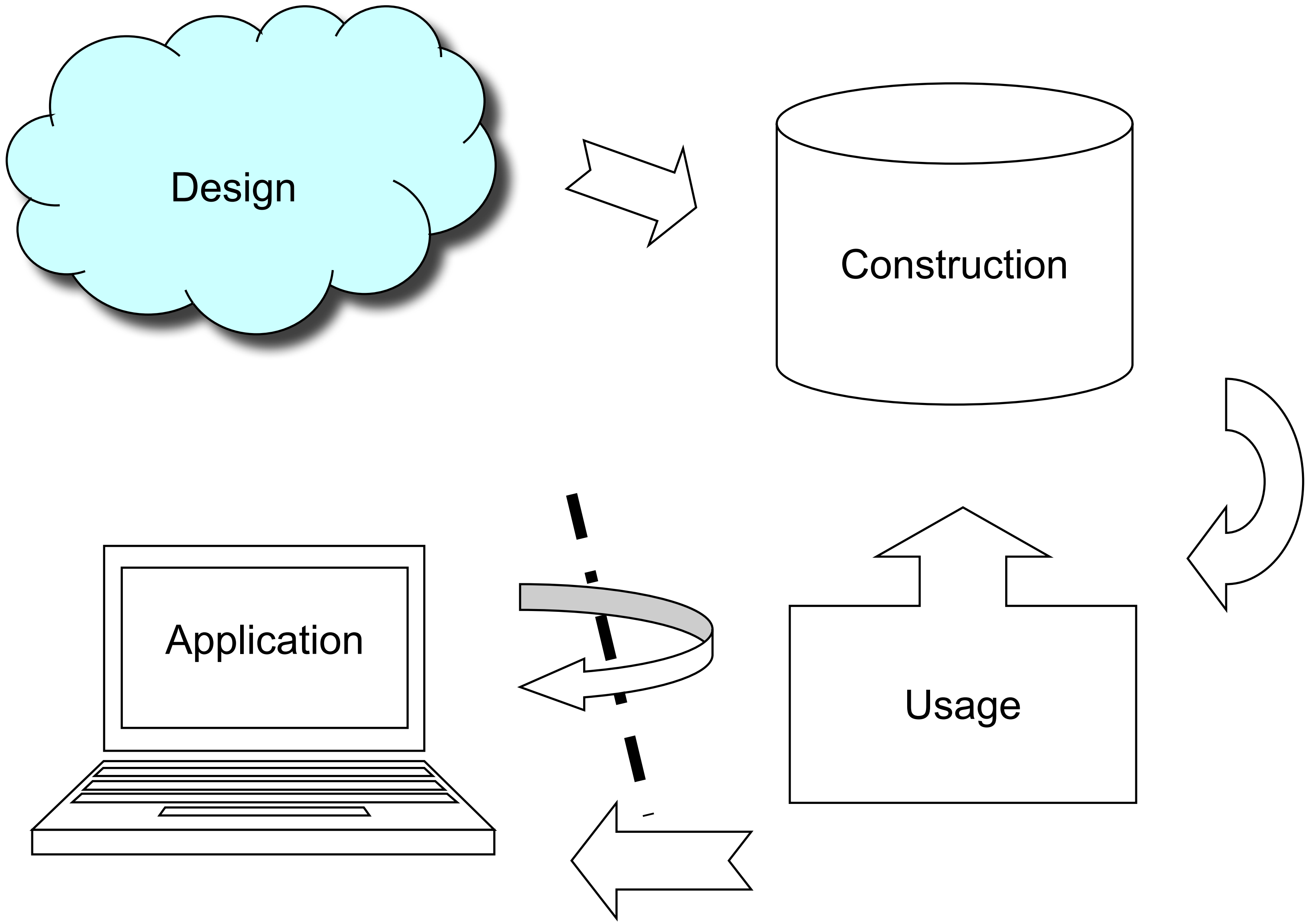
