Database design

The Entity-Relationship model

Naive approach

- Not using a structured design method means it's easy to make errors.
- Learn from the mistakes of others, then you won't have to repeat them yourself!

Scheduler database

"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

First attempt

- Course codes and name, and the period the course is 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
- Weekday and hour of lectures

```
Schedules(code, name, period, numStudents,
  teacher, room, numSeats, weekday, hour)
```

Quiz: What's a key of this relation?

First attempt

```
Schedules(code, name, period, numStudents, teacher,
  room, numSeats, 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

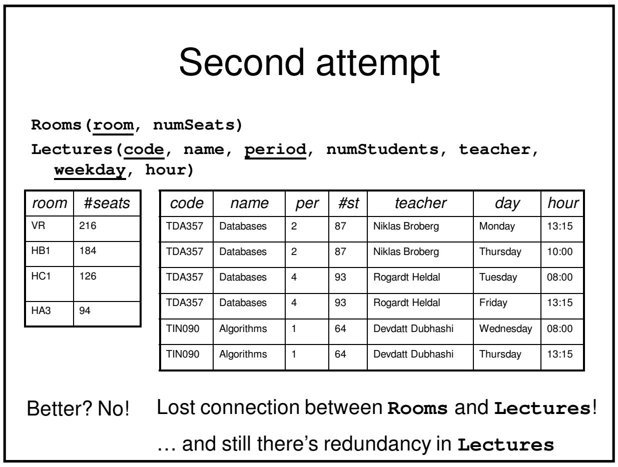
Quiz: What's wrong with this approach?

Anomalies

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	13:15
TIN090	Algorithms	1	64	Devdatt Dubhashi	**HC1**	**126**	Wednesday	08:00
TIN090	Algorithms	1	64	Devdatt Dubhashi	HA3	94	Thursday	13:15

- Redundancy – same thing stored several times
- Update anomaly – we must remember to update all tuples

- Deletion anomaly – if no course has lectures in a room, we lose track of how many seats it has



Second attempt

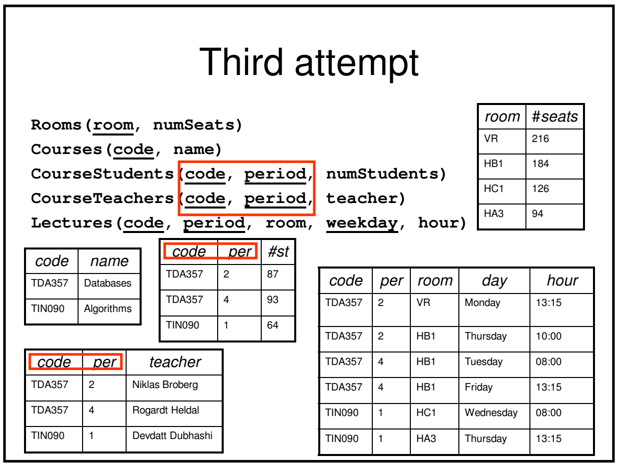
`Rooms(room, numSeats)`
`Lectures(code, name, period, numStudents, teacher,`
`  weekday, hour)`

room	#seats
VR	216
HB1	184
HC1	126
HA3	94

code	name	per	#st	teacher	day	hour
TDA357	Databases	2	87	Niklas Broberg	Monday	13:15
TDA357	Databases	2	87	Niklas Broberg	Thursday	10:00
TDA357	Databases	4	93	Rogardt Heldal	Tuesday	08:00
TDA357	Databases	4	93	Rogardt Heldal	Friday	13:15
TIN090	Algorithms	1	64	Devdatt Dubhashi	Wednesday	08:00
TIN090	Algorithms	1	64	Devdatt Dubhashi	Thursday	13:15

Better? No! Lost connection between **Rooms** and **Lectures**!

… and still there's redundancy in **Lectures**



