Databases

TDA357/DIT621– LP3 2023

Lecture 5

Ana Bove

(much of the material is based on material from
both Thomas Hallgren and Jonas Duregård)

January 26th 2023

Recall Last Lecture

- Foreign keys;
- More about consistency:
  - Policies on referencial contraints;
  - Assertions;
- Summary of **SQL**;
- Summary of relational schemas;
- Example.

Overview of Today's Lecture

- Entity-relationship (ER) model:
  - Entities and attributes;
  - Many-to-many relationships;
  - Many-to-exactly-one relationships;
  - Many-to-at-most-one relationships.
  - Multiway relationships;
  - Self-relationships;
  - Weak entities;
  - ISA relationships;
- (ER-example/exercise.)

Modelling/Designing the Database

How do we *correctly interpret the domain description*?

And how do we know *which tables* to implement? ...

... or what is a *good model/design* for our database?

Domain descriptions are usually:

- Informal;
- Ambiguous;
- Incomplete;
- Given by those who understand the problem but not databases.

We will now take a step back and look into how to *model/design a database* from its domain description.

Example of a Bad Database Design

Here is a table to keep track of booking in courses:

CourseBookings:

code	name	day	time	room	seats
TMV028	Automata	Monday	10	HB1	108
TDA357	Databases	Monday	15	HC4	104
TMV028	Automata	Tuesday	13	HB1	108
TDA357	Databases	Wednesday	10	HC2	115
TDA357	Databases	Thursday	10	HC4	104
...	...	...	...	...	...

What problems can we identify?

Redundancy: Nr. of seats in a room and name of a code is repeated;

Update anomaly: If codes/names/room/seats/... change we might forget to update all relevant entries;

Delete anomaly: If one removes all the bookings in a room we might loose the information on its number of seats.

A Better Design for this Example

Courses:

code	name
TMV028	Automata
TDA357	Databases
...	...

Rooms:

room	seats
HB1	108
HC2	115
HC4	104
...	...

Bookings:

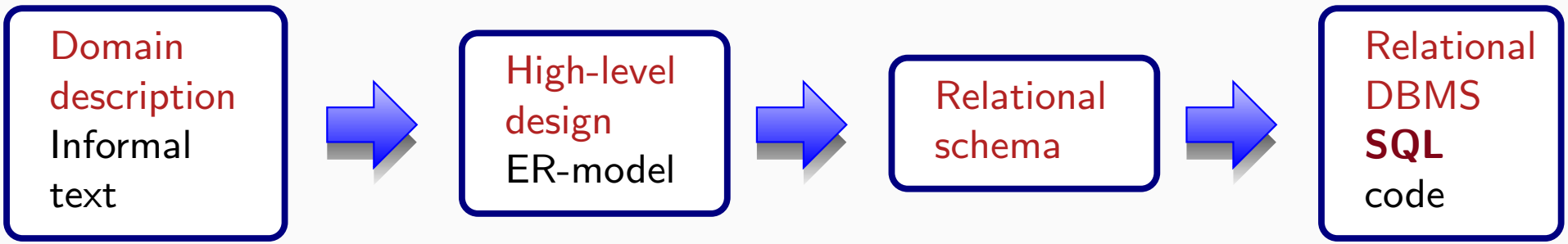
code	day	time	room
TMV028	Monday	10	HB1
TDA357	Monday	15	HC4
TMV028	Tuesday	13	HB1
TDA357	Wednesday	10	HC2
TDA357	Thursday	10	HC4
...	...	...	...

Using joins we then obtain the view/information in the previous slide.

Note: How easy will the modelling process be for larger domains?

Modelling the Domains

We will look into how to *transform the informal domain* description into a *formal model* of it, which can then easily be translated into a relational schema and from there to **SQL** code.



Entity-Relationship Model

ER-model was proposed by Peter Chen in 1976.

It is a high-level design consisting of:

Entity sets: The "things" we have information/talk about.

Relationships:

- How entities are related to each other, connections among entities;
- **Note:** relationships vs. relations (tables): they are *NOT* the same!

Attributes:

- The information we have that will go into the database tables in the end;
- Both entities and relationships can have attributes.
- (Attributes are NOT NULL unless otherwise stated.)

Entities and their Attributes

Entities are the objects in our domain.

They (in general) exist independently of other entities.

Example: Courses, Rooms, Students, Teachers, Countries, Animals, ...

Properties that every instance of the entity have are called *attributes*.

Values of these attributes will be stored in a table cell.

They can be numbers, text, ...

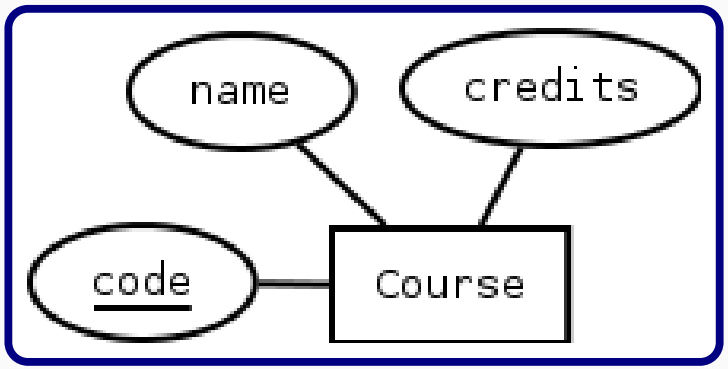
... but they are *NOT* collections of values or "pointers" that refer to attributes of other entities (they could have the same values as attributes in other entities though!).

Example: Course's codes and name, Room's name and size, Student's id number, name and telephone number, Teacher's name and address, ...

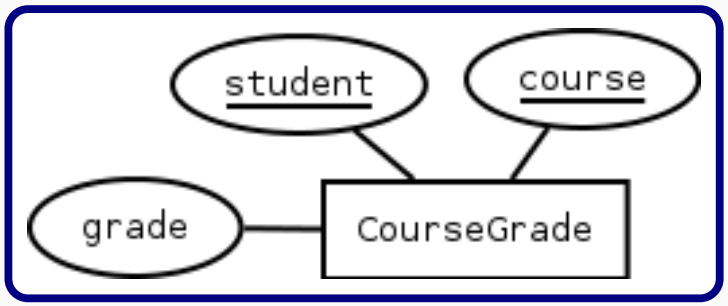
ER-diagramas and Schemas for Entities and Attributes

Entities become *boxes*.

Attributes become *elipses*. Attributes that are primary keys are underlined.



Courses (code, name, credits)

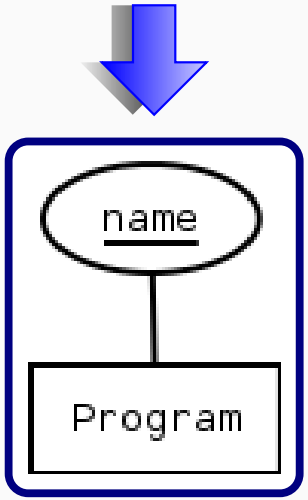


CourseGrades (student, course, grade)

Note: By convention, entities are named in singular while their relations in plural.