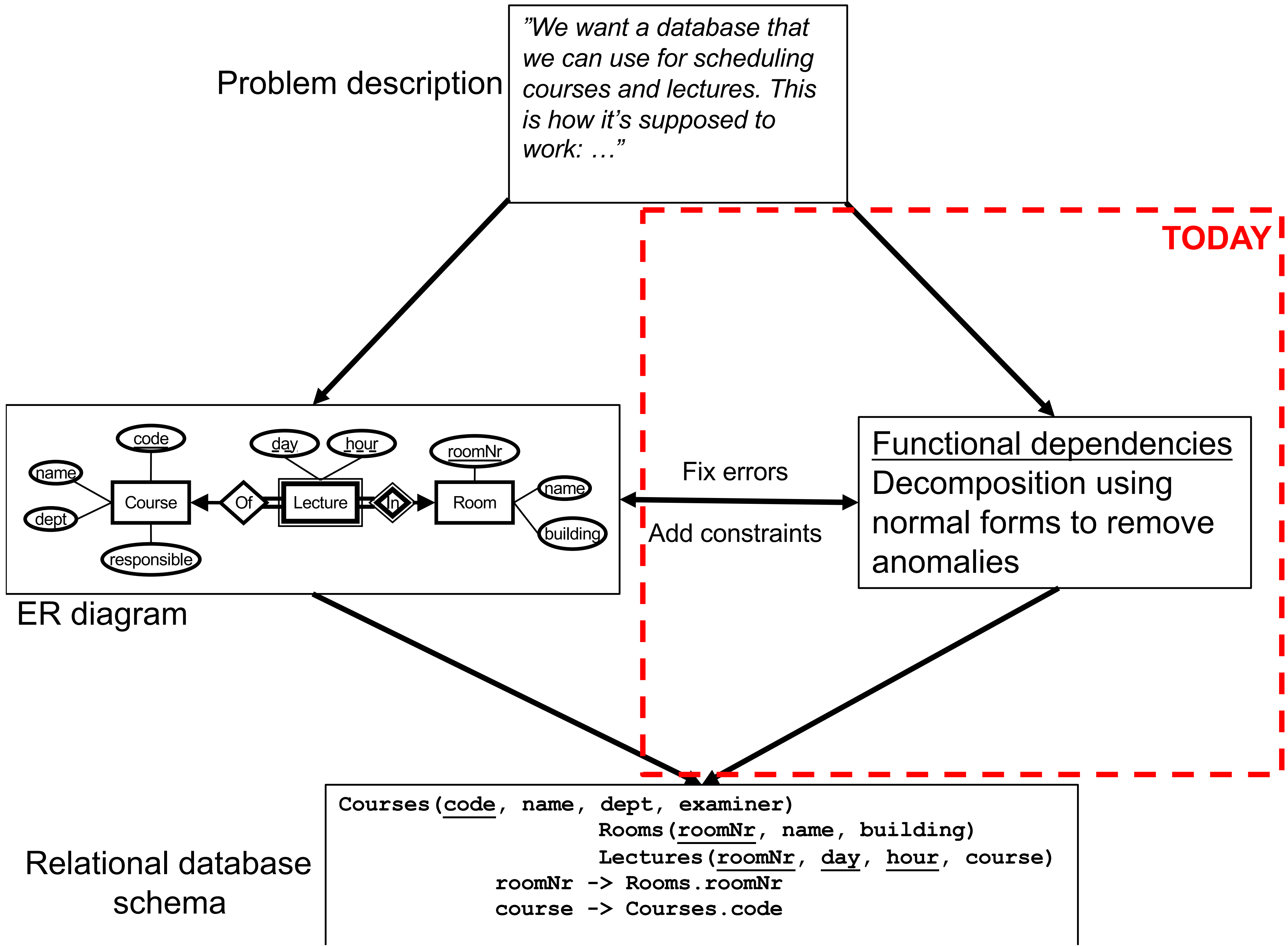
Lecture 3

