Lecture 10: Database Design (E/R Translation)

CS348 Spring 2025:
Introduction to Database Management

Instructor: **Xiao Hu**

Sections: 001, 002, 003

Announcements

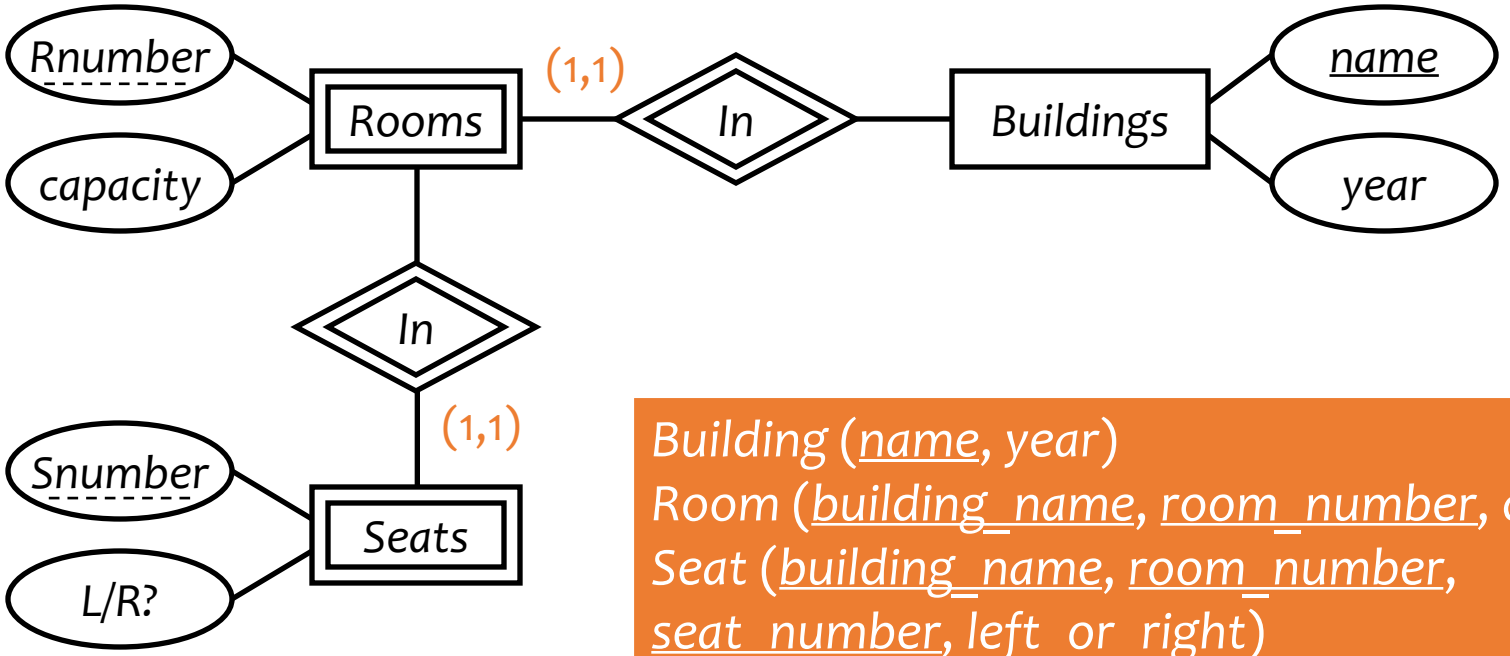
- Assignment 2 released on Learn
  - Due on 11:59 PM Jun 24
  - Coverage: Lectures 4 – 12
  - Content: SQL and Database design
  - Crowdmark and Marmost are already open

(Recap) 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:
  - Composite and Multi-valued attributes
  - Aggregation

E/R Model

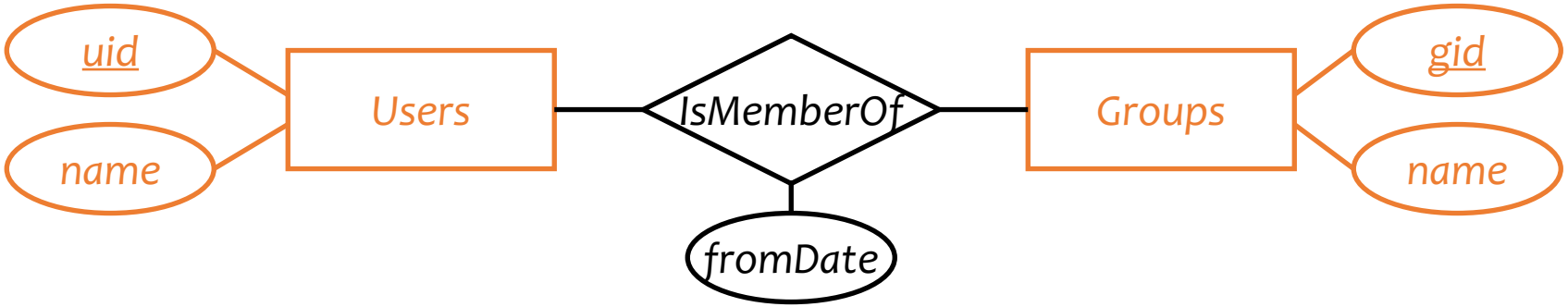
- E/R Concepts
- E/R Schema Design
- Next: Translating E/R to relational schema



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

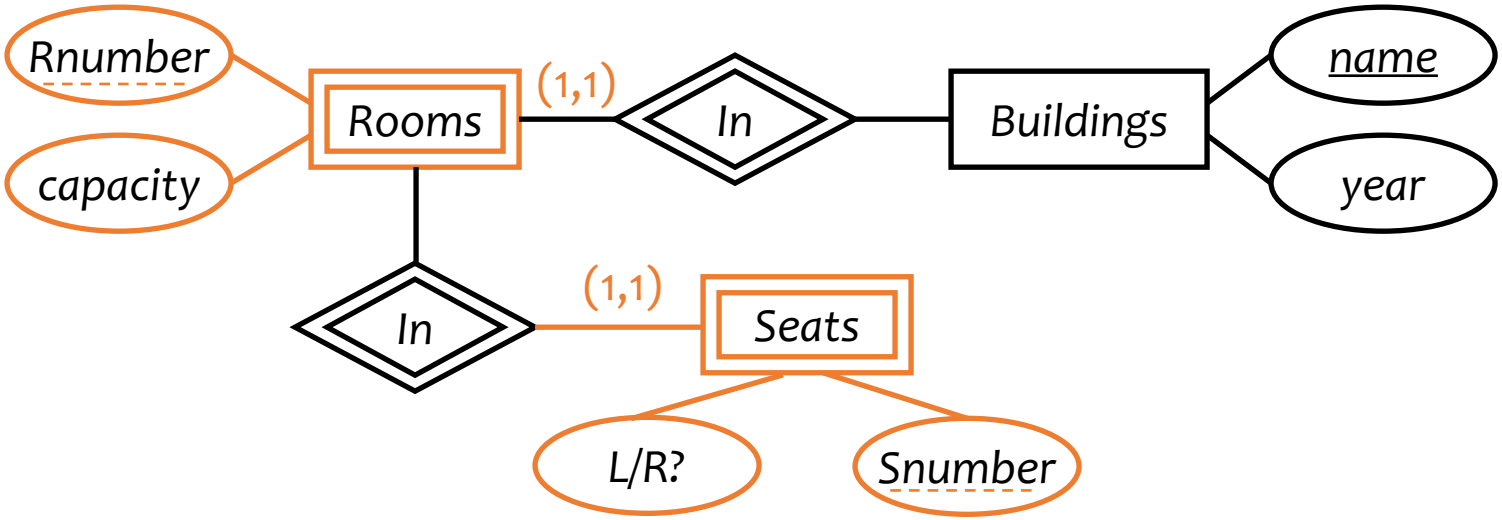


User (uid, name)

Group (gid, name)

Translating weak entity sets

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



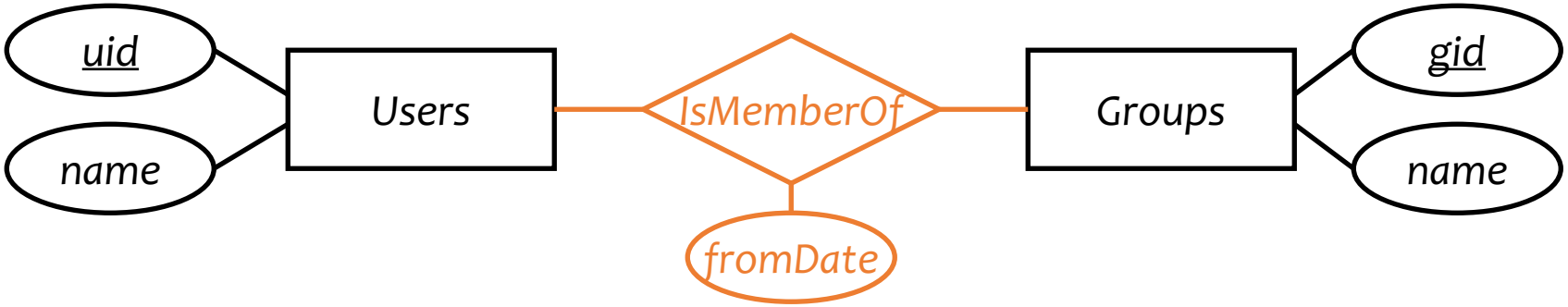
Building (<u>building_name</u>, *year*)