From Domain Description to **SQL**

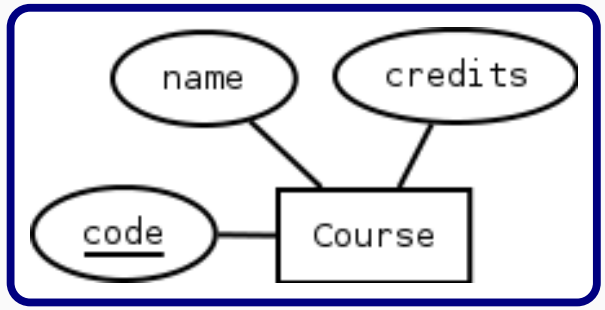
Example: *Each program has a name that identifies it.*



Programs (<u>name</u>)

```
CREATE TABLE Programs (
    name TEXT PRIMARY KEY );
```

Example: *Each course has a code that identifies it, a name and a nr of credits.*



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

```
CREATE TABLE Courses (
    code CHAR(6) PRIMARY KEY,
    name TEXT NOT NULL,
    credits FLOAT NOT NULL );
```

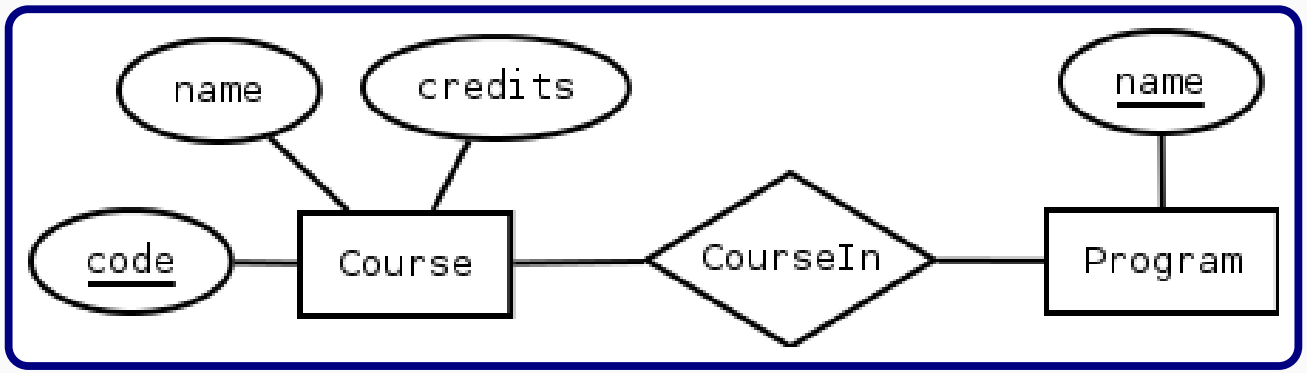
A Simple Many-to-Many Relationship

Example: *Every program has a set of courses.*

How do we model this? Recall that lists are not attributes!

We introduce a *relationship* containing pairs <course, program>!

Relationships have *diamond* shapes in ER-diagrams.



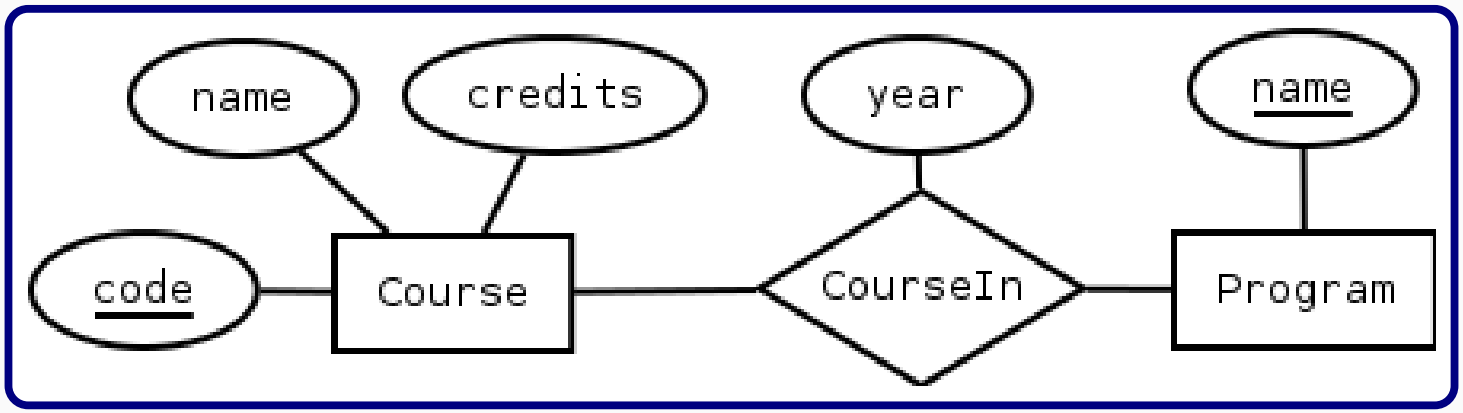
Courses (<u>code</u>, name, credits)

Programs (<u>name</u>)

CoursesIn (<u>course</u>, <u>program</u>)
course → Courses.code
program → Programs.name

Attributes of a Many-to-Many Relationship

Example: *Every program has a set of courses. The year in which the program starts offering the course needs to be remembered.*



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

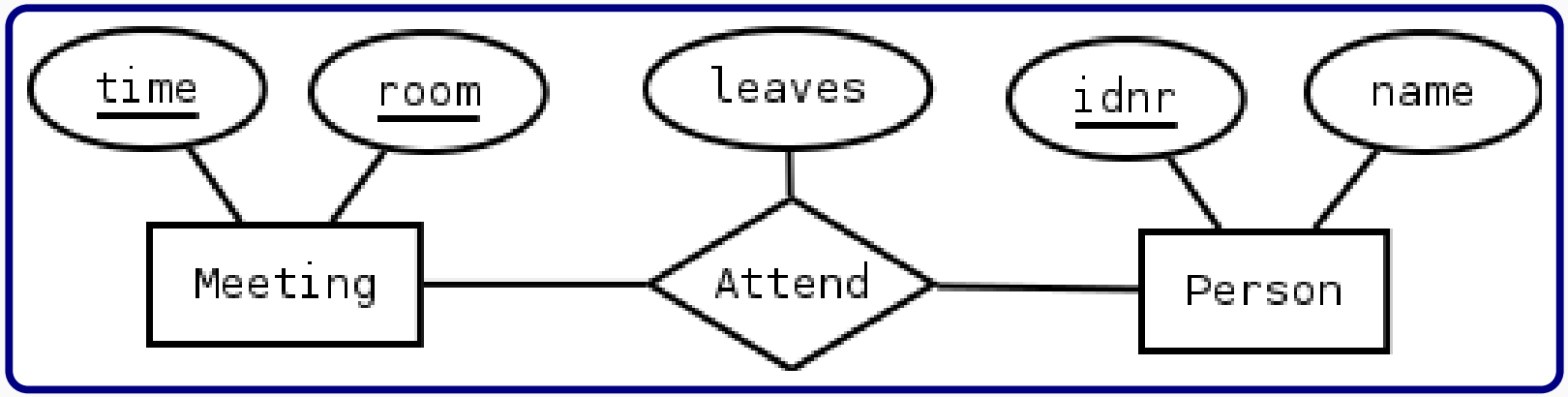
Programs (<u>name</u>)

CoursesIn (<u>course</u>, <u>program</u>, year)
course → Courses.code
program → Programs.name

Note: The year is not an attribute of neither a course nor a program but of the relationship between courses and programs!

