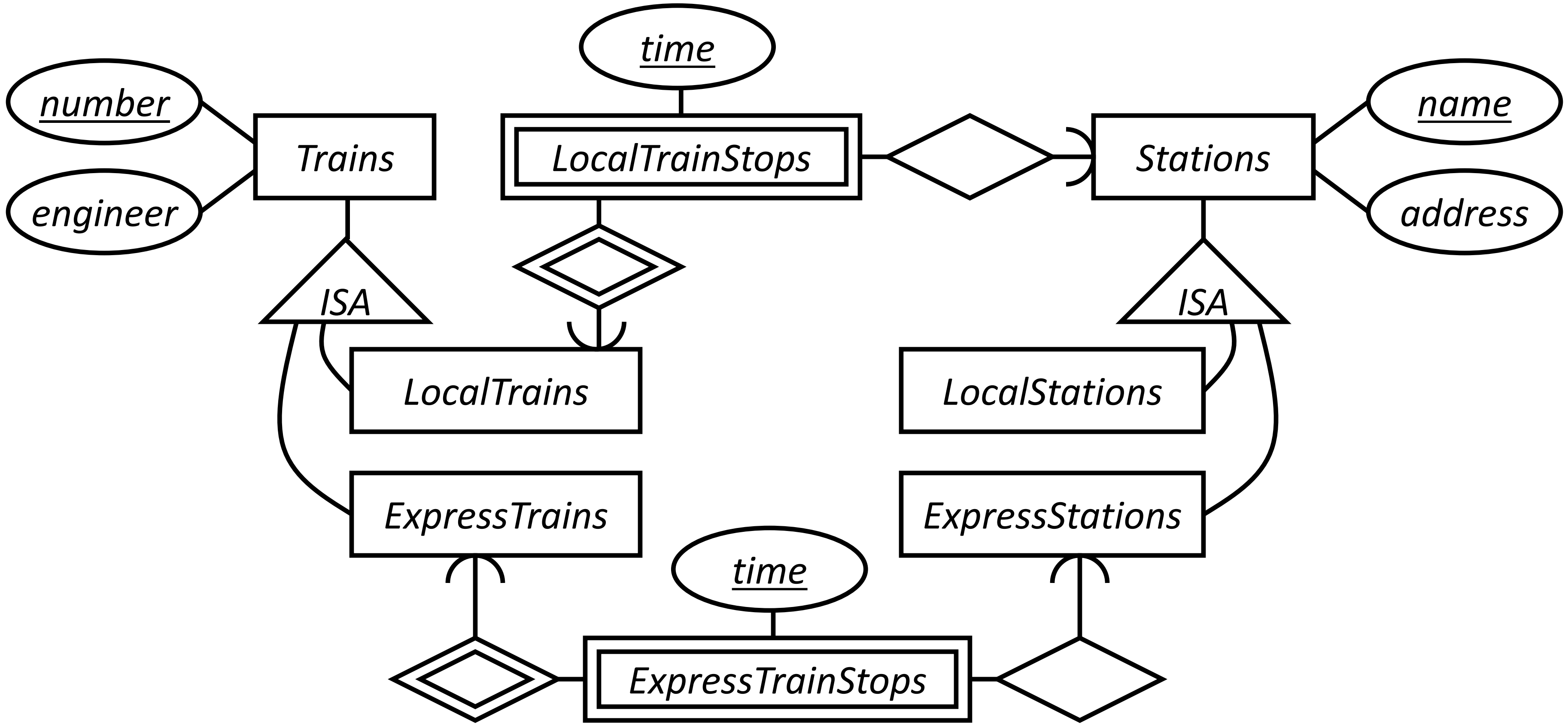
- All-entities-in-one-table
  - *User (uid, [type,]name, avatar)*
  - Pro: Everything is in one table
  - Con: Lots of NULL's; complicated if class hierarchy is complex

If you do not have a default value which allows you to distinguish paid and free users, than they will appear to be equivalent