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

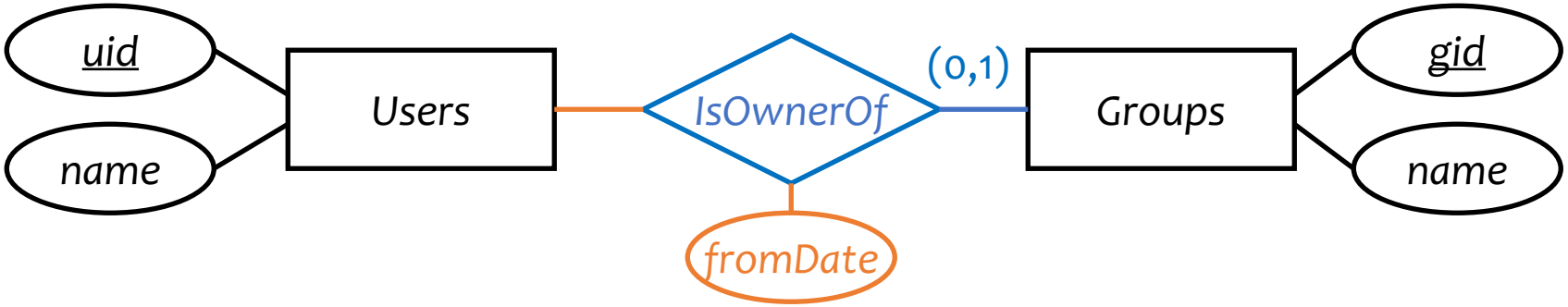


Member (uid, gid, fromDate)

- 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

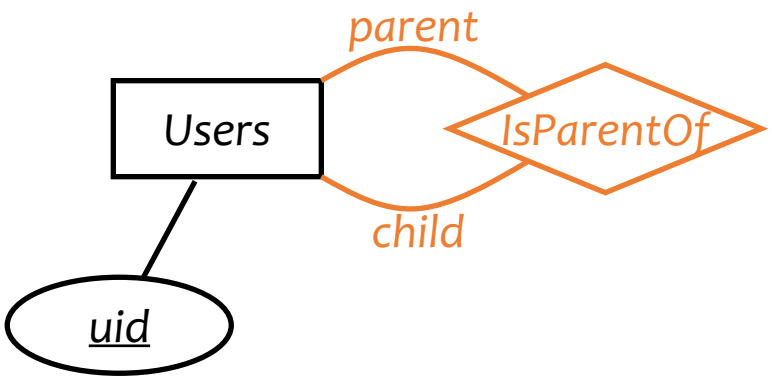
- A relationship set translates to a table
  - Keys of connected entity sets → columns
  - Attributes of the relationship set (if any) → columns
  - Multiplicity of the relationship set determines the key of the table



Owner (uid, gid, 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

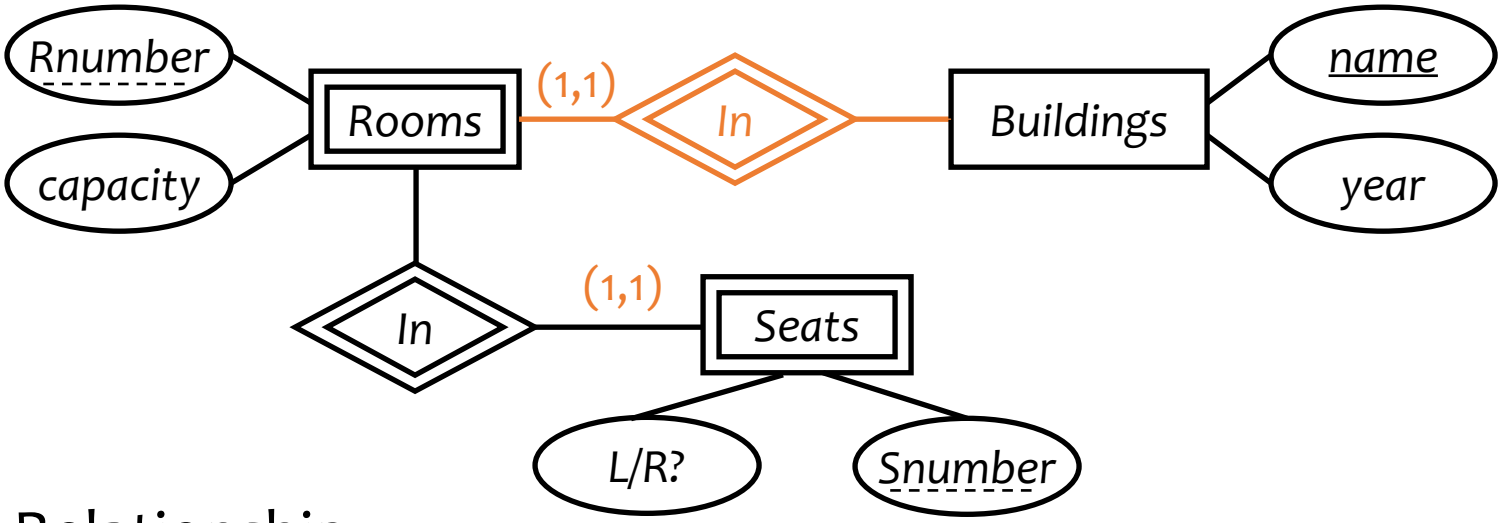
More examples



Parent (*parent_uid*, *child_uid*)

Translating double diamonds?

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



Relationship

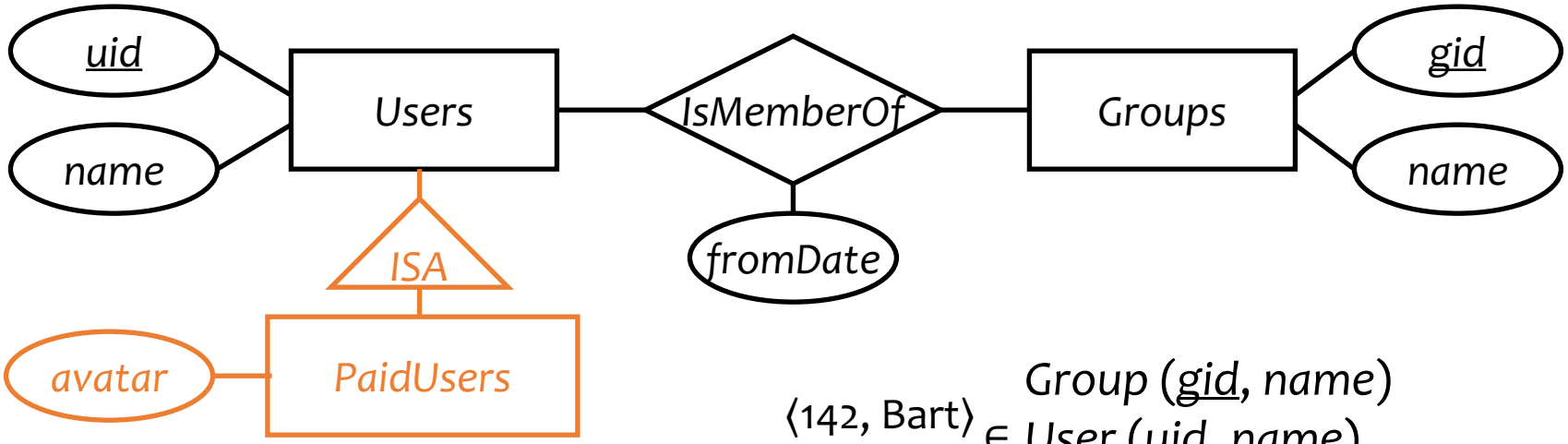
RoomInBuilding (*building_name*, *room_number*)

is subsumed by entity

Room (*building_name*, *room_number*, *capacity*)

