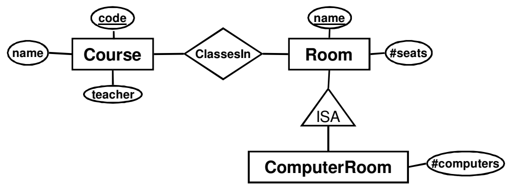
Generalisation/Specialisation

- Subclass = sub-entity = special case.
- More attributes and/or relationships.
- A subclass shares the key of its parent.

- Drawn as an entity connected to the superclass by a special triangular relationship called *ISA.* Triangle points to superclass.
  - ISA = ”is a”

Example:



- A computer room *is a* room.
- Not all rooms are computer rooms.
- Computer rooms share the extra property that they have a number of computers.

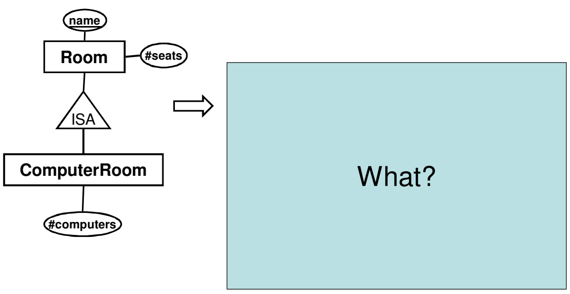
Subclass/Superclass Hierarchy

- We assume that subclasses form a tree hierarchy.
  - A subclass has only one superclass.
  - Several subclasses can share the same superclass.
    - E.g. Computer rooms, lecture halls, chemistry labs etc. could all be subclasses of Room.
  - One class can have several (orthogonal) subclass hierarchies.

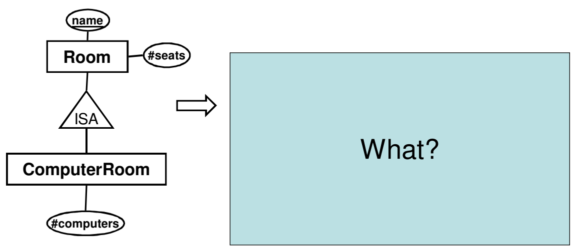
Translating ISA to relations

- Three different approaches
  - *E-R:* An ISA relationship is a standard one-to-”exactly one” relationship. Each subclass becomes a relation with the key attributes of the superclass included.
  - *NULLs:* Join the subclass(es) with the superclass. Entities that are not part of the subclass use NULL for the attributes that come from the subclass.
  - *Object-oriented:* Each subclass becomes a relation with all the attributes of the superclass included. An entity belongs to either of the two, but not both.

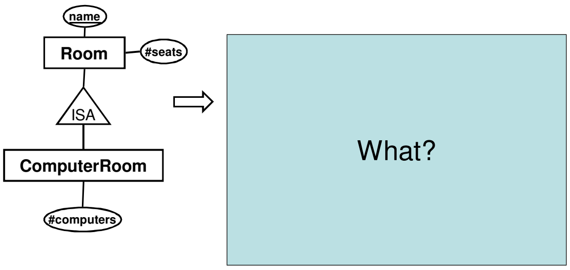
The E-R approach:



The NULLs approach:



The object-oriented approach:



Comparison

- E-R approach
  - Good when searching for general information about all entities in the class hierarchy.
    - *"List the number of seats in all rooms"*
- OO approach
  - Good when searching for information about entities in a subclass only.
    - *"List the number of seats in all computer rooms"*
- NULLs approach
  - Could save space in situations where most entities in the hierarchy are part of the subclass (e.g. most rooms have computers in them).
  - Reduces the need for *joins* (see later).

