TDA357/DIT621 – Databases

Lecture 4 & 5 – Design using ER

Jonas Duregård

Week 2: Design using ER

- We now know how to implement a design in SQL and query it for data
- We have used constraints to prevent certain operations
  - Primary keys and unique constraints to prevent duplicate values
  - Reference constraints (FOREIGN KEY) to check that values exist in other tables
- This week we will take a step back and learn how to design a database from a domain description

Recall: Relational schemas

- A database schema is a collection of relation schemas like this one:

```
Grades(student, course, grade)
    student -> Students.idNumber
    course -> Courses.code
    grade ∈ {0,3,4,5}
```

- Relation schemas can be translated into SQL tables

```
CREATE TABLE Grades (
    student TEXT,
    course CHAR(6),
    grade INT NOT NULL,
    PRIMARY KEY (student, course),
    FOREIGN KEY (student) REFERENCES Students(idNumber),
    FOREIGN KEY (course) REFERENCES Courses(code),
    CHECK (grade IN (0,3,4,5))
);
```

Spotting bad schemas

- Here is a database for keeping track of room bookings. Why is it bad?

Table: Bookings

courseCode	name	day	timeslot	room	seats
TDA357	Databases	Tuesday	0	GD	236
TDA357	Databases	Tuesday	1	GD	236
ERE033	Reglerteknik	Tuesday	0	HB4	224
ERE033	Reglerteknik	Friday	0	GD	236

- Redundancy
  - Mentions 3 times that GD has 236 seats and twice that TDA357 is Databases
- Risk of:
  - Update anomaly: I change one of the seat values but not the other two
  - Deletion anomaly: Remove all bookings in GD and loose knowledge of its size

Decomposing the table

Table: Bookings

courseCode	name	day	timeslot	room	seats
TDA357	Databases	Tuesday	0	GD	236
TDA357	Databases	Tuesday	1	GD	236
ERE033	Reglerteknik	Tuesday	0	HB4	224
ERE033	Reglerteknik	Friday	0	GD	236

Table: Courses

courseCode	name
TDA357	Databases
ERE033	Reglerteknik

Table: Bookings

courseCode	day	timeslot	room
TDA357	Tuesday	0	GD
TDA357	Tuesday	1	GD
ERE033	Tuesday	0	HB4
ERE033	Friday	0	GD

Table: Rooms

room	seats
GD	236
HB4	224

Fairly simple process for this tiny example

Not so much for large databases with hundreds of columns

How do we design database schemas?

Domain descriptions

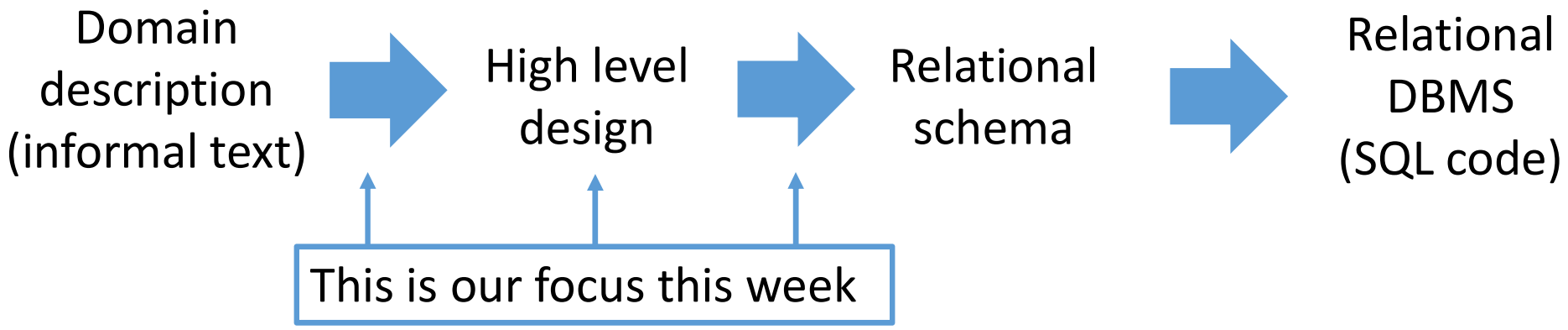
- The domain of a database is an informal description of everything a database should contain
- Typically given by a client (an expert in the domain, but not in databases)
- Natural language, no reliable automatic way to translate to SQL code
- Sometimes ambiguous on important details
- Sometimes hard to understand if you are not a domain expert
- In real life, there is continuous communication between the client who wants the database and the developers of the database
  - In this course, you get domain descriptions that are more precise (and much shorter) than typical domain descriptions

Modelling domains

- The database we create is based on a model of the domain
- A model contains formal, well defined definitons
- Look at a sentence from the domain like "Rooms can be booked for courses at particular timeslots.", when designing the database we have to decide how to model that sentence in the database

The problem of design

- Going from an informal domain description directly to SQL or even to a schema is difficult and error prone
- It's also hard to use the tables directly to describe the database to other developers
- We want an intermediate high level design that is easy for humans to understand, and still easy to translate into tables



The Entity-Relationship model

- The ER model is a high level design model that expresses every aspect of the database as entities (entity sets to be more precise) and relationships (not the same as relations!)
  - A third category that is easy to forget (not in the name) are attributes
- Entities are things from the domain: courses, employees, car models, cars …
- Relationships are things that connect entities, like "a car is of a car model", or "an employee has another employee as its boss", "courses have students" …
- Attributes are simple properties of entities: names of courses, salaries of employees, …

Entities

- What is an entity?
  - Entities are objects with attributes (e.g. courses with name, credits..)
  - Rule of thumb: Entities can exist independently from other entities, for instance courses and teachers, or players and teams etc.
  - Car models are entites (can exist even if there are no cars of that model)
- Objects that are inherently dependent on other entities are not entities
  - Course names are not entities (what is a course name without a course?)
  - Course result can not exist without courses, so results are not entities (but also not attributes like course names!)
- This distinction is not clear-cut and there are some tricky cases
- As a guideline: most basic objects you want in your database tend to be entities

Entities, attributes and deletion anomalies

- One way to express the problem with the deletion anomaly we saw before:
  - We treated rooms as attributes of bookings, when they should be entities
  - Rooms are logically separate entities that can exist independently of bookings, same thing goes for courses

Table: Bookings

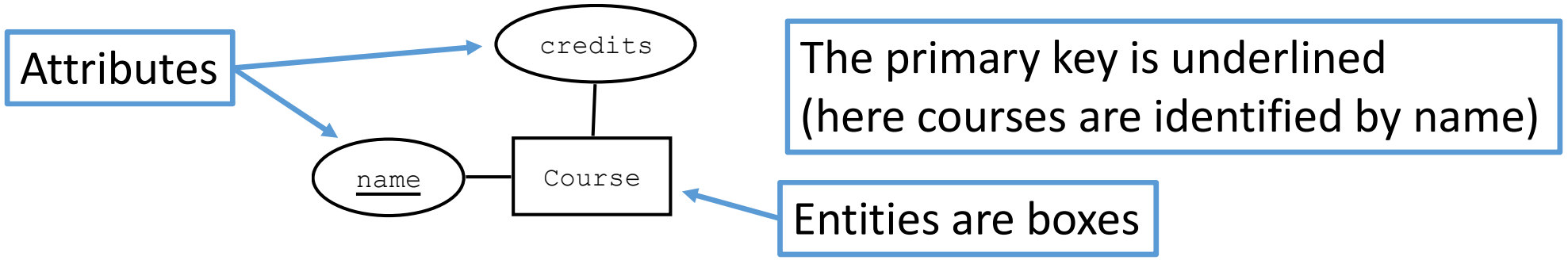
courseCode	name	day	timeslot	room	seats
TDA357	Databases	Tuesday	0	GD	236
TDA357	Databases	Tuesday	1	GD	236
ERE033	Reglerteknik	Tuesday	0	HB4	224
ERE033	Reglerteknik	Friday	0	GD	236

Attributes

- Attributes are "simple" properties of entities
  - Can be numbers, text strings, etc.
- Something that every instance of that entity has, like a each course having a name or each car having a license number
- What are NOT attributes
  - Collections of values (like a list of students)
  - Optional values (if cars *sometimes* have an owner, owner is not an attribute of the car entity)
  - Things that refer to attributes of other entities (see relationships)

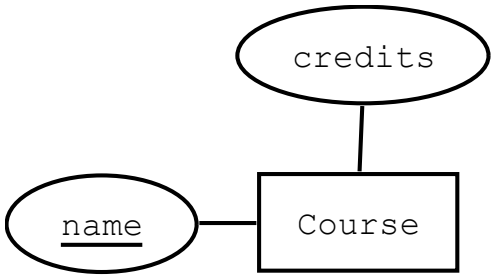
Drawing ER-diagrams

- ER-diagrams are visual representations of ER-models
- ER-diagrams are popular because they are:
  - Graphical, human-friendly
  - Mechanically translatable into SQL (thus machine-friendly)
  - Quite expressive (a lot of design concepts can be expressed using ER)
- Our first ER-diagram, an entity (Course) with two attributes (name, credits):



Translating to a relational schema

- Entities become relations (become tables in SQL)
  - Attributes become attributes (become columns in SQL)
  - Primary keys are translated in the obvious way



Courses(name, credits)

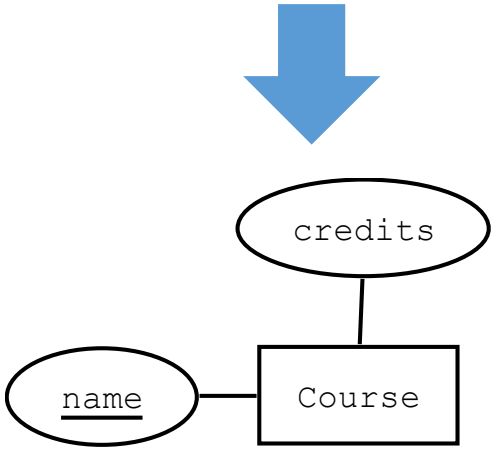
```
CREATE TABLE Courses(
   name TEXT,
   credits INT NOT NULL,
   PRIMARY KEY (name)
);
```

Mechanical translation into schema

Semi-mechanical translation to SQL

Domain description (informal text) → High level design (ER) → Relational schema → Relational DBMS (SQL code)

"Each course has its own name, and a number of credits."