Translating ISA: Approach 1

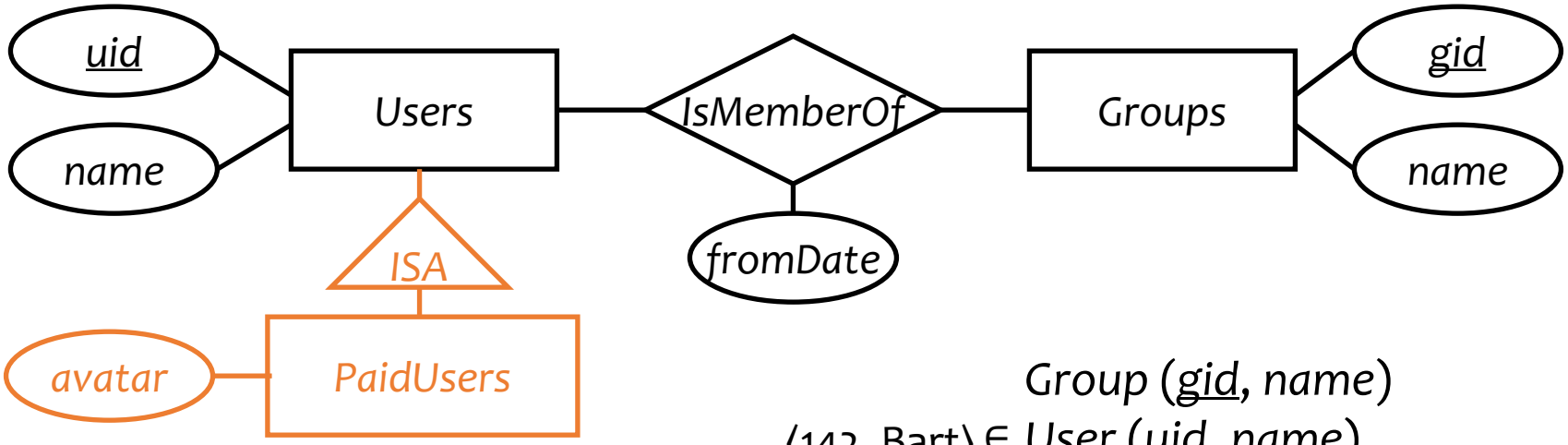
- Entity-in-all-superclasses approach ("E/R style")
  - An entity is represented in the table for each subclass to which it belongs
  - A table includes only the attributes directly attached to the corresponding entity set, plus the inherited key



⟨142, Bart⟩ ∈ Group (gid, name)
⟨456, Ralph⟩ ∈ User (uid, name)
Member (uid, gid, from_date)
⟨456, ☺⟩ ∈ PaidUser (uid, avatar)

Translating ISA: Approach 2

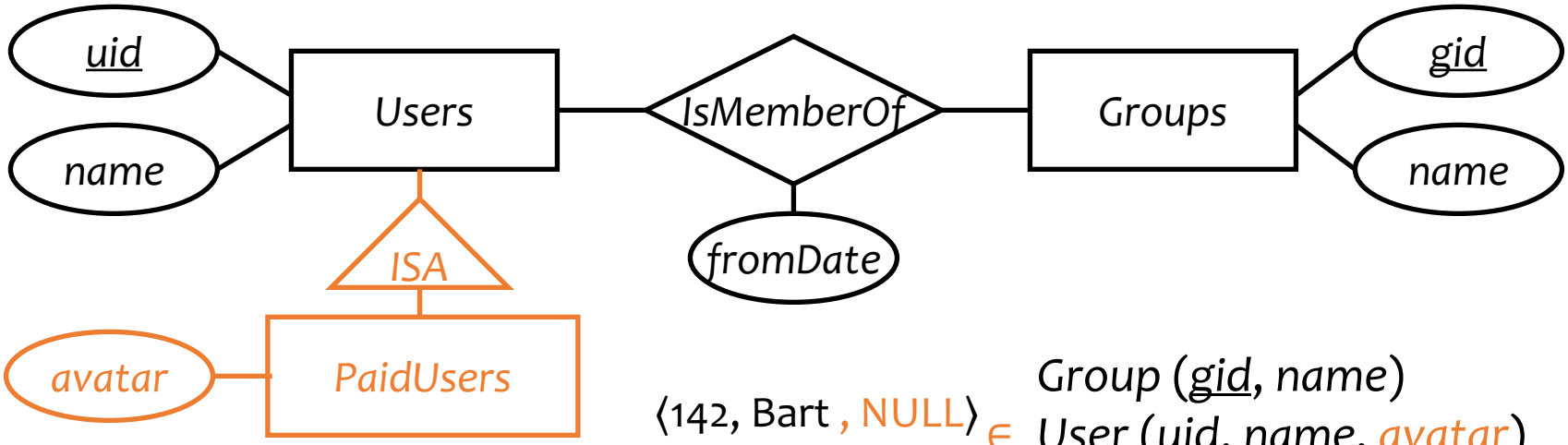
- Entity-in-most-specific-class approach ("OO style")
  - An entity is only represented in one table (the most specific entity set to which the entity belongs)
  - A table includes the attributes attached to the corresponding entity set, plus all inherited attributes



Group (*gid*, *name*)
⟨142, Bart⟩ ∈ *User* (*uid*, *name*)
Member (*uid*, *gid*, *from_date*)
⟨456, Ralph, ☺⟩ ∈ *PaidUser* (*uid*, *name*, *avatar*)

Translating 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 irrelevant columns for a particular entity



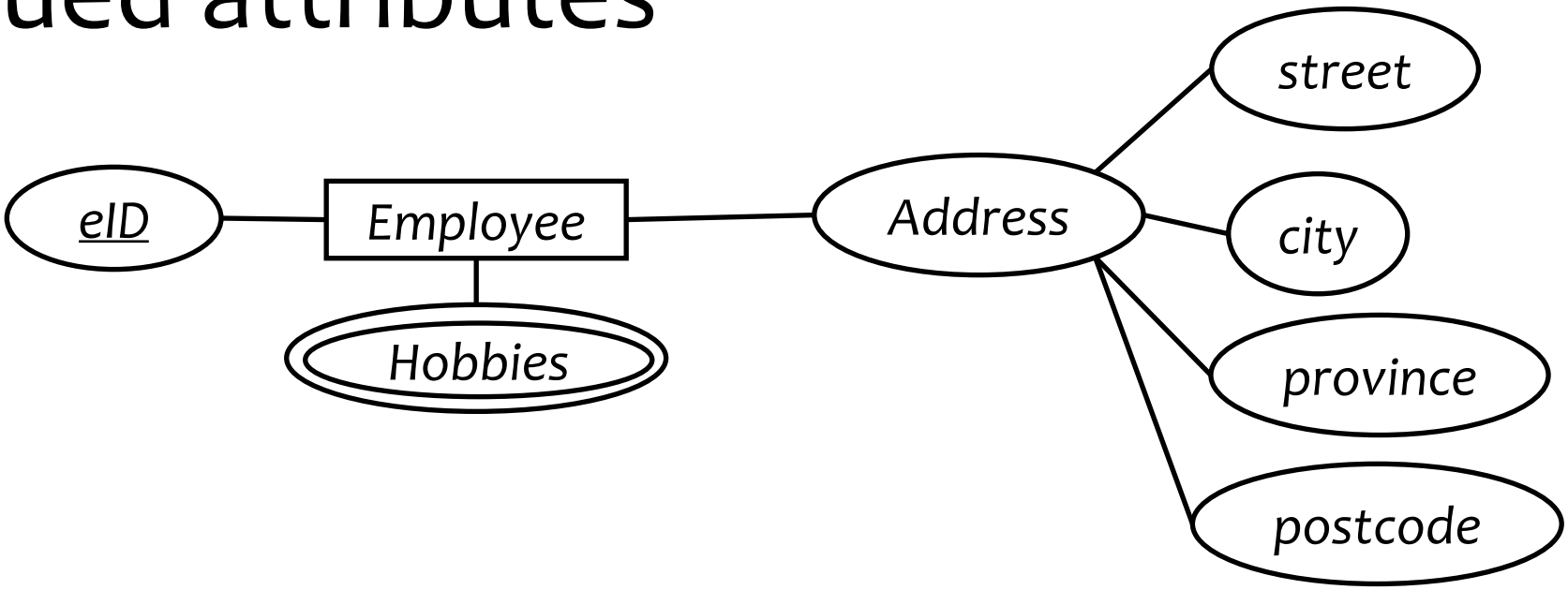
⟨142, Bart , NULL⟩ ∈
⟨456, Ralph, ☺⟩

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

Comparison of three approaches

- Entity-in-all-superclasses
  - *User* (*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
- 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

Translating composite and multi-valued attributes

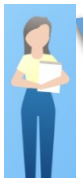


Composite: *Employee(eID, street, city, province, postcode)*

Multi-valued: *EmployeeHobbies(eID, hobby)*

Foreign key: *eID references Employee*

Case study 2

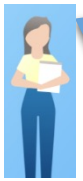


Design a database consistent with the following:

- 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

Can you draw a E/R diagram?

