CS 348 Lectures 12-13

Entity-Relationship Model for Relational Database Design

Semih Salihoğlu

Feb 12-24 2025

Motivating Example

I want to have a registrar's database. Can you help?

It has these requirements …

Zero or more sections of a course are offered each term. Courses have names and numbers. In each term, the sections of each course are numbered starting with 1.
Most course sections are taught on-site, but a few are taught at off-site locations.
Students have student numbers and names. Each course section is taught by a professor. A professor may teach more than one section in a term, but if a professor teaches more than one section in a term, they are always sections of the same course. Some professors do not teach every term.
Up to 50 students may be registered for a course section. Sections with 5 or fewer students are cancelled.
A student receives a mark for each course in which they are enrolled. Each student has a cumulative grade point average (GPA) which is calculated from all course marks the student has received.

I know how to use SQL now!

What tables do you want me to create? What are the primary keys, constraints, queries, … ..?

We still need to learn about database design ☺

Database Design

Step 1: Understand the real-world domain being modeled

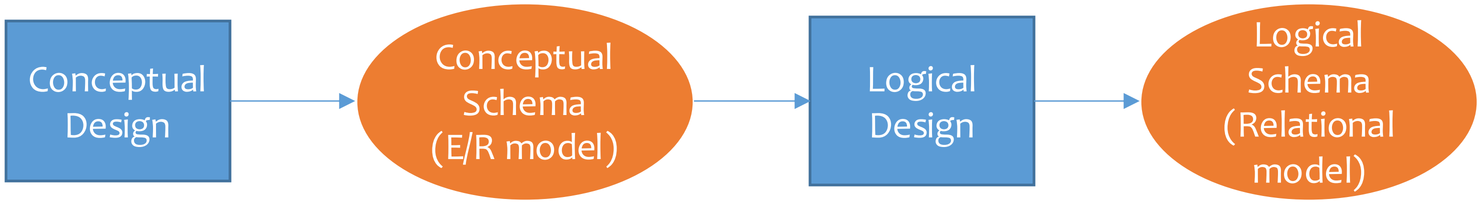
→Specify it using a database design model

- E.g., Entity/Relationship (E/R) model, Object Definition Language (ODL), UML (Unified Modeling Language)

Step 2: Translate specification to the data model of DBMS

- Relational, XML, object-oriented, etc.

→ Create DBMS schema


Conceptual Design
Conceptual Schema (E/R model)
Logical Design
Logical Schema (Relational model)

Database Design

- Entity-Relationship (E/R) model
- Translating E/R to relational schema

Database Design

- Entity-Relationship (E/R) model

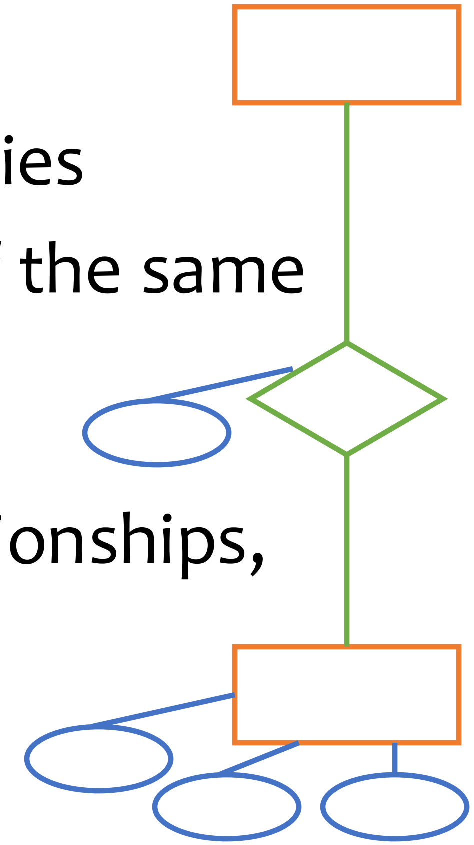
- Translating E/R to relational schema

Entity-relationship (E/R) model

- Historically and still very popular
- 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 textbook book; there are other styles/extensions
  - Very similar to UML diagrams

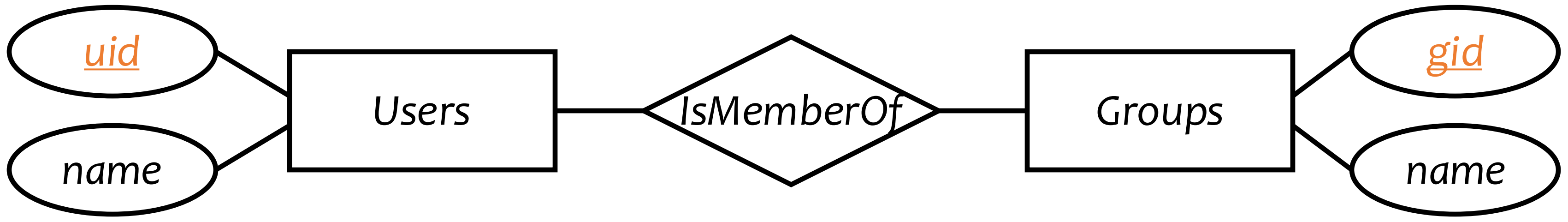
E/R basics

- 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

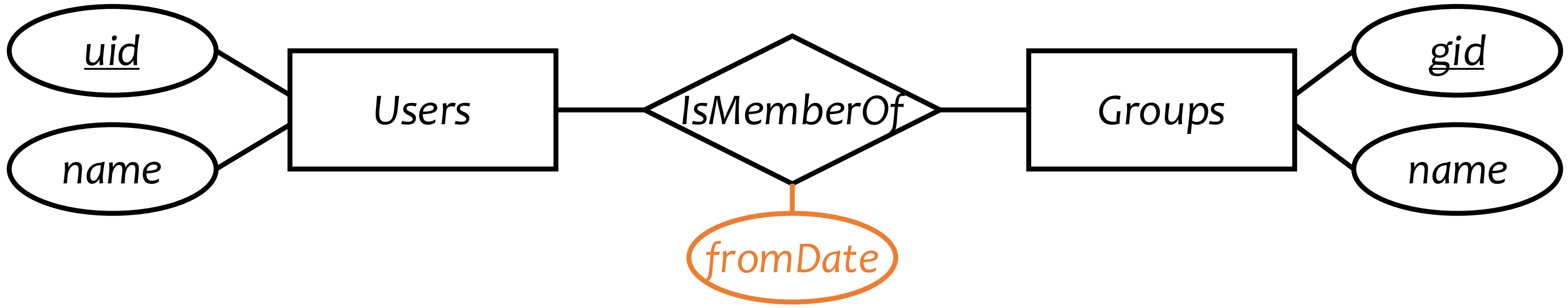
- Users are members of groups



- 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

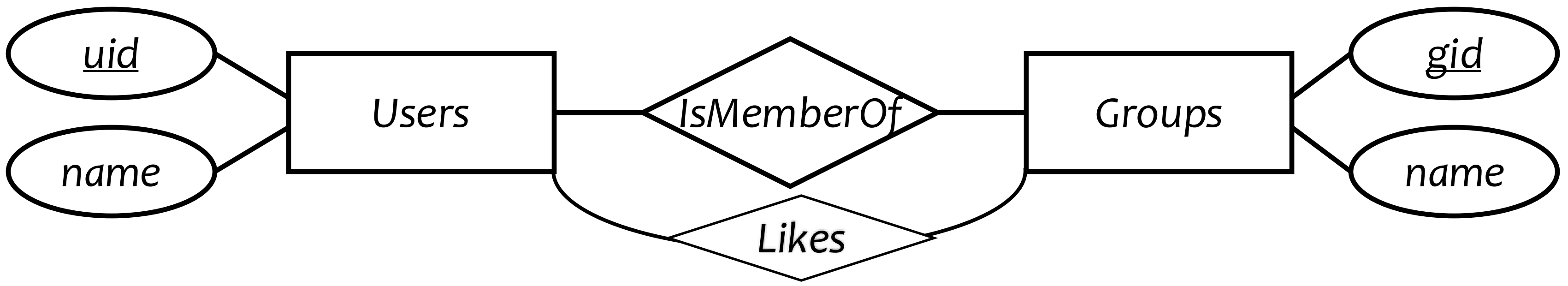
- Example: a user belongs to a group since a particular date



- 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

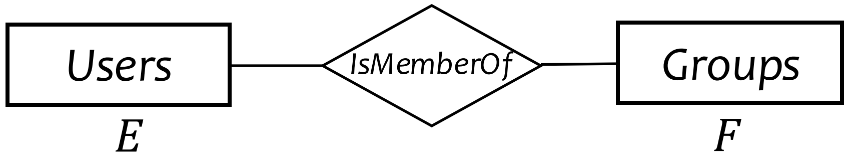


More on relationships

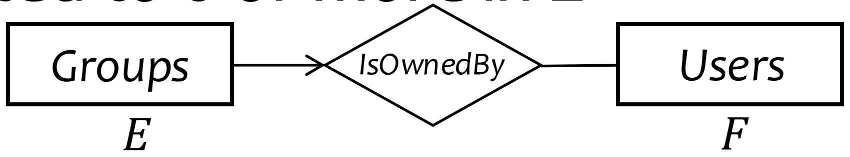
- There could be multiple relationship sets between the same entity sets
  - Example: *Users IsMemberOf Groups*; *Users Likes Groups*
- In a relationship set, each relationship is uniquely identified by the entities it connects
  - Example: Between Bart and "Dead Putting Society", there can be at most 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

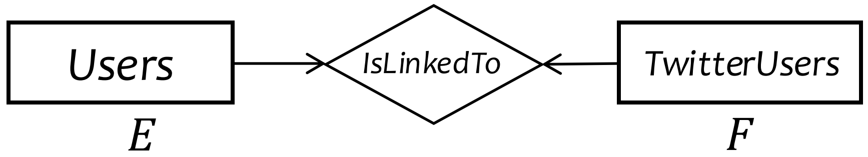
- E and F: entity sets
- Many-many: Each entity in E is related to 0 or more entities in F and vice versa
  - Example:



- 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:



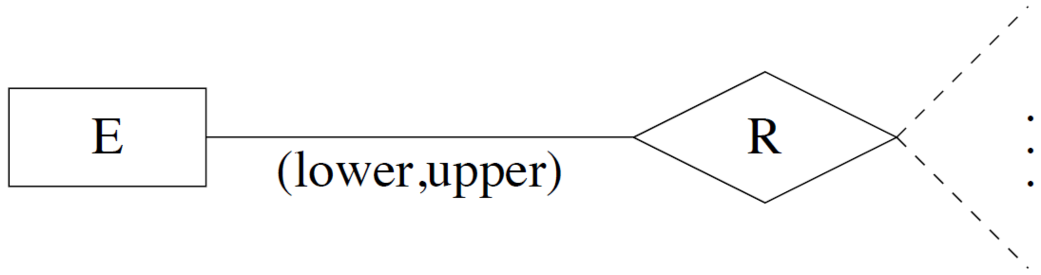
- One-one: Each entity in E is related to 0 or 1 entity in F and vice versa
  - Example:



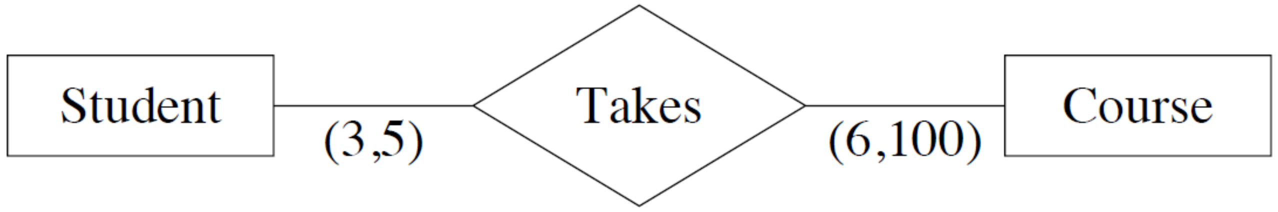
- "One" (0 or 1) is represented by an arrow $\longrightarrow$

General cardinality constraints

- General cardinality constraints determine lower and upper bounds on the number of relationships of a given relationship set in which a component entity may participate



- Example:

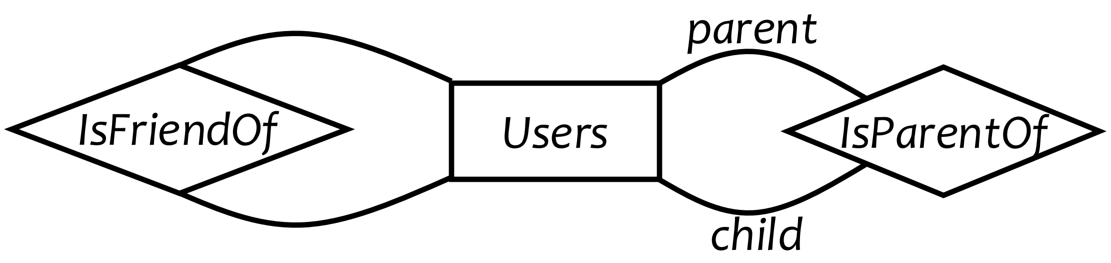


Roles in relationships

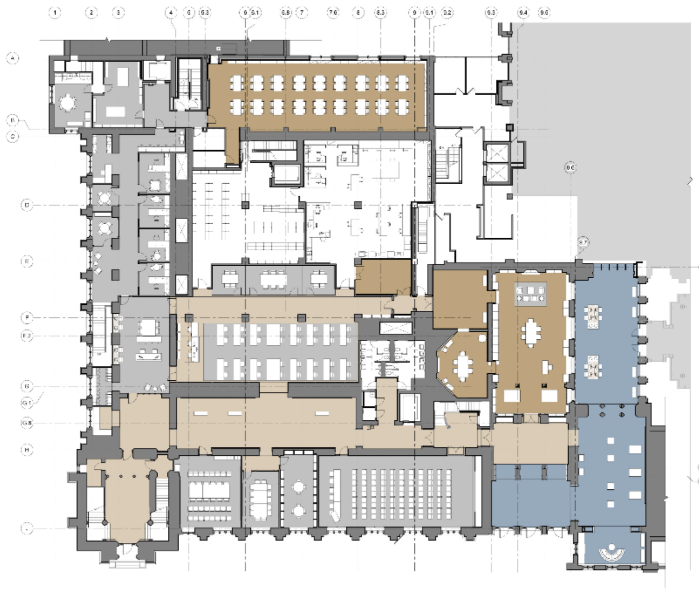
- An entity set may participate more than once in a relationship set

☞ May need to label edges to distinguish roles

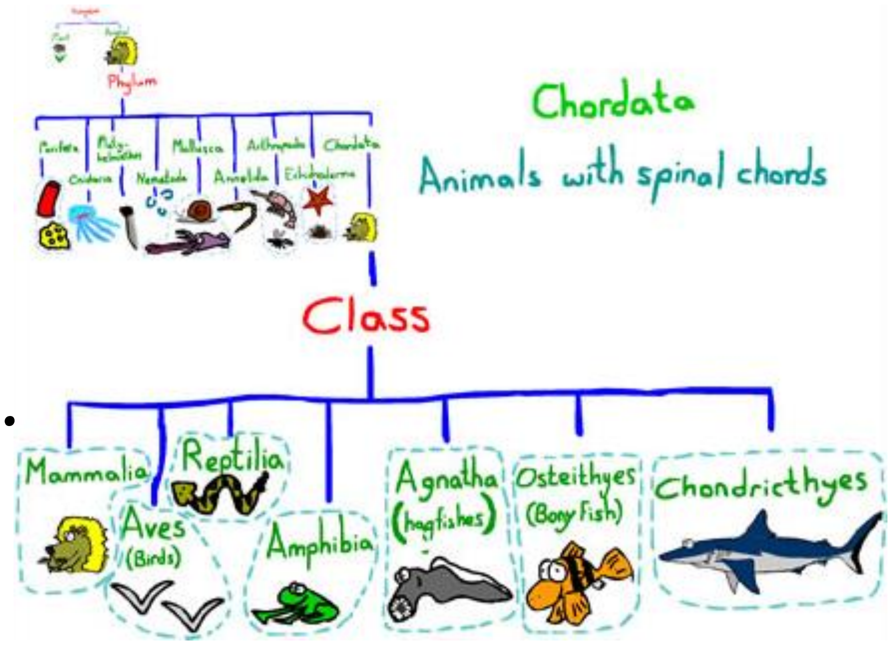
- Examples
  - Users may be parents of others; label needed
  - Users may be friends of each other; label not needed