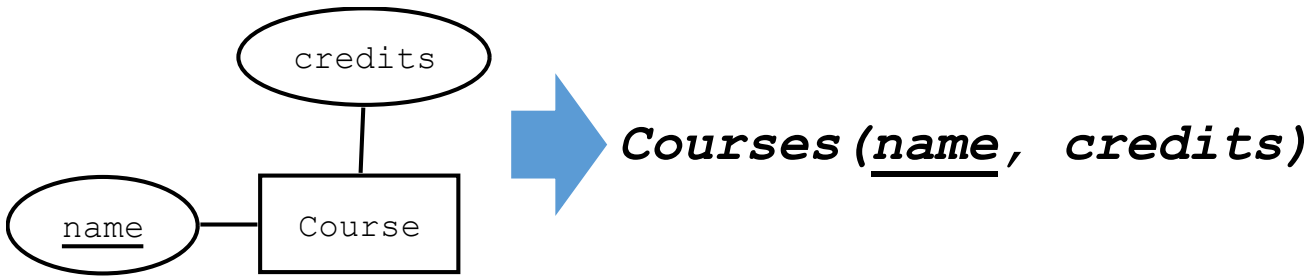


Courses(<u>name</u>, credits)

```
CREATE TABLE Courses(
  name TEXT,
  credits INT NOT NULL,
  PRIMARY KEY (name)
);
```

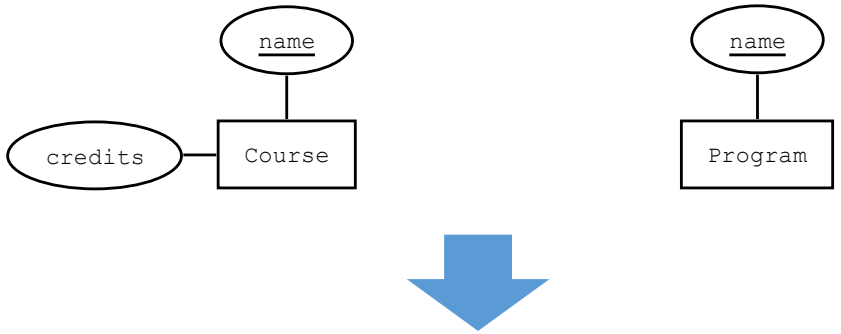
Naming in ER vs relational schemas

- Entities are by convention named in singular (Course)
- Relations are named in plural (Courses)
- We also follow the convention we are using for relations/tables, meaning capitalized entity names and common attribute names



Multiple entities

- Separate entities are (for now) translated independently



Courses(<u>name</u>, credits)
Programs(<u>name</u>)

The rest of this and tomorrows lecture

- I will present lots of features of ER-diagrams
- For each feature I will state something from a domain description (an informal description of the database) and …
  - … show how to model it in ER-diagrams
  - … show how it is translated into a relational schema
  - … say something about how to identify the feature in domains
- If there is time left, I will spend it demonstrating some more examples common mistakes

A simple relationship

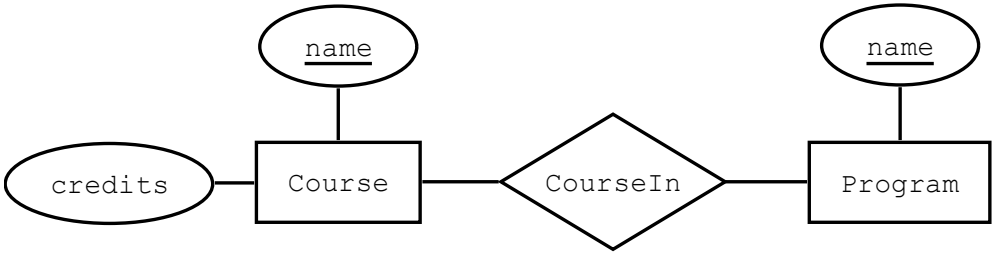
```
Courses(name, credits)
Programs(name)
```

- I want to extend my database so every program has a set of courses
- First question: How would I modify my schema?
- Do I add an attribute "listOfCourses" to Programs?
  - No! Lists are not attributes. And if they where, how do we state they need to be names of courses?
- Do I add an attribute "listOfPrograms" to Courses?
  - No! Same reasons!
- Solution: Add a new relation ☺
  - Essentially a list of (course,program)-pairs
- In ER, we express CourseIn as a *relationship*

```
Courses(name, credits)
Programs(name)
CourseIn(course,program)
   course -> Course.name
   program -> Program.name
```

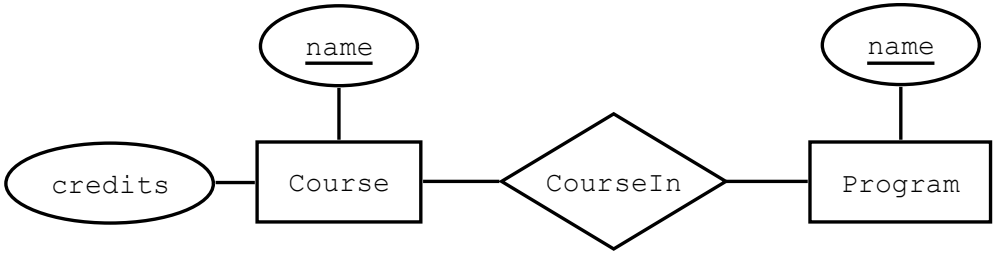
Relationships in ER-diagrams

- Relationships are drawn using a diamond-shapes
- Names typically describe the relationship (OwnedBy, RegisteredTo, …), or sometimes just adding the related entities names (like ProgramCourse)
  - Should relay the intent of the relationship, and work as a table name



Constraints from relationships

- The references ensure only existing courses and programs can be in CourseIn
- The primary key of CourseIn is the PKs of both referenced relations:
  - A course can be in several programs (same course, different program)
  - A program can have multiple courses (same program, different course)
  - A program can not have the same course multiple times (both equal)

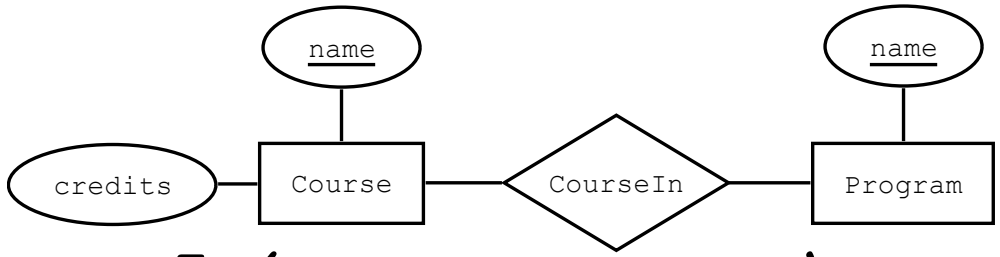


Courses(<u>name</u>, credits)
Programs(<u>name</u>)
CourseIn(<u>course</u>,<u>program</u>)
course -> Course.name
program -> Programs.name

Note that the credits attribute is not here, only the key

Naming and relationships

- During translation you need to decide a few names in the schema
- You can use the same name for attributes everywhere, but this is sometimes confusing (like having an volumn called name in courses, that is actually the name of a teacher) and sometimes does not work
- I like using the (lowercase) names of the connected entities:



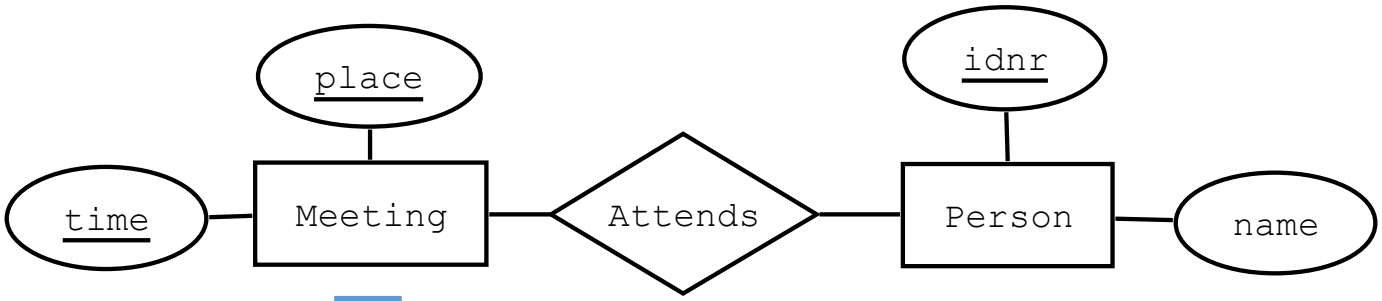
CourseIn(course,program)

course -> Course.name

program -> Programs.name

Compound keys and relationships

- Remember: Always include the whole key of both the related relations!



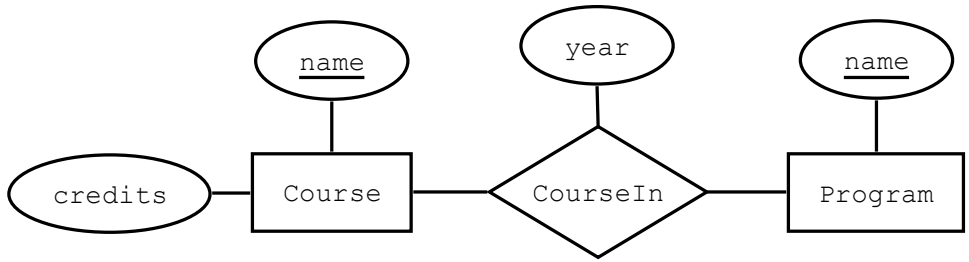
Meetings(time, place)
Persons(idnr, name)
Attends(person,time,place)
person -> Persons.idnr
(time,place) -> Meetings.(time,place)

Compound keys (2 or more attributes)

Compound reference

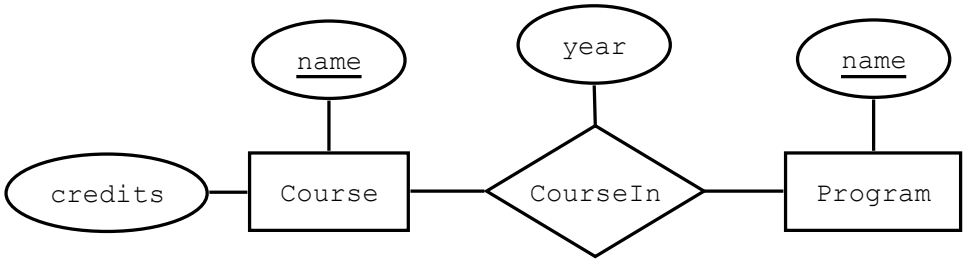
Attributes on relationships

- What if we wanted to add an attribute stating which year in a program a certain course is?
  - The year is not an attribute of course (because different programs may have the course in different years)
  - The year is not an attribute of program (because different courses may be in different years of the program)
  - The year is an attribute of the relationship between them!