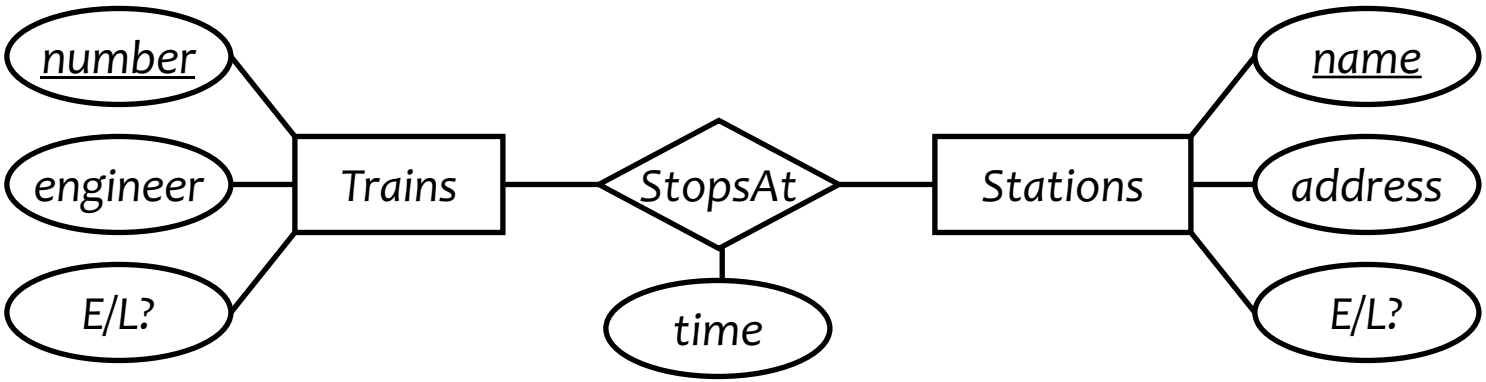
Case study 2: first design



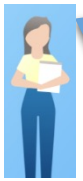
Design a database consistent with the following:

- 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?



Case study 2: first design

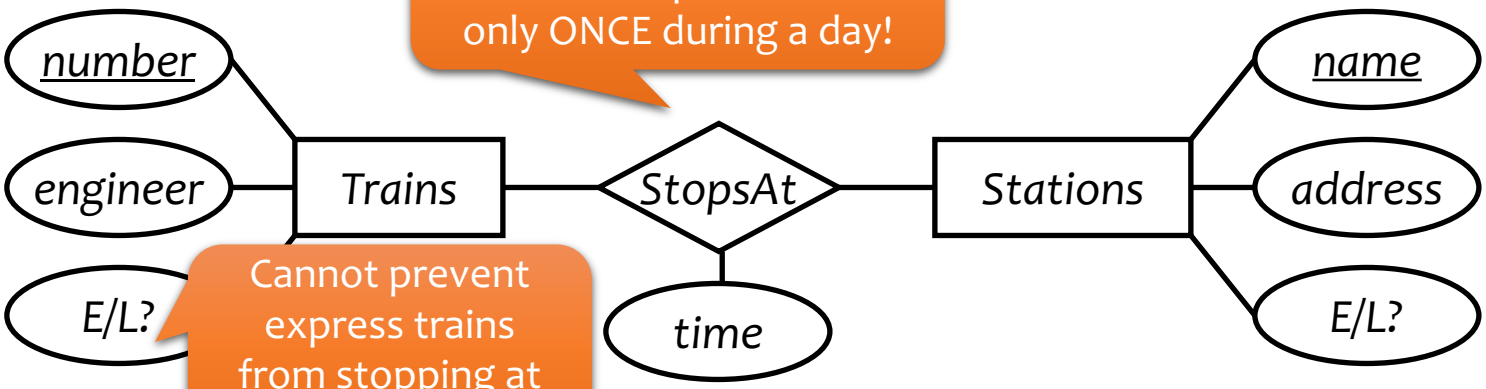


Design a database consistent with the following:

- 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!

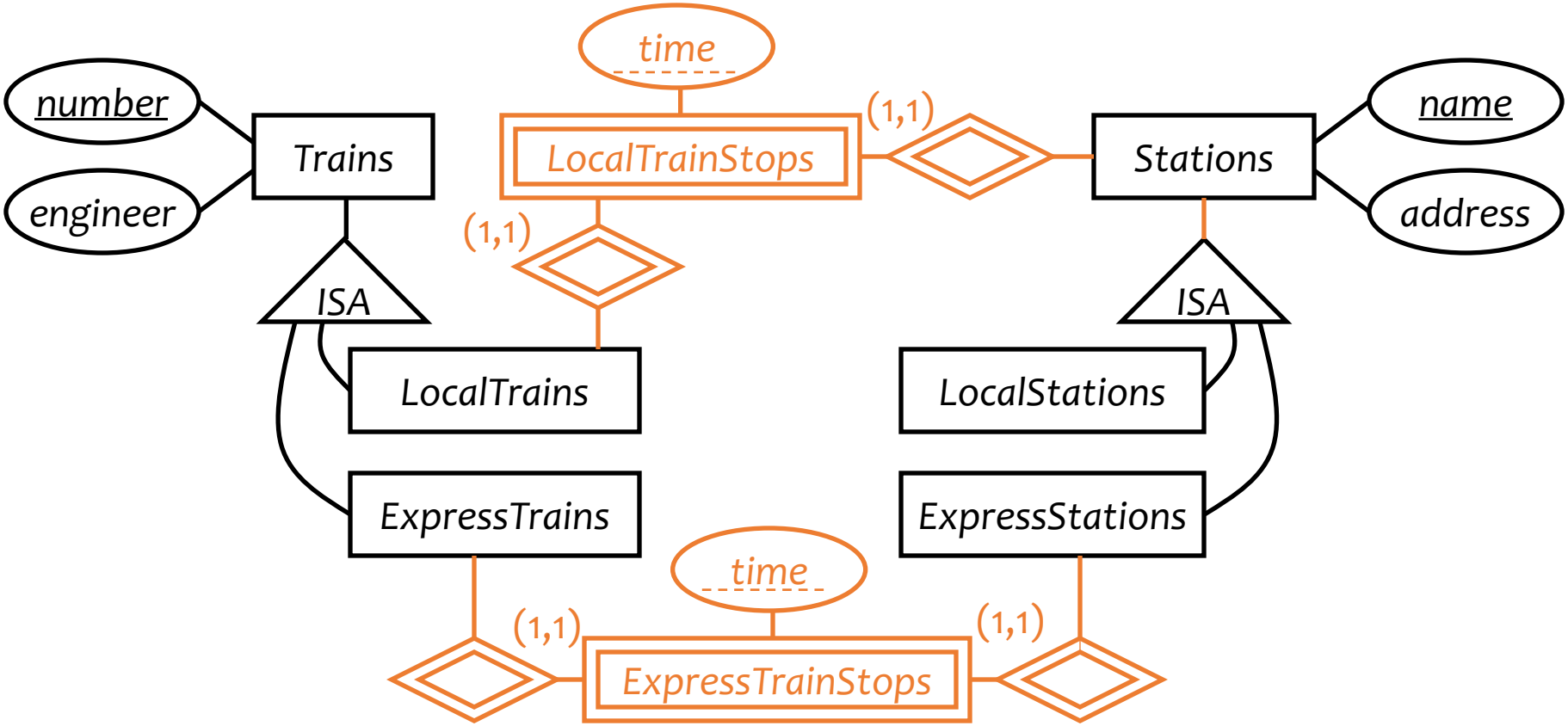


Cannot prevent express trains from stopping at any local stations

Case study 2: second design

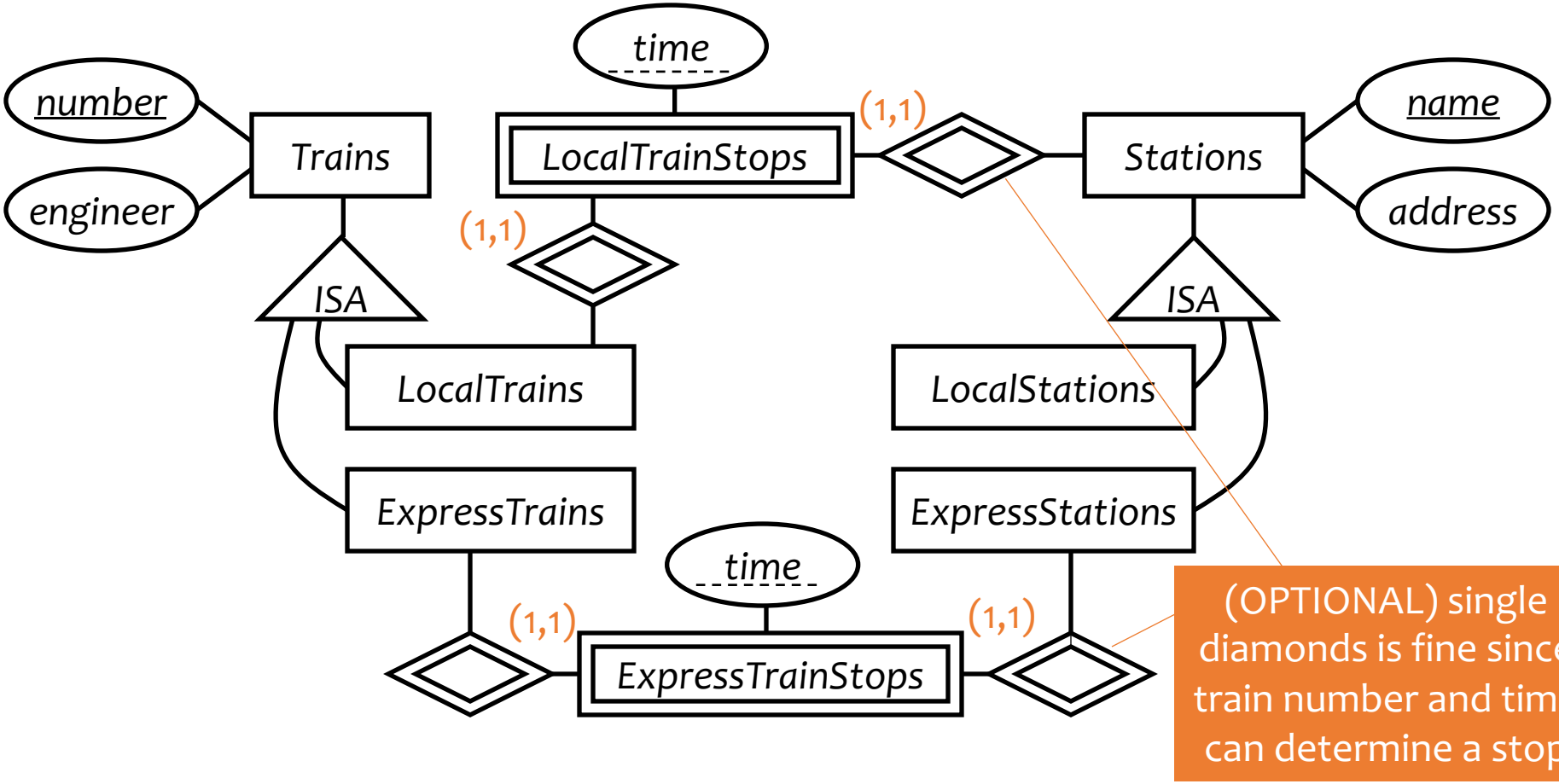
- A station has a unique name and an address, and is either an express station or a local station
- A train has a unique number and an engineer, and is either an express train or a local train
- …..