Many-to-Many Relationships

- The primary key of the new relation consists of the (complete) primary keys of the entities it relates;
- None of the (other) attributes of the entities are part of this relation;
- The relation itself can have attributes.



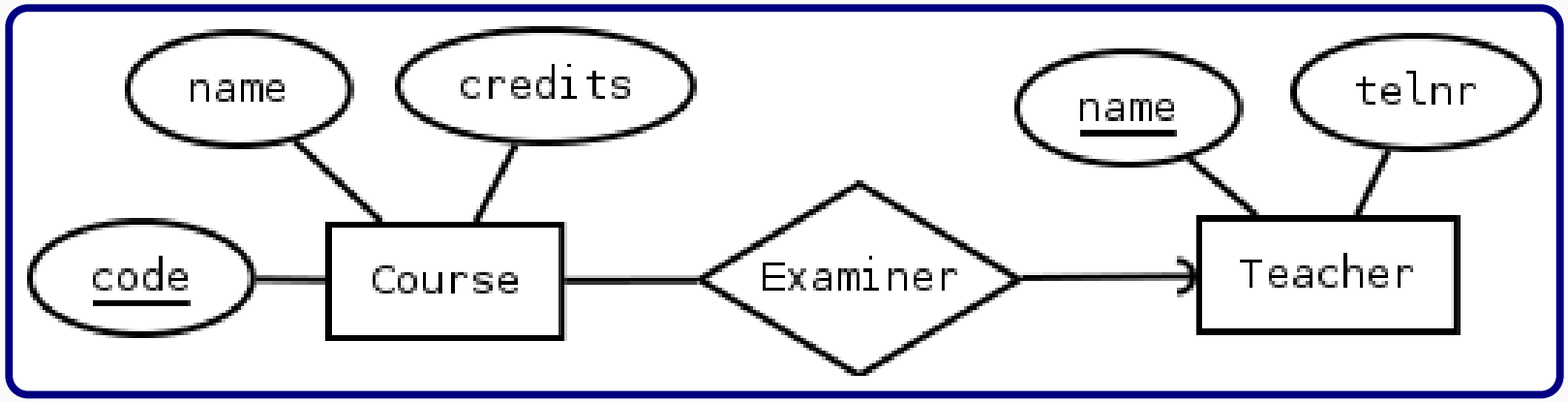
Meetings (<u>times</u>, <u>room</u>)

Persons (<u>idnr</u>, name)

Attends (<u>person</u>, <u>mtime</u>, <u>mroom</u>, leaves)
person $\rightarrow$ Persons.idnr
(mtime, mroom) $\rightarrow$ Meetings.(time, room)

Many-to-exactly-one Relationships

Example: *Each course has (exactly) one teacher as examiner.*

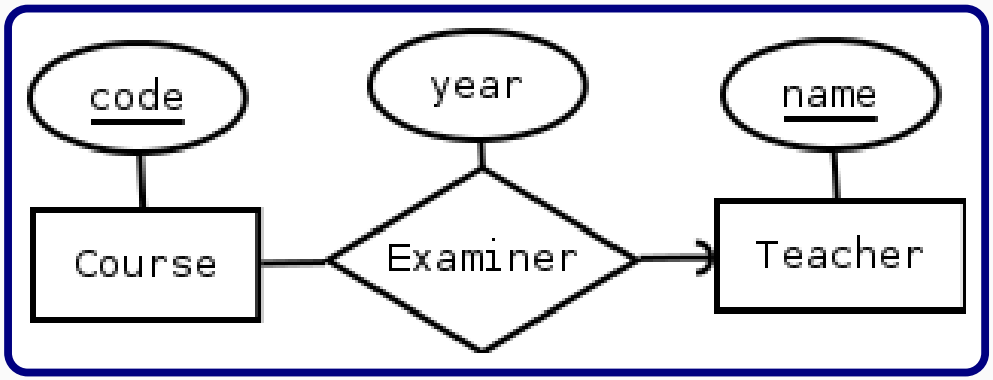


Teachers (<u>name</u>, telnr)

Courses (<u>code</u>, name, credits, examiner)
examiner → Teachers.name

Attributes of a Many-to-exactly-one Relationships

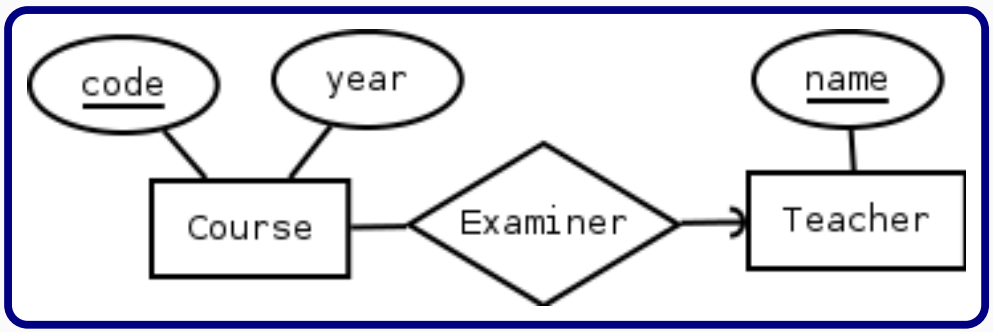
Compare this:



Courses (<u>code</u>, teacher, year)
teacher → Teachers.name

Teachers (<u>name</u>)

with this:



Courses (<u>code</u>, year, teacher)
teacher → Teachers.name

Teachers (<u>name</u>)

Different diagrams give the same schema!

Many-to-exact-one relationships could (in principle) have attributes but we need to make sure it is clear how the attributes in the relationship are to be understood.