Translating relationships with attributes

- Just add the extra attributes to the created relation
- Note that relationships can never have key attributes!
  - Always identified by the related entities



Courses(<u>name</u>, credits)
Programs(<u>name</u>)
CourseIn(<u>course</u>, <u>program</u>, year)
course -> Courses.name
program -> Programs.name

When I add a (course,program)-pair to CourseIn, I also have to specify a year

Identify attributes on relationships in domains

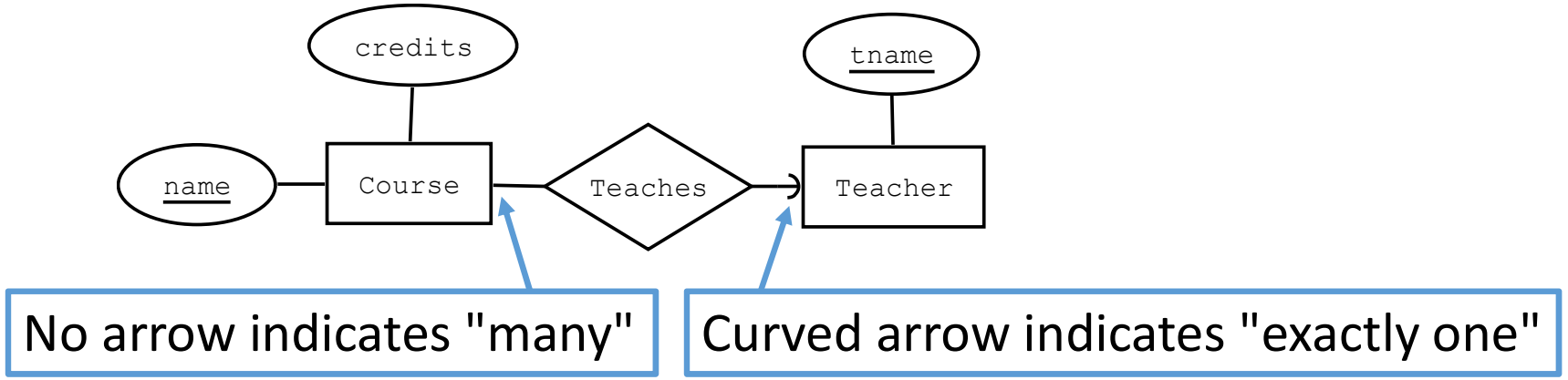
- Things like "X may have a z in/for/at a Y" where X and Y are entities and z is an attribute typically signify that z is an attribute of a relation
- Example: "Teachers can be assigned roles in courses"
  (role is an attribute of a relationship between teachers and courses, unless roles should be entity…)

Multiplicity

- The relationships we have seen so far are called many-to-many
  - E.g. A course can belong to many programs and programs have many courses
- What if I wanted to model something like "Each course has a single teacher" or "Some of the teachers have an office"?
- These examples describe relationships, but they are not many-to-many

Many-to-exactly-one

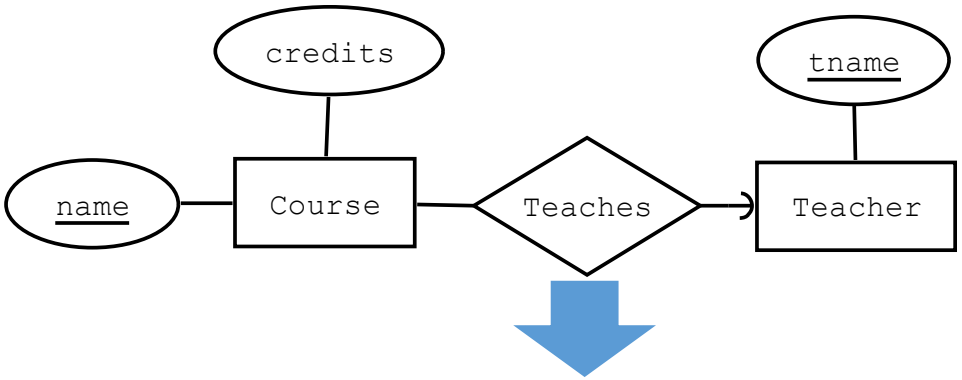
- The Teaches relationship models "each course has a single teacher"
- If the arrow was in the other direction it would model "each teacher has a single course"



- An exception to the guideline that entities can exist independently (since courses can not exist without teachers)

Translating many-to-exactly-one

- If you had a table of courses, where each row is a separate course. How would you express that each course has a teacher?
  - By adding a column for teacher of course!



Not every relationship becomes a relation/table!

Teachers(<u>tname</u>)

Courses(<u>name</u>, credits, teacher)

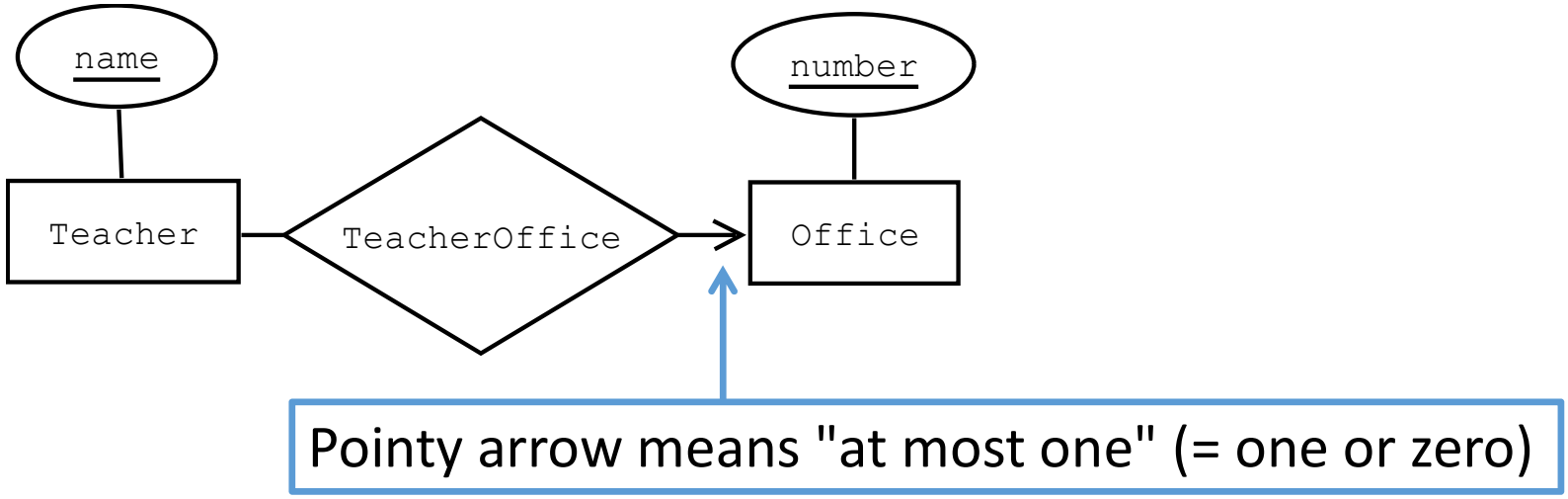
teacher -> Teachers.tname

Identify many-to-exactly-one in domains

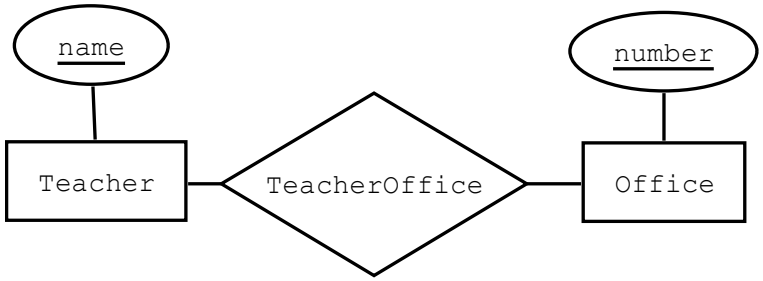
- Anything like "Every X has a Y" (where Y is another entity) are typically many-to-exactly-ones
- It's common that the language is ambiguous, e.g. "Xs have Ys" can mean that each X can have multiple ys, or that they each have one.
- If you want to add an attribute, but that attribute is more accurately modelled as an entity, you should use many-to-one relationships

Many-to-at-most-one

- This diagram expresses "Some teachers have an office", or "A teacher may have an office" or equivalent
- Also called many-to-one-or-zero

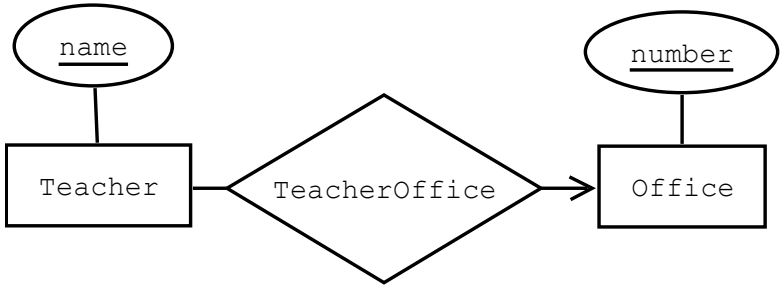


Translating many-to-at-most-one: ER-approach



Teachers (***name***)
Offices (***number***)
TeacherOffice (***teacher***, ***office***)
  teacher -> Teachers.name
  office -> Offices.number

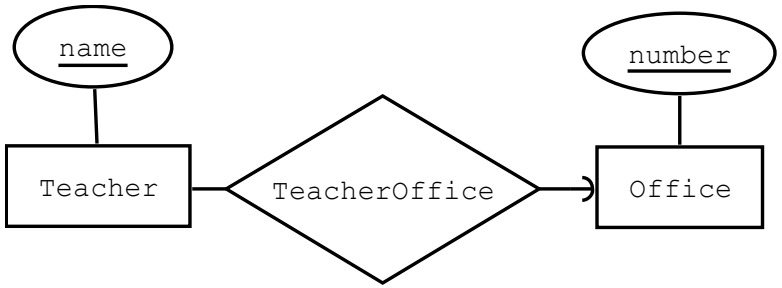
- How can we modify the translation of a many-to-many relationship to make it many-to-at-most-one?



Teachers (***name***)
Offices (***number***)
TeacherOffice (***teacher***, ***office***)
  teacher -> Teachers.name
  office -> Offices.number

Change the key!

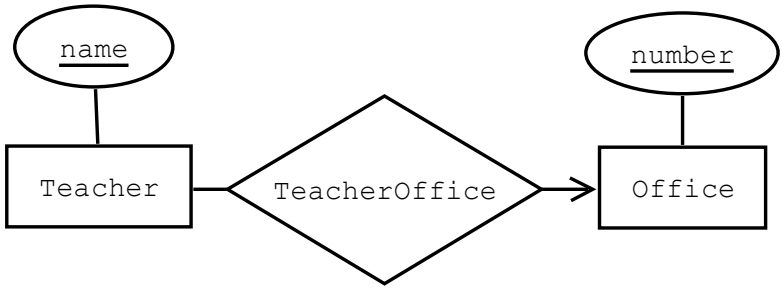
Translating many-to-at-most-one: Null-approach



Offices(<u>number</u>)
Teachers(<u>name</u>, office)
office -> Offices.number

- How can we modify a many-to-one to make it many-to-at-most-one?