E-R summary

- Entities
- Attributes
- Relationships
  - Multiplicity
- Weak entities
- Generalisation/specialisation

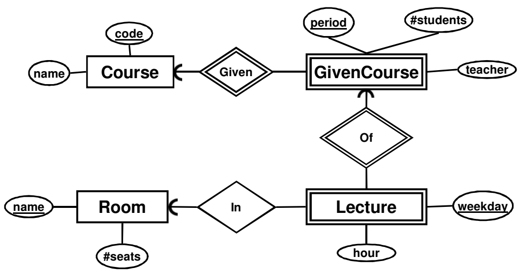
- Translation to relations

Scheduler database revisited

"We want a database for an application that we will use to schedule courses. …"

- Course codes and names, and the period the courses are given
- The number of students taking a course
- The name of the course responsible
- The names of all lecture rooms, and the number of seats in them
- Weekdays and hours of lectures

E-R diagram for Scheduler



Translate to relations

```
Courses(code, name)
GivenCourses(course, period, #students, teacher)
  course -> Courses.code
Lectures(course, period, room, weekday, hour)
  (course, period) -> GivenCourses.(course, period)
  room             -> Rooms.name
Rooms(name, #seats)
```

Compare with the "good" one from the previous lecture – we've reached the same conclusion using the structured and well-defined method.

Exam – E-R diagrams

"A small train company wants to design a booking system for their customers. …"

- Given the problem description above, construct an E-R diagram.
- Translate the E-R diagram into a database schema.

Programming Assignment

- Write a "student portal" application in Java
  - Part I: Design
    - Given a domain description, design a database schema using an E-R diagram and functional dependencies.
  - Part II: Construction and Usage
    - Implement the schema from Part I in Oracle.
    - Insert relevant data.
    - Create views.
  - Part III: Construction
    - Create triggers.
  - Part IV: Interfacing from external Application
    - Write a Java application that uses the database from Part III.

Programming Assignment

- Each task must be completed and approved before the next can be started.
  - Submit in good time!
- Preferrably, work in pairs.

System Specification

- Your final application should have the following functionality:
  - Info: A student should be able to ask the system for info about herself, including what courses she has read or is registered to.
  - Register: A student should be able to register for a course. If there is no room on the course, she should be put in a waiting list.
  - Unregister: A student should be able to withdraw a registration. If some other student is on the waiting list, that student should be registered instead.

Part I - Design

- Design the database schema by drawing an E-R diagram of the domain, and then translating your diagram to relations.
- Verify your schema by identifying all functional dependencies that you expect to hold on the domain, and check them against the schema.

Part I - Design

- Hand in:
  - a diagram
  - a database schema
  - the FDs of the domain
  - a text report where you argue the correctness of your solution.
- Submission deadline: 18 November 2014

Database design II

Functional Dependencies
BCNF

Design theory for relational databases

- Offers ways to "improve" a relational design
- ("improve" usually means reducing the amount of redundancy)
- Chapter 3 of the textbook introduces the concepts:
  - functional dependencies
  - normalization

Functional dependencies (FDs)

- X → A
  - "X determines A", "X gives A"
  - "A depends on X"
- X is a set of attributes, A is a single attribute
- Examples:
  - **`code`** → **`name`**
  - **`code, period`** → **`teacher`**

Why "functionally" dependent?

- X → A is a (deterministic) function from X to A. Given values for the attributes in the set X, we get the value of A.
- Example:
  - **`code`** → **`name`**
  - imagine a function f(code) which returns the name associated with a given code.

A note on syntax

- A **functional dependency** exists between attributes in the <u>same</u> relation

  e.g. in relation Courses we have FD:

  `code` → **`name`**
- A **reference** exists between attributes in two different relations, e.g. for relation GivenCourses we have reference:

  `course -> Courses.code`

- Two completely different things, but with similar syntax. Clear from the context which is intended.

Assertions on a schema

- X → A is an assertion about a schema R
  - If two tuples in R agree on the values of the attributes in X, then they must also agree on the value of A.
- Example: **`code, period`** → **`teacher`**
  - If two tuples in the GivenCourses relation have the same course code and period, then they must also have the same teacher.

Quiz!

What are reasonable FDs for the scheduler domain?