Next: two special relationships



… is part of/belongs to …

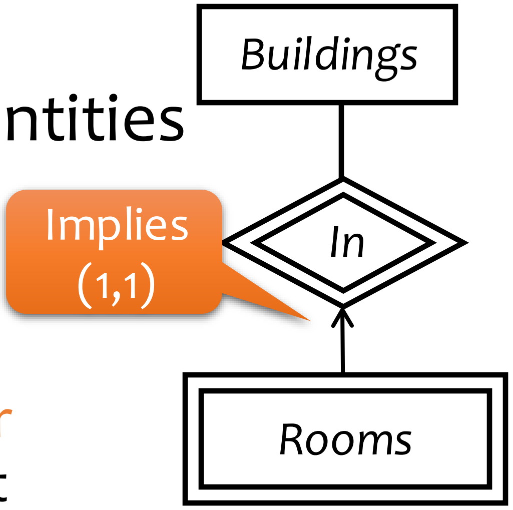


… is a kind of …

http://blogs.library.duke.edu/renovation/files/2012/08/Rubenstein-Library-First-Floor-Floorplan.jpg
http://www.sharky-jones.com/Sharkyjones/Artwork/taxonomy%20artwork/Class1.jpg

Weak entity sets

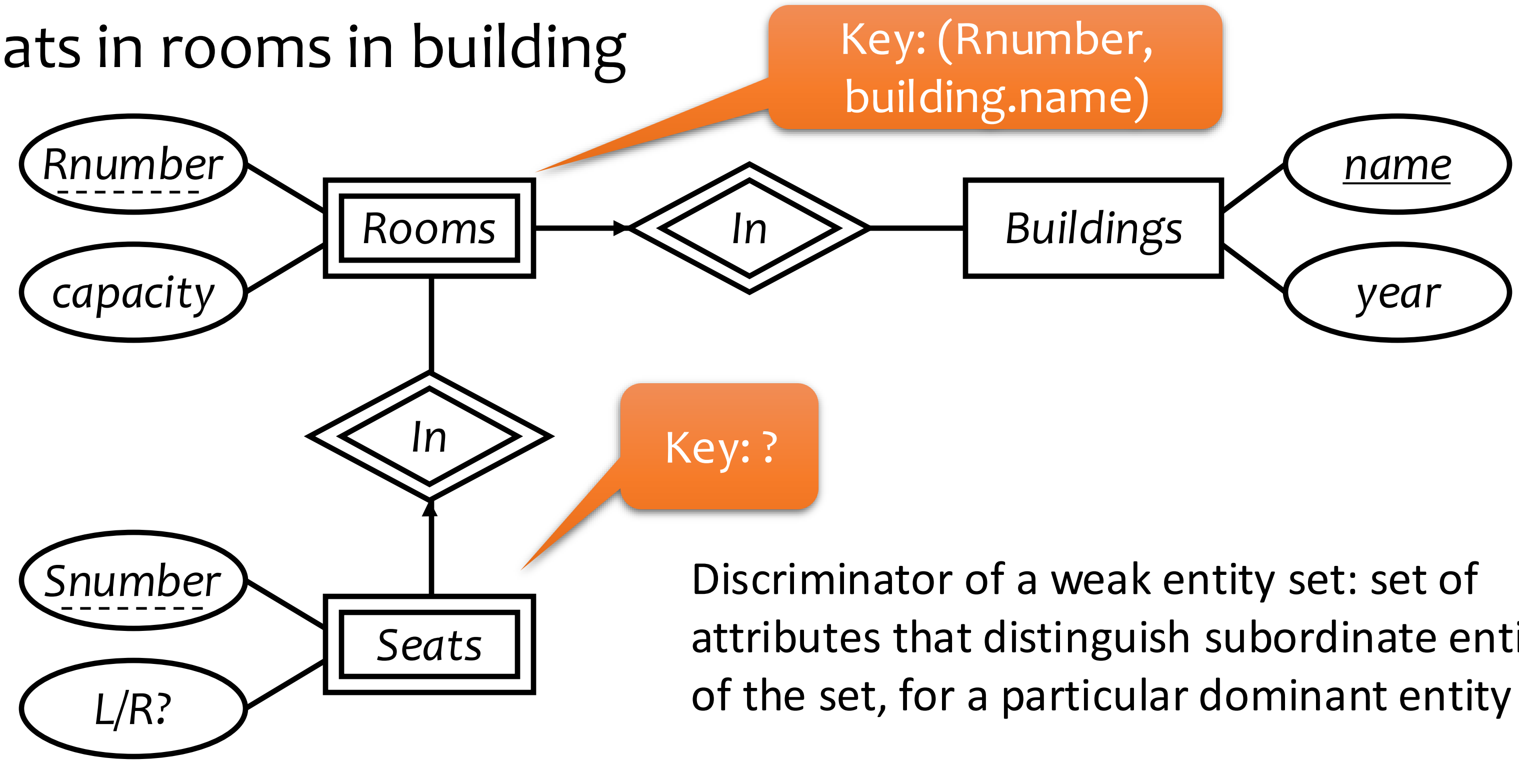
- If entity E is existence dependent on entity F, then
  - F is a dominant entity
  - E is a subordinate entity
  - Example: *Rooms* inside *Buildings* are partly identified by *Buildings*' name
- Weak entity set: containing subordinate entities
  - Drawn as a double rectangle
  - The relationship sets are called supporting relationship sets, drawn as double diamonds
  - A weak entity set must have a many-to-one or one-to-one relationship to a distinct entity set



- Strong entity set: containing no subordinate entities

Weak entity set examples

- Seats in rooms in building

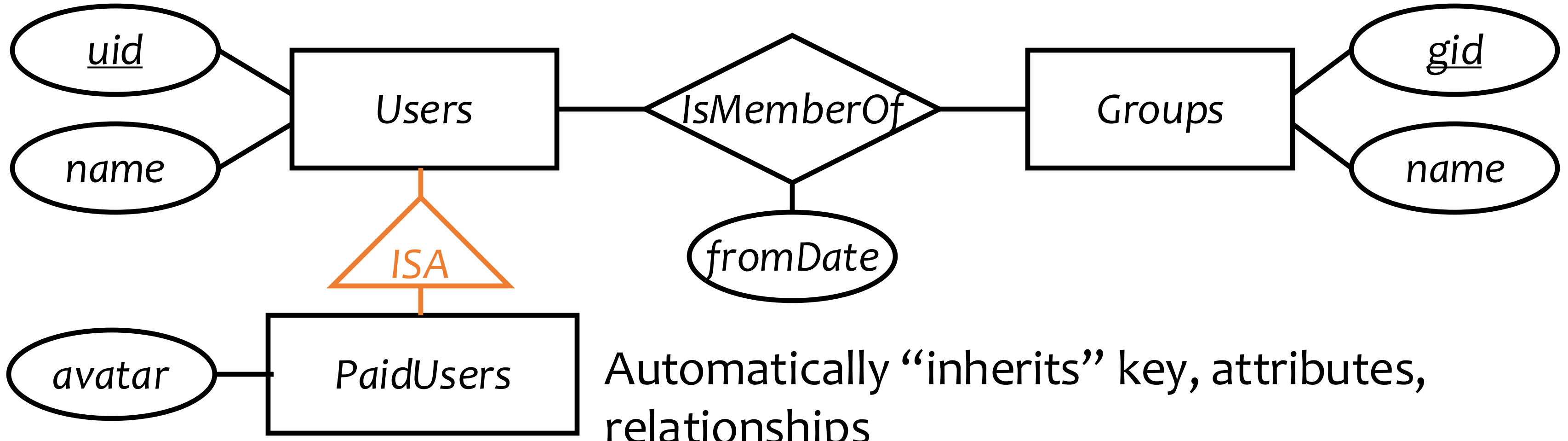


Discriminator of a weak entity set: set of attributes that distinguish subordinate entities of the set, for a particular dominant entity

- Attributes of weak entity sets only form key relative to a given dominant entity → discriminator (dotted underline)
- Primary key of a weak entity set: discriminator + primary key of entity set for dominant entities

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!)



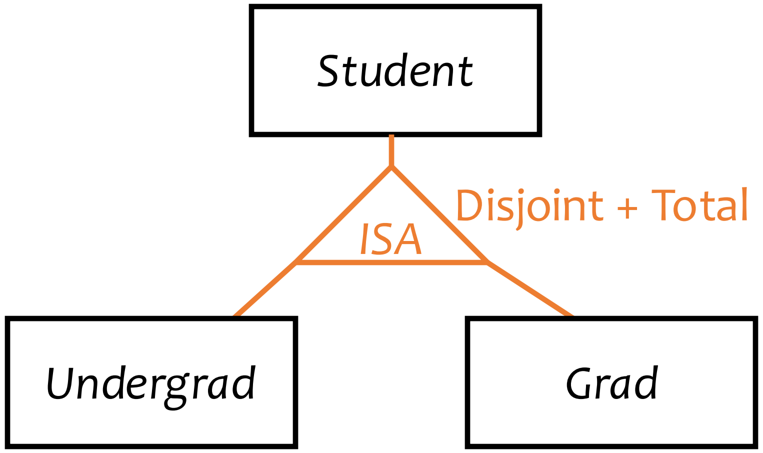
Automatically “inherits” key, attributes, relationships

More on ISA relationships

- Two constraints:
  1. Overlapping vs Disjoint:
     - Overlapping: An entity may belong to multiple subclass
     - Disjoint: An entity can belong to at most 1 subclass
  2. Partial vs Total:
     - Partial: An entity of a superclass does not have to belong to any subclass (can only belong to a superclass)
     - Total: An entity of a superclass has to belong to a subclass.

More on ISA relationships

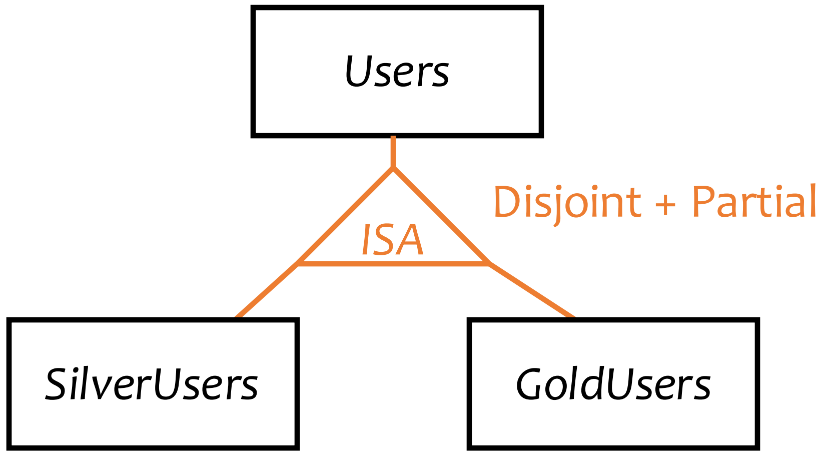
- Ex: Student must be either undergrad or grad



- Ex: If students can be both undergrad + grad, then Overlaps + Total
  - (e.g., Waterloo CS's accelerated MMath CS degree)

More on ISA relationships

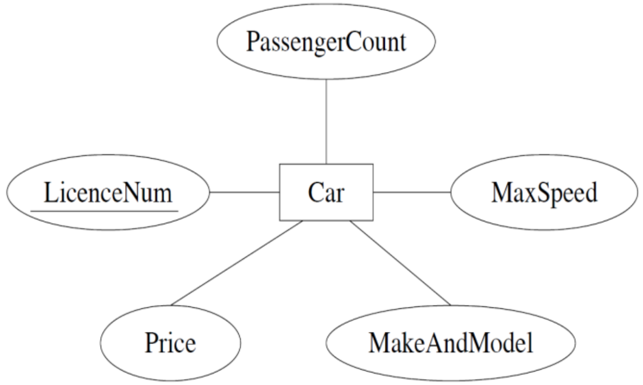
- Ex: Some users can be either Silver or Gold (but not both) but they can also just be “regular” users.

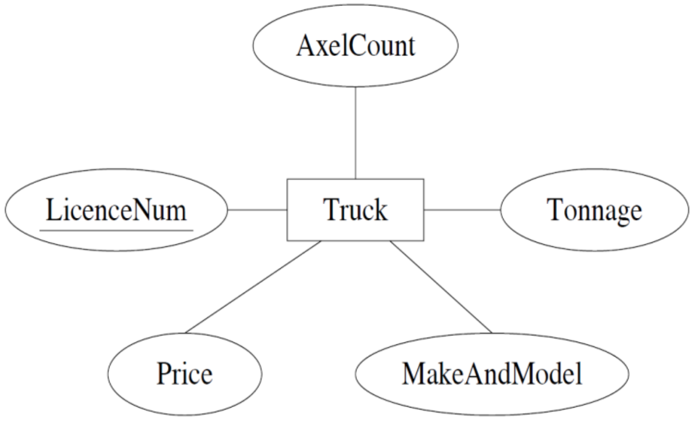


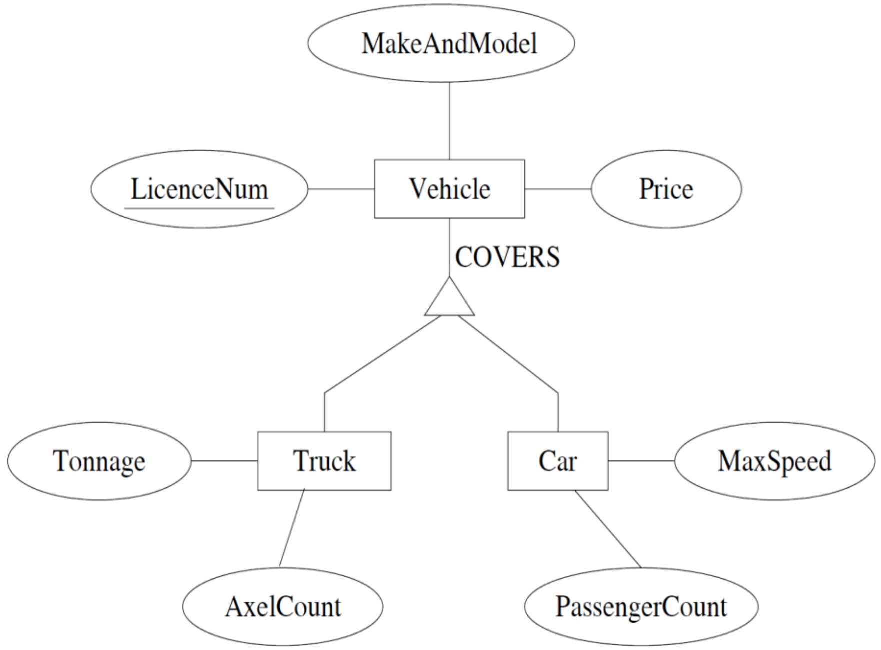
- By default: Disjoint + Partial.
- But good practice to explicitly state as in the above examples.