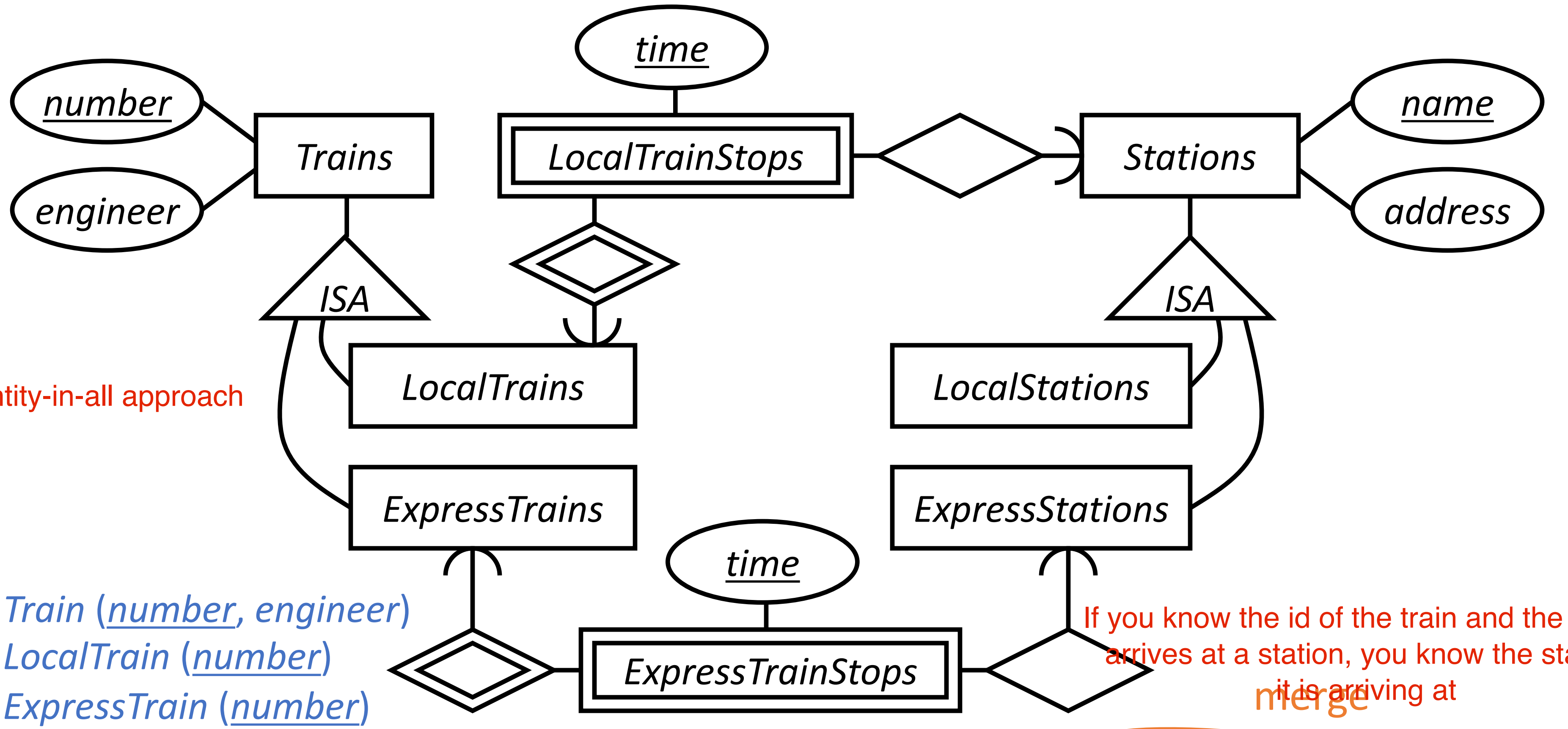
A complete example: Case study 2

- 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 every day

A complete example



A complete example



Using the entity-in-all approach

If you know the id of the train and the time it arrives at a station, you know the station it is arriving at

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

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

merge

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

Simplifications and refinements

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

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

An alternative design

Train (<u>number</u>, engineer, type)

Station (<u>name</u>, address, type)

TrainStop (<u>train_number</u>, station_name, <u>time</u>)

- 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 (as we will see later in course)
- Arguably a better design because it is simpler!

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
  - Warning: mechanical translation procedures given in this lecture are no substitute for your own judgment
- Practice inclusive design—more to come
  - Your database design influences everything built around it and can have profound effects on real people

http://ungenius.files.wordpress.com/2010/03/thehomer.jpg