Allow the foreign key to be null



Offices(<u>number</u>)
Teachers(<u>name</u>, office (or null))
office -> Offices.number

When can the Null approach be used?

The null approach can only be used under the following conditions:

- The "at-most-one"-side does not have a compound key
- The relationship does not have any attributes
- You are morally OK with having null values in your database

The ER-approach always works ☺

- Side-note: Is the Null-approach more efficient?
  - Depends on usage, but often it's less efficient
    (slightly simplified: SELECTs are slower and INSERTS are faster)

Identify many-to-at-most-one in domains

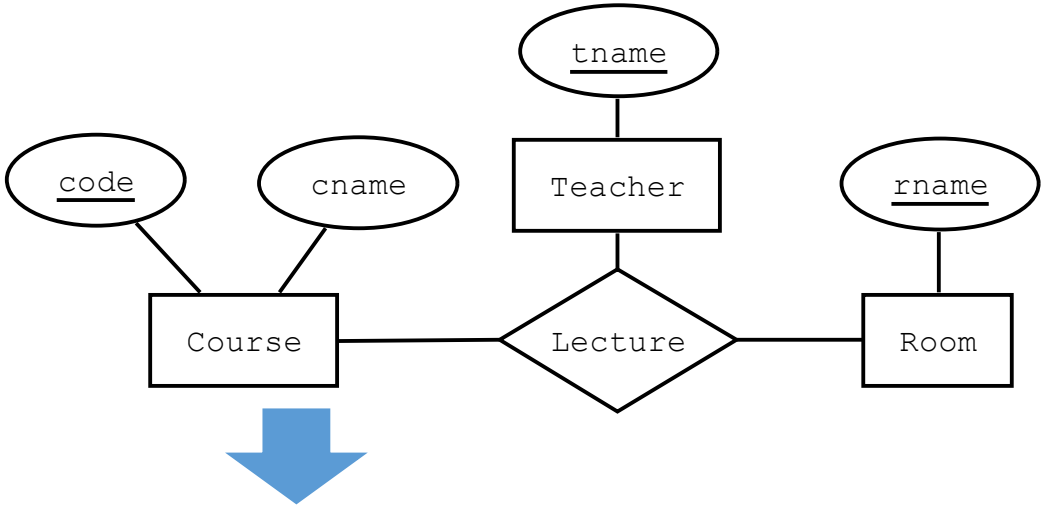
- Anything like "X may have a Y" or "Some Xs have a Y" (where Y is an entity)
- Often text is ambiguous about many/exactly one (e.g. "courses have teachers" could sometimes mean that many courses do, but not all)

A side-note on naming attributes

- In several of these examples I have had the same name for attributes of different entities (e.g. name for both courses and programs)
- This is allowed, but there may be advantages to having globally unique attribute names in diagrams (less confusion, fewer qualifed names later on in SQL etc.)

Multiway relationships

- A relationship can connect to more than two entities, but this is fairly rare



Note that I have chosen globally unique names here

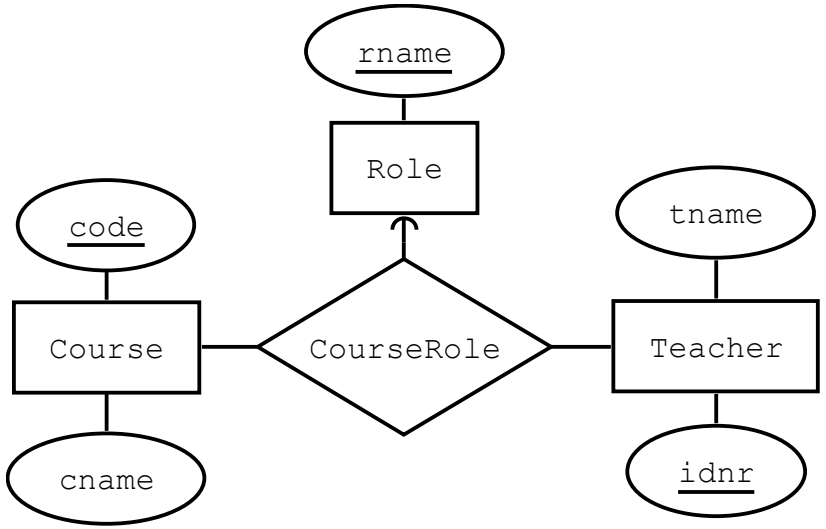
Courses(<u>code</u>, cname)
Teachers(<u>tname</u>)
Rooms(<u>rname</u>)
Lectures(<u>course</u>,<u>teacher</u>,<u>room</u>)
course -> Courses.code
teacher -> Teachers.tname
room -> Rooms.rname

This may not be exactly what we wanted (each lecture can have multiple teachers and rooms?)

A better example of a multiway relationship

- "Teachers can be assigned any of a list of roles (e.g. examiner, course responsible, assistant) for any course."
- Here, roles should be an entity (the concept of examiner exists even if there are currently no courses)
- Ambiguity: Can a teacher have roles on multiple courses?
  - Reasonable assumption: yes (from our knowledge of the domain)
- Ambiguity: Can a teacher have multiple roles in the same course? (does "any of" mean "any one of" or "any subset of"?)
  - Let's assume a teacher can only have one role (and there is a special role for being both examiner and course responsible for instance)

Multiway relationship



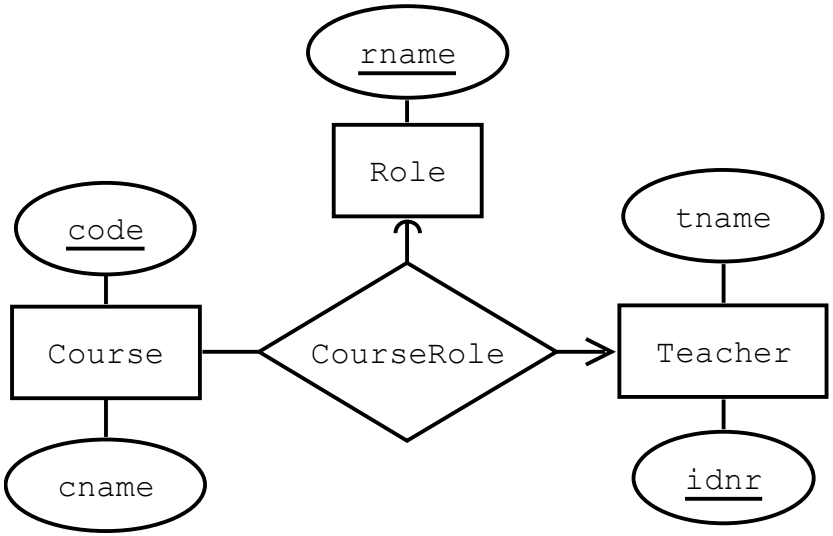
Key ensures we can associate any number of teachers with any number of courses, and for each association we have to select a valid role

```
Courses(code, cname)
Teachers(idnr,tname)
Roles(rname)
CourseRole(course,teacher,role)
   course -> Courses.code
   teacher -> Teachers.idnr
   role -> Roles.rname
```

Very similar to a role attribute on the relationship, but with a reference

Weird cases: What are we saying here?

- Probably "Some courses have a teacher with a role (from a list of roles)?"
- Switching the arrows expresses the same thing?



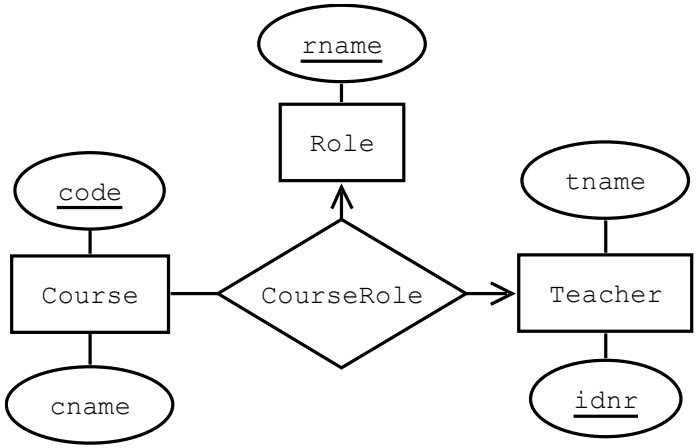
Courses(<u>code</u>, cname)
Teachers(<u>idnr</u>,tname)
Roles(<u>rname</u>)
CourseRole(<u>course</u>,teacher,role)
course -> Courses.code
teacher -> Teachers.idnr
role -> Roles.rname

Key ensures: At most one (role,teacher)-pair per course

Then what does this mean?

- Same as the one on the last slide?
- If we enterpret the course-teacher connection as the main one it may mean "courses may have a teacher and if they do they may have a role",

  … but then it would look exactly the same as "courses may have a role and if they do that role may have a teacher", which is different…



Confusion conclusion

- Multiway relationships that are many-to-many-to-many and many-to-many-to-one seem to make sense, other combinations are more difficult to interpret
- There are even weirder examples like all connections having pointed arrow
- Some that are obviously the same as having two separate relationships, like many-to-exactly-one-to-exactly-one

Identifying multiway relationships in domains

- Something like "Xs have Ys in Zs" where X, Y and Z are all entities
- Ambiguity: "Xs have Ys and Zs" usually mean two separate relationships (X-Y and X-Z) but sometimes a multiway relationship
- "Teachers have roles in courses" (assuming role is an entity not an attribute), "A person must have a contract for each project they are involved in" (assuming persons, contracts and projects are entities)
- Identifying multiplicity of the various connects is often difficult

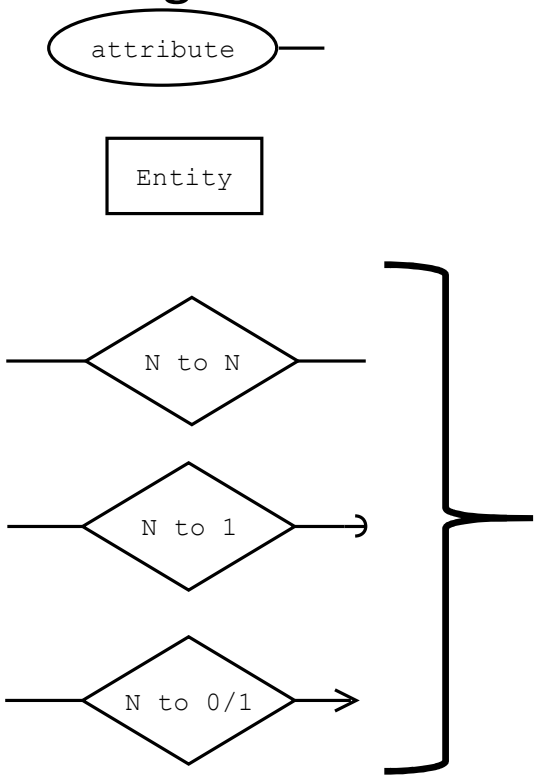
Start of lecture 5 – The story so far

Domain description → ER-diagram → schema → SQL code

Modelling

Translation

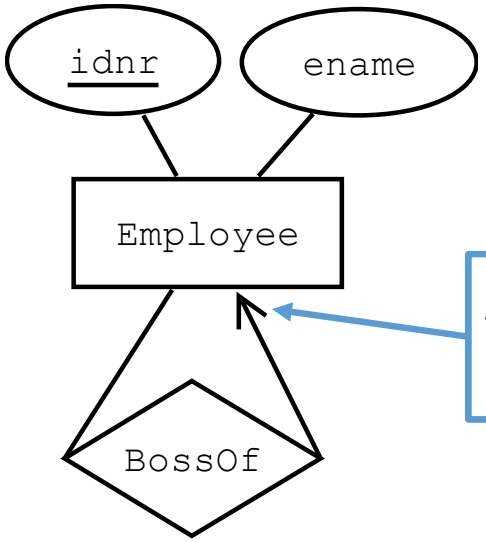
ER-diagrams:



Relationships
(many-to-many,
many-to-one,
many-to-at-most-one)

New problem!