Database design II

Functional Dependencies

Course Objectives




Problem description
"We want a database that we can use for scheduling courses and lectures. This is how it's supposed to work: …"
TODAY
code
name
dept
responsible
Course
Of
day
hour
Lecture
In
roomNr
Room
name
building
ER diagram
Fix errors
Add constraints
Functional dependencies
Decomposition using normal forms to remove anomalies
Relational database schema
Courses(code, name, dept, examiner)
Rooms(roomNr, name, building)
Lectures(roomNr, day, hour, course)
roomNr -> Rooms.roomNr
course -> Courses.code

Functional dependencies (FDs)

- $X \rightarrow A$
  - ”X determines A”, ”X gives A”
  - ”A depends on X”
- X and A are sets of attributes
- Examples:
  - **`code`** $\rightarrow$ **`name`**
  - **`code, period`** $\rightarrow$ **`teacher`**

Assertions on a schema

- $X \rightarrow A$ 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`** $\rightarrow$ **`teacher`**
  - If two tuples in the GivenCourses relation have the same course code and period, then they must also have the same teacher.

Assertions on a domain

- $X \rightarrow A$ is really an assertion about a *domain* D
  - Let D be the relation that is the join (along references) of all relations in the database of the domain.
    - E.g. The Scheduler domain
  - If two tuples in D agree on the values of the attributes in X, then they must also agree on the value of A.

- Example: **`code, period`** $\rightarrow$ **`teacher`**
  - If two tuples in the D relation (i.e. the domain) have the same course code and period, then they must also have the same teacher.

What are FDs really?

- Functional dependencies represent a special kind of constraints of a domain – *dependency constraints*.
- The database we design should properly capture all constraints of the domain.
- We can use FDs to verify that our design indeed captures the constraints we expect, and add more constraints to the design when needed.

What's so functional?

- $X \rightarrow A$ 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`** $\rightarrow$ **`name`**
  - imagine a deterministic 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` $\rightarrow$ **`name`**

- A **reference** exists between attributes in two <u>different</u> relations, e.g. for relation GivenCourses we have reference:

  ```
  course -> Courses.code
  ```

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

Quiz!

What are reasonable FDs for the scheduler domain?

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

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
```

Multiple attributes on R/LHS

- $X \rightarrow A,B$
  - Short for $X \rightarrow A$ and $X \rightarrow B$
  - If we have both $X \rightarrow A$ and $X \rightarrow B$, we can combine them to $X \rightarrow A,B$.
  - **`code, period`** $\rightarrow$ **`teacher, #students`**
- Multiple attributes on LHS can be crucial!
  - **`code, period`** $\rightarrow$ **`teacher`**
    - **`code`** $\not\rightarrow$ **`teacher`**
    - **`period`** $\not\rightarrow$ **`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).

Example

Schedules(<u>code</u>, name, <u>period</u>, numStudents, teacher, <u>room</u>, numSeats, <u>weekday</u>, hour)

code	*name*	*per.*	*#st*	*teacher*	*room*	*#seats*	*day*	*hour*
TDA357	Databases	2	200	Steven Van Acker	HB2	186	Tuesday	10:00
TDA357	Databases	2	200	Steven Van Acker	HB2	186	Wednesday	8:00
TDA357	Databases	3	93	Graham Kemp	HC4	216	Tuesday	10:00
TDA357	Databases	3	93	Graham Kemp	VR	228	Friday	10:00
TIN090	Algorithms	1	64	Devdatt Dubhashi	HC1	126	Wednesday	08:00
TIN090	Algorithms	1	64	Devdatt Dubhashi	HC1	126	Thursday	13:15

code, period → teacher ? Yes! This is a FD

…**but {code, period} is not a key**…

Example (decomposed)