Many-to-exactly-one Relationships

Observe that:

- The *round arrow* points to the teacher who examines a course;

  An arrow in the other direction would say that "each teacher examines a single/exactly one course";

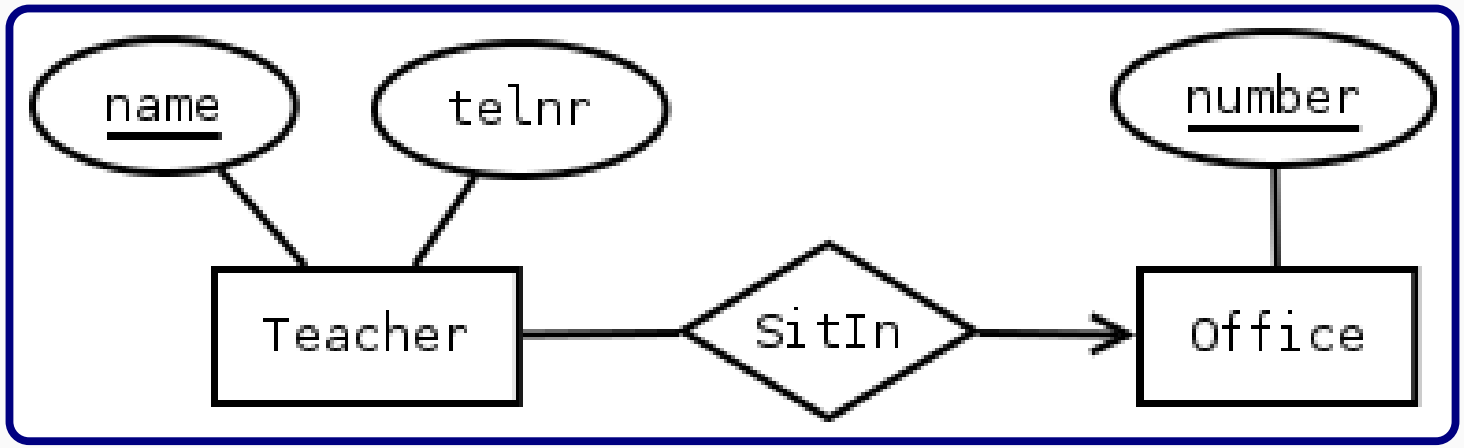
- In principle, "many" courses could have "exactly the same (one)" teacher as examiner;

- The table/relation for courses has one entry per course;

  The "examination" relationship becomes then an attribute of courses: each course has the/its examiner as attribute.

Many-to-at-most-one Relationships

Example: *Some teachers have an office/A teacher might have an office.*



ER-approach (preferable):

Offices (<u>number</u>)

Teachers (<u>name</u>, telnr)

SitsIn (<u>teacher</u>, office)
teacher → Teachers.name
office → Offices.number

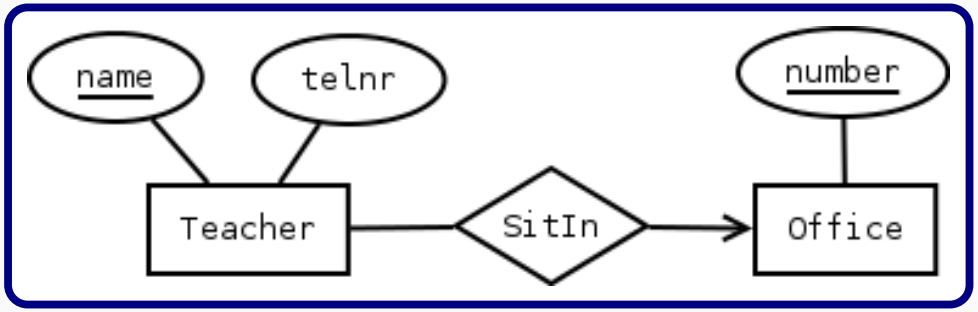
Null approach:

Offices (<u>number</u>)

Teachers (<u>name</u>, telnr, office (or null))
office → Offices.name

Attributes of a Many-to-at-most-one Relationships

Compare this:

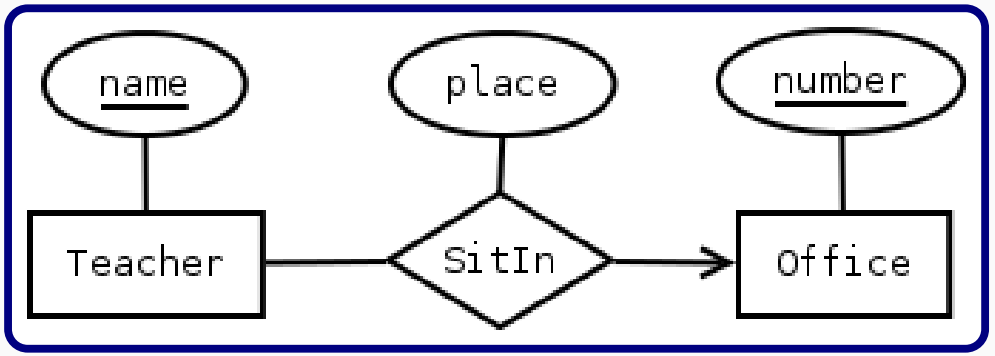


Offices (<u>number</u>)

Teachers (<u>name</u>, telnr)

SitsIn (<u>teacher</u>, office)
office → Offices.number
teacher → Teachers.name

with this:



Offices (<u>number</u>)

Teachers (<u>name</u>)

SitsIn (<u>teacher</u>, office, place)
office → Offices.number
teacher → Teachers.name

With the ER approach different diagrams give different schemas!

With the null approach we have the same problem as in the many-to-exactly-one case and maybe more (recall: what if the place is not empty but the office is?)

Many-to-at-most-one Relationships

Observe that:

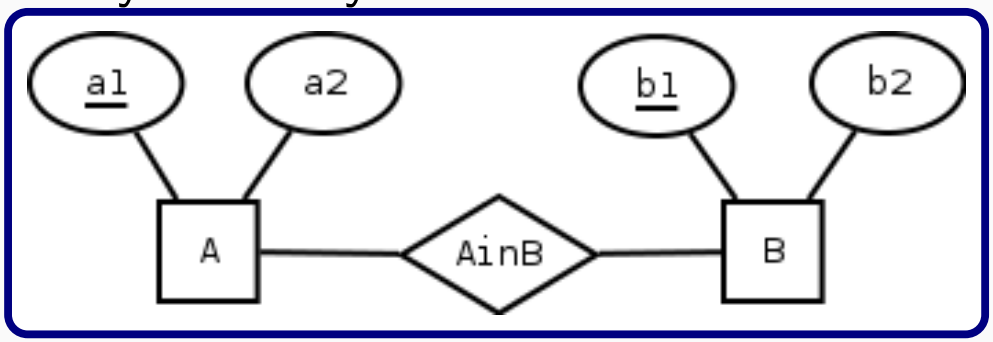
- The *pointy arrow* points to the possible office of a teacher;
- Many teachers could have the same office...
- ... however some teachers might not have an office;
- We can translate this relationship in two different ways:

  ER approach: always works!

  Null approach: Might not work well when:
  - The "at-most-one" side has a compound key: what would it mean if one of the attributes in the key is null?
  - The relationship has attributes: what will it mean if the key is empty but not the attribute? One would need to add a constraint in this case...