- Model: "Most employees have another employee as boss."
- Being boss of someone is a relationship
- So this is a relationship between employees and employees

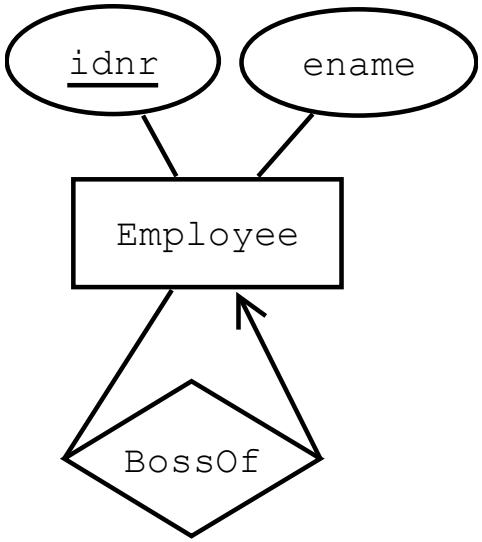


Arrow indicates at most one boss per employee

Translating self relationships

… is done exactly like other relationships!

- Here using the null-approach (add the key from the at-most-one side as an extra attribute on the many side, and make it nullable):



Employees(idnr,ename, boss (or null))
boss -> Employees.idnr

Self reference!

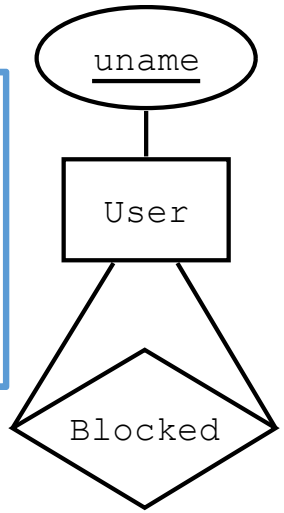
```
CREATE Table Employees(
 idnr INT PRIMARY KEY,
 ename TEXT NOT NULL,
 boss INT REFERENCES Employees
);
```

Nullable

A naming problem

- Suppose we want to model "Users can block other users"
- How do we translate this?

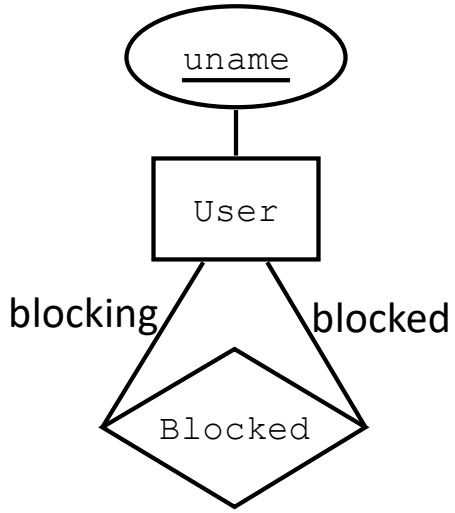
No arrows, many users can block many users



Problem: Who is blocking who here?

```
Users(uname)
Blocked(uname1,uname2)
  uname1 -> Users.uname
  uname2 -> Users.uname
```

Solution: Annotate connections with names



Remember: The keys of both sides become the key for many-to-many

```
Users(uname)
Blocked(blocking,blocked)
  blocking -> Users.uname
  blocked -> Users.uname
```

Limitations of self-relationships

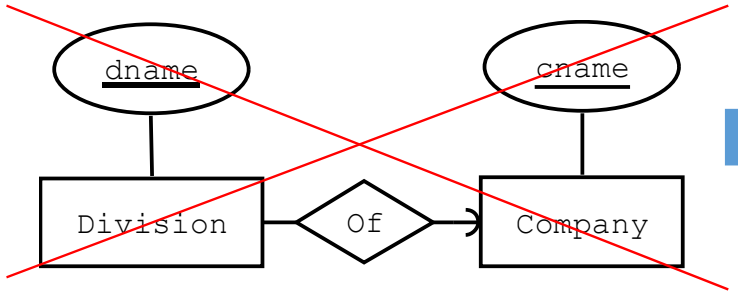
- Some things that can not be expressed in ER-diagrams:
  - Can a value be related to itself? (I'm my own boss? I block myself?)
  - Can a there be cycles (I'm the boss of my bosses boss? I'm blocked by a person I block?)
  - Is the relationship symmetric, e.g. for a Sibling-relationship:
    If *a* is a sibling of *b*, then *b* must also be a sibling of *a*
- These can be expressed in side-notes/comments, but may be difficult to implement in SQL

Identifying self relationships in domain

- Anything on the form "X has … to [an]other X"
- If you want to model any kind of tree-structure (many-to-at-most-one) or graph structures (many-to-many) on values of an entity

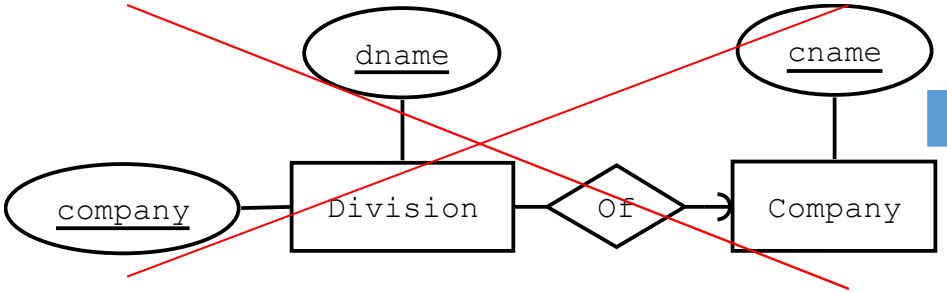
Weak entities

- Model this: "Each company has divisions, and each division has a name. Two divisions in the same company can not have the same name."
- Problem: What is the primary key of Divisions?



Too strong key

Companies(<u>cname</u>)
Divisions(<u>dname</u>,company)
company -> Companies.cname



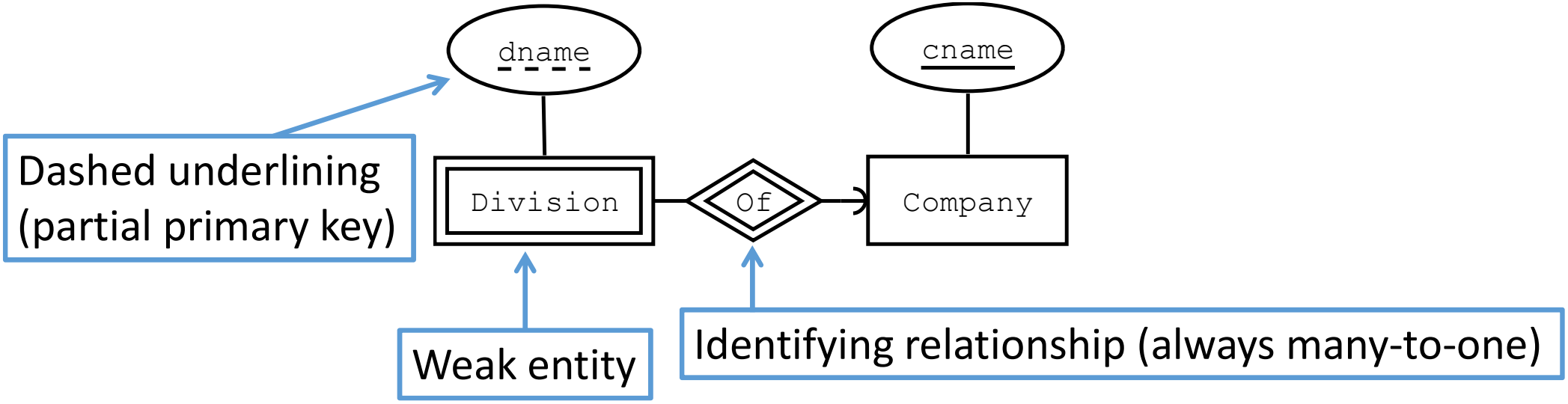
???

Companies(<u>cname</u>)
Divisions(<u>dname</u>,<u>company</u>, company2)
company2 -> Companies.cname

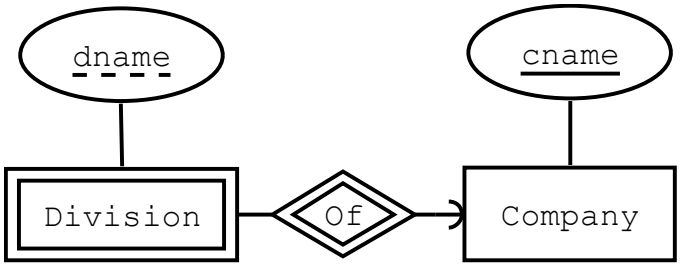
Missing reference for company

Weak entities

- To deal with this situation, we add a feature called Weak entities
- A weak entity can not be identified only by its own attributes
  - It requires support from at least one other entity
- The diagram below expresses that a division is identified by its name along with the identity of the company it belongs to:



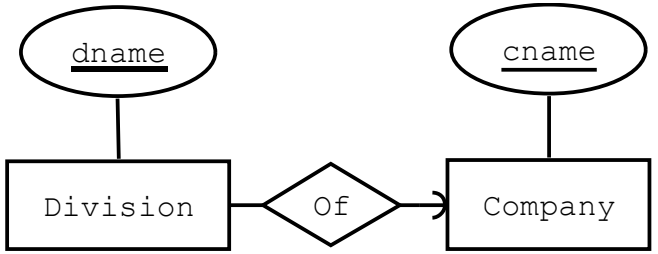
Translating Weak entities



Companies(<u>cname</u>)
Divisions(<u>dname</u>,<u>company</u>)
company -> Companies.cname

The key is the only difference!