Case study 2: second design



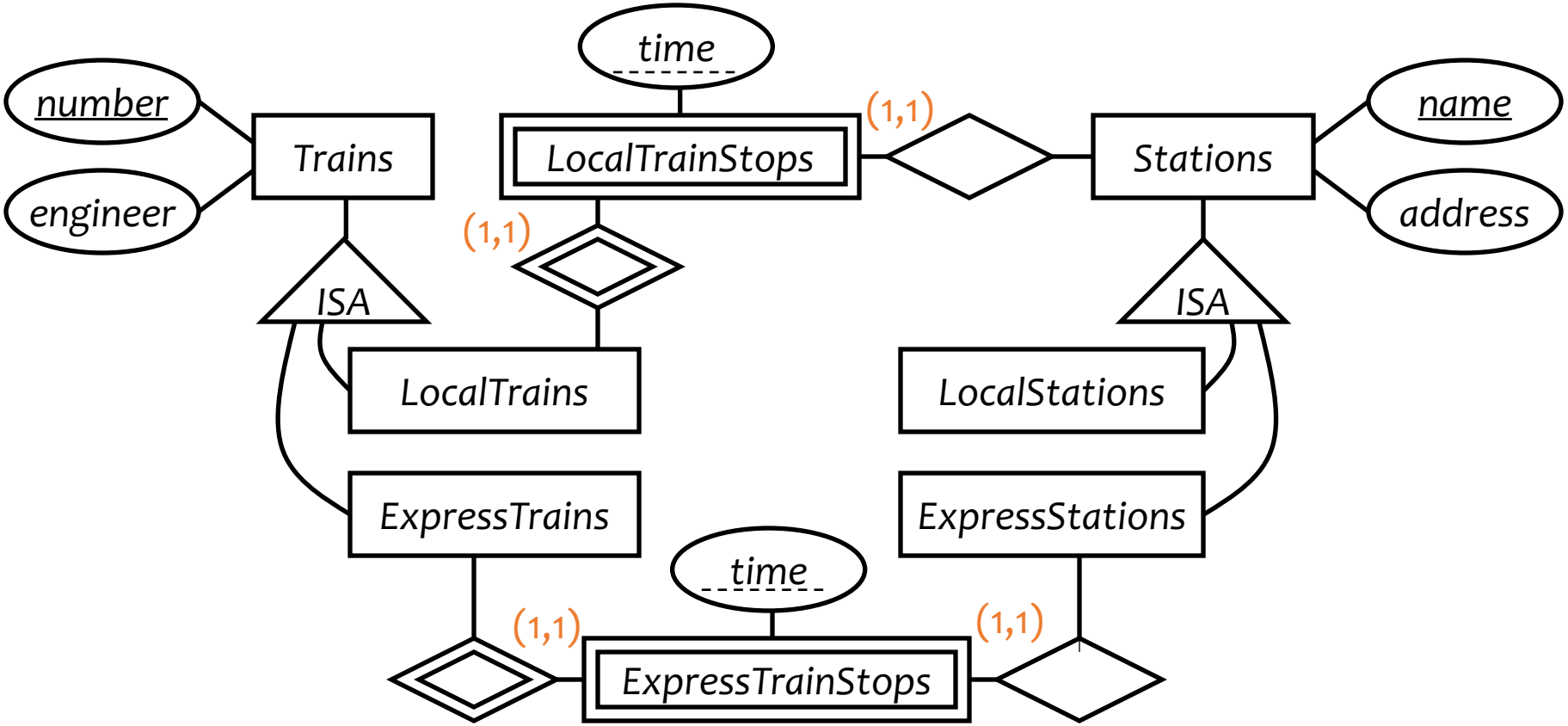
- …
- A local train can stop at any station
- An express train only stops at express stations
- A train can stop at a station for any number of times during a day
- …

Case study 2: second design

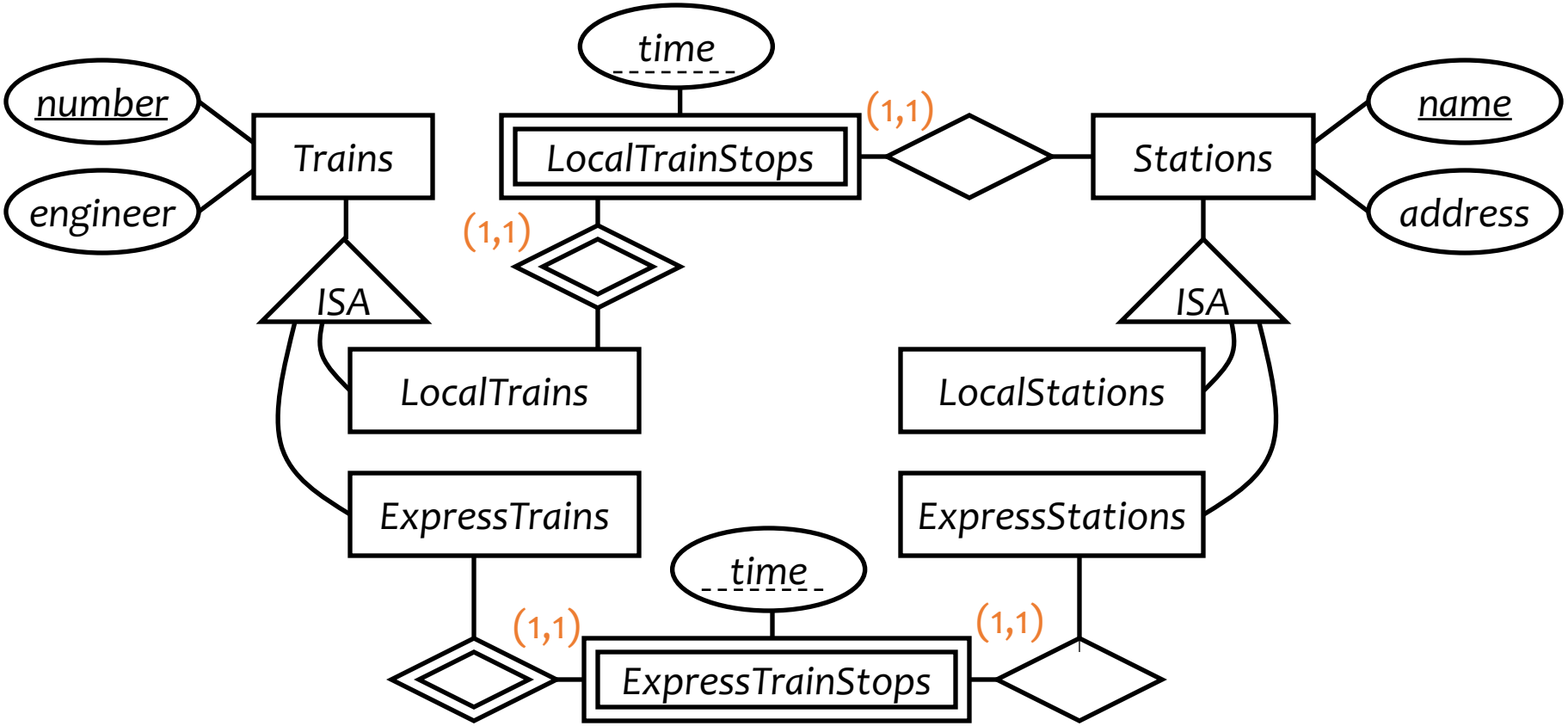


Is the extra complexity worth it? Yes! Captures more constraints and avoids unintended info