`Schedules(code, name, period, #students, teacher, room, #seats, weekday, hour)`

code	*name*	*per.*	*#st*	*teacher*	*room*	*#seats*	*day*	*hour*
TDA357	Databases	2	87	Niklas Broberg	VR	216	Monday	13:15
TDA357	Databases	2	87	Niklas Broberg	HB1	184	Thursday	10:00
TDA357	Databases	4	93	Rogardt Heldal	HB1	184	Tuesday	08:00
TDA357	Databases	4	93	Rogardt Heldal	HB1	184	Friday	08:00
TIN090	Algorithms	1	64	Devdatt Dubhashi	HC1	126	Wednesday	08:00
TIN090	Algorithms	1	64	Devdatt Dubhashi	HA3	94	Thursday	13:15

Quiz: (an) answer

What are reasonable FDs for the scheduler domain?

```
code → name
code, period → #students
code, period → teacher
room → #seats
code, period, weekday → hour
code, period, weekday → room
room, period, weekday, hour → code
```

Where do FDs come from?

- ”Keys” of entities
  - If code is the key for the entity Course, then all other attributes of Course are functionally determined by code, e.g. **`code`** → **`name`**
- Relationships
  - If all courses hold lectures in just one room, then the key for the Course entity also determines all attributes of the Room entity, e.g. **`code`** → **`room`**
- Physical reality
  - No two courses can have lectures in the same room at the same time, e.g. **`room, period, weekday, hour`** → **`code`**

Multiple attributes on RHS

- X → A,B
  - Short for X → A and X → B
  - If we have both X → A and X → B, we can combine them to X → A,B.
  - **`course, period`** → **`teacher, #students`**
- Multiple attributes on LHS can be crucial!
  - **`course, period`** → **`teacher`**
    - **`course`** ↛ **`teacher`**
    - **`period`** ↛ **`teacher`**

Quiz!

- What’s the difference between the LHS of a FD, and a key?
  - both uniqely determine the values of other attributes.
  - …but a key must determine *all* other attributes in a relation!
  - We use FDs when determining keys of relations (will see how shortly).

Trivial FDs

- A FD is *trivial* if the attribute on the RHS is also on the LHS.
  - Example: course, period → course

Quiz: Is this a trivial FD?

`course, period` → **`course, name`**

Shorthand for

`course, period` → **`course`** (trivial)
`course, period` → **`name`** (not trivial)

Armstrong's axioms

Suppose X, Y and Z are sets of attributes in relation R.

1. Reflexivity.
   If Y is a subset of X, then $X \rightarrow Y$ is a trivial FD.
2. Augmentation.
   If $X \rightarrow Y$ holds, then $XZ \rightarrow YZ$ holds.
3. Transitivity.
   If $X \rightarrow Y$ and $Y \rightarrow Z$ hold, then $X \rightarrow Z$ holds.

Basis

Suppose S is a set of FDs that hold for a given relation.

- A *basis* for S is any set of FDs that is equivalent to S.
- S and B are equivalent if and only if S follows from B and B follows from S.

Minimal basis

B is a *minimal basis* if:

1. All FDs in B have a single attribute on the right side.
2. The result of removing any FD from B is not a basis.
3. The result of removing any attribute from the left side of any FD in B is not a basis.

Closure of a set of attributes

- Computing the *closure* of X means finding all FDs that have X as the LHS.
- If A is in the closure of X, then $X \rightarrow A$.
- The closure of X is written X^+.

Computing the closure

- Given a set of FDs, F, and a set of attributes, X:
  1. Start with $X^+ = X$.
  2. For all FDs $Y \rightarrow B$ in F where Y is a subset of X^+, add B to X^+.
  3. Repeat step 2 until there are no more FDs that apply.

Quiz!

What is the closure of {code, period, weekday}?

```
code → name
code, period → #students
code, period → teacher
room → #seats
code, period, weekday → hour
code, period, weekday → room
room, period, weekday, hour → code
```

```
{code, period, weekday}+ =
  {code, period, weekday, name, #students,
   teacher, hour, room, #seats}
```

What are FDs really?

- Functional dependencies represent a special kind of constraints of a domain – dependency constraints.
- We can use FDs to verify that our design indeed captures the constraints we expect.