```
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)
```

code, period → teacher ?

Quiz: Given values for a code and a period, starting from any relation where they appear, is it possible to reach more than one teacher value by following keys and references?

Answer: No, so the FD constraint is properly captured. No need to fix schema

Trivial FDs

- A FD is *trivial* if all attributes on the RHS are 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)

Inferring FDs

- In general we can find more FDs
  - course, period, weekday $\rightarrow$ room
  - room $\rightarrow$ #seats

  $\Rightarrow$ course, period, weekday $\rightarrow$ #seats

- We will need *all* FDs for doing a proper design.

Closure of attribute set X

- 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 → A.

  E.g. If `teacher` is in the closure of `code, period`

  Then `code, period → teacher`

- The closure of X is written X^+.
  - X^+ = all attributes that ”follow” from 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}

Finding all implied FDs: F^+

- F^+ is also called the closure of F

- Simple, exponential algorithm

  For each set of attributes X in a relation R:

  1. compute X^+.
  2. Add $X \rightarrow A$ to F^+ for all A in X^+ - X.
  3. However, drop $XY \rightarrow A$ whenever we discover $X \rightarrow A$.
     - Because $XY \rightarrow A$ follows from $X \rightarrow A$.

Example: Finding F^+

code, period, weekday → name	(1)
code, period, weekday → #students	(2)
code, period, weekday → teacher	(3)
code, period, weekday → hour	(5)
code, period, weekday → room	(6)
code, period, weekday → #seats	Implied: (6)+(4)

```
(1) code → name
(2) code, period → #students
(3) code, period → teacher
(4) room → #seats
(5) code, period, weekday → hour
(6) code, period, weekday → room
(7) room, period, weekday, hour → code
```

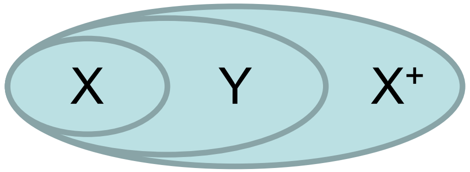
X = {code, period, weekday} (remember: must repeat for all other X too)

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

X^+ - X= {name, #students, teacher, hour, room, #seats}

Finding F^+: a simplifying trick

If $X \subseteq Y \subseteq X+$



then $Y^+ == X^+$ and no new FDs will be found

e.g.
X = `{code, period, weekday}`
Y = `{code, period, weekday, name, room}`
X^+= `{code, period, weekday, name, #students, teacher, hour, room, #seats}`

In particular, if X^+ is the set of all attributes, then the closure of all supersets of X will also be the set of all attributes.

Finding keys

- For a relation R, any subset X of attributes of R such that X^+ contains all 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

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

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.

Quiz!

What is the key of Schedules?

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

Two keys exist:

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

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).

Where do FDs come from?

- ”Keys” of entities (from ER diagram)
  - 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 (from ER diagram)
  - 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 (domain description)
  - No two courses can have lectures in the same room at the same time, e.g.
    `room, period, weekday, hour` → **`code`**

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.

How NOT to find FDs

- Do an E-R diagram, look at the entities and many-to-one relationships, pick the proper FDs.

Quiz: Why not?

- FDs should be used to find *more* constraints, and also to check that your diagram is correct. If the FDs are taken from the diagram, no more constraints will be added, and it will contain the same errors!

Example: Scheduler domain

```
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)
```

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

Quiz: Fix the schema!

Scheduler domain (fixed)

```
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
   (room, period, weekday, hour) unique
Rooms(name, #seats)
```

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

Add a key to Lectures!

CSE wants you! Student TA for LP3

- Why?
  - Students help students best
  - Also:
    - 156 SEK/h without Master degree
    - 184 SEK/h with Master degree
- Apply before November 13th 2016!
  - http://www.chalmers.se/en/departments/cse/Pages/TeachingAssistants.aspx
  - https://goo.gl/TfFTxA

Break! In part 2:

BCNF decomposition

3NF