Case study 2: final E/R diagram



Case study 2: E/R translation



Train (<u>train_number</u>, *engineer*)
LocalTrain (<u>local_train_number</u>)
ExpressTrain (<u>express_train_number</u>)

Station (<u>station_name</u>, *address*)
LocalStation (<u>local_station_name</u>)
ExpressStation (<u>express_station_ame</u>)

LocalTrainStopsAtStation (<u>local_train_number</u>, <u>time</u>, *station_name*)

ExpressTrainStopsAtStation (<u>express_train_number</u>, <u>time</u>, *express_station_name*)

Case study 2: Simplification

- Eliminate *the 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 $\pi_{name}(Station) - ExpressStation$

Train (<u>train_number</u>, *engineer*)
LocalTrain (<u>local_train_number</u>)
ExpressTrain (<u>express_train_number</u>)

Station (<u>station_name</u>, *address*)
LocalStation (<u>local_station_name</u>)
ExpressStation (<u>express_station_ame</u>)

LocalTrainStopsAtStation (<u>local_train_number</u>, <u>time</u>, *station_name*)

ExpressTrainStopsAtStation (<u>express_train_number</u>, <u>time</u>, *express_station_name*)

Case study 2: An alternative design

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

- 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

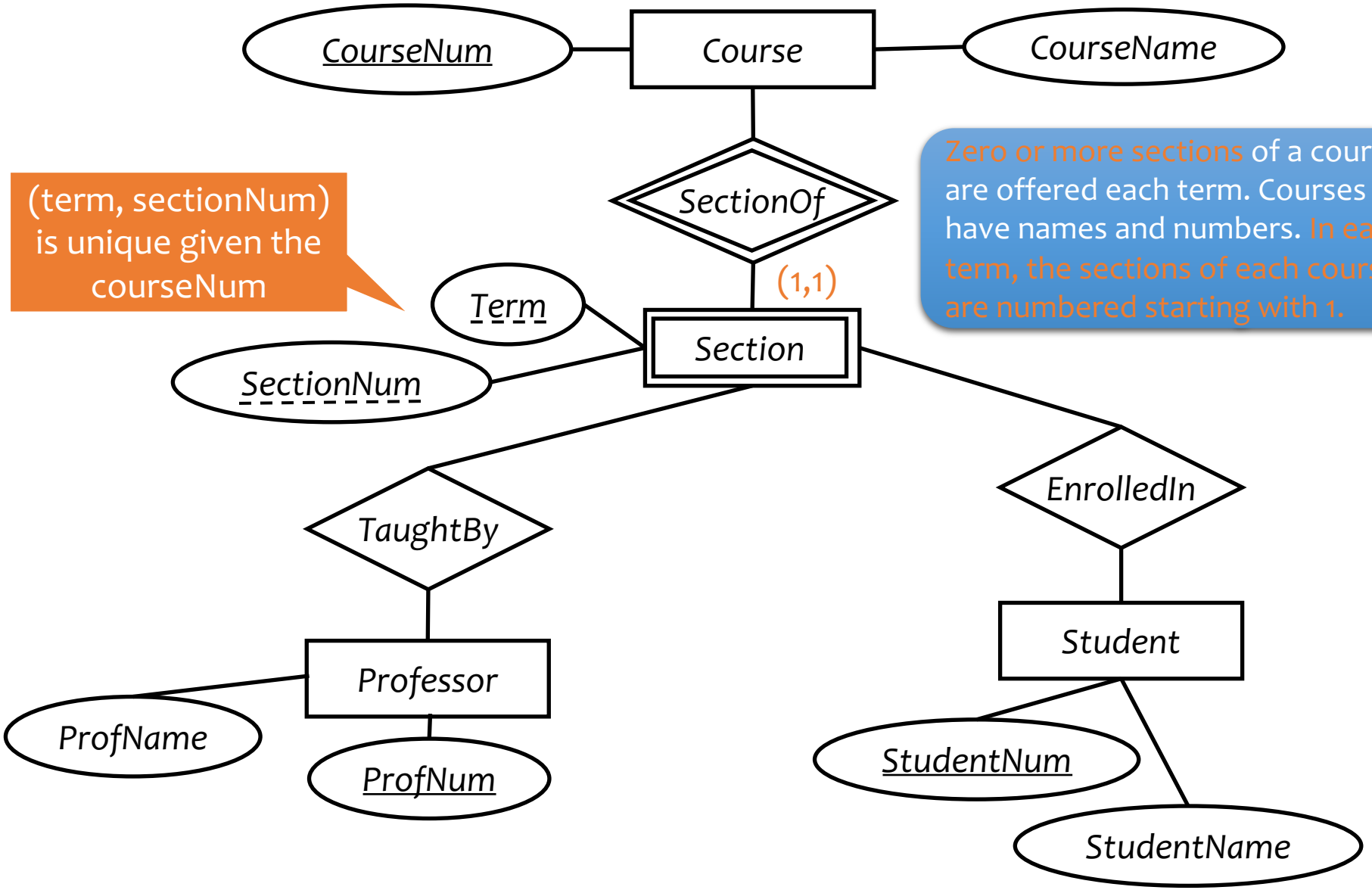
- Avoid redundancy
- Capture essential constraints, but don't introduce unnecessary restrictions
- Use your common sense
  - Warning: mechanical translation procedures given in this lecture are no substitute for your own judgment

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

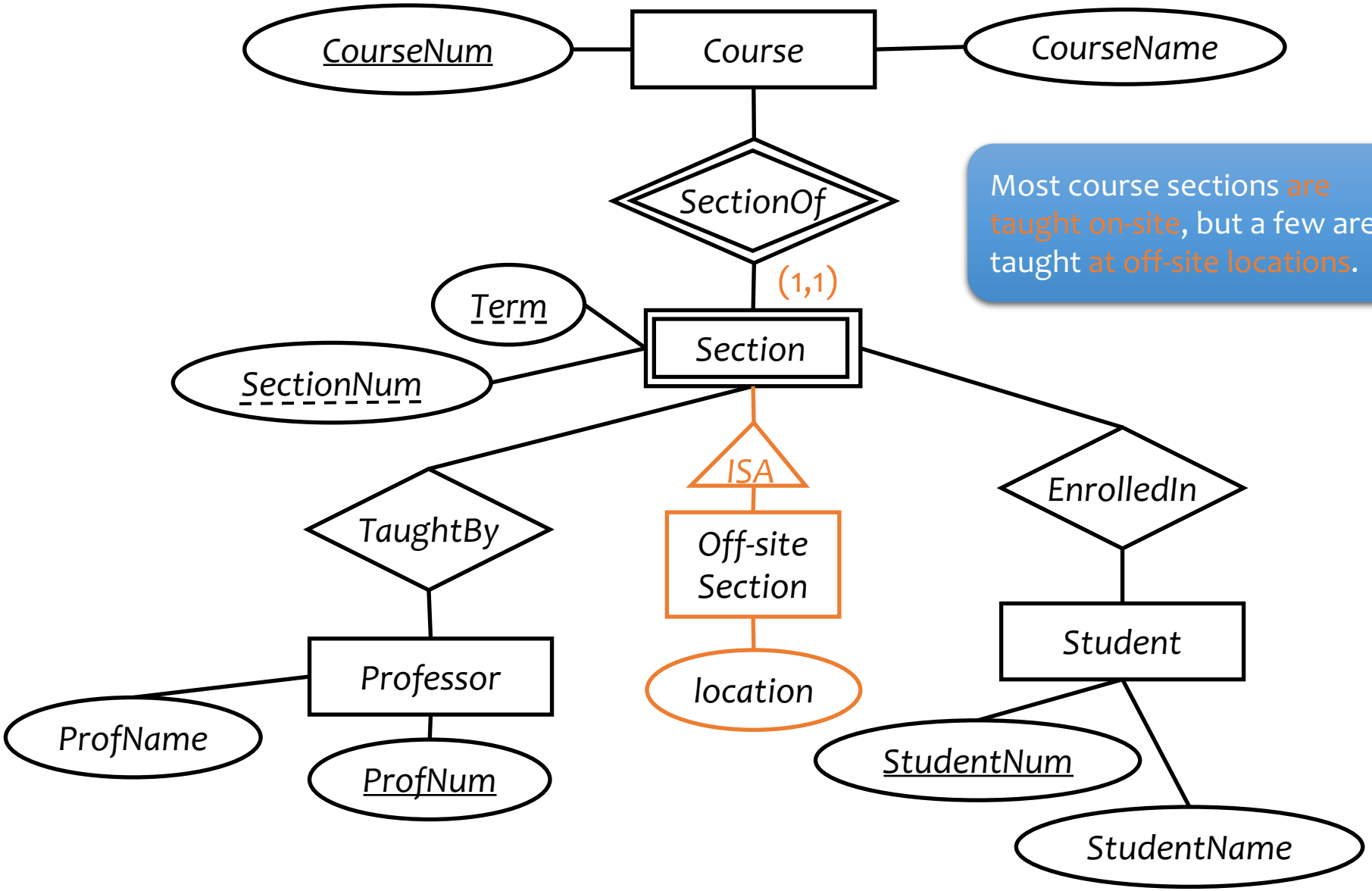
Case study 3

- A Registrar's Database:
  - 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. Professors have professor numbers and names. 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.