Third attempt

`Rooms(room, numSeats)`
`Courses(code, name)`
`CourseStudents(code, period, numStudents)`
`CourseTeachers(code, period, teacher)`
`Lectures(code, period, room, weekday, hour)`

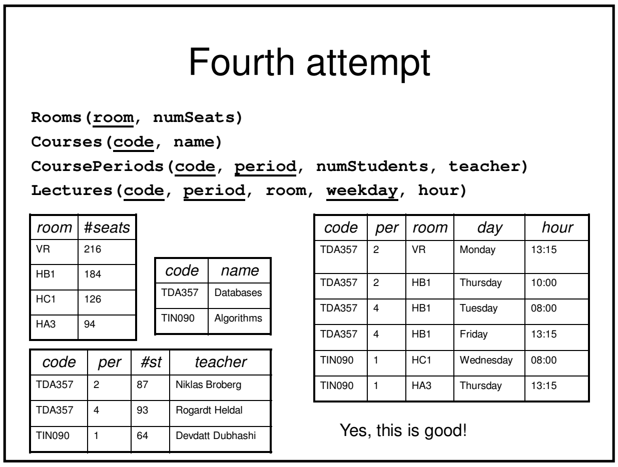
room	#seats
VR	216
HB1	184
HC1	126
HA3	94

code	name
TDA357	Databases
TIN090	Algorithms

code	per	#st
TDA357	2	87
TDA357	4	93
TIN090	1	64

code	per	teacher
TDA357	2	Niklas Broberg
TDA357	4	Rogardt Heldal
TIN090	1	Devdatt Dubhashi

code	per	room	day	hour
TDA357	2	VR	Monday	13:15
TDA357	2	HB1	Thursday	10:00
TDA357	4	HB1	Tuesday	08:00
TDA357	4	HB1	Friday	13:15
TIN090	1	HC1	Wednesday	08:00
TIN090	1	HA3	Thursday	13:15



Fourth attempt

`Rooms(room, numSeats)`
`Courses(code, name)`
`CoursePeriods(code, period, numStudents, teacher)`
`Lectures(code, period, room, weekday, hour)`

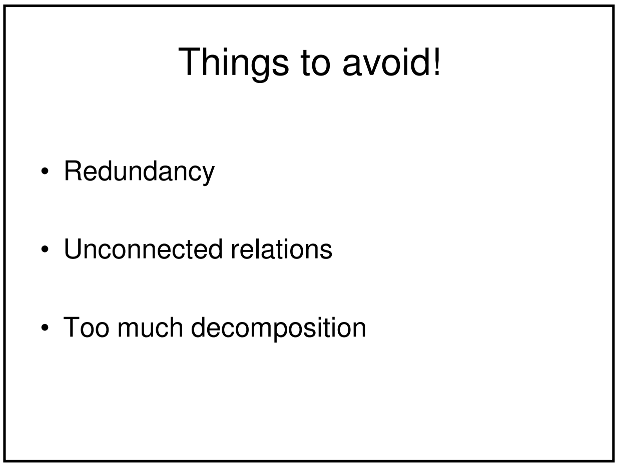
room	#seats
VR	216
HB1	184
HC1	126
HA3	94

code	name
TDA357	Databases
TIN090	Algorithms

code	per	#st	teacher
TDA357	2	87	Niklas Broberg
TDA357	4	93	Rogardt Heldal
TIN090	1	64	Devdatt Dubhashi

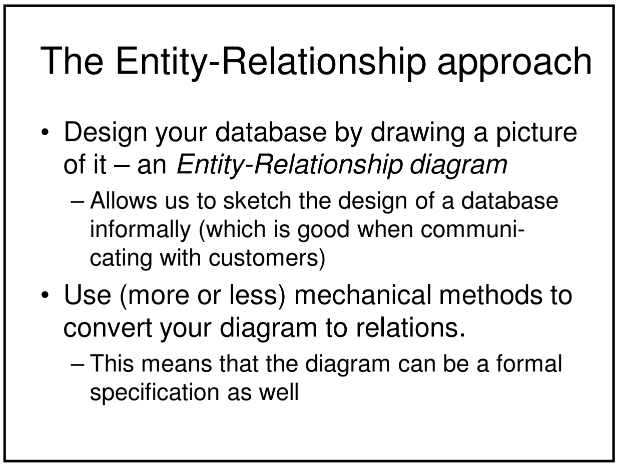
code	per	room	day	hour
TDA357	2	VR	Monday	13:15
TDA357	2	HB1	Thursday	10:00
TDA357	4	HB1	Tuesday	08:00
TDA357	4	HB1	Friday	13:15
TIN090	1	HC1	Wednesday	08:00
TIN090	1	HA3	Thursday	13:15

Yes, this is good!



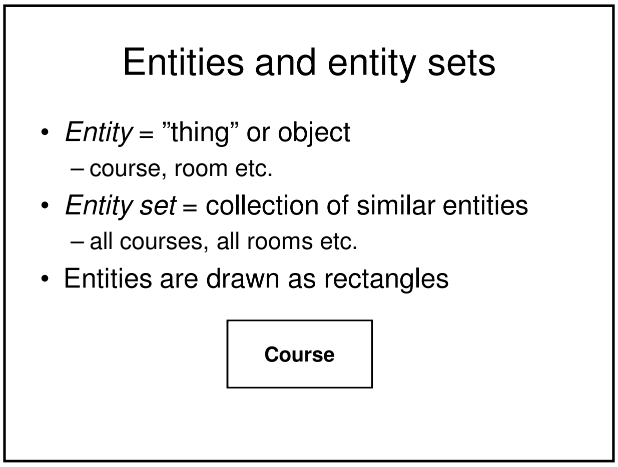
Things to avoid!

- Redundancy
- Unconnected relations
- Too much decomposition



The Entity-Relationship approach

- Design your database by drawing a picture of it – an *Entity-Relationship diagram*
  - Allows us to sketch the design of a database informally (which is good when communicating with customers)
- Use (more or less) mechanical methods to convert your diagram to relations.
  - This means that the diagram can be a formal specification as well



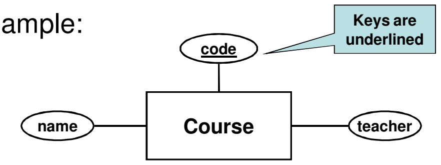
Entities and entity sets