- Compare to a regular many-to-one:



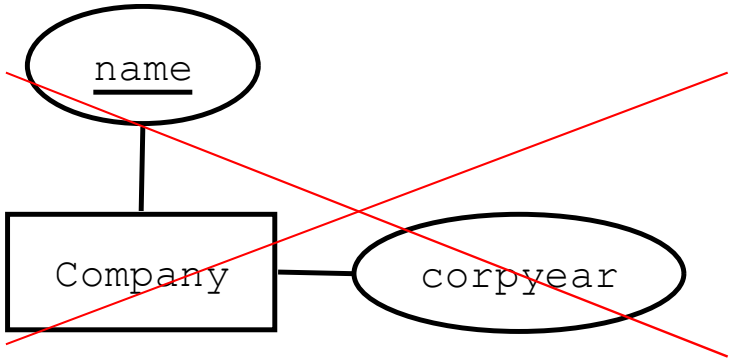
Companies(<u>cname</u>)
Divisions(<u>dname</u>,company)
company -> Companies.cname

Identify weak entities in domain descriptions

- Things like "Player numbers are unique within teams" or "players can have the same number assuming they are on different teams"
- If you notice that the attributes you have determined for an entity are not sufficient to identify members, perhaps it should be a weak entity

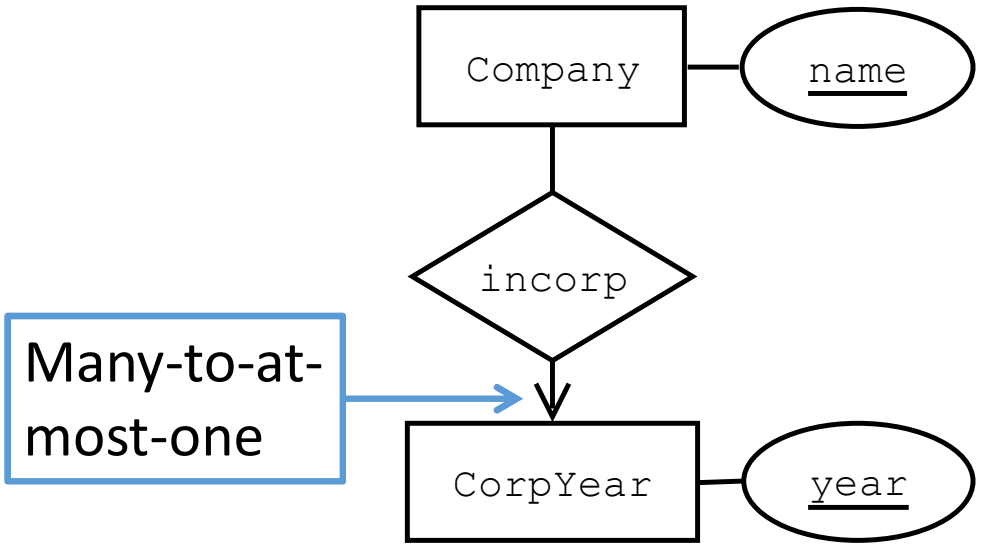
Inheritance and subentities

- Next up: What if I have something like "Some companies are corporations. Each corporations has a year of incorporation."
- Adding a year attribute to Company does not work, that implies all companies have a year of incorporation, and it's impossible to tell which companies are corporations.



Why is this bad?

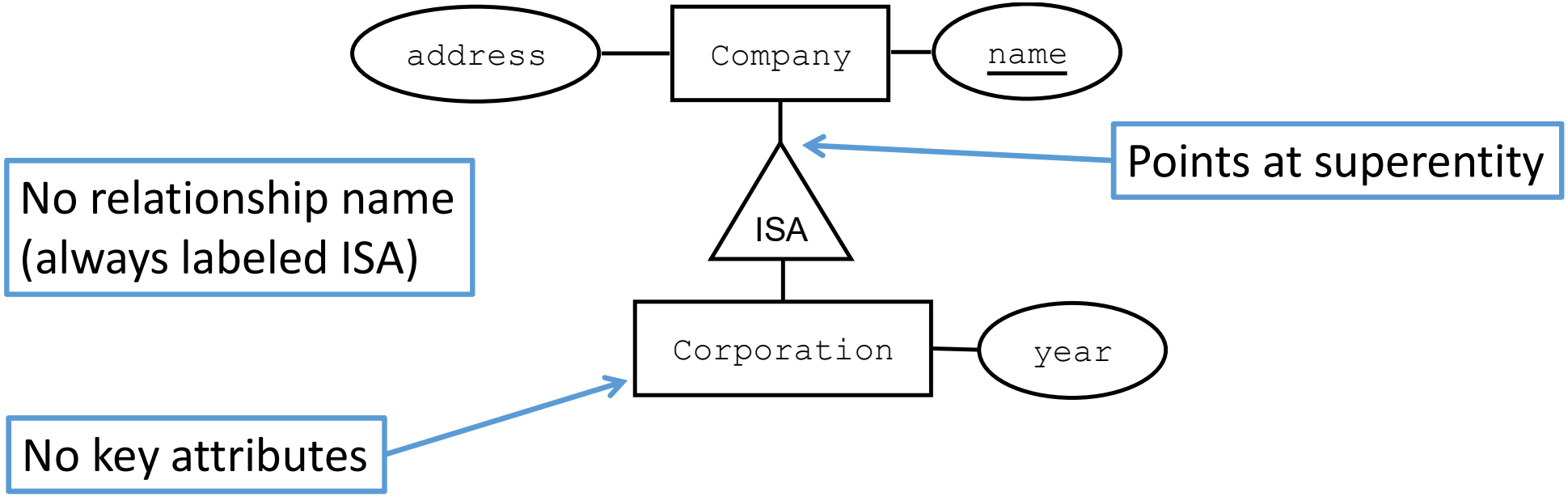
- In this design some companies have corporation years.
- Problem: Look at the CorpYear entity in isolation. What does it contain? All years? All years where at least one corporation has been formed (and no other years can be added)?



Incoporation years can not exist independently, and should not be modelled as entities!

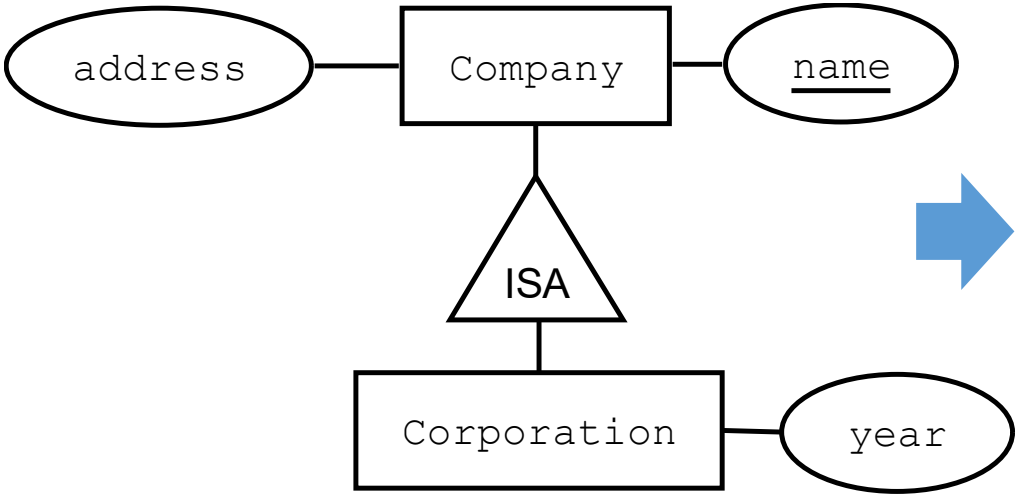
Inheritance in ER: ISA-relationships (ISA = "is a")

- This ER diagram expresses "Corporations are a special kind of companies, they have a year in addition to all properties of other companies."
- We call Corporation a subentity, and Company its superentity



Translating ISA-relationships: ER-approach

- Make a new relation that only has the primary key of the superentity and the extra attributes added by the subentity, and a reference to the superentity

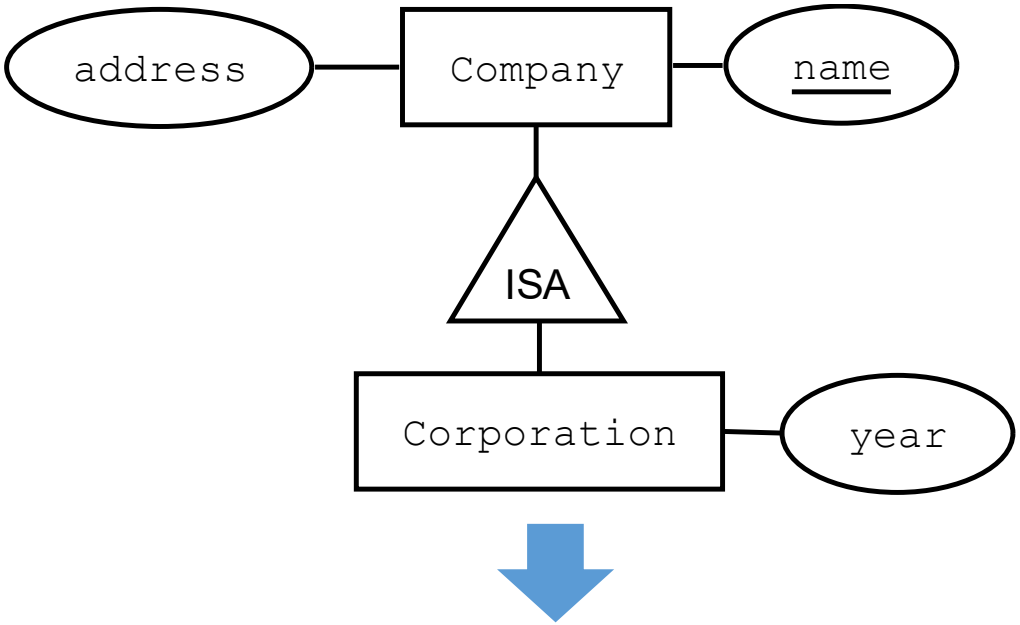


Companies(<u>name</u>, address)
Corporations(<u>name</u>, year)
name -> Companies.name

The reference means each corporation also has an address etc. - it really IS A company

Translating ISA-relationships: Null approach

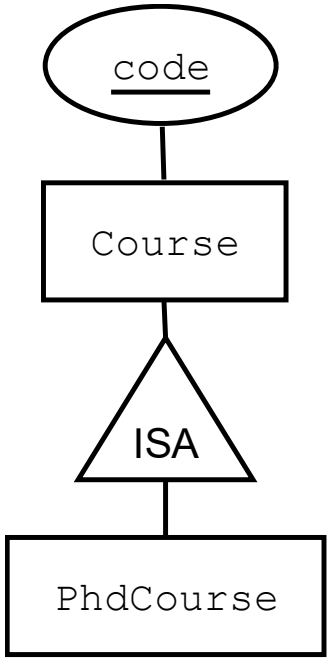
- Just add the extra attribute to the superentity, but nullable