Other extensions to E/R models

- Generalization: several entity sets can be abstracted by a more general entity set
    - Example: “ a vehicle abstracts the notion of a car and a truck”

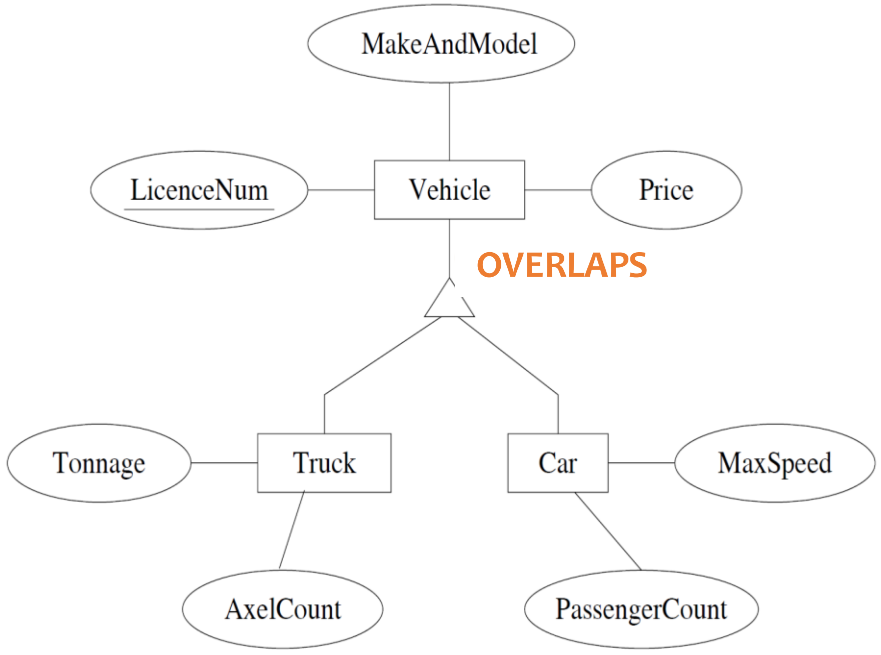






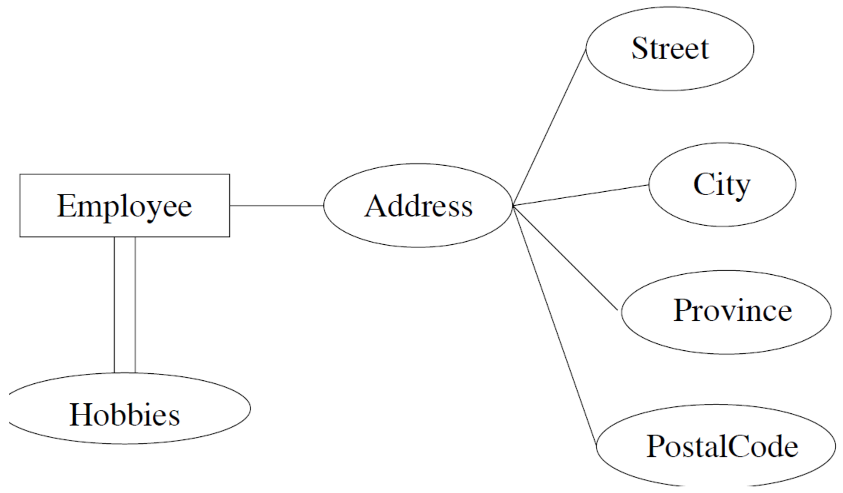
Other extensions to E/R models

- Specialized entity sets are usually disjoint but can be declared to have entities in common
- By default, specialized entity sets are disjoint.
  - Example: We may decide that nothing is both a car and a truck.
  - However, we can declare them to overlap (to accommodate utility vehicles, perhaps).



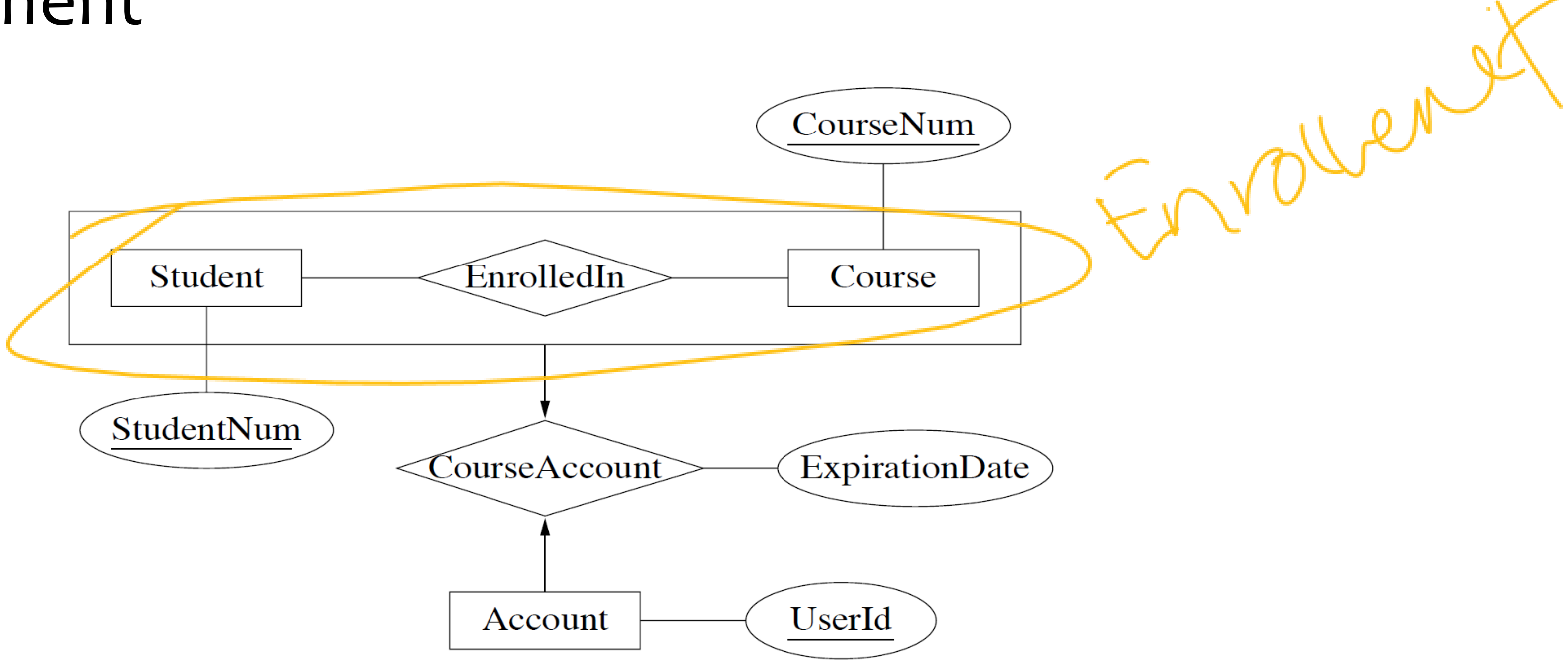
Other extensions to E/R models

- Structured attributes:
  - Composite attributes: composed of fixed number of other attributes
    - E.g. Address
  - Multi-valued attributes: attributes that are set-valued
    - e.g. Hobbies (double edges)



Other extensions to E/R models

- Aggregation: relationships can be viewed as high-level entities
- Example: "accounts are assigned to a given student enrollment"



Summary of E/R concepts

- Entity sets
  - Keys
  - Weak entity sets
- Relationship sets
  - Attributes of relationships
  - Multiplicity
  - Roles
  - Supporting relationships (related to weak entity)
  - ISA relationships
- Other extensions:
  - Generalization
  - Structured attributes
  - Aggregation

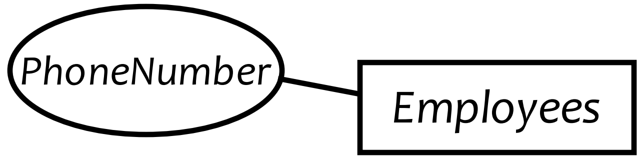
Designing an E/R schema

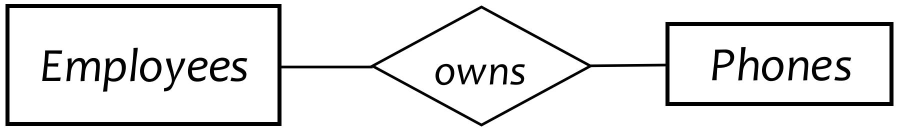
- Usually many ways to design an E-R schema

- Points to consider
  - use attribute or entity set?
  - use entity set or relationship set?
  - degrees of relationships?
  - extended features?

Attributes or Entity Sets?

- Example: How to model employees’ phones?



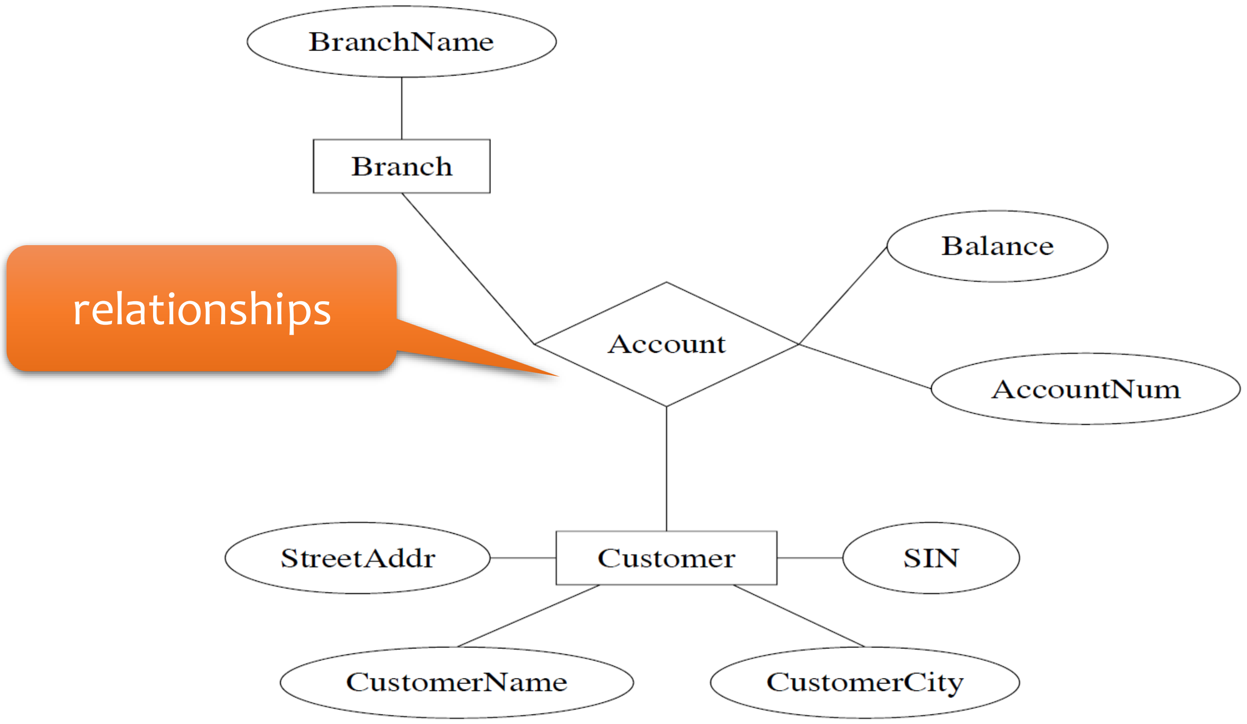


- Rules of thumb:
  - Is it a separate object?
  - Do we maintain information about it?
  - Can several of its kind belong to a single entity?
  - Does it make sense to delete such an object?
  - Can it be missing from some of the entity set’s entities?
  - Can it be shared by different entities?

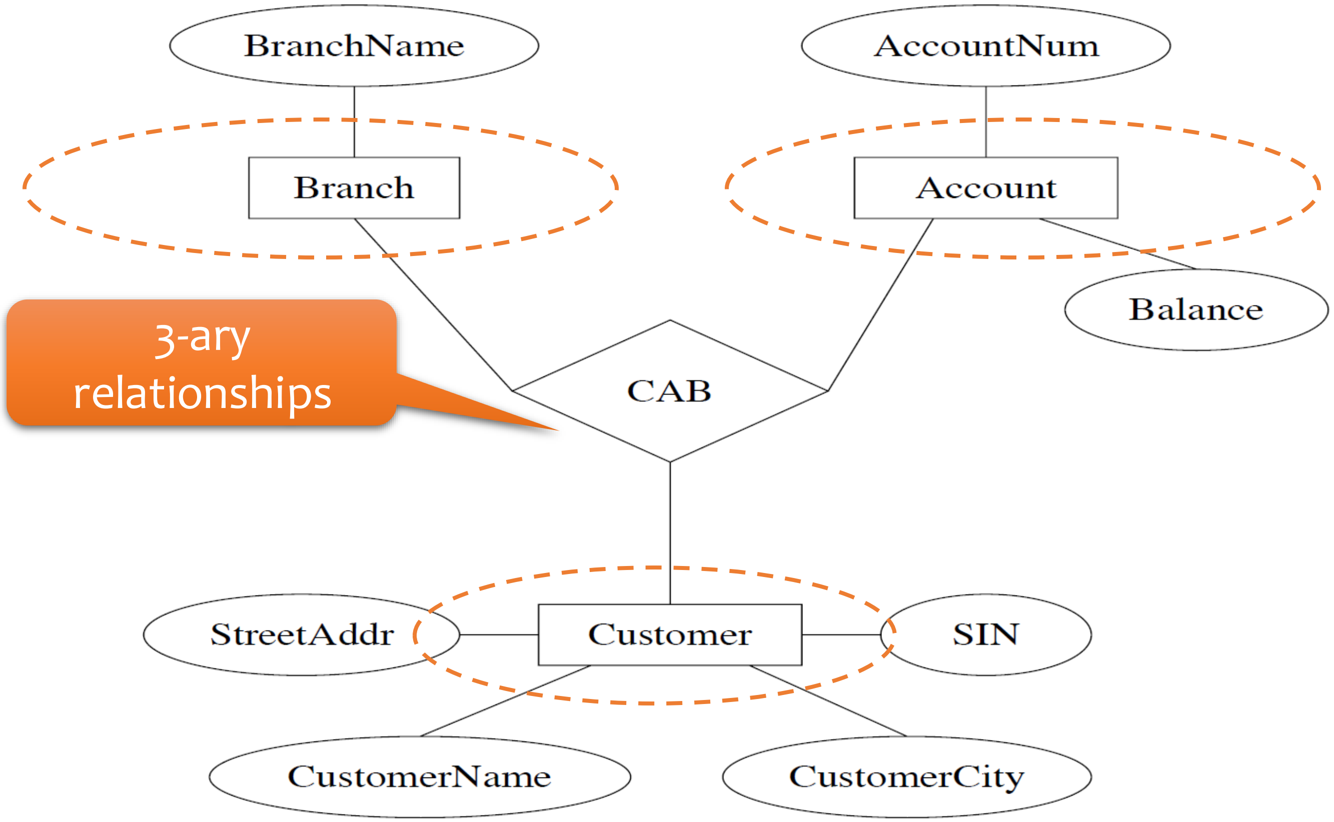
→ An affirmative answer to any of the above suggests a new entity set.

Entity Sets or Relationships?