- *Entity* = "thing" or object
  - course, room etc.
- *Entity set* = collection of similar entities
  - all courses, all rooms etc.
- Entities are drawn as rectangles

Attributes

- Entities have attributes.
- All entities in an entity set have the same attributes (though not the same values)
- Attributes are drawn as ovals connected to the entity by a line.

Example:

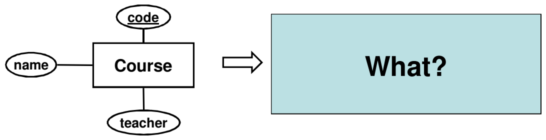


Keys are underlined

- A course has three attributes – the unique course code, a name and the name of the teacher.
- All course entities have values for these three attributes, e.g. (TDA357, Databases, Niklas Broberg).

Translation to relations

- An E-R diagram can be mechanically translated to a relational database schema.
- An entity becomes a relation, the attributes of the entity become the attributes of the relation, keys become keys.



A note on naming policies

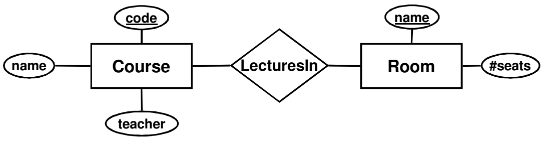
- My view: A rectangle in an E-R diagram represents an entity, hence it is put in singular (e.g. Course).
  - Fits the intuition behind attributes and relationships better.
- The book: A rectangle represents an entity set, hence it is put in plural (e.g. Courses)
  - Easier to mechanically translate to relations.

Relationships

- A *relationship* connects two (or more) entities.
- Drawn as a diamond between the related entities, connected to the entities by lines.

- Note: Relationship ≠ Relation!!

Example:

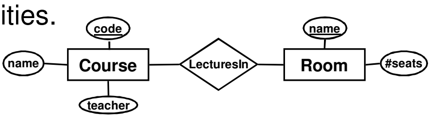


- A course has lectures in a room.
- A course is related to a room by the fact that the course has lectures in that room.

- A relationship is often named with a verb (e.g. HasLecturesIn)

Translation to relations

- A relationship between two entities is translated into a relation, where the attributes are the *keys* of the related entities.



⇨ **What?**

References