Summary: Many-to-many/exactly-one/at-most-one Relationships

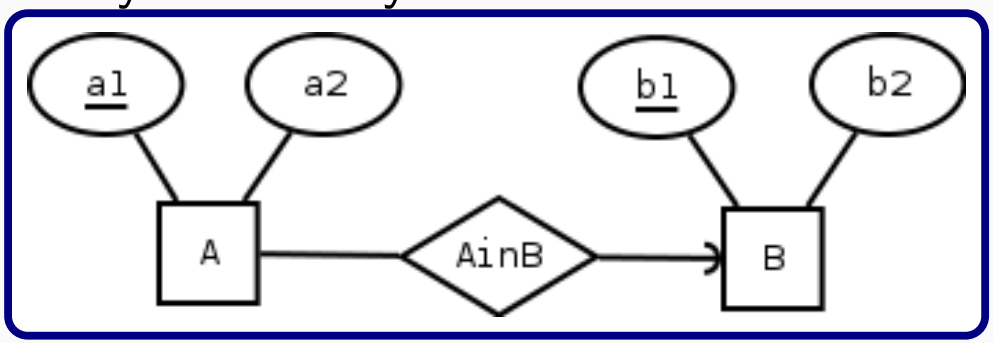
Many-to-many:



As (<u>a1</u>, a2)
Bs (<u>b1</u>, b2)

AsInBs (<u>a1</u>, <u>b1</u>)
a1 → As.a1
b1 → Bs.b1

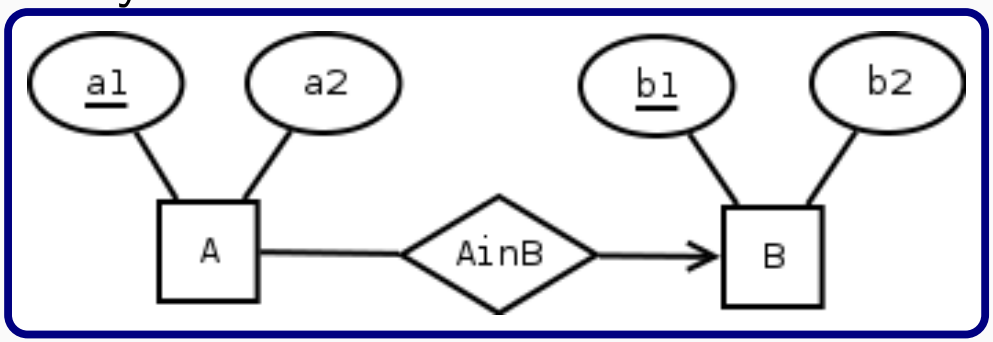
Many-to-exactly-one:



As (<u>a1</u>, a2, b1)
b1 → Bs.b1

Bs (<u>b1</u>, b2)

Many-to-at-most-one:

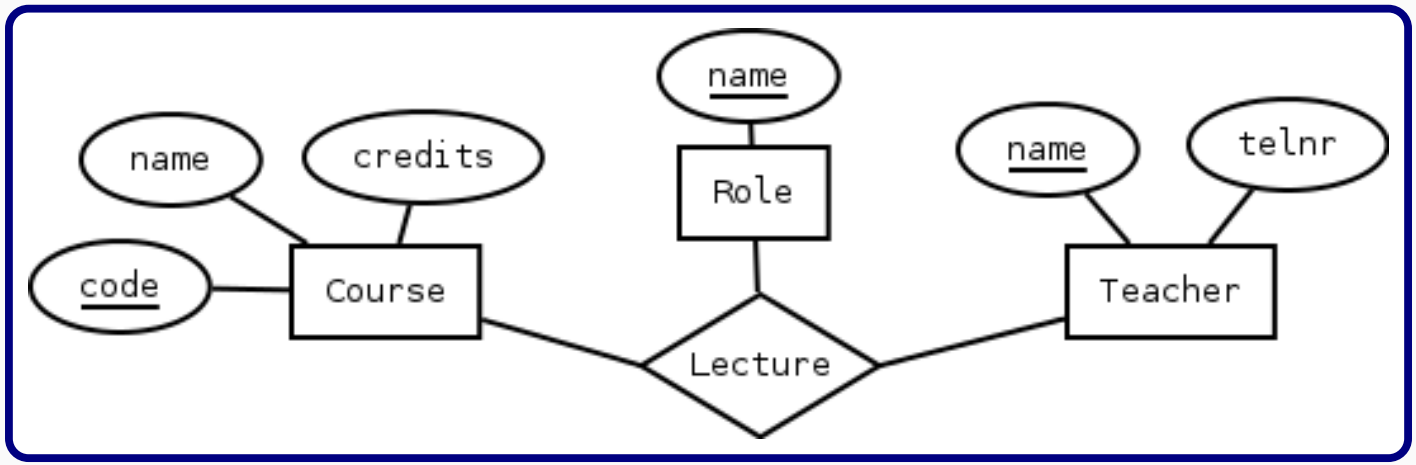


As (<u>a1</u>, a2)
Bs (<u>b1</u>, b2)
AsInBs (<u>a1</u>, b1)
(add references!)
Alternative:
As (<u>a1</u>, a2, b1 (or null))
Bs (<u>b1</u>, b2)

Multiway Relationships

Relationships can connect more than two entities.

Example: *Teachers can have different roles in a course*.



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

Teachers (<u>name</u>, telnr)

Roles (<u>name</u>)

Lectures (<u>course</u>, <u>teacher</u>, <u>role</u>)

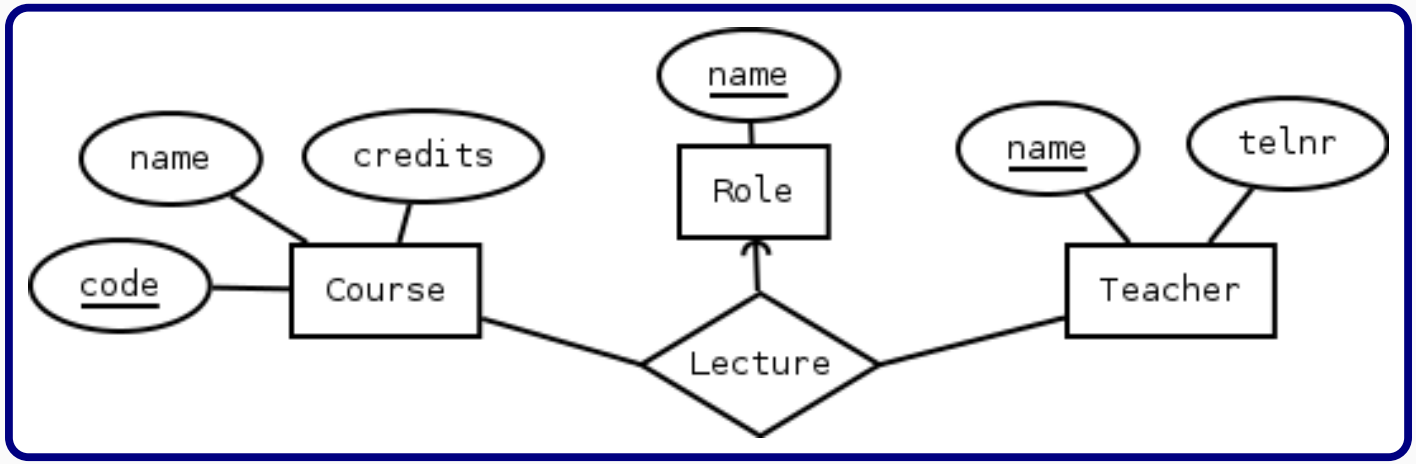
course → Courses.code
teacher → Teachers.name
role → Roles.name

Note: Is this really what we wanted?