- Instead of representing accounts as entities, we could represent them as relationships

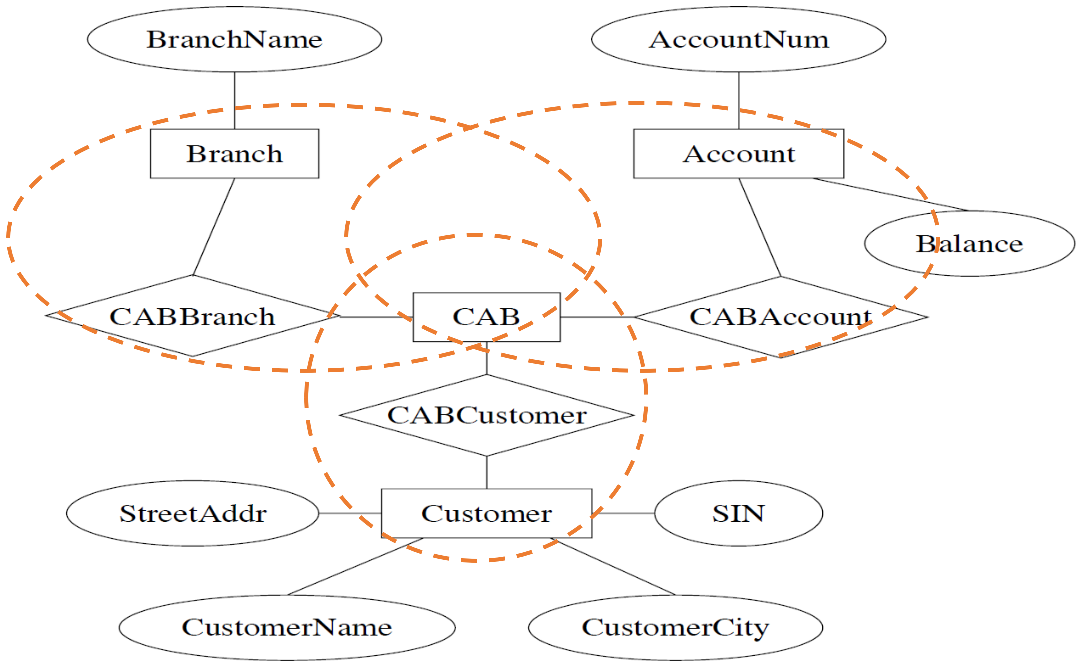


Binary vs. N-ary Relationships?



Binary vs. N-ary Relationships (cont'd)

- We can always represent a relationship on n entity sets with n binary relationships



A simple methodology

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, keys and discriminators
6. Draw diagram

- For each step, maintain a log of assumptions motivating the choices, and of restrictions imposed by the choices

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

What are the entity sets, relationship sets, and their attributes? What are the types of relationships and cardinality constraints, keys, discriminators?

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)

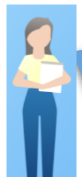
Assume the following:

- 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

Cities
Counties
States

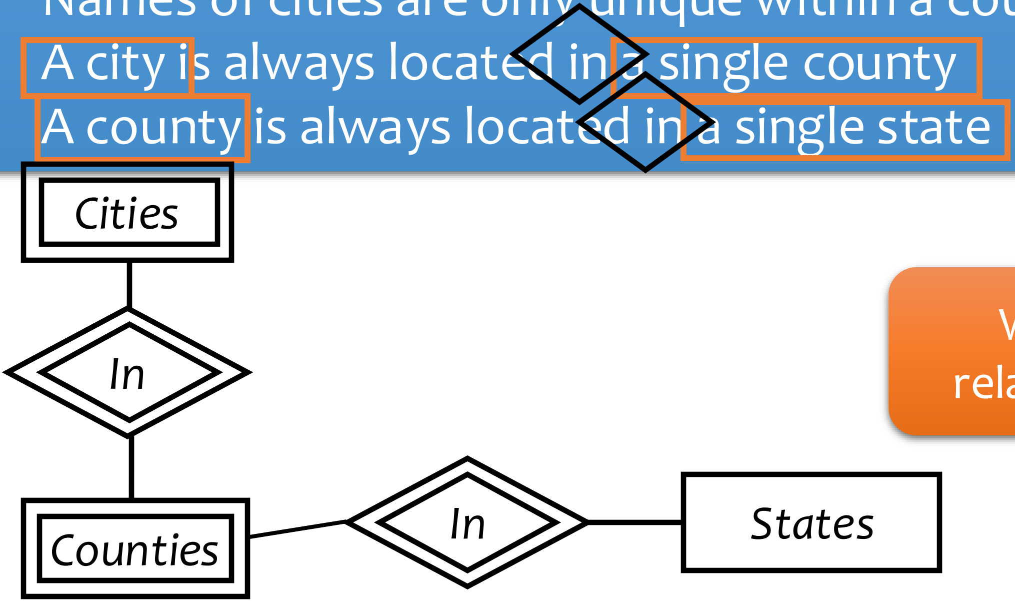
What are my entity sets? (slide 26)

Case study 1



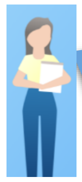
Design a database representing cities, counties, and states
- For states, record name and capital (city)
- For counties, record name, area, and location (state)
- For cities, record name, population, and location (county and state)

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



What are my relationship sets?

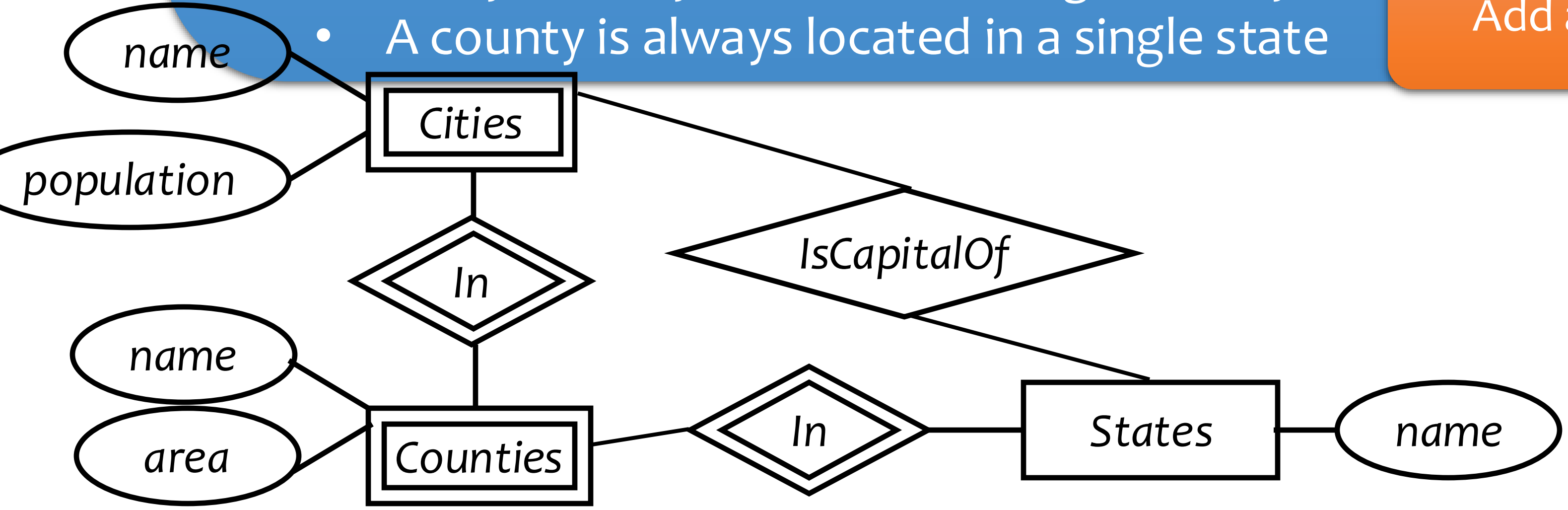
Case study 1



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

Add attributes!



Case study 1

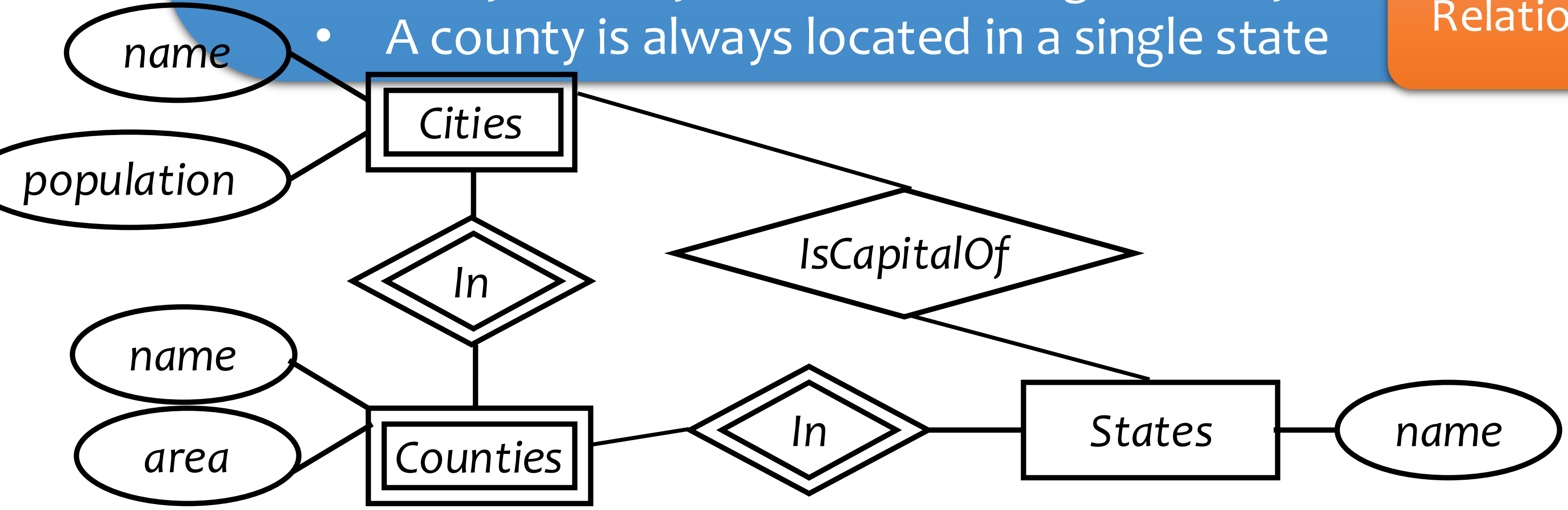
Design a database representing cities, counties, and states

- 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

Relationship types?



Case study 1

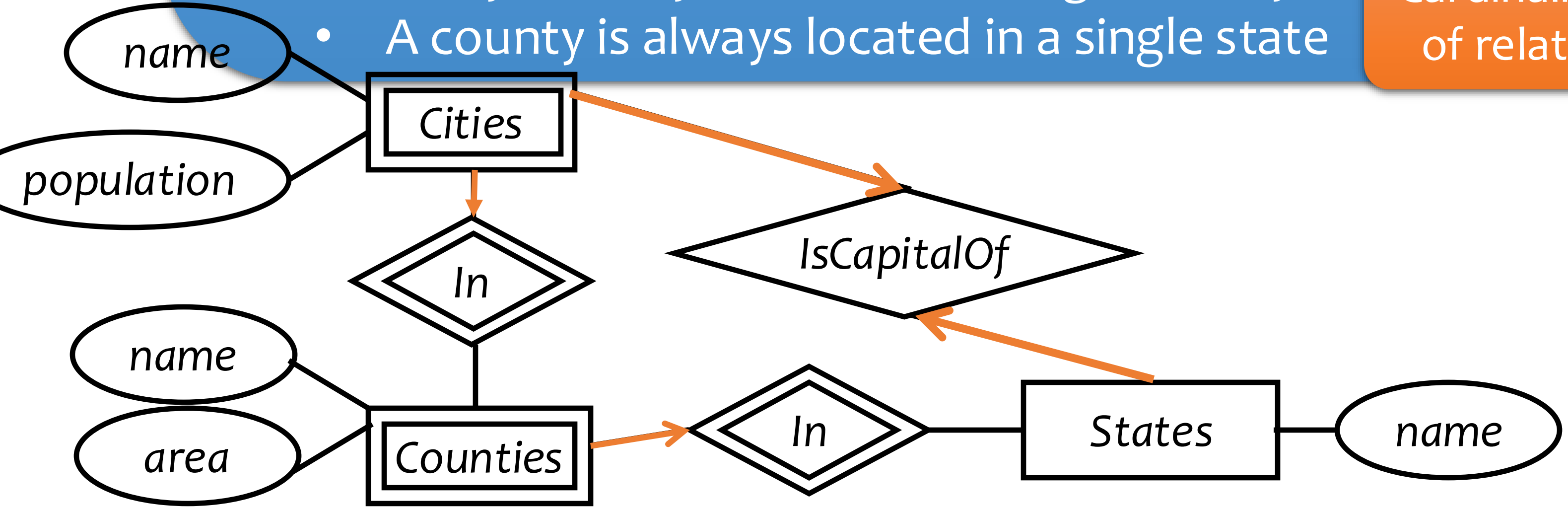
Design a database representing cities, counties, and states

- 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

Cardinality constraints of relationship sets?



Case study 1

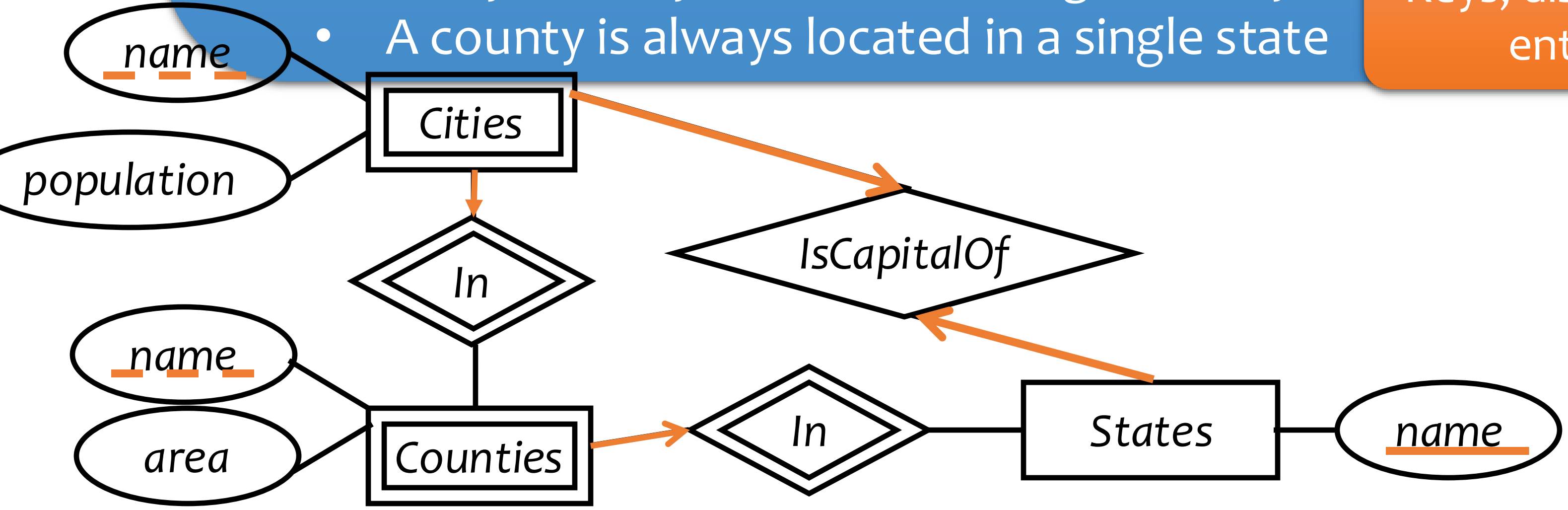
Design a database representing cities, counties, and states

- 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)

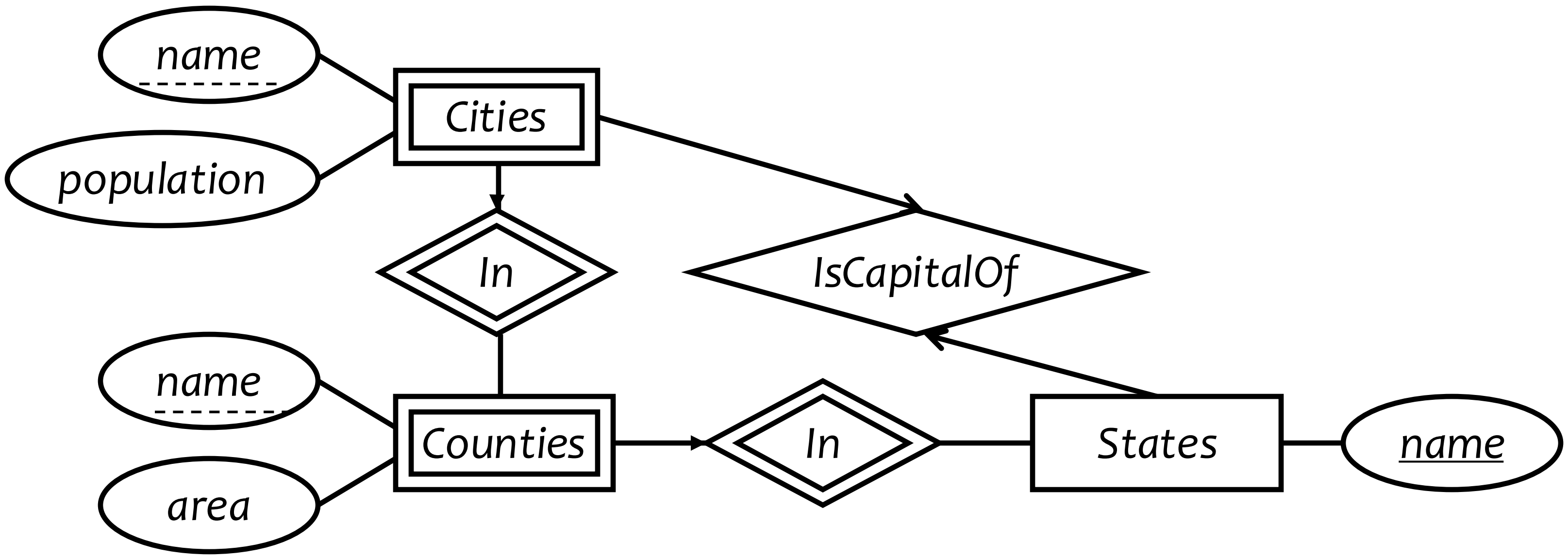
Assume the following:

- 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

Keys, discriminator of entity sets?

