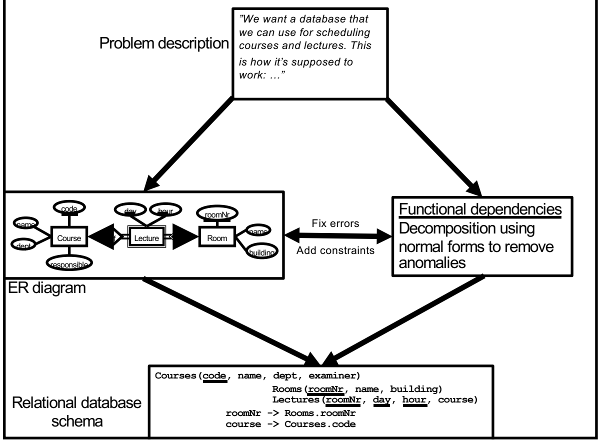
Lecture 4

Database design IV

INDs and 4NF
Design wrapup



Problem description

"We want a database that we can use for scheduling courses and lectures. This is how it's supposed to work: …"

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
```

Work flow

- **DRAW** your diagram of the domain.
- **TRANSLATE** to relations forming a schema
- **IDENTIFY** dependencies from domain!
- **RELATE** the dependencies to the schema, to find more constraints, and to validate your design.

Last time

- Functional dependencies (FDs) X → A
- X^+ = Closure of X = all derivable (from X) attributes
- F^+ = Closure of F = all implied (from F) FDs
- Superkeys, keys and primary keys
- Boyce-Codd Normal Form (BCNF):
  - The LHS (X) of every non-trivial FD (X → A) must be a superkey
- Decomposition:
  - Split up relations until normal form (e.g. BCNF) holds
  - Make sure to preserve recovery!!! No lossy joins allowed

Normal Forms?!

- Use normal forms to detect anomalies (e.g. Redundancy)
- Use decomposition to remove anomalies

- 1NF + ... = 2NF
- 2NF + ... = 3NF
- 3NF + ... = BCNF (3.5NF)
- BCNF + ... = 4NF
- 4NF + ... = 5NF
- 5NF + ... = 6NF

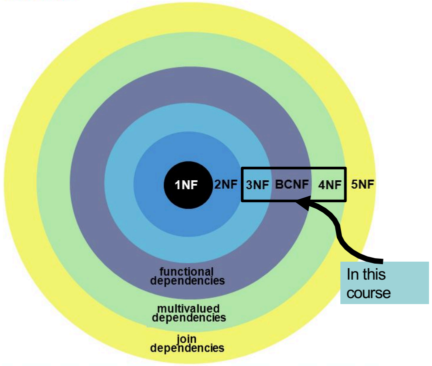
In this course

Stronger Requirements

- 6NF ⊆ 5NF ⊆ 4NF ⊆ BCNF ⊆ 3NF ⊆ 2NF ⊆ 1NF

(e.g. a database in 6NF is also in 5NF, etc.)

Normal Forms?!



BCNF Example

Decompose Courses into BCNF.

Courses(code, period, name, teacher)

code → name
code, period → teacher

Violates BCNF, so we will kick it out of the relation

{code}$^+$ = {code, name}

Courses1(code, name)

Create new relation

Courses2(code, period, teacher)
  code -> Courses1.code

Remove 'name' from old relation and add reference

No BCNF violations left, so we're done!

Tricky example of BCNF decomposition:

GivenCourses(course, period, teacher)
  course -> Courses.code

course, period → teacher
teacher → course

Violation!

Two keys:
{course, period}
{teacher, period}

Decompose:

Teaches(teacher, course)
  course -> Courses.code
GivenCourses(period, teacher)
  teacher -> Teaches.teacher

Quiz: What just went wrong?

Teaches(teacher, course)
  course -> Courses.code
GivenCourses(period, teacher)
  teacher -> Teaches.teacher

teacher	course
Niklas Broberg	TDA357
Graham Kemp	TDA357

per	teacher
2	Niklas Broberg
2	Graham Kemp

course	per	teacher
TDA357	2	Niklas Broberg
TDA357	2	Graham Kemp

course, period → teacher ??

Third Normal Form (3NF)

- 3NF is a weakening of BCNF that handles this situation.
  - An attribute is *prime* in relation R if it is a member of any key of R.
    - e.g. keys: {course, period}{teacher, period}

    Prime attributes: {course, period, teacher}

X → A is in **BCNF**
iff either:
- X → A is a trivial FD
- X is a superkey

X → A is in **3NF**
iff either:
- X → A is a trivial FD
- X is a superkey
- A-X has only prime attributes

Different algorithm for 3NF

- Given a relation R and a set of FDs F:
  - Compute the *minimal basis* of F.
    - Minimal basis means F$^+$, except remove A → C if you have A → B and B → C in F$^+$.
  - Group together FDs with the same LHS.
  - For each group, create a relation with the LHS as the key.
  - If no relation contains a key of R, add one relation containing only a key of R.

Example:

Courses(code, period, name, teacher)

code → name
code, period → teacher
teacher → code

teacher → name

Two keys:
{course, period}
{teacher, period}

Prime attributes:
{course, period, teacher}

Decompose:

Courses(code, name)
GivenCourses(course, period, teacher)
  course -> Courses.code
  teacher -> Teaches.teacher
Teaches(teacher, course)
  course -> Courses.code

GivenCourses contains a key for the original Courses relation, so we are done.

Earlier tricky example revisited:

```
GivenCourses(course, period, teacher)
  course -> Courses.code

course, period → teacher
teacher → course
```

Two keys:
`{course, period}`
`{teacher, period}`

Since all attributes are members of some key, i.e. all attributes are prime, there are no 3NF violations. Hence GivenCourses is in 3NF.

Quiz: What's the problem now then?

One 3NF solution for scheduler

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

Quiz: What's the problem now then?

Redundancy with 3NF

```
GivenCourses(course, period, teacher)
  course -> Courses.code

course, period → teacher
teacher → course
```

Two keys:
`{course, period}`
`{teacher, period}`

GivenCourses is in 3NF. But **`teacher → course`** violates BCNF, since teacher is not a key. As a result, **`course`** will be redundantly repeated!

3NF vs BCNF

- Three important properties of decomposition:
  1. Recovery (loss-less join)
  2. No redundancy
  3. Dependency preservation
- 3NF guarantees 1 and 3, but not 2.
- BCNF guarantees 1 and (almost) 2, but not 3.
  - 3 can sometimes be recovered separately through "assertions" (costly). More on this later.

Almost?

Example:

```
Courses(code, name)
  code → name
LecturesIn(code, room, teacher)
  code -> Courses.code
```

code	*room*	*teacher*
TDA357	VR	Niklas Broberg
TDA357	VR	Graham Kemp
TDA357	HC1	Niklas Broberg
TDA357	HC1	Graham Kemp

code	*name*
TDA357	Databases

These two relations are in BCNF, but there's lots of redundancy!

Let's start from the bottom…

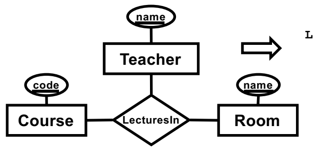
code	*room*
TDA357	HC1
TDA357	VR

code	*teacher*
TDA357	Niklas Broberg
TDA357	Graham Kemp

code	*room*	*teacher*
TDA357	VR	Niklas Broberg
TDA357	VR	Graham Kemp
TDA357	HC1	Niklas Broberg
TDA357	HC1	Graham Kemp

- No redundancy before join
- The two starting tables are what we really want to have

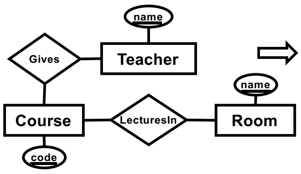
Compare with E/R



```
LecturesIn(code, teacher, room)
  code  -> Courses.code
  room -> Rooms.name
  teacher -> Teachers.name
```

Before decomposition

After decomposition



```
LecturesIn(code, room)
  code -> Courses.code
  room -> Rooms.name
Gives(code, teacher)
  code -> Courses.code
  teacher -> Teachers.name
```

BREAK

Independencies (INDs)

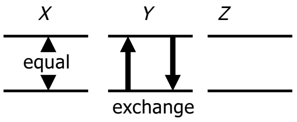
- Some attributes are not uniquely defined (as with FDs), but are still independent of the values of other attributes.
  - In our example: code does not determine room, there can be several rooms for a course. But the rooms a course uses is *independent* of the teachers on the course.
- X ↠ Y | Z states that from the point of view of X, Y and Z are independent.
  - Just X ↠ Y means that X's relationship to Y is independent of all other attributes.

(INDs are called Multivalued Dependencies (MVDs) in the book, but no need to remember that name)

Independent how?

- An IND *X* ↠ *Y* is an assertion that if two tuples of a relation agree on all the attributes of *X*, then their components in the set of attributes *Y* may be swapped, and the result will be two tuples that are also in the relation.
- If (for some X) all values of Y (for that X) can be combined with all values of Z (for that X), then (from X) Y and Z are independent.

Picture of IND *X* ↠ *Y* | *Z*



If two tuples have the same value for X, different values for Y and different values for the Z attributes, then there must also exist tuples where the values of Y are exchanged, otherwise Y and Z are not independent!

Implied tuples

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

`code → name` `code ↠ room | teacher`

If we have:

code	*name*	*room*	*teacher*
TDA357	Databases	VR	Niklas Broberg
TDA357	Databases	HC1	Graham Kemp

we must also have:

TDA357	Databases	HC1	Niklas Broberg
TDA357	Databases	VR	Graham Kemp

otherwise room and teacher would not be independent!

Compare with joining

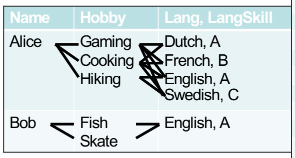
code	room
TDA357	HC1
TDA357	VR

code	teacher
TDA357	Niklas Broberg
TDA357	Graham Kemp

code	room	teacher
TDA357	VR	Niklas Broberg
TDA357	VR	Graham Kemp
TDA357	HC1	Niklas Broberg
TDA357	HC1	Graham Kemp

- Joining two independent relations yields a relation with all combinations of values!

Another example



Name	Hobby	Lang	LangSkill
Alice	Gaming	Dutch	A
Alice	Gaming	French	B
Alice	Gaming	English	A
Alice	Gaming	Swedish	C
Alice	Cooking	Dutch	A
Alice	Cooking	French	B
Alice	Cooking	English	A
Alice	Cooking	Swedish	C
Alice	Hiking	Dutch	A
Alice	Hiking	French	B
Alice	Hiking	English	A
Alice	Hiking	Swedish	C
Bob	Fish	English	A
Bob	Skate	English	A

name ↠ hobby | lang, langskill

- For a given name, hobby and {language, langskill} are independent
- For a given name, all combinations of hobby and {lang, langskill} must be able to exist

FDs are INDs

- Every FD is an IND (but of course not the other way around). Compare the following cases:
  - If X ↠ Y holds