Multiway Relationships (Cont.)

Is this better?

Example: *Teachers can have different roles in a course*.



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

Teachers (<u>name</u>, telnr)

Roles (<u>name</u>)

Lectures (<u>course</u>, <u>teacher</u>, role)

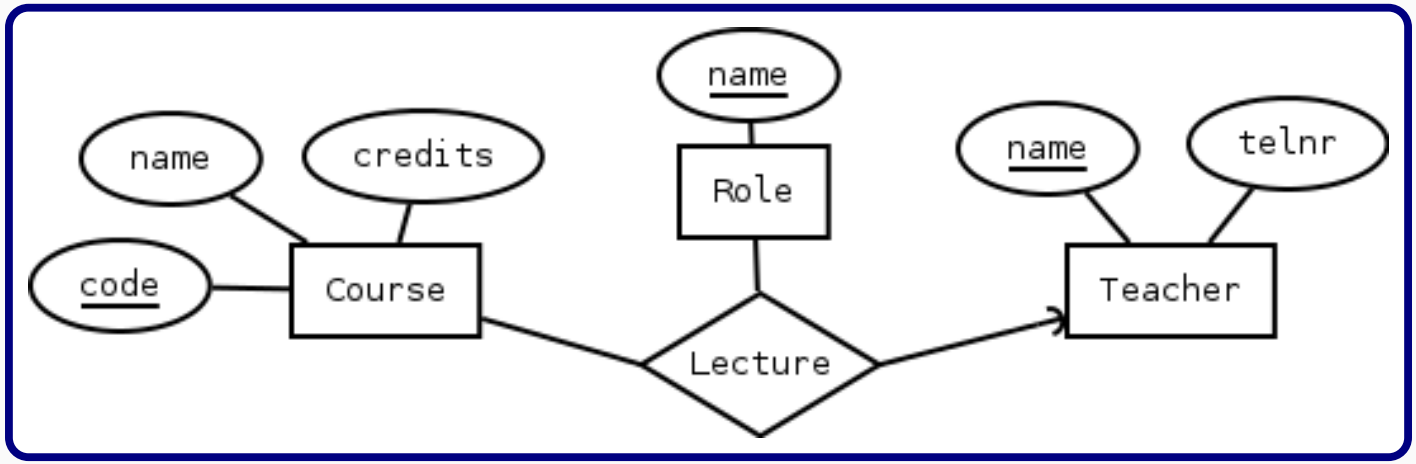
course → Courses.code
teacher → Teachers.name
role → Roles.name

Note: Similar to having an attribute in a relationship but with a reference!

Multiway Relationships (Cont.)

What about this?

Example: *Teachers can have different roles in a course*.



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

Teachers (<u>name</u>, telnr)

Roles (<u>name</u>)

Lectures (<u>course</u>, teacher, <u>role</u>)

course → Courses.code
teacher → Teachers.name
role → Roles.name

Note: Similar to having an attribute in a relationship but with a reference!

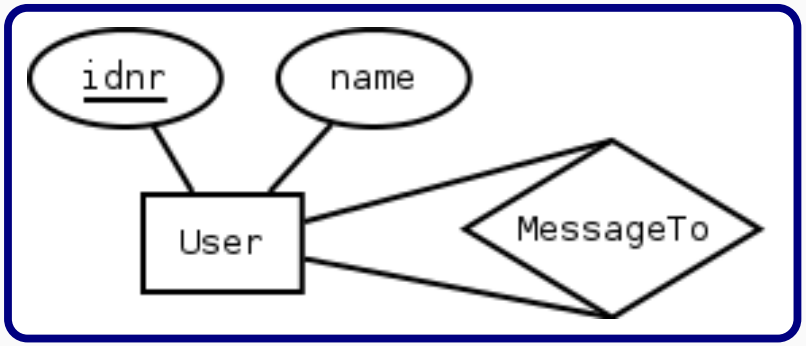
Multiway Relationships: Some Conclusions

- One could have relationships connecting more than two entities but they are sometimes tricky;
- It is important to understand exactly the domain we want to model;
- Domain descriptions might be ambiguous, maybe there are separate relationships instead?
- Or would it help to add a few UNIQUE constraints?
- Many-to-many-to-many and many-to-many-to-exactly one are easy to understand.

  Other variants are not that easy: try to put different kind of arrows and give them a meaning!

Self-relationships

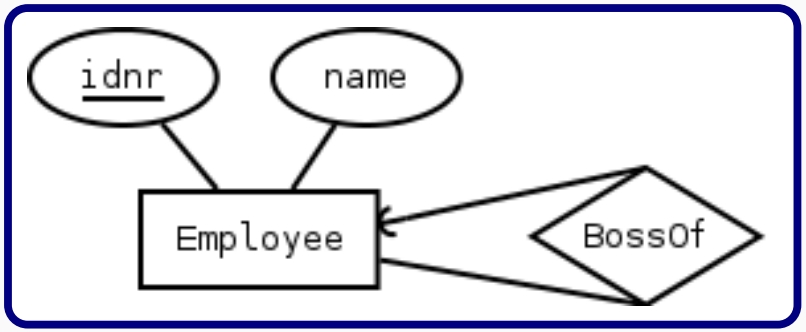
Many-to-many:



Users (<u>idnr</u>, name)

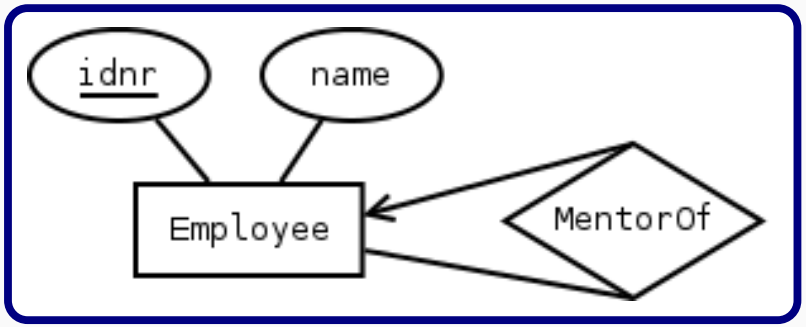
MessagesTo (<u>sender</u>, <u>receiver</u>)
sender → Users.idnr
receiver → Users.idnr

Many-to-exactly-one:



Employees (<u>idnr</u>, name, boss)
boss → Employees.idnr

Many-to-at-most-one:



Employees (<u>idnr</u>, name)

MentorsOf (<u>mentee</u>, mentor)
mentee → Employees.idnr
mentor → Employees.idnr

Self-relationships: Some Observations

There are certain issues of a self-relationship that cannot be expressed with ER-diagrams:

- Can a value be related to itself?
  - Can a user send a message to him/herself?
  - Can an employee be his/her own boss?
- Can there be cycles?
  - Can an employee be the boss of his/her own boss?
  - Can a mentee be the mentor of his/her own mentor?
- How to deal with symmetric relationships?
  For example when we use self-relationships to model "is a sibling of".