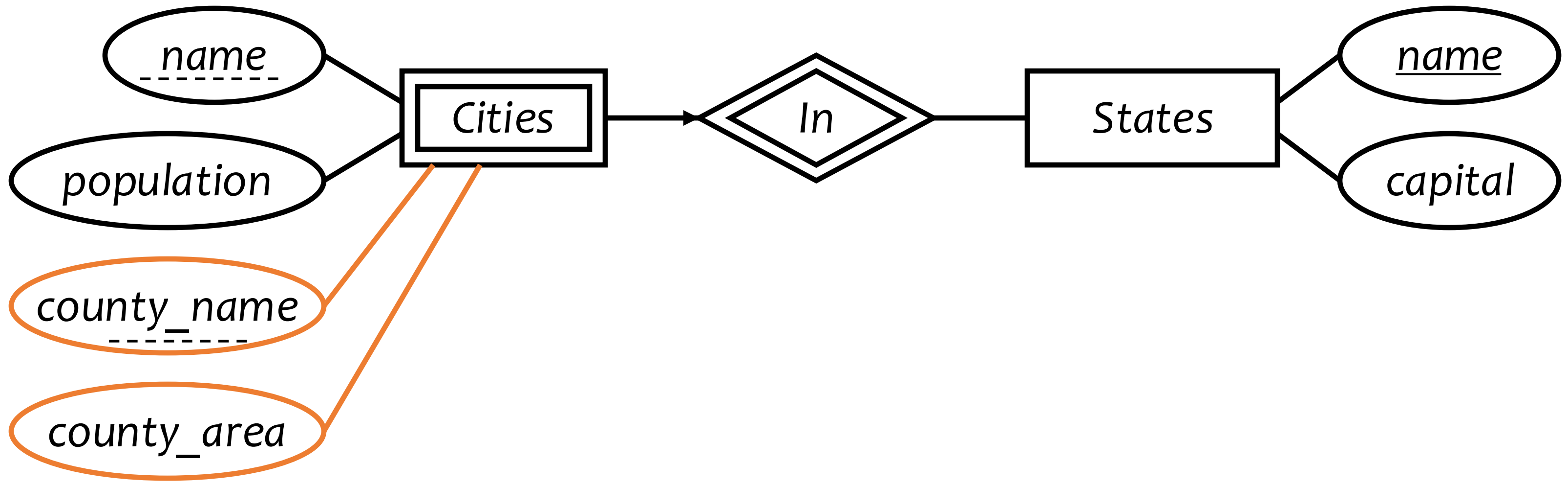


Case study 1: final design



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

Case study 1: why not good?



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

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

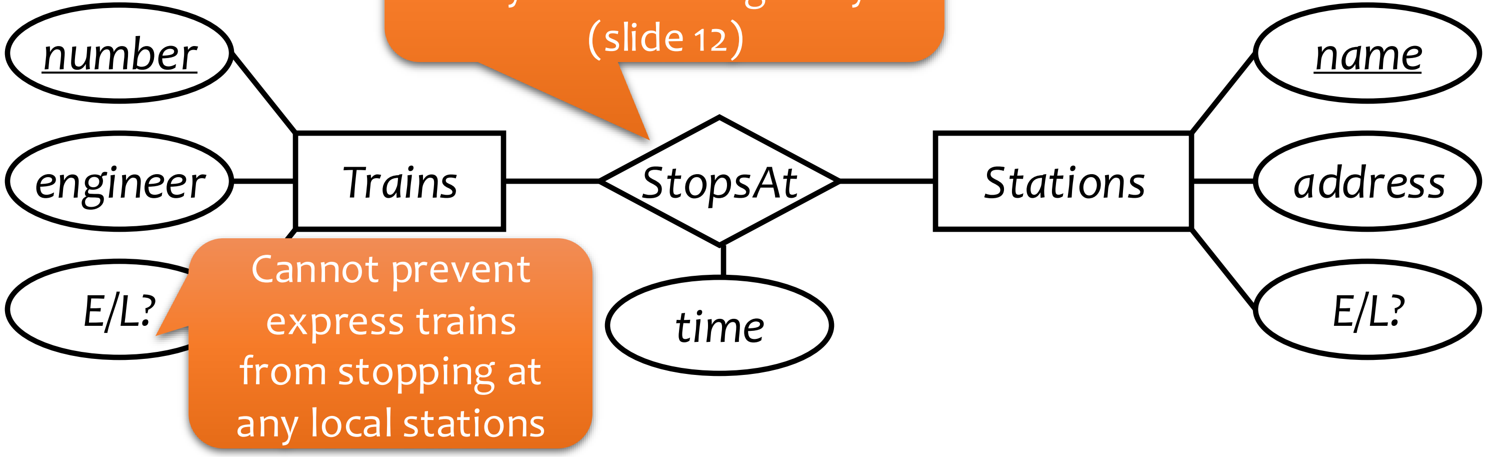
What are the entity sets, relationship sets, and their attributes? What are the types of relationships and cardinality constraints, keys, discriminators?

Case study 2: first design

Design a database consistent with the following:

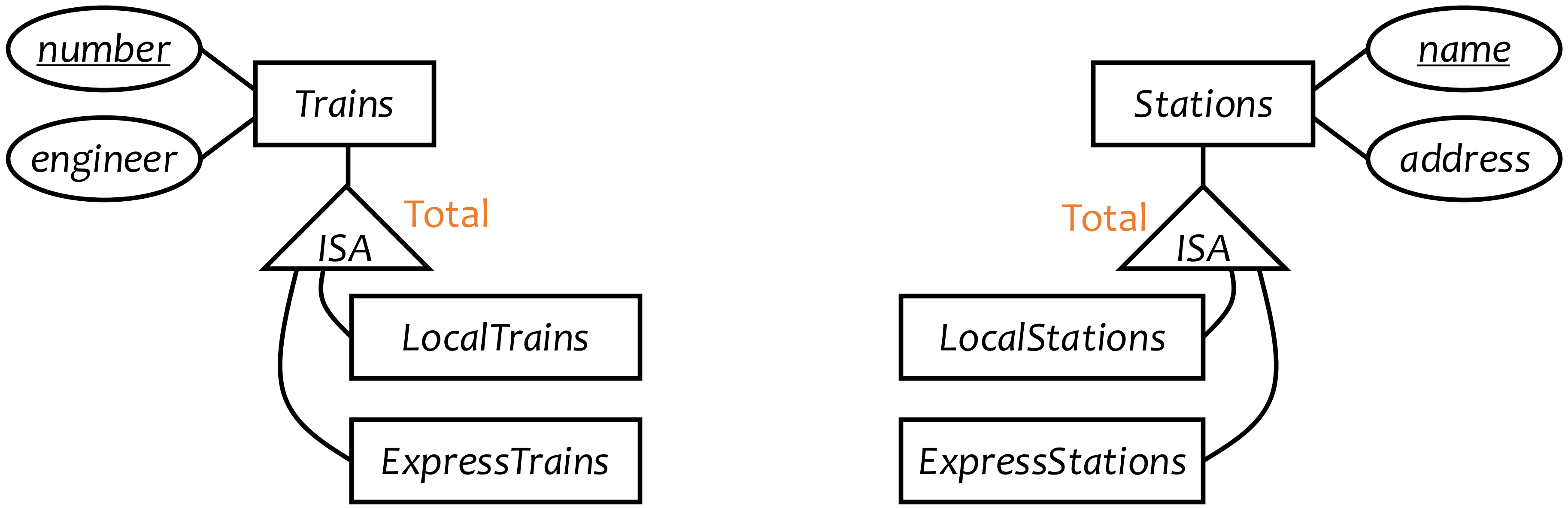
- A station has a unique name and an address, and is either an express station or a local station
- A train has a unique number and an engineer, and is either an express train or a local train
- A local train can stop at any station
- An express train only stops at express stations
- A train can stop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

Why not good?



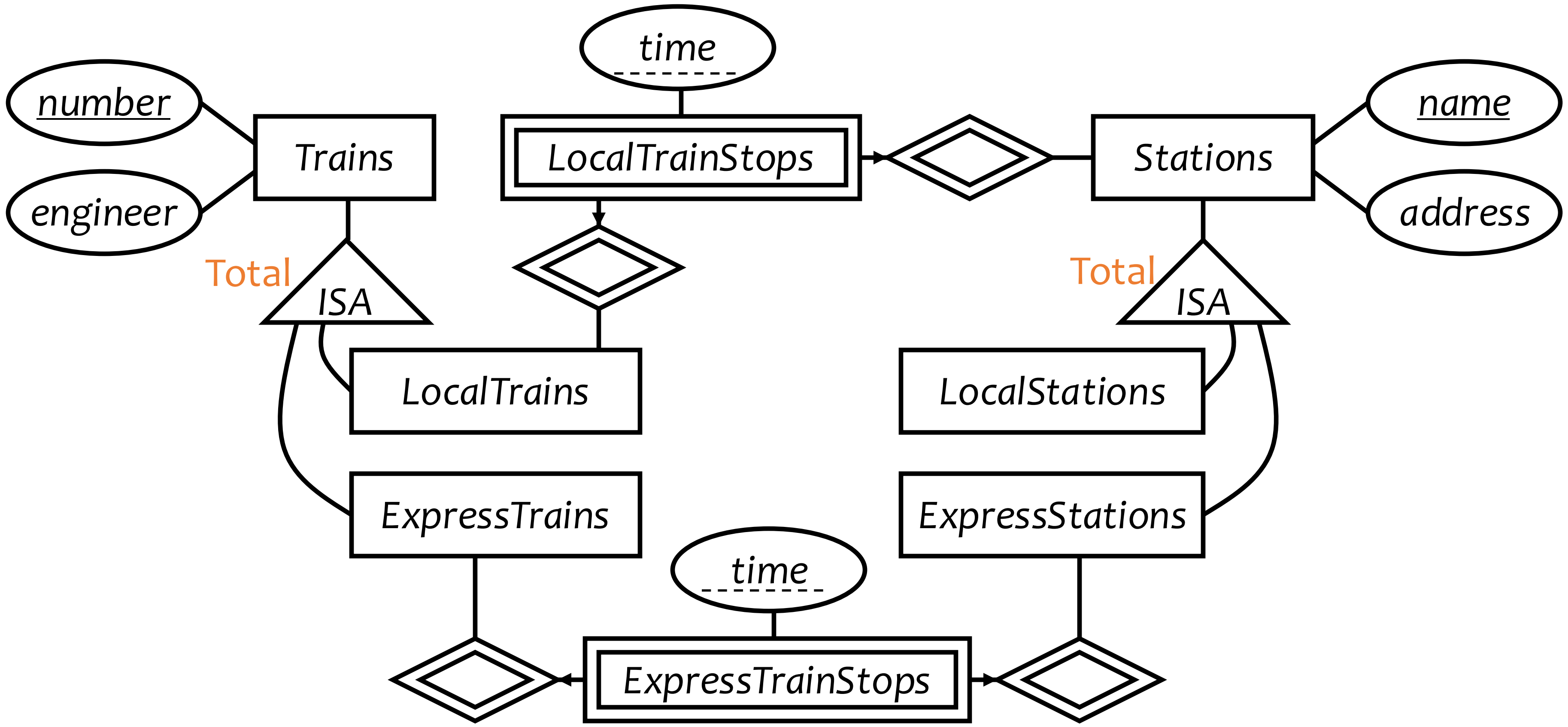
Unintended constraint: A train can stop at a station only ONCE during a day! (slide 12)
number
engineer
E/L?
Trains
StopsAt
time
Stations
name
address
E/L?
Cannot prevent express trains from stopping at any local stations

Case study 2: second design



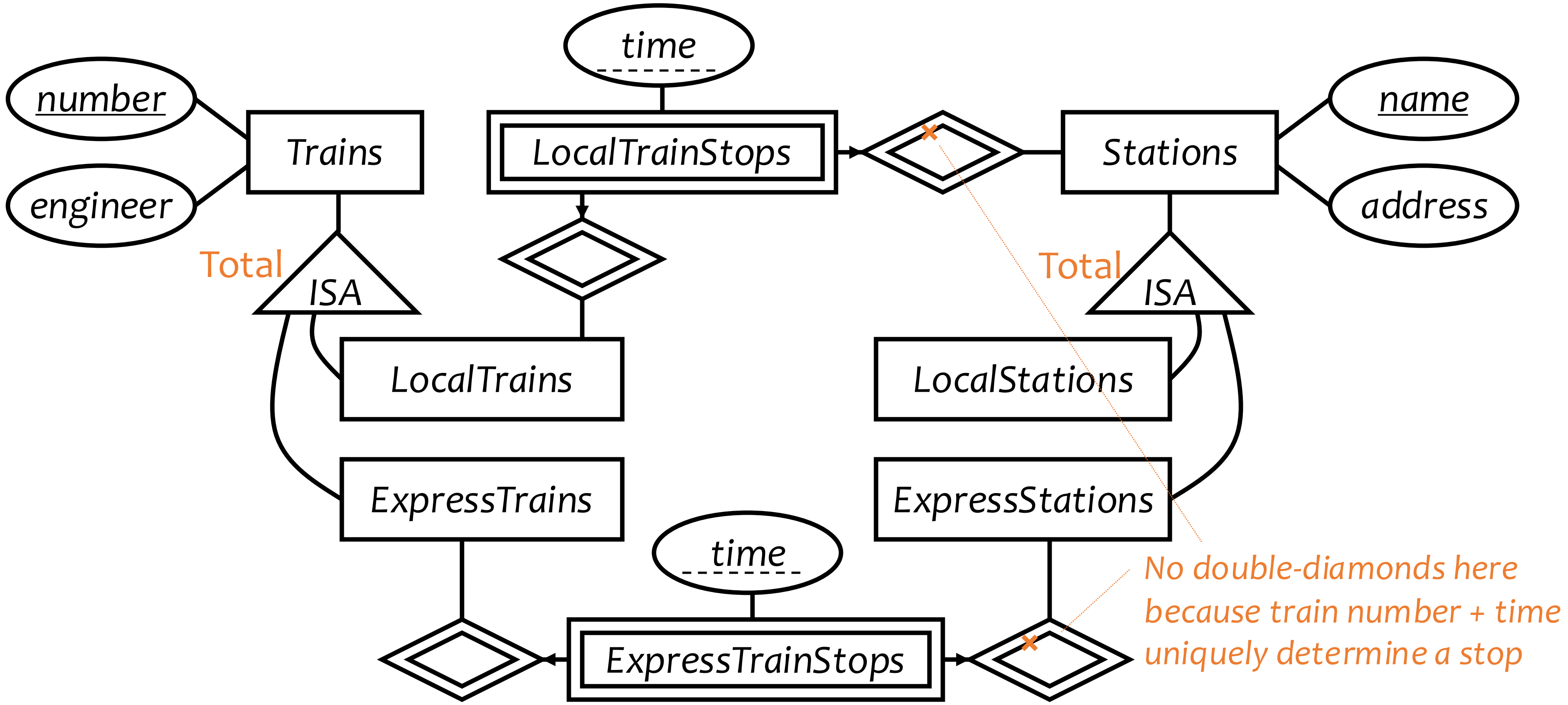
- A station has a unique name and an address, and is either an express station or a local station
- A train has a unique number and an engineer, and is either an express train or a local train
- …..