Case study 3


CourseNum
Course
CourseName
SectionOf
(1,1)
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.
(term, sectionNum) is unique given the courseNum
Term
Section
SectionNum
TaughtBy
EnrolledIn
Professor
Student
ProfName
ProfNum
StudentNum
StudentName

Case study 3

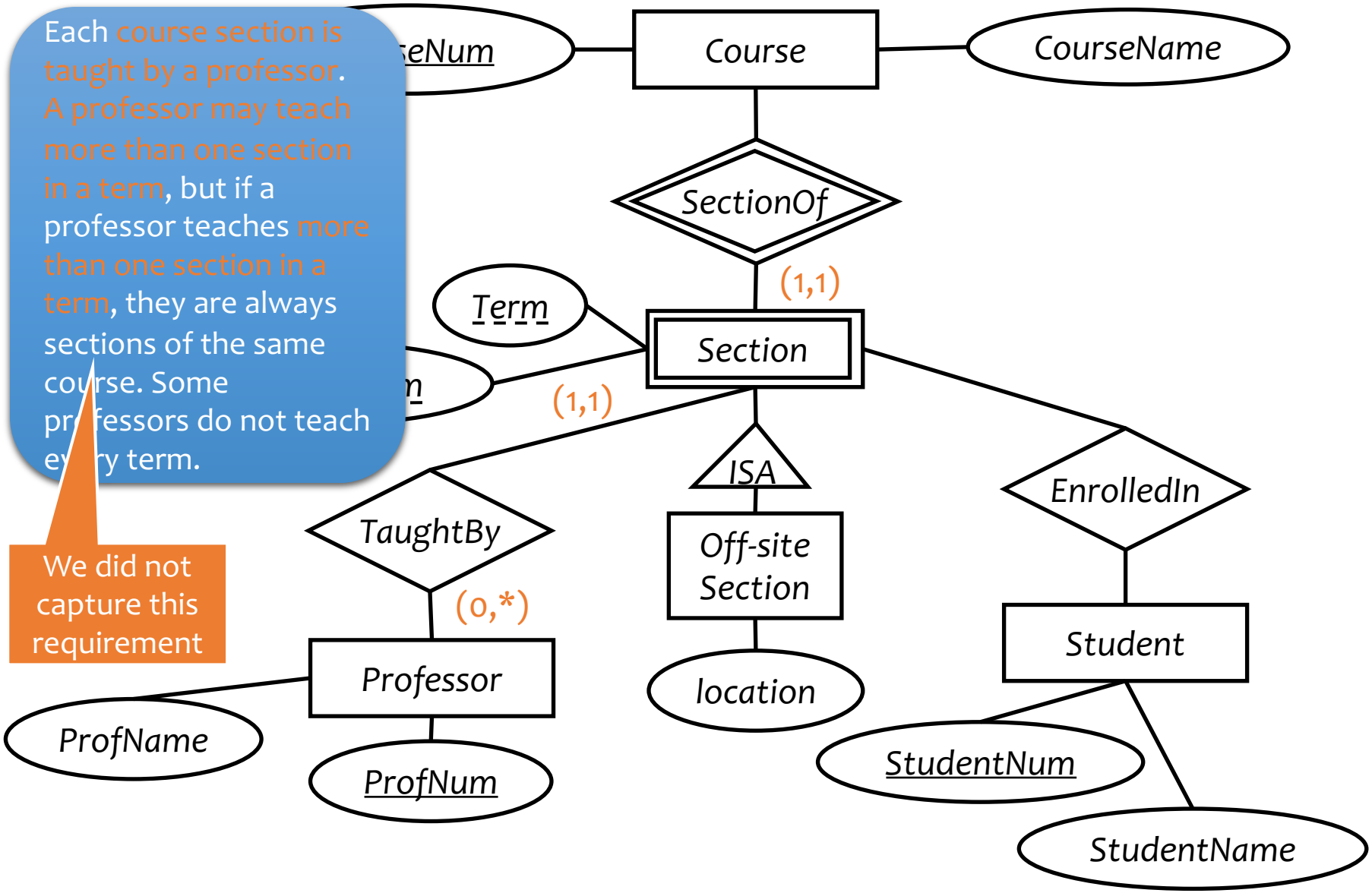


Most course sections are taught on-site, but a few are taught at off-site locations.

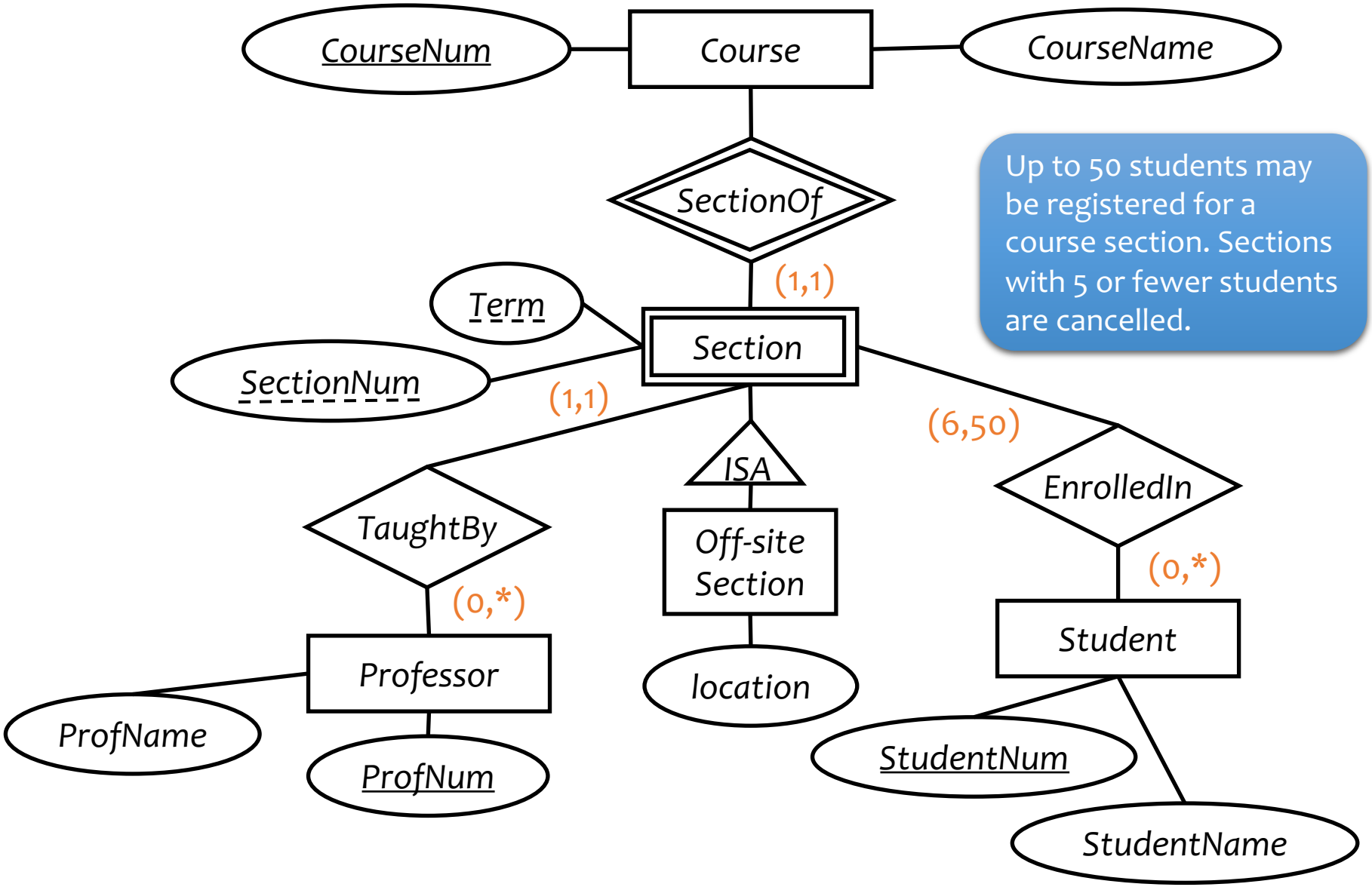
Case study 3

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.