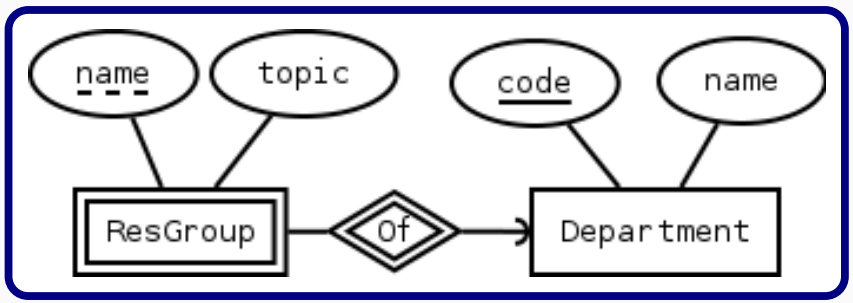
One could have some "side notes" but it is not always easy to implement them in **SQL**.

Weak Entities

Example: *Departments have research groups; each research group has a unique name within its department.*

Two research groups in different departments could have the same name.

So a research group is a *weak entity* since it cannot be identify only by its own attributes; it needs "support" from at least another entity (in this case departments) for identification!



Departments (<u>code</u>, name)

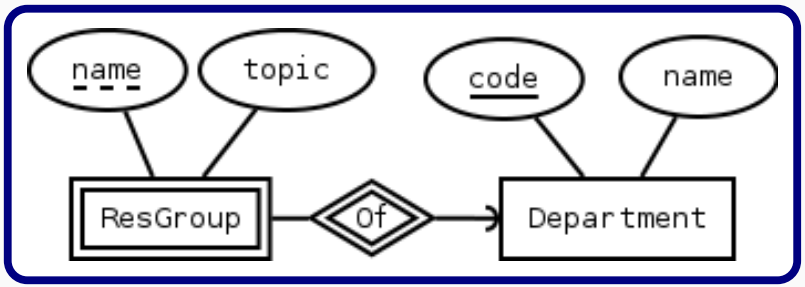
ResGroups (<u>name</u>, <u>department</u>, topic)
department → Departments.code

Note: Observe the *double lines* in the weak entity and in its relationship to the other entity, the *round arrow* and the *dashed line* on the weak entity's partial key.

Weak Entities vs Many-to-exactly-one Relationships

The difference lays on how we translate the key.

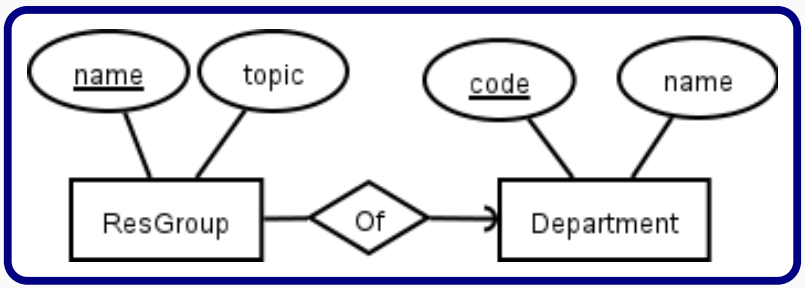
Weak entity:



Departments (<u>code</u>, name)

ResGroups (<u>name</u>, <u>department</u>, topic)
department → Departments.code

Many-to-exactly-one relationship:



Departments (<u>code</u>, name)

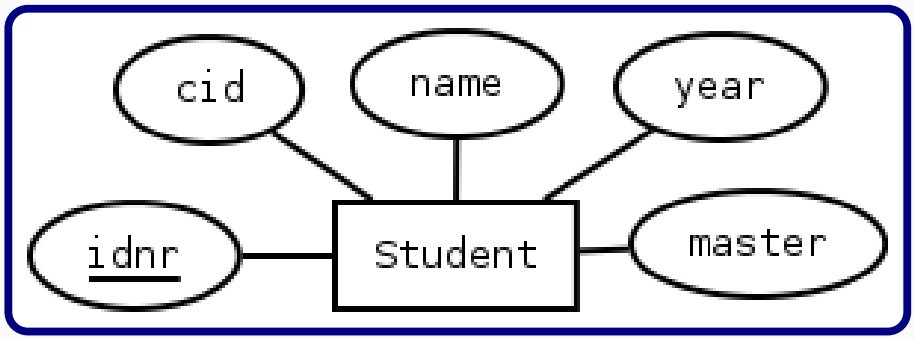
ResGroups (<u>name</u>, department, topic)
department → Departments.code

Inheritance and Sub-entities: Null Approach

Example: *Students are identified by their id number. They have a unique cid and a name.*

Some students are Ph.D. students and they have a date in which they joined the Ph.D. program.

Some students are master students and they have the name of the master program they read.



Students (idnr, cid, name,
year (or null),
master (or null))

Unique cid

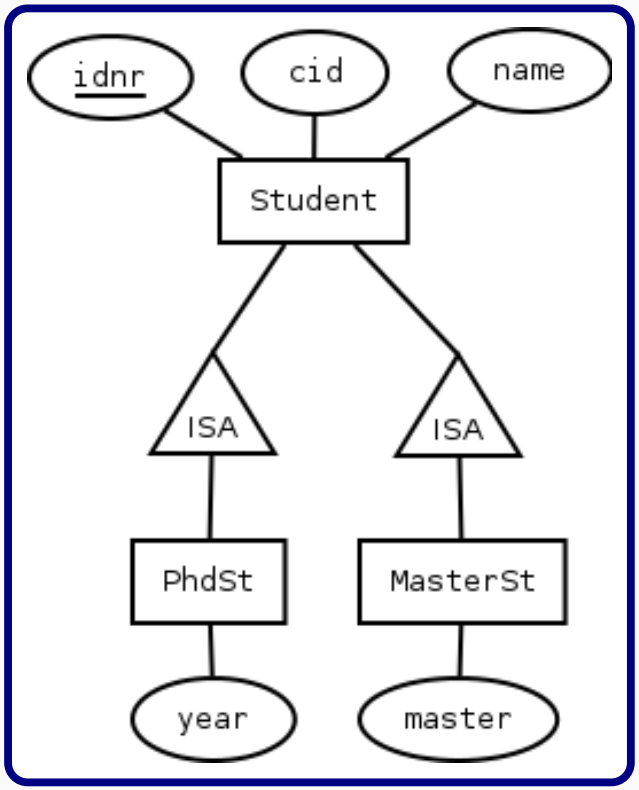
Some Problems with Null Approach

As usually, there are some problem with the null approach:

- When a student has null in the year/master name, is this because the student is not a PhD/mater student or because we forgot to add that information?
- What if neither year nor master name is null? Is the student both a master and a Phd? Or was it an error?
- If the sub-entities would *not* have extra attributes, how can we distinguish the different kind of students?
- If a sub-entity would have a relationship with another entity (for example the teacher that supervises a PhD student), how would this be modelled? A relationship between (all) students and teachers is not a good solution...
- What if the sub-entity has itself a sub-entity?