Case study 2: second design



- …
- 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
- …

Case study 2: second design



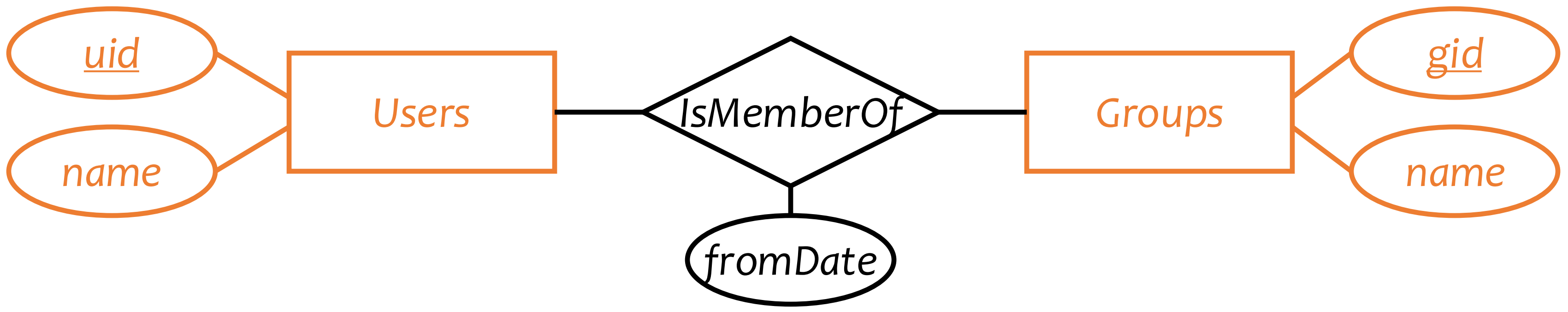
Is the extra complexity worth it?

Database Design

- Entity-Relationship (E/R) model
- Translating E/R to relational schema

Translating entity sets

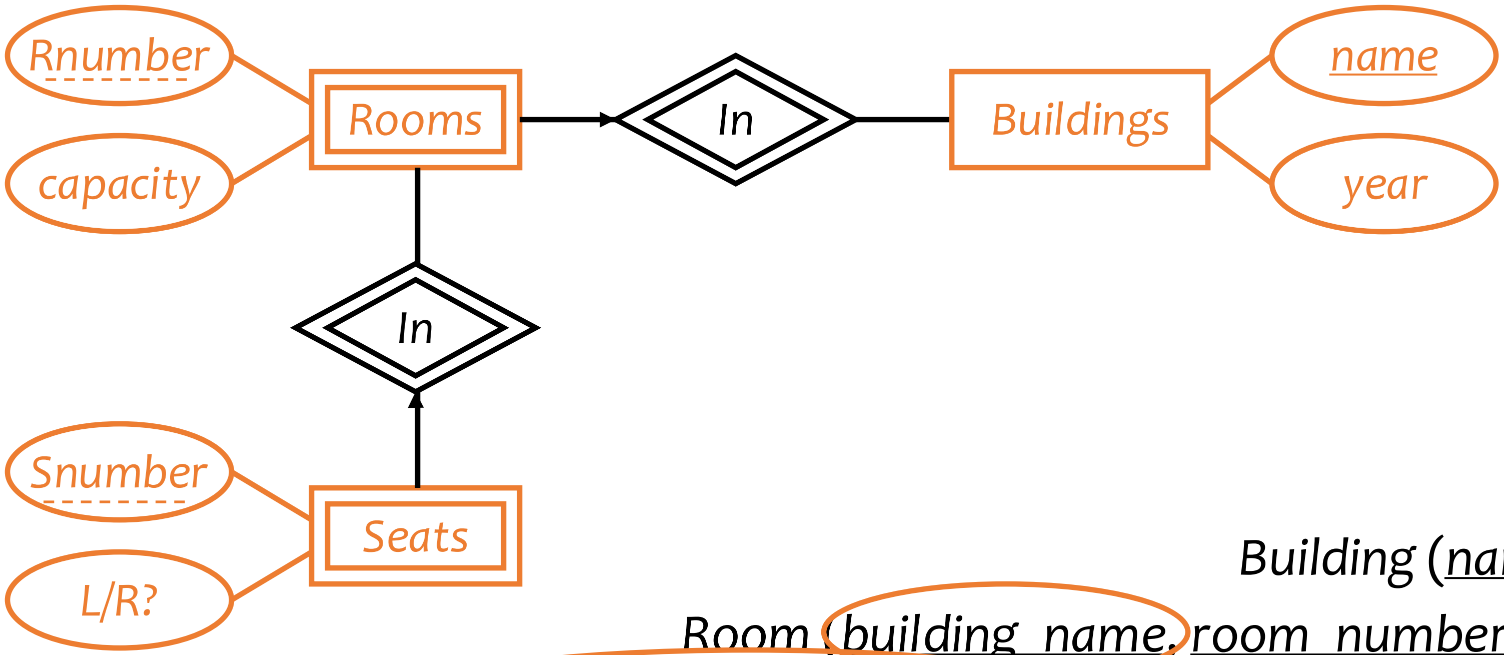
- An entity set translates directly to a table
  - Attributes → columns
  - Key attributes → key columns



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

Translating weak entity sets

- Remember the "borrowed" key attributes
- Watch out for attribute name conflicts
- Foreign key from weak to dominating entity set



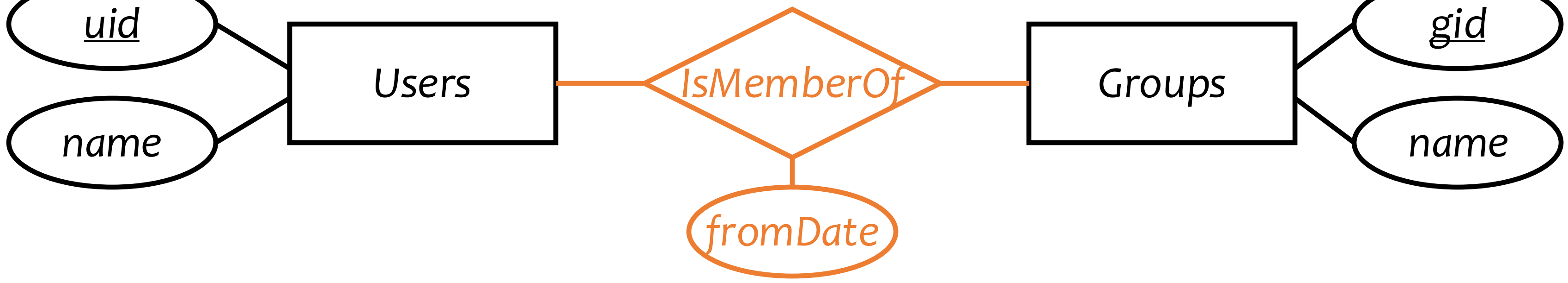
Building (*<u>name</u>*, *year*)

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 relationship set determines primary key:

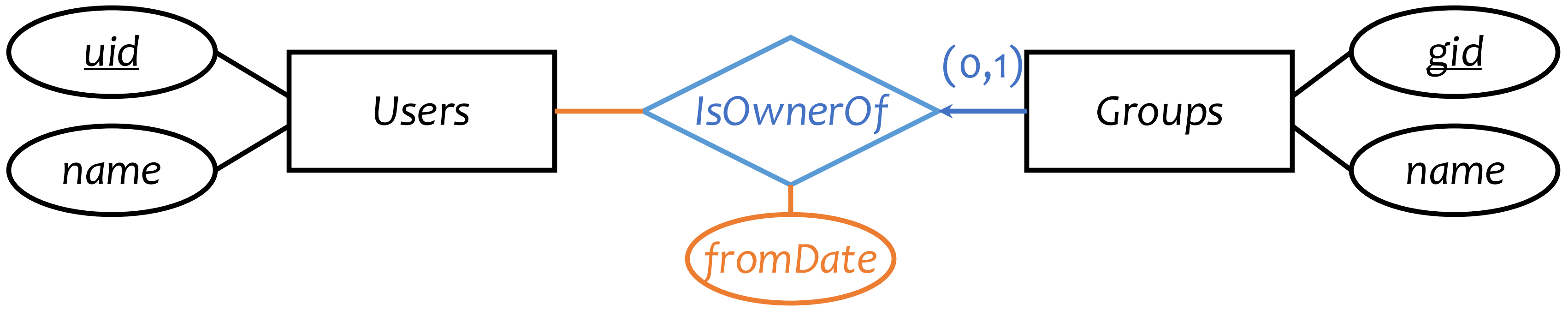


Member (<u>uid</u>, <u>gid</u>, fromDate)

- If we can deduce the general cardinality constraint (0,1) for a component 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 relationship set determines primary key:



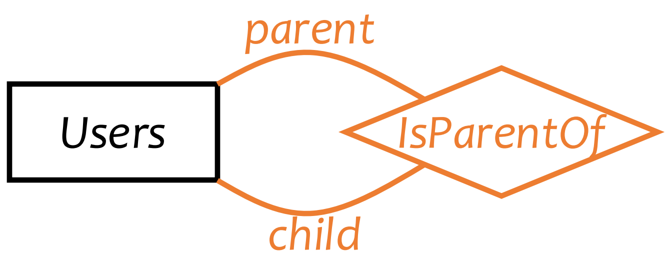
Owner (uid, <u>gid</u>, fromDate)

- If we can deduce the general cardinality constraint (0,1) for a component entity set E, then take the primary key attributes for E
- Otherwise, choose primary key attributes of each component entity

Translating relationship sets

- Also for each entity set E, whose primary key appears in the relationship table, add a foreign key back to E

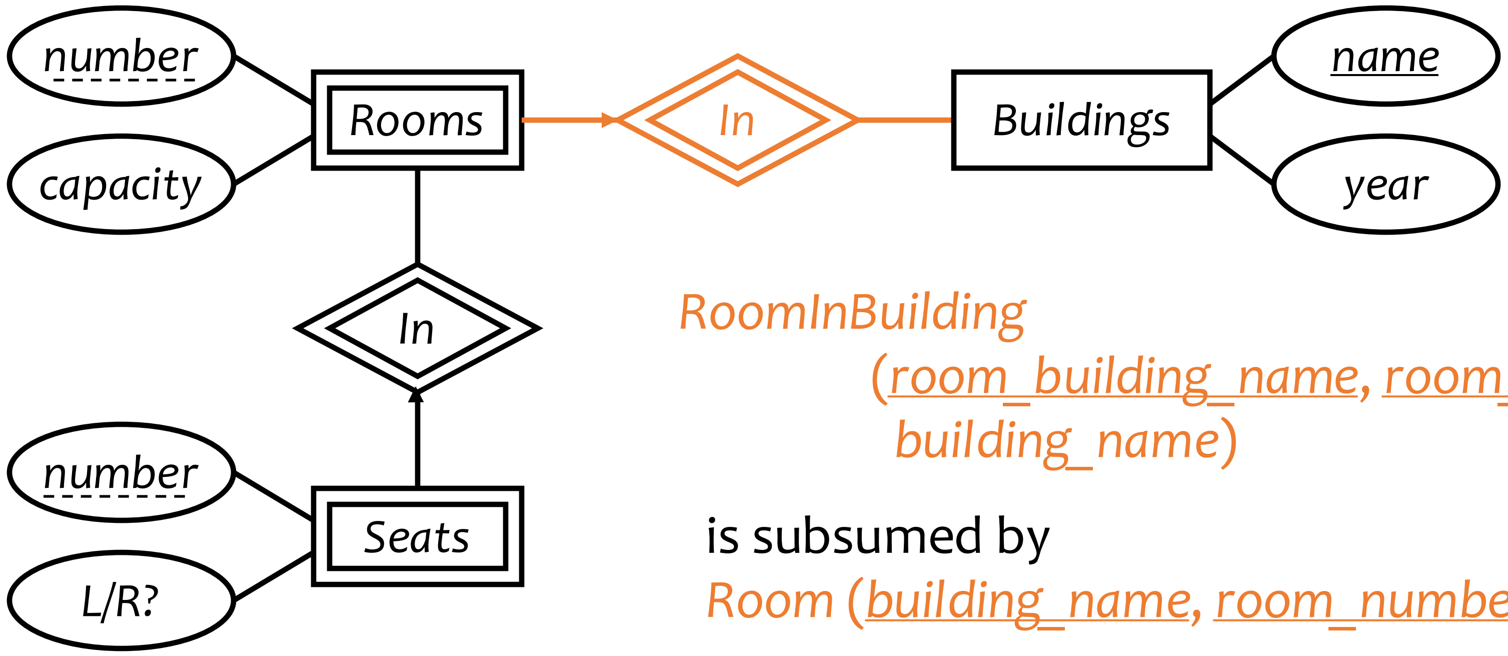
More examples



Parent (<u>parent_uid</u>, <u>child_uid</u>)

Translating double diamonds?

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



RoomInBuilding
(*room_building_name*, *room_number*,
building_name)

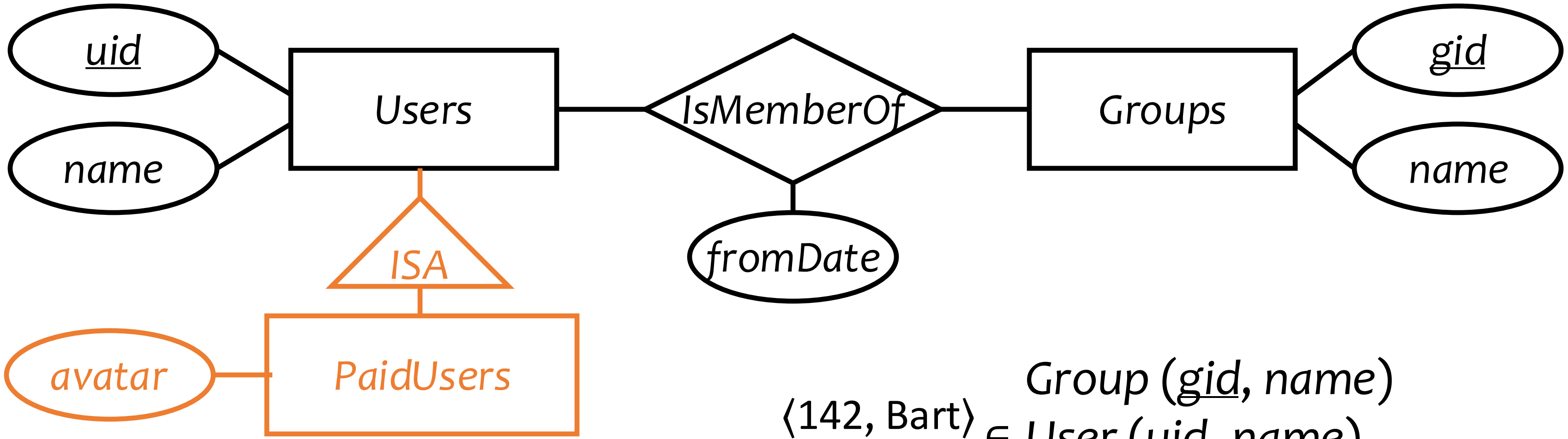
is subsumed by
Room (*building_name*, *room_number*, *capacity*)

Translating subclasses & ISA:

- 3 approaches (see next slides)
- Each can be more appropriate depending on whether the sub-superclass relationship is especially in terms of capturing “foreign key relationships”.
    - Disjoint or Overlapping
    - Partial or Total
- Ultimately you have a choice here and need to weigh pros and cons though I’ll make a recommendation as well.