```
Courses(code, name, teacher)
Rooms(name, #seats)
LecturesIn(code, name)
```

- We must ensure that the codes used in **`LecturesIn`** matches those in **`Courses`**.
  - Introduce *references* between relations.
  - e.g. the course codes used in **`LecturesIn`** *reference* those in **`Courses`**.

```
Courses(code, name, teacher)
Rooms(name, #seats)
LecturesIn(code, name)
  code  -> Courses.code
  name -> Rooms.name
```

References

”Foreign” keys

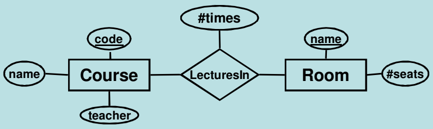
- Usually, a reference points to the key of another relation.
  - E.g. **`name`** in **`LecturesIn`** references the key **`name`** in **`Rooms`**.
  - **`name`** is said to be a *foreign key* in **`LecturesIn`**.

Relationship (non-)keys

- Relationship relations have no key attributes of their own!
  - The ”key” of a relationship relation is the combined keys of the related entities
  - Follows from the fact that entities are either related or not.
  - If you at some point think it makes sense to put a key on a relationship, it should probably be an entity instead.

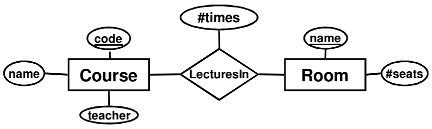
Quiz

Suppose we want to store the number of times that each course has a lecture in a certain room. How do we model this?



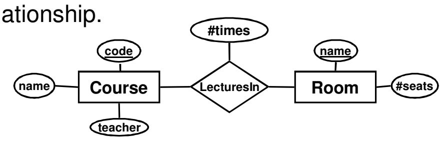
Attributes on relationships

- Relationships can also have attributes.
- Represent a property of the relationship between the entities.
  - E.g. **`#times`** is a property of the relationship between a course and a room.



Translation to relations

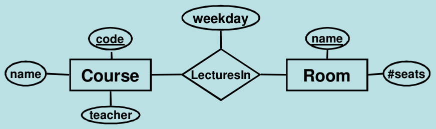
- A relationship between two entities is translated into a relation, where the attributes are the *keys* of the related entities, plus any attributes of the relationship.



⇨ **What?**

Quiz

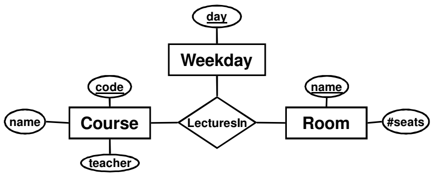
Why could we not do the same for weekday?



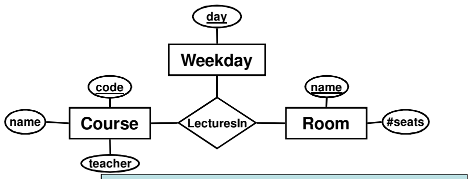
- Not a property of the relationship – a course can have lectures in a given room on several weekdays!
- A pair of entities are either related or not.

Multiway relationships

- A course has lectures in a given room on different weekdays.



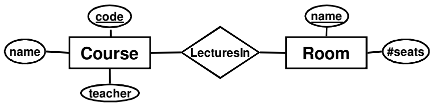
- Translating to relations:



⇨ What?

Many-to-many relationships

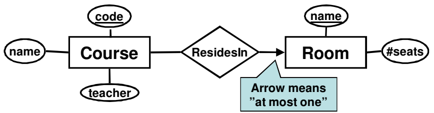
- Many-to-many (n-to-m, N-M) relationships
  - Each entity in either of the entity sets can be related to any number of entities of the other set.



  - A course can have lectures in many rooms.
  - Many courses can have lectures in the same room.

Many-to-one relationships

- Many-to-one (n-to-1, N-1) relationships
  - Each entity on the "many" side can only be related to (at most) one entity on the "one" side.

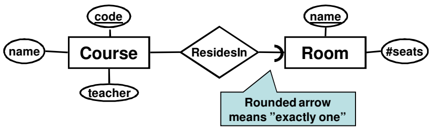


Arrow means "at most one"

  - Courses have all their lectures in the same room.
  - Many courses can share the same room.

Many-to-”exactly one”

- All entities on the ”many” side *must* be related to one entity on the ”one” side.
  - This is also known as ***total participation***

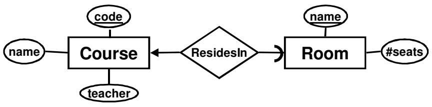


Rounded arrow means ”exactly one”

- Courses have all their lectures in some room.
- Many courses can share the same room.

One-to-one relationships

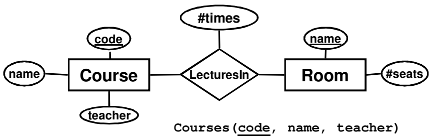
- One-to-one (1-to-1, 1-1) relationships
  - Each entity on the either side can only be related to (at most) one entity on the other side.



- Courses have all their lectures in the same room.
- Only one course in each room.
- Not all rooms have courses in them.

Translating multiplicity

- A *many-to-many* relationship between two entities is translated into a relation, where the attributes are the *keys* of the related entities, and any attributes of the relation.

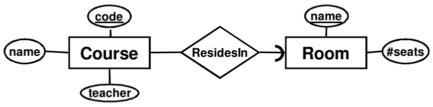


⇨