Finding keys

- For a relation R, any subset X of attributes of R such that X^+ contains all the attributes of R is a *superkey* of R.
  - Intuitively, a superkey is any set of attributes that determine all other attributes.
  - The set of all attributes is a superkey.
- A *key* for R is a *minimal* superkey.
  - A superkey X is minimal if no proper subset of X is also a superkey.
    - Minimal – no subset is a key
    - Minimum – the smallest, i.e. the one with the fewest number of attributes

Using attribute closures to find all FDs, superkeys and keys (1)

Suppose we have relation R(A,B,C) and FDs AB → C and C → A.

A systematic way to find all other FDs is to consider the closures of all sets of attributes:

$\{A\}^+ = \{A\}$	$\{A,B\}^+ = \{A,B,C\}$	$\{A,B,C\}^+ = \{A,B,C\}$
$\{B\}^+ = \{B\}$	$\{A,C\}^+ = \{A,C\}$	
$\{C\}^+ = \{A,C\}$	$\{B,C\}^+ = \{A,B,C\}$	

One extra (non-trivial) FD: BC → A

Using attribute closures to find all FDs, superkeys and keys (2)

$\{A\}^+ = \{A\}$	$\{A,B\}^+ = \{A,B,C\}$	$\{A,B,C\}^+ = \{A,B,C\}$
$\{B\}^+ = \{B\}$	$\{A,C\}^+ = \{A,C\}$	
$\{C\}^+ = \{A,C\}$	$\{B,C\}^+ = \{A,B,C\}$	

- Superkeys: {A,B}, {B,C}, {A,B,C}
- Keys: {A,B}, {B,C}
- {A,B,C} is not a key, since subset(s) of it's attributes are (super)keys.

Primary keys

- There can be more than one key for the same relation.
- We choose one of them to be the *primary key*, which is the key that we actually use for the relation.
- Other keys could be asserted through uniqueness constraints.
  - E.g. for the self-referencing relation

Example:

```
Rooms(name, #seats)
NextTo(right, left)
  right -> Rooms.name
  left  -> Rooms.name
  left unique
```

For NextTo we have both

- `left → right`
- `right → left`

Both `left` and `right` are keys, but we have chosen `right` to be the primary key for `NextTo`. We can add a constraint stating that `left` should be unique.

Note: The syntax for constraints is not well specified. Both the reference syntax, as well as the uniqueness assertion, are my suggestions only (but they're rather good).

Quiz!

What is the key of Schedules?

```
Schedules(code, name, period, #students,
  teacher, room, #seats, weekday, hour)

code → name
code, period → #students
code, period → teacher
room → #seats
code, period, weekday → hour
code, period, weekday → room
room, period, weekday, hour → code
```

Example:

- X = `{code, period, weekday, hour}` is a superkey of the relation Schedules since X^+ is the set of all attributes of Schedules.
- However, Y = `{code, period, weekday}` is also a superkey, and is a subset of X, so X is not a key of Schedules.
- No subset of Y is a superkey, so Y is also a key.

Two keys exist:

```
{code, period, weekday}
{room, period, weekday, hour}
```

Make reality match theory

- In some cases reality is not suitably deterministic. We may need to invent key attributes in order to have a key at all.

Quiz: Give examples of this phenomenon from reality!

Social security numbers, course codes, product numbers, user names etc.

Quiz time!

What's wrong with this schema?

```
Courses(code, period, name, teacher)
code → name
code, period → teacher

{('TDA356', 2, 'Databases', 'Niklas Broberg'),
 ('TDA356', 4, 'Databases', 'Rogardt Heldal')}
```

Redundancy!

Using FDs to detect anomalies

- Whenever X → A holds for a relation R, but X is not a key for R, then values of A will be redundantly repeated!

```
Courses(code, period, name, teacher)

{('TDA356', 2, 'Databases', 'Niklas Broberg'),
 ('TDA356', 4, 'Databases', 'Rogardt Heldal')}

code → name
code, period → teacher
```

Quiz: What kind of anomaly could this relational schema lead to?

Next Lecture

BCNF decomposition

3NF, 4NF