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
  - Reasonable for Partial ISA relationships



Also add foreign key: From PaidUser to User

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

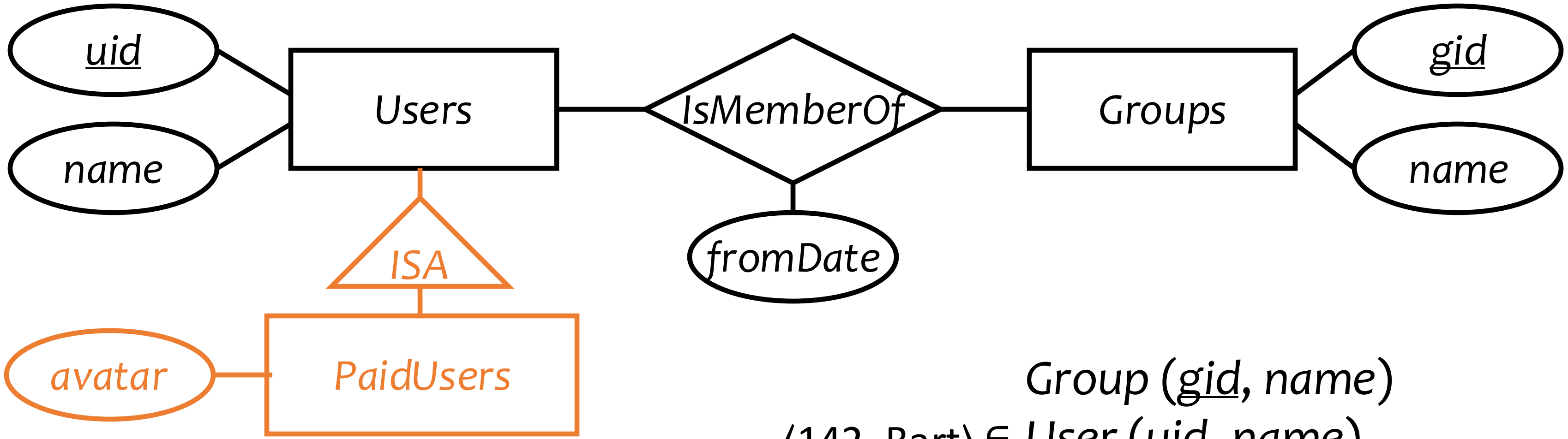
⟨142, Bart⟩
⟨456, Ralph⟩ ∈ User (<u>uid</u>, name)

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

⟨456, ☺⟩ ∈ PaidUser (<u>uid</u>, 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
  - Reasonable for Total and Disjoint cases



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

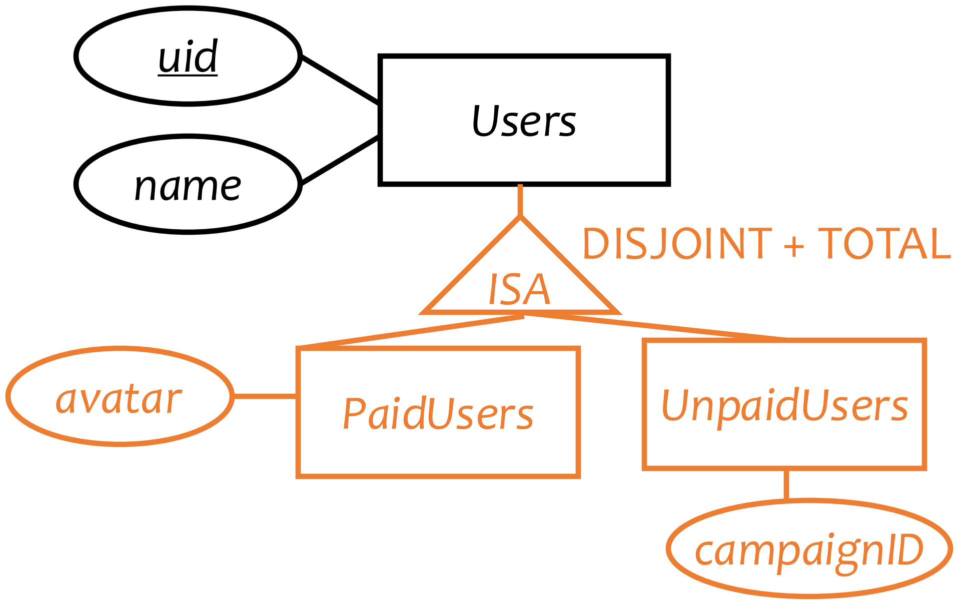
⟨142, Bart⟩ ∈ User (<u>uid</u>, name)

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

⟨456, Ralph, ☺⟩ ∈ PaidUser (<u>uid</u>, name, avatar)

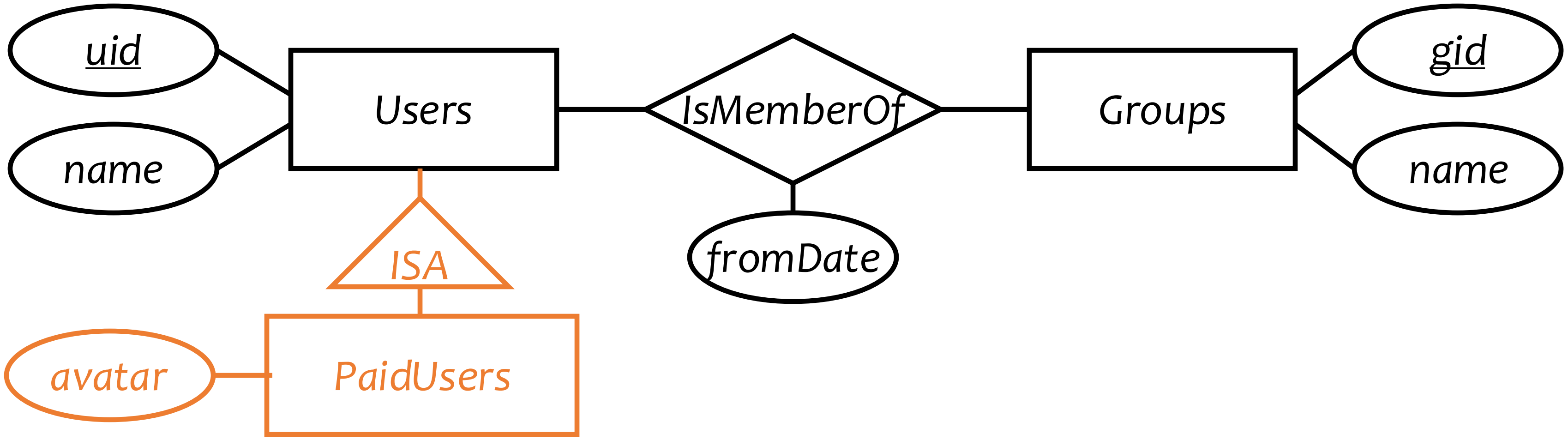
More on Entity-in-most-specific approach

- In this case, since each user has to be either Paid or Unpaid, might makes sense to compile to:
  - PaidUsers(uid, name, avatar)
  - UnpaidUsers(uid, name, campaignID)



Problem with Entity-in-most-specific approach

- Foreign keys of relationship sets:
  - Consider the foreign key of IsMemberOf to User.
  - We can no longer add this constraint because some users are only represented in PaidUsers

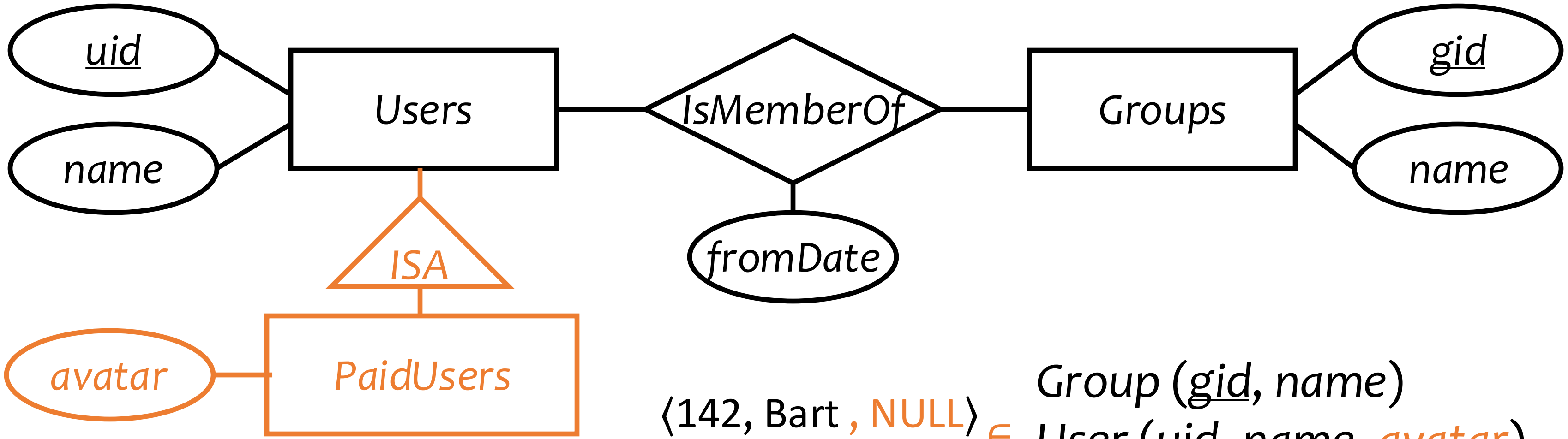


Possible Solution: Keep superclass tables with PKs

- We can still keep User(uid) to capture FKs of IsMemberOf.
- Not “really” entity-in-most-specific-class anymore but attributes of an entity are all in superclasses

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

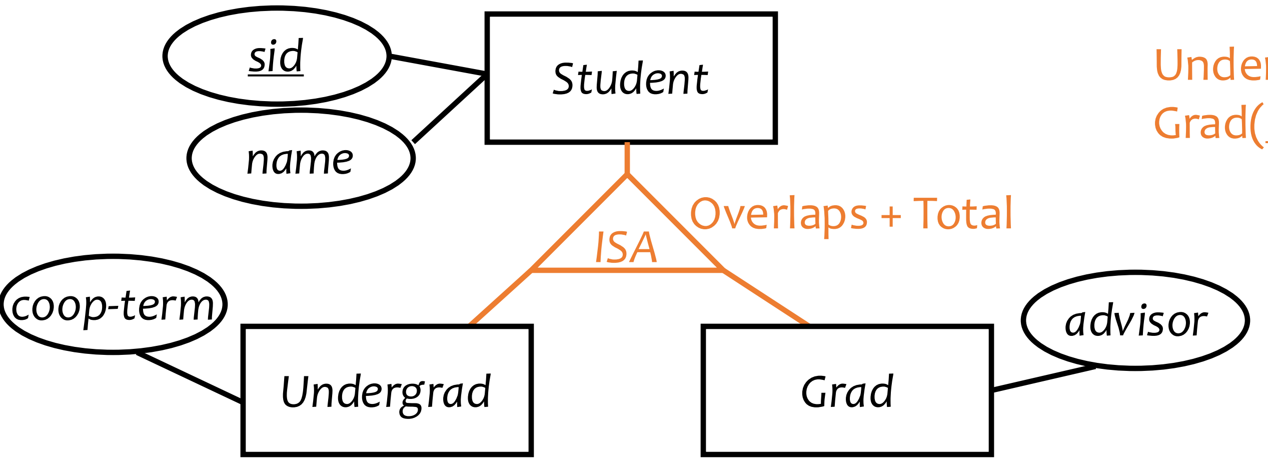
- Entity-in-all-superclasses
  - *User* (*uid*, *name*), *PaidUser* (*uid*, *avatar*)
  - Pro: All users are found in one table
  - Con: Attributes of paid users are scattered in different tables
- Entity-in-most-specific-class
  - *User* (*uid*, *name*), *PaidUser* (*uid*, *name*, *avatar*)
  - Pro: All attributes of paid users are found in one table
  - Con: Users are scattered in different tables & May lose FKs of the relationshipsets of the superclass.
- 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

General Recommendations

- Try to choose entity-in-all-superclasses:
  - Compared to Entity-in-most-specific-class: can lead to attributes of entities to be distributed
  - But there is value in terms of data cleanness for adding foreign keys if superclasses participate in relationships (recall the foreign key problem of entity-in-most-specific-class approach)
  - Compared to all-entities-in-one-table: NULLs are also a nightmare in data cleanness. So you should specialize when possible.

General Recommendations

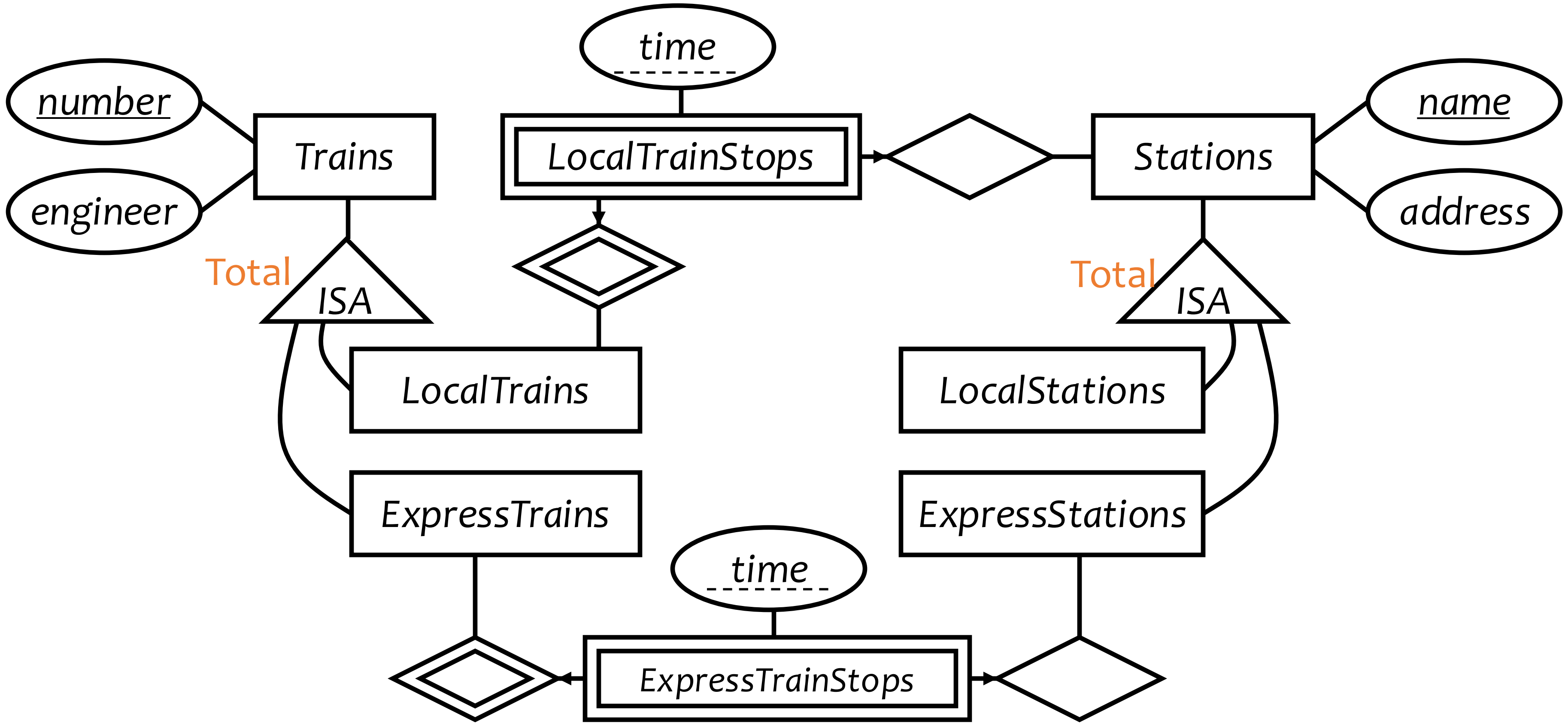
- Don’t recommend choosing entity-in-most-specific if OVERLAPPING sub-classes exist