```
Courses(code, name, teacher)
Rooms(name, #seats)
LecturesIn(code, name, #times)
  code  -> Courses.code
  name -> Rooms.name
```

Translating multiplicity

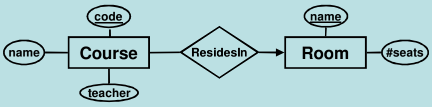
- A *N-to-”exactly one”* relationship between two entities is translated as part of the ”many”-side entity.



⇨ What?

Quiz

How do we translate an *N-to-one* (meaning ”at most one”) relationship?



```
Courses(code, name, teacher, room)
Room(name, #seats)
```

or

```
Courses(code, name, teacher)
Room(name, #seats)
ResidesIn(code, room)
```

?

Aside: the NULL symbol

- Special symbol NULL means either
  - we have no value, or
  - we don’t know the value
- Use with care!
  - Comparisons and other operations won’t work.
  - May take up unnecessary space.

Translation comparison

```
Courses(code, name, teacher, room)
Rooms(name, #seats)
```

– Will lead to NULLs for courses that have no room.

– Typically used when *not* having a room is the exception to the rule.

```
Courses(code, name, teacher)
Rooms(name, #seats)
ResidesIn(code, room)
```

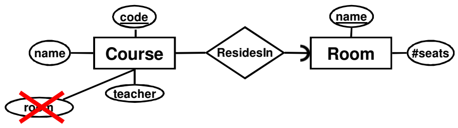
Note that ”name” is not a key here

– No NULLs anywhere.

– May lead to much duplication of the course code.

– Typically used when *having* a room is the exception to the rule.

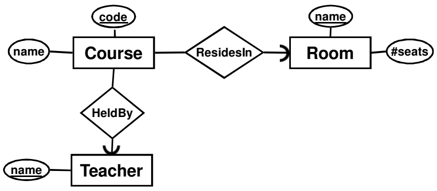
Bad E-R design



- Room is a related entity – not an attribute as well!

Attribute or related entity?

What about teacher? Isn't that an entity?



Quiz!

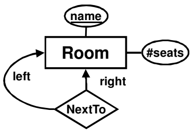
When should we model something as an entity in its own right (as opposed to an attribute of another entity)?

At least one of the following should hold:

- Consists of more than a single (key) attribute
- Used by more than one other entity
- Part of an X-to-many relation as the many side
- Generally entity-ish, is important on its own

Relationships to ”self”

- A relationship can exist between entities of the same entity set.
- Use *role* annotations for attributes.

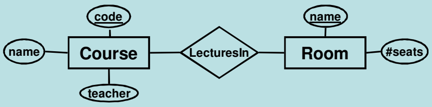


⇨

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

Quiz!

How would we add study periods to this diagram?



- Teacher can vary depending on period, but name will not.