Companies(<u>name</u>, address, year (or null))

null approach is (sometimes) bad m'kay

"Some courses are PhD Courses"



ER

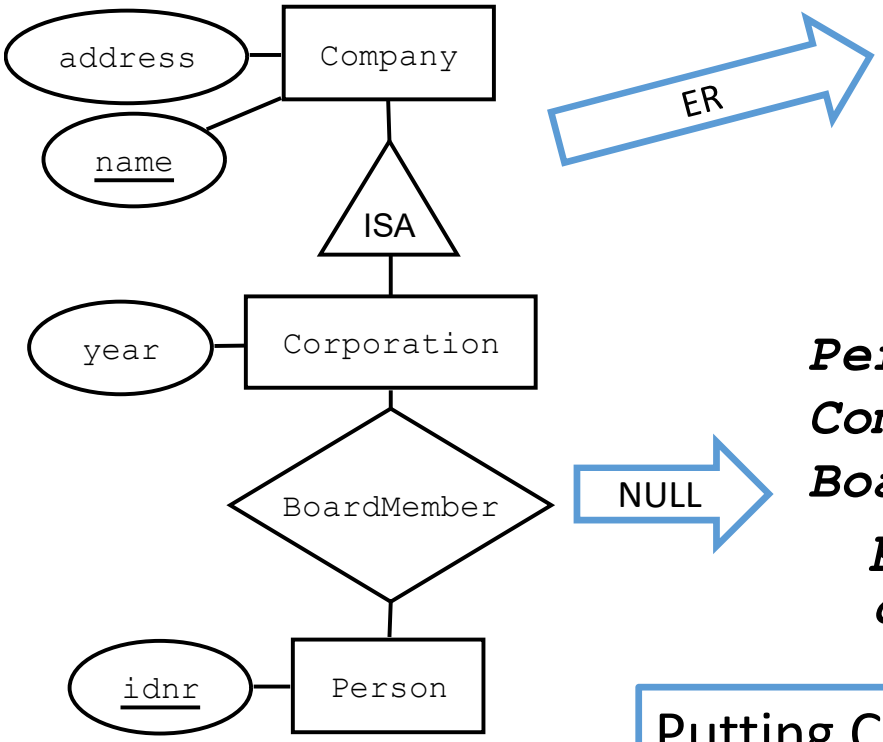
Course (***code***)
PhdCourse (***code***)
code -> Course.code

NULL

Course (***code***)

Sanity check: Is every design choice I made in my diagram reflected in my schema?

Another null-problem



ER

```
Persons(idnr)
Companies(name, address)
Corporations(name, year)
  name -> Companies.name
BoardMembers(person,corp)
  person -> Persons.idnr
  corp -> Corporations.name
```

Reference to Corporations ☺

NULL

```
Persons(idnr)
Companies(name, address, year (or null))
BoardMembers(person,corp)
  person -> Persons.idnr
  corp -> ???
```

Putting Companies.name here would be bad
(means all companies can have board members)

Problems with the NULL-approach

The NULL approach can only be used if:

- The subentity has a single attribute
- The subentity has no relationships to other entities
  - Includes not having its own subentitites

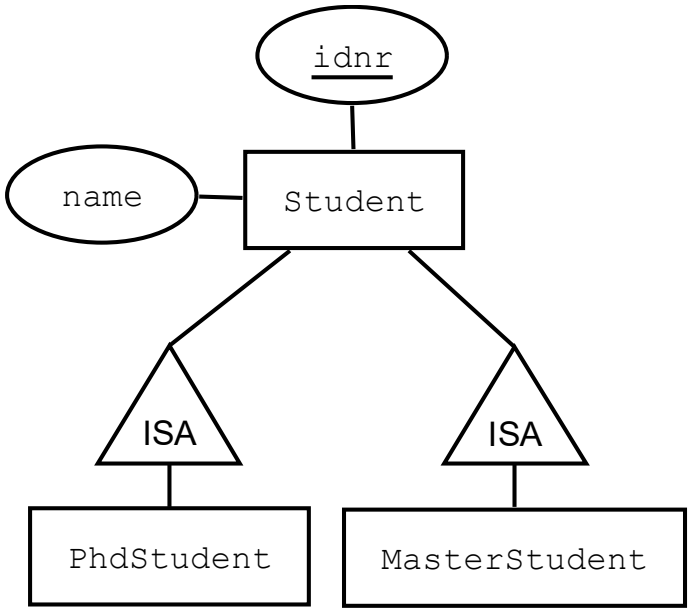
- Once again, the ER-approach always works...

Object Oriented translation of ISA

- A third type of translation is possible, where the subentity gets all the attributes of the superentity, and every value is a member of just one of the relations (in the ER-approach it is a member of the superentity or both)
- This translation is even more problematic than the Null-approach, so I will not explain it in more detail

Multiple subentities

- In ER (unlike Object Oriented Programming), a value can be a member of several subentities (if it also a member of the superentity)
- No easy way to limit membership to a single subentity

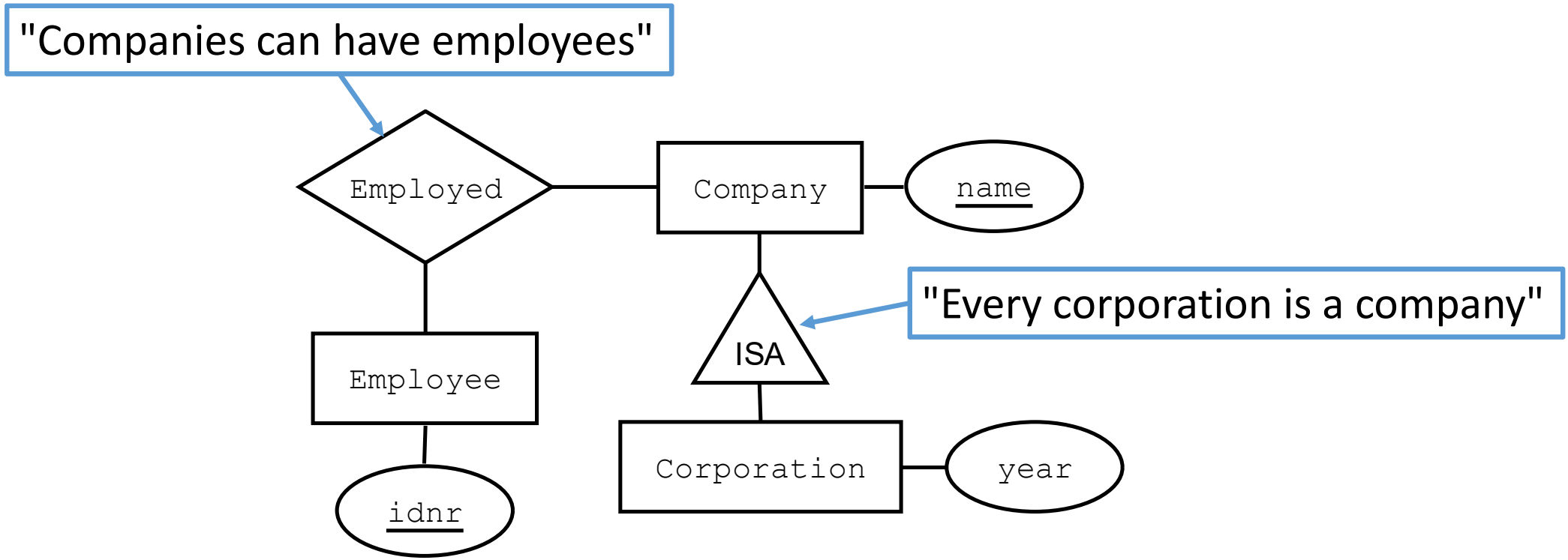


Students(<u>idnr</u>,name)
PhdStudents(<u>idnr</u>)
idnr -> Students.idnr
MasterStudents(<u>idnr</u>)
idnr -> Students.idnr

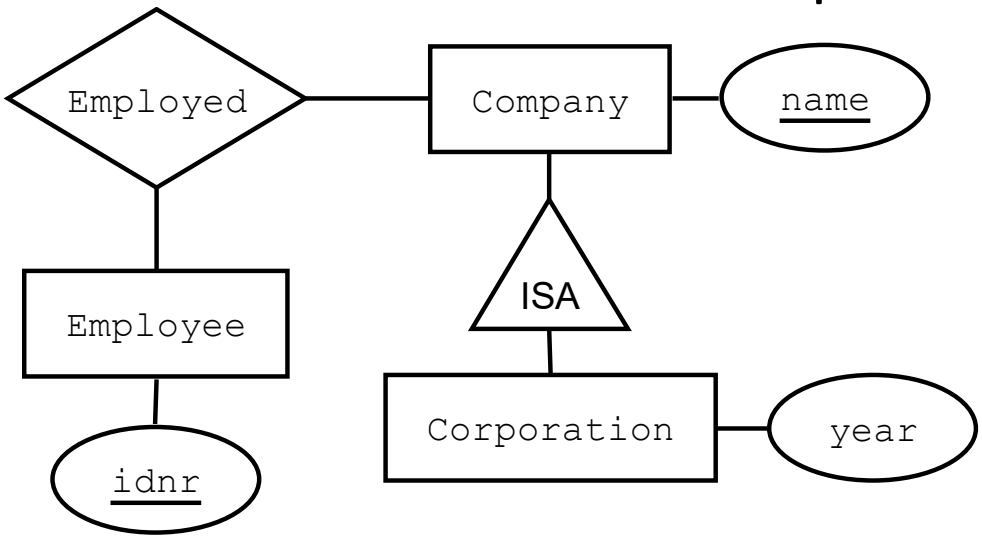
Four possibilites for a student X:
X is only in Student
X is in Student and PhdStudent
X is in Student and MasterStudent
X is in all three (!)

Quiz

- Can employees be employed by corporations in this design?
  - Answer: Yes!