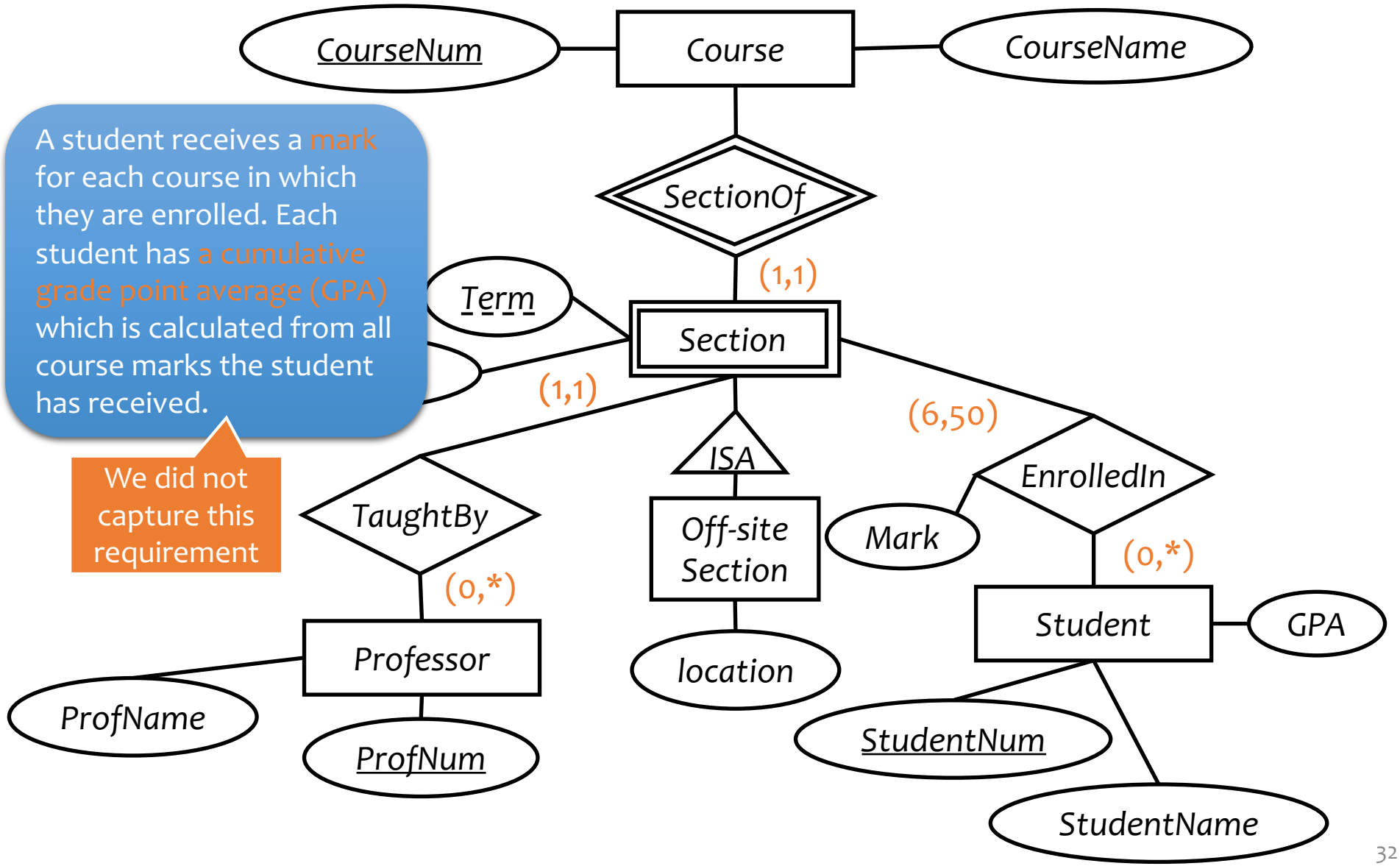
We did not capture this requirement



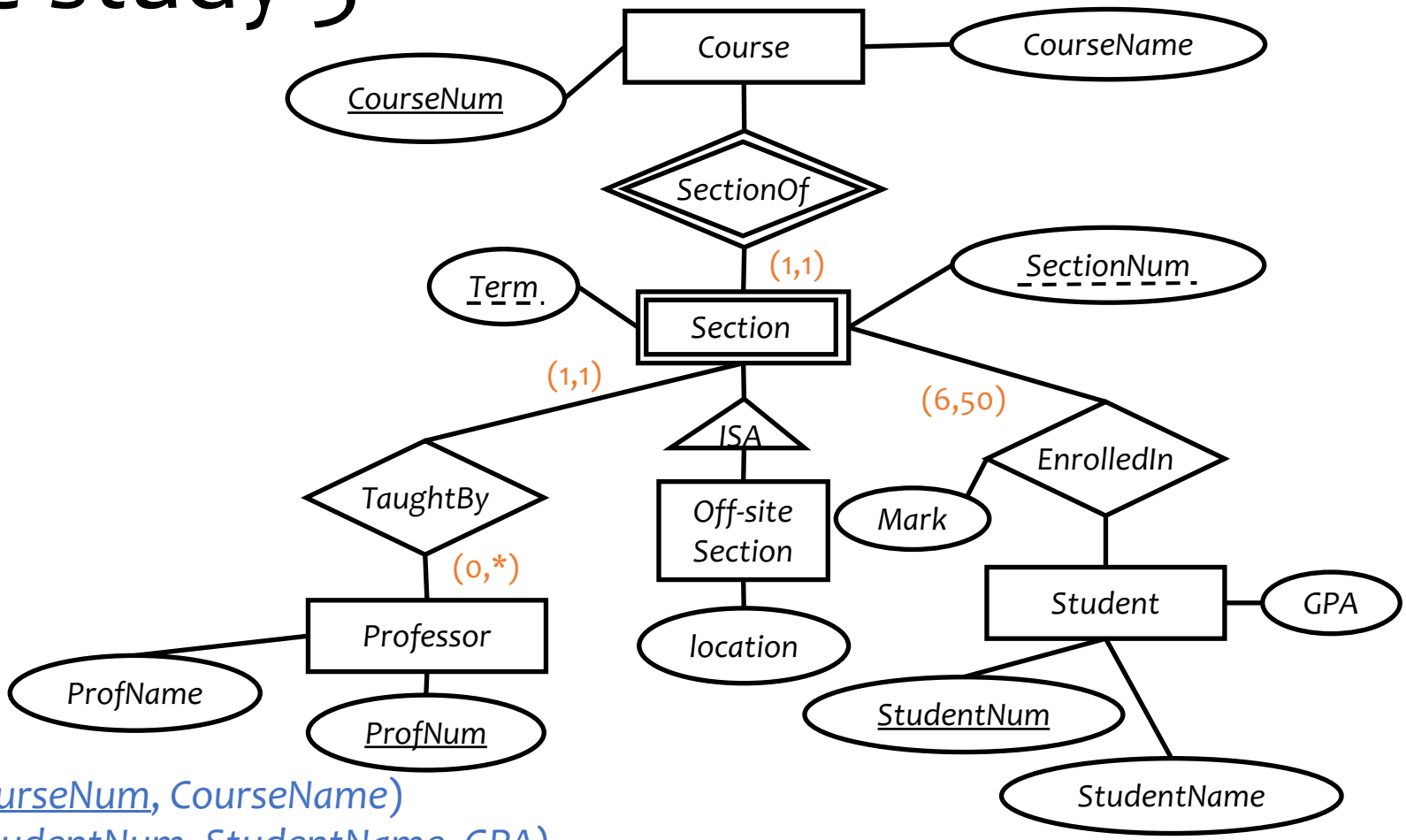
Case study 3


CourseNum
Course
CourseName
SectionOf
(1,1)
Term
Section
SectionNum
(1,1)
(6,50)
ISA
TaughtBy
Off-site Section
EnrolledIn
(0,*)
(0,*)
Professor
Student
location
ProfName
ProfNum
StudentNum
StudentName
Up to 50 students may be registered for a course section. Sections with 5 or fewer students are cancelled.

Case study 3


CourseNum
Course
CourseName
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.
We did not capture this requirement
SectionOf
(1,1)
Term
Section
(1,1)
(6,50)
ISA
TaughtBy
EnrolledIn
Off-site Section
Mark
(0,*)
(0,*)
Professor
Student
GPA
location
ProfName
StudentNum
ProfNum
StudentName

Case study 3



Course (CourseNum, CourseName)
Student(StudentNum, StudentName, GPA)
Professor(ProfNum, ProfName)
Section(CourseNum, Term, SectionNum, ProfNum)
EnrolledIn(CourseNum, Term, SectionNum, StudentNum, Mark)
OffSiteSection(CourseNum, Term, SectionNum, location)

Case study 3

Course (CourseNum, CourseName)
Student(StudentNum, StudentName, GPA)
Professor(ProfNum, ProfName)
Section(CourseNum, Term, SectionNum, ProfNum)
EnrolledIn(CourseNum, Term, SectionNum, StudentNum, Mark)
OffSiteSection(CourseNum, Term, SectionNum, ProfNum)

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

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

Database Design

- Entity-Relationship (E/R) model
- Translating E/R to relational schema
- Next lecture: principles of relational schema