- Rooms for lectures can vary depending on period.

Weak entities

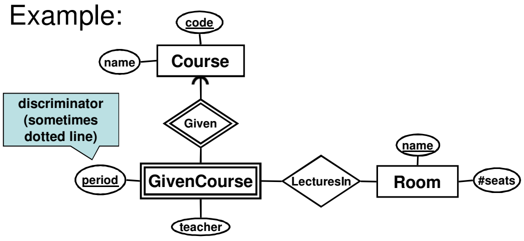
- Some entities depend on other entities.
  - A course is an entity with a code and a name.
  - A course does not have a teacher, rather it has a teacher for each time the course is given.
  - We introduce the concept of a given course, i.e. a course given in a particular period. A given course is a *weak entity*, dependent on the entity course. A given course has a teacher.

Weak entities

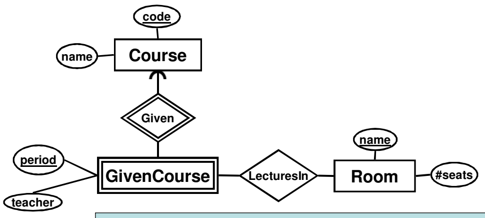
- A *weak entity* is an entity that depends on another entity for help to be ”uniquely” identified.
  - E.g. an airplane seat is identified by its number, but is not uniquely identified when we consider other aircraft. It depends on the airplane it is located in.
- Drawn as a rectangle with double borders.
- Related to its *supporting entity* by a *supporting relationship*, drawn as a diamond with double borders. This relationship is always many-to-”exactly one”.

Weak entities in E-R diagrams

Example:



Translating to relations:

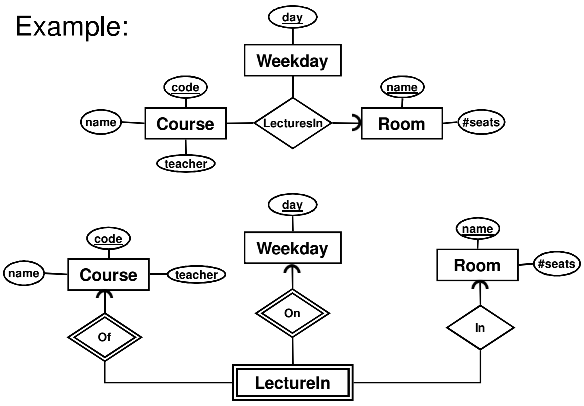


⇨ What?

Multiway relationships as WEs

- Multiway relationships can be transformed away using weak entities
  - Subtitute the relationship with a weak entity.
  - Insert supporting relationships to all entities related as ”many” by the original relationship.
  - Insert ordinary many-to-one relationships to all entities related as ”one” by the original relationship.

Example:

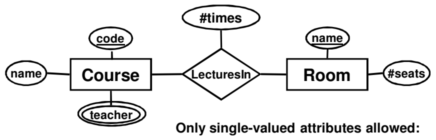


What’s the point?

- Usually, relationships work just fine, but in some special cases, you need a weak entity to express all multiplicity constraints correctly.
- A weak entity is needed when a **part** of an entity’s key is a foreign key.

Multivalued Attributes

- If an attribute can have more than one value it is called multivalued:



Only single-valued attributes allowed:

```
Courses(code, name)

Teachers(code,t_name)
  code  -> Courses.code
Rooms(name, #seats)
LecturesIn(code, name, #times)
  code  -> Courses.code
  name -> Rooms.name
```

Next lecture

More on E-R Modelling

Functional Dependencies

BCNF