Translation of example in last slide



Employees(idnr)
Companies(name)
Corporations(name, year)
name -> Companies.name
Employed(employee, company)
employee -> Employees.idnr
company -> Companies.name

All corporation names are also company names

Any company name can be used here

ISA relationships and primary keys

- Subentities can never have any key attributes of their own!
- Defies the concept of inheritance, if entity X is not identified by the same attributes as entity Y, we can never claim "X is a Y", it is something else
- If you need additional identifying attributes, use weak entities
  - In a way, weak entities are "subentities with extra identifying attributes"

Identifying ISA-relationships in domains

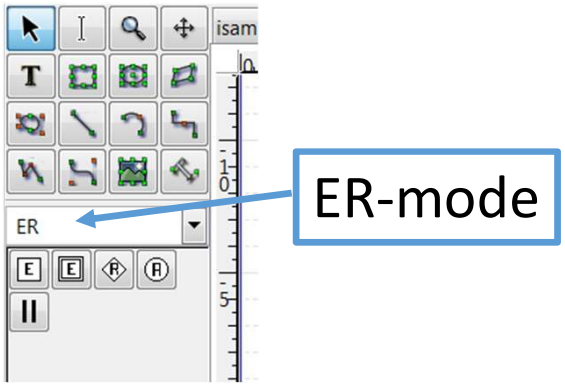
- When it makes sense to say "X is a Y", like "Corporation is a Company" or "Employee is a Person" or "Car is a Vehicle"
- In domain texts, it may also be stated as variations of "some X have y", where y is an attribute and not another entity (optional attributes)
- Sometimes when you want to model a subset of some entity even if they don't have extra attributes

Mission Accomplished!

You now know (or have at least heard of) all features of ER required for task 2!

Practicalities of making ER-diagrams

- We recommend using Dia to make diagrams (http://dia-installer.de/)
- Make sure you set it to ER-mode
- Pro-tip: Double-click an entity/relation to set its properties (e.g. weak etc)
- Pro-tip: Double-click an attribute to set primary key etc.



- Pro-tip: Double-click a line to change arrows, the curved arrow is available under "more arrows"
- Pro-tip: There are no ISA-relationships in the ER, but you can use merge/extract from flowchart (and just write ISA on them)
- Pro-tip: Always connect lines to the points of diamonds, not their center

Order of translation

- Always start with an entity whose translation does not depend on knowing the keys of other entities
  - Example: If two entities have a many-to-exactly-one relationship, always start with the entity on the exactly-one side
  - Example2: Translate superentities before you translate subentities
  - Basically: Do translations in reverse order of how the arrows are pointing
- Do many-to-many relationships last
- If you find yourself writing a reference to a relation you have not yet translated, you are doing things in the wrong order

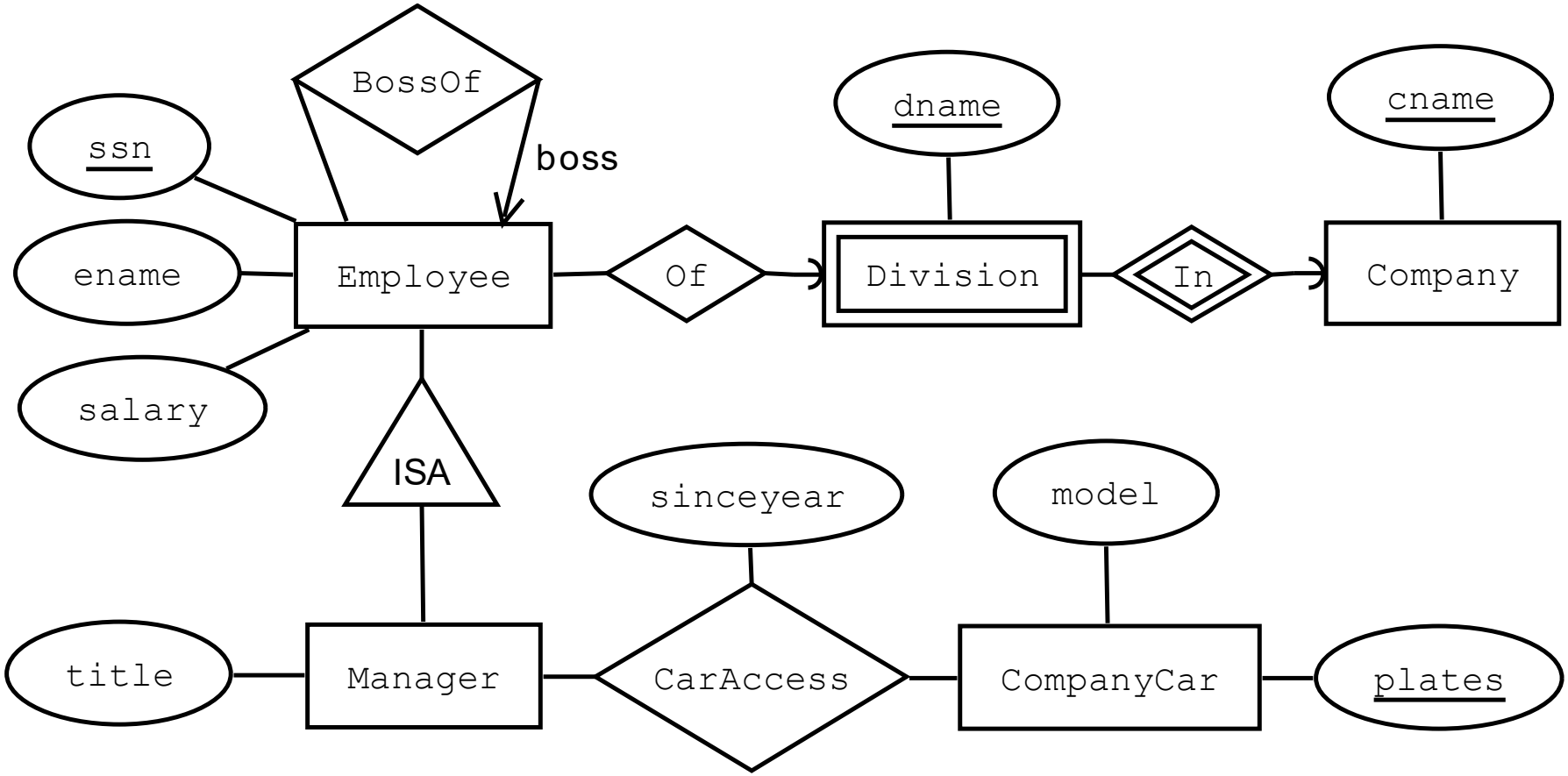
A small domain example

"*Make a database of companies in a corporate group. The companies have employees, that are divided into divisions within each company. Divisions in the same company can not have the same names. Most employees have a boss.*

Some employees are given a managerial position with a special title.

Also make an inventory of the company cars, who has access to which cars (only managers can have access to company cars) and for how long they have had access."

An ER-diagram with almost everything



```
Companies(cname)
Divisions(dname,company)
  company -> Companies.cname
Employees(ssn,division,company,ename,salary)
  (division,company) -> Divisions.(dname,company)
Managers(ssn,title)
  ssn -> Employees.ssn
CompanyCars(plates,model)

CarAccess(ssn,car,sinceyear)
  ssn -> Managers.ssn
  car -> CompanyCars.plates
BossOf(employee, boss)
  employee -> Employees.ssn
  boss -> Employees.ssn
```

Entities (including many-to-exactly-one relationships)

Remaining relationships

Used ER-approach here

Assumptions made in this model

- Several assumptions were made that where not clearly stated in the domain description:
  - Each employee has a unique social security number (ssn)
  - It's enough to know which year a car access was issued
  - The model of each company car is known
  - Titles for managers do not exist if no one has them (there is no Title entity)
- Making different assumptions would give a different diagram/schema
- It's always good to keep track of the assumptions you make!

Things that are not in the model (but should be?)

- Cars belonging to specific companies or divisions
  - Managers can only have access to cars from their own company?
- Bosses and their employees belong to the same company?
- An employee can not be the boss of themselves
  - More general: No cycles in Boss-relation (two employees being eachothers bosses etc)

- Some of these would be impossible to model in ER without loosing other constraints we have, and some would be very difficult to implement in SQL

Clever ways to patch up the schema

- Make sure that bosses and their employees work in the same company.

```
BossOf(employee, boss)
   employee -> Employees.ssn
   boss -> Employees.ssn
```

```
BossOf(employee, boss, company)
   (employee,company) -> Employees.(ssn,company)
   (boss,company) -> Employees.(ssn,company)
```

References express:
"employee works at company"
and "boss works at company"

- Difficult or impossible to express in the ER-Diagram ☹
- In SQL we have to add (sssn,company) UNIQUE to Employees for the reference to work (you can only have references to values that are explicitly stated to be unique)

Some things that can not be expressed in ER

- Secondary keys/uniqueness constraints
  - Can be identified using functional dependencies (next weeks topic)
- Value constraints (like "grade is one of 'U','3','4','5' ")
  - Can be added as side-notes/comments
- "Relationships between relationships"
  - Example: If we have one relationship for access to a company car, and one for currently using it, we can not express that the latter is a subset of the former
  - Can sometimes be implemented by cleverly modifying the schema (e.g. adding/modifying references) after translation from ER

Can we extend ER to express more things?

- Yes, but some of the limitations are due to limitations in the underlying relational model, or in the end limitations in SQL
  - Means we may get diagrams that can not easily be translated into executable code

Common mistakes – check before you submit!

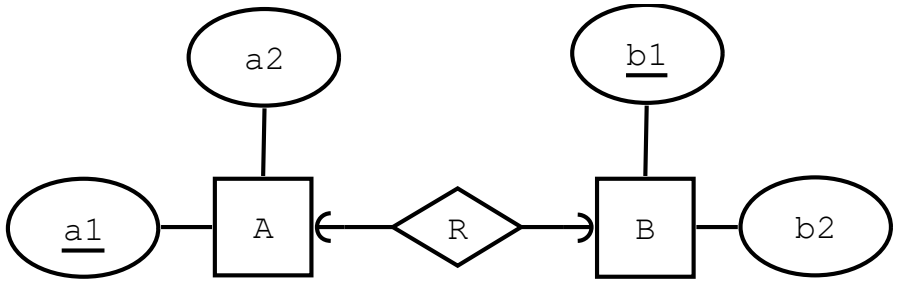
- Connecting entities directly without a relationship
- Connecting relationships directly to eachother
- Pointing the ISA the wrong way or pointing it at empty space
- Ignoring multiplicity and translating everything as many-to-many
  - Sanity check: Is every choice you made in the diagram reflected in your schema?
- Not using inheritance for optional attributes
  - If an entity has an attribute, it means every instance of it has a value
  - "I just set the attribute to 1000 000 if there is no limit"
  - Sanity check: If your diagram looks exactly as it would without a design choice, that design choice does not exist

Common mistakes: Inheritance

- Overusing Inheritance, e.g. introducing a "WinningTeam" or "CompletedCourse" entity when wins/grades are already stored in relationships
  - Introuduces redundancy!
  - The inheritance should be an inherent property of the entity (ask yourself: "can I look at a course and determine if it's a completed course or not?")
- "Branch ISA Program" or similar things that don't make sense to say out loud

Common mistakes: Playing with arrows

- If you ever get the feeling you should have something like this:



… more likely A and B should be a single entity?

- It's hard to understand the exact intention here, are the values in A and B in 1 to 1 correspondence? Or can several values in A map to the same value in B? (Then it should be two relationships!)