ER Approach: ISA Relationships

A Phd/master student *IS A* special kind of student.
They have all the attributes of students and a few more.
They have no key on their own, only that of the "super entity".



Students (<u>idnr</u>, cid, name)
Unique cid

PhdSts (<u>idnr</u>, year)
idnr → Students.idnr

MasterSts (<u>idnr</u>, master)
idnr → Students.idnr

Note: It is possible for a student to be both a PhD and a master student!

Making ER-Diagrams

- One could use Dia to make the diagrams.
- In *Tools and Tips* under *Modules* in the Canvas page of the course, you will find some information on how to install the tool and a video on how to use it.

Example/Exercise (from Exam): ER-Model from a Domain

You are making a database for recipes and ingredients. Every recipe consists of components, like "filling" or "glaze". Each component has its own set of ingredients, and an amount for each such ingredient.

Additional features:

- Each recipe has its own unique name.
- Each component has a name that is unique for the recipe it's in;
- Each recipe has an instruction text;
- The database should store the unit of measurement of each ingredient (e.g. butter is always measured in gram so if a component contains 100 of ingredient "butter", that means 100 grams);
- Some recipes require using an oven, for these recipes one should store temperature and time;
- Ingredients can have any number of alternate ingredients (globally for all recipes that use that ingredient); there is a conversion factor for each such alternate, e.g. 1 gram of sugar can be replaced by 3/4 gram of honey (so the factor is 0.75).

Example/Exercise: Recipe for Cinnamon buns

Note: Uses an oven at 220 degrees Celsius for 6 minutes.

Components:

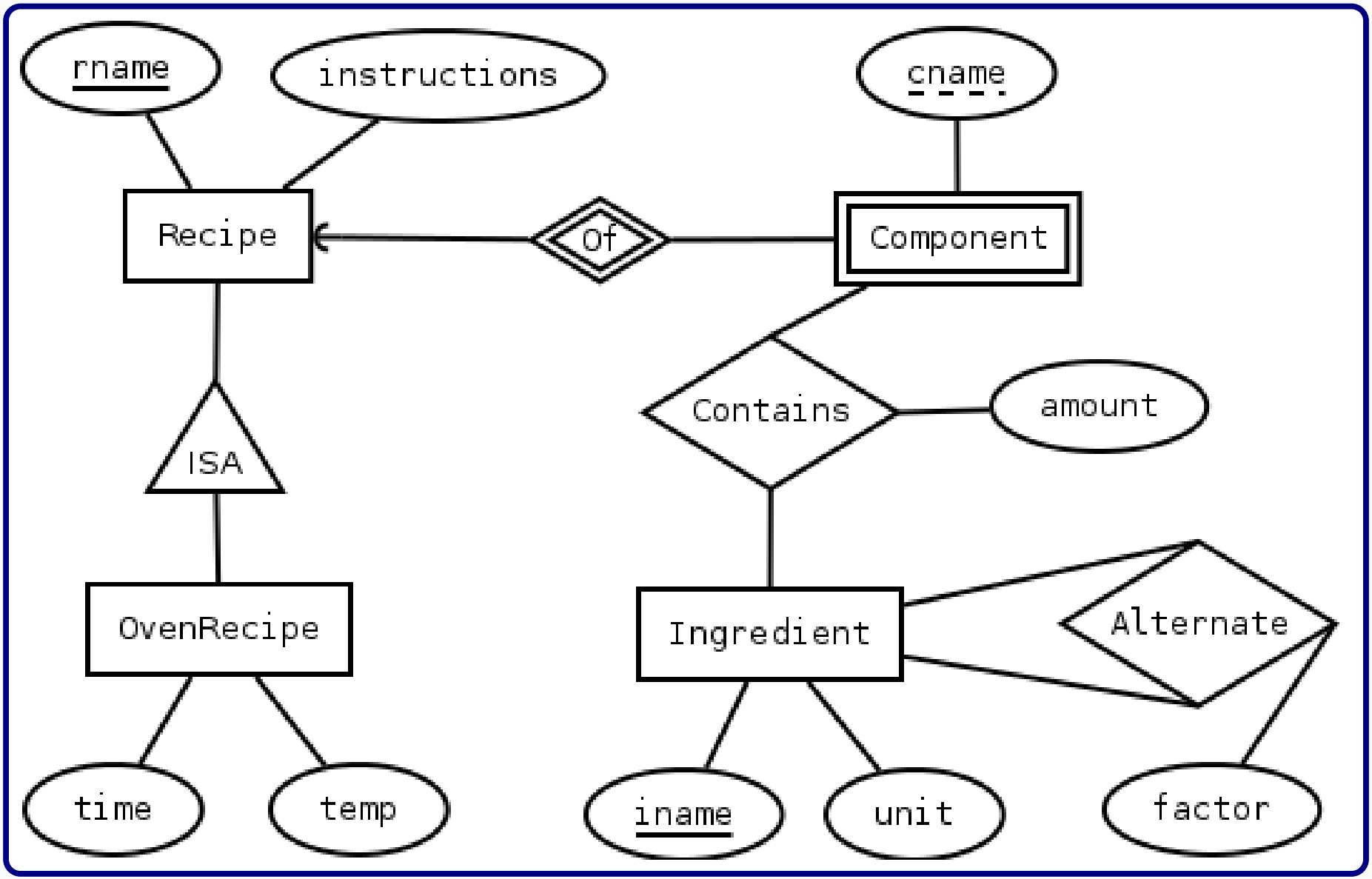
Main ingredients: 35 g yeast, 100 g sugar, 3 dl milk, 1 x egg, 120 g butter, 1 tsp salt, 1 tsp ground cardamom 750 g flour.

Filling: 100 g butter, 50 g sugar, 2 tsp cinnamon.

Glaze: 1 egg, 0.1 dl water, 100 g pearl sugar.

Instructions: Crumble the yeast in a bowl and stir in a few tablespoons of milk ...

Example: ER-Diagram



Example: Relational Schema of the ER-Diagram

Recipes (<u>rname</u>, instructions)

OvenRecipes (<u>rname</u>, time, temp)
  rname $\rightarrow$ Recipes.rname

Components (<u>cname</u>, <u>recipe</u>)
  recipe $\rightarrow$ Recipes.rname

Ingredients (<u>iname</u>, unit)

Contains (<u>component</u>, <u>recipe</u>, <u>ingredient</u>, amount)
  (component, recipe) $\rightarrow$ Components.(cname, recipe)
  ingredient $\rightarrow$ Ingredients.iname

Alternate (<u>base</u>, <u>alternate</u>, factor)
  base $\rightarrow$ Ingredients.iname
  alternate $\rightarrow$ Ingredients.iname

Overview of Next Lecture

- Functional dependencies:
  - Reflexivity, transitivity and augmentation;
  - Closures;
  - Superkeys and keys;
  - Minimal basis;
  - Boyce-Codd Normal form (BCNF) ...
  - ... and its normalisation algorithm.

Reading:

Book: chapter 3

Notes: chapters 4.1–4.3 and 5