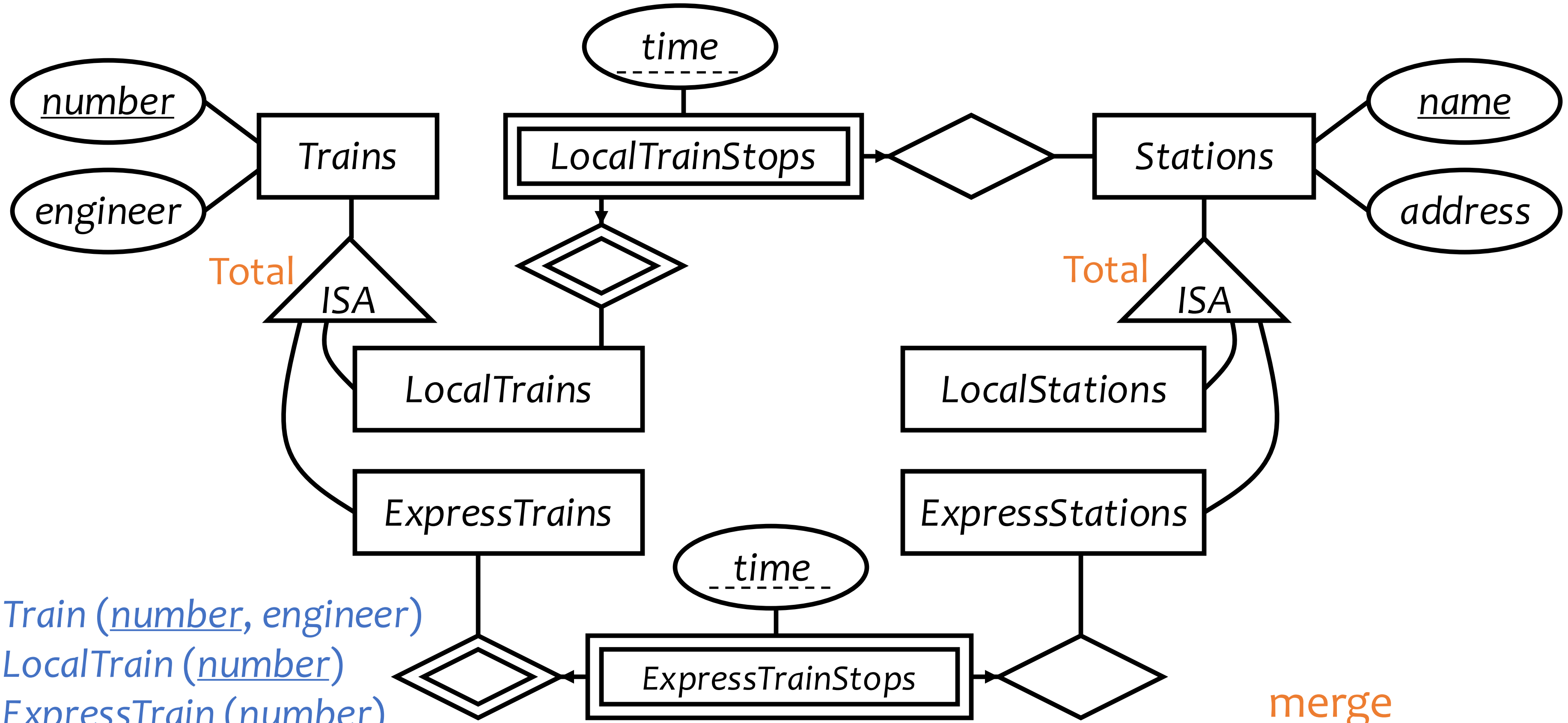
Undergrad(sid, name, coop-term)
Grad(sid, name, advisor)

- name replicated for a student who is both undergrad & grad
- *Ultimately these are rules of thumb. Need to weigh pros/cons of each approach.*

A complete example



A complete example



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

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

  ☞Arguably a better design because it is simpler!

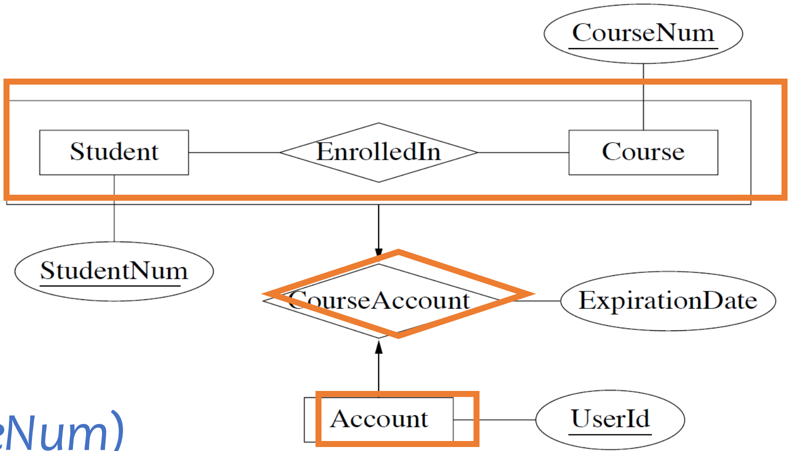
Design principles

- KISS
  - Keep It Simple, Stupid
- Avoid redundancy
- 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

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

More examples

- Representing aggregation
  - No separate relation for aggregation A
  - Represent tables and relationships in *R as before (including FKs)*
  - To represent relationship set, say CourseAccount, involving an aggregation A, treat A as an entity set whose PK is the PK of its defining relationship (EnrolledIn in this case). Also apply the rules of simplifying the PK of CourseAccount if any of its participating entity sets have (0,1) multiplicity



Student (<u>StudentNum</u>)
Couse(<u>CourseNum</u>)
Account(<u>UserID</u>)

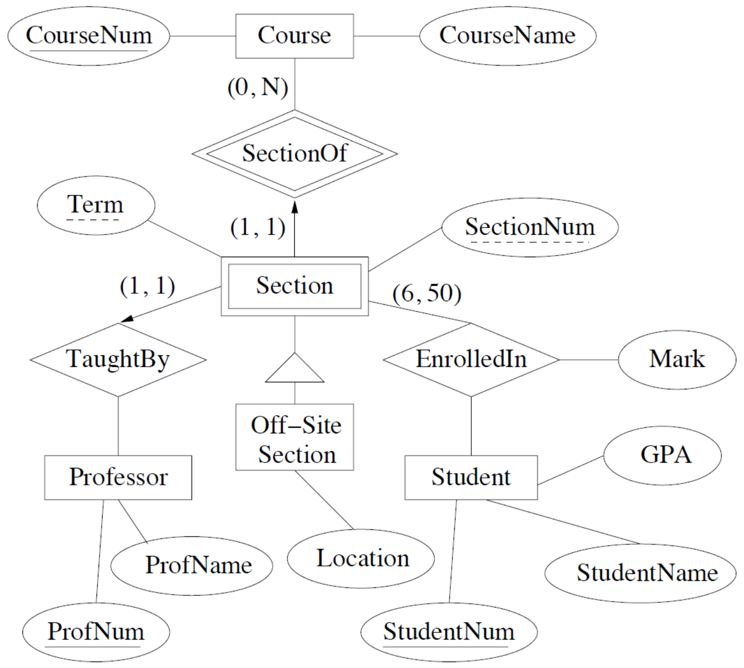
EnrolledIn(<u>StudentNum,CouseNum</u>)

CouseAccount(<u>UserId, StudentNum, CourseNum</u>, ExpirationDate)

One-to-one relationships → We can simply take UserId or (StudentNum, CourseNum) as the key

More examples (Exercise)

- ER Diagram

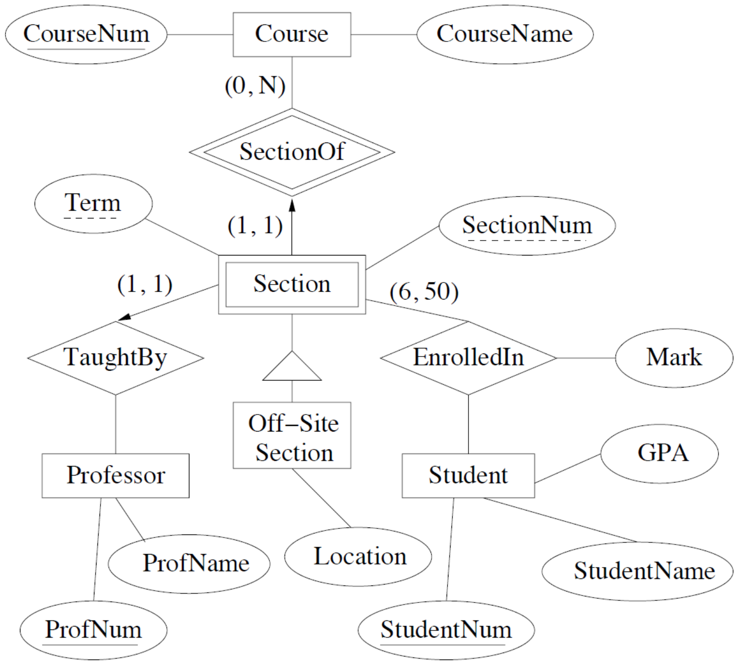


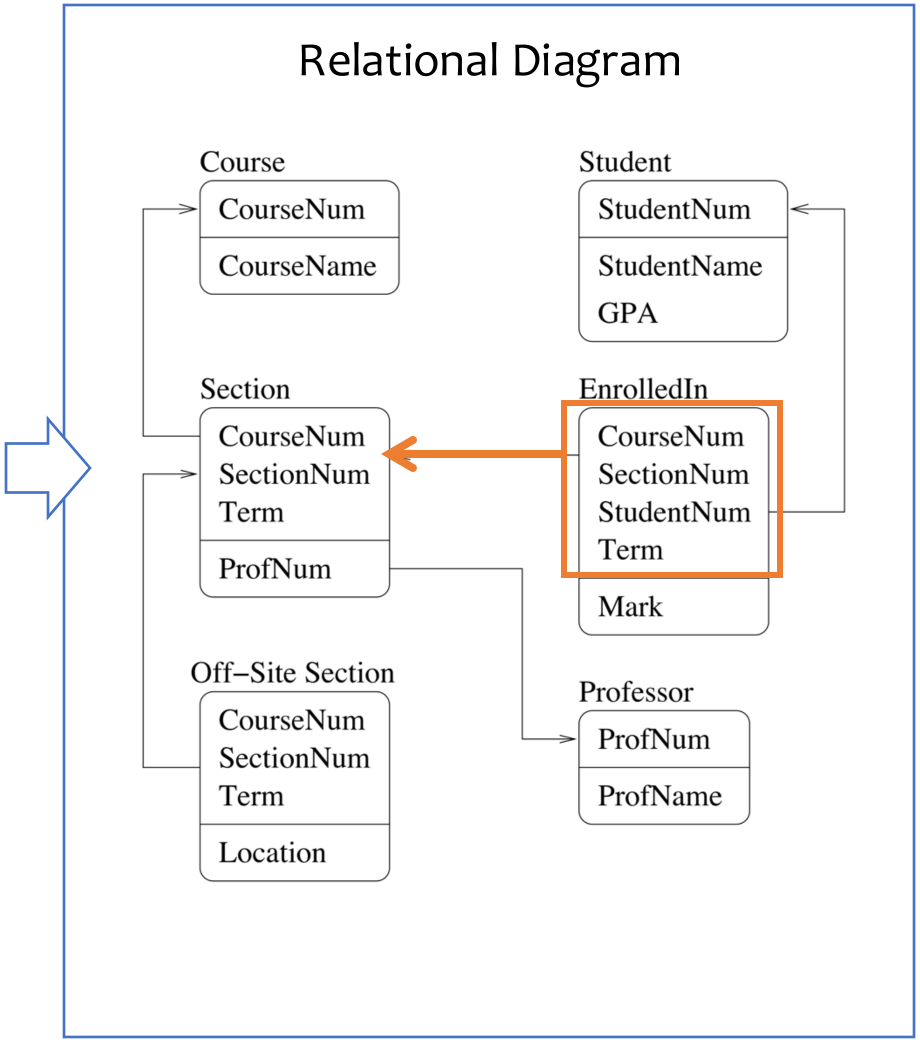
Relational Schema

?

More examples

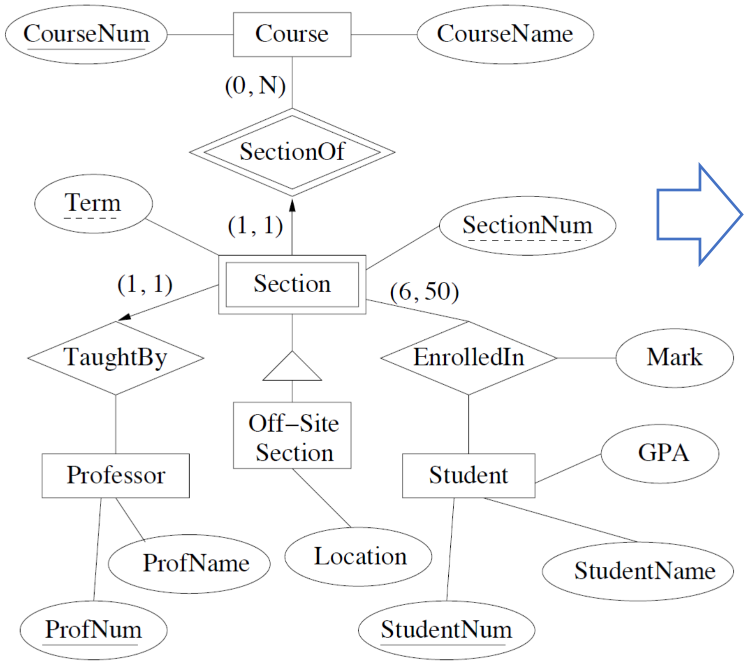
- ER Diagram





More examples

- ER Diagram



Relational DDL Commands

```
CREATE TABLE Course
(CourseNum INTEGER PRIMARY KEY,
 CourseName CHAR(50));

CREATE TABLE Student
(StudentNum INTEGER PRIMARY KEY,
StudentName CHAR(50),
GPA FLOAT);

CREATE TABLE Professor
(ProfNum INTEGER PRIMARY KEY,
 ProfName CHAR(50));

CREATE TABLE Section
(CourseNum INTEGER  NOT NULL REFERENCES Course(CourseNum),
SectionNum INTEGER  NOT NULL,
Term INTEGER  NOT NULL,
PRIMARY KEY(CourseNum, SectionNum, Term),
ProfNum INTEGER NOT NULL REFERENCES Professor(ProfNum));

CREATE TABLE Off-SiteSection
(CourseNum INTEGER  NOT NULL,
SectionNum INTEGER NOT NULL,
Term INTEGER NOT NULL,
FOREIGN KEY(CouseNum,SectionNum,Term) REFERENCES
                    Section(CouseNum,SectionNum,Term),
Location CHAR(50));

CREATE TABLE EnrolledIn
(CourseNum INTEGER  NOT NULL,
SectionNum INTEGER NOT NULL,
Term INTEGER NOT NULL,
StudentNum INTEGER NOT NULL REFERENCES Student(StudentNum),
FOREIGN KEY(CouseNum,SectionNum,Term) REFERENCES
                    Section(CouseNum,SectionNum,Term),
Primary Key(CouseNum,SectionNum,Term,StudentNum),
Mark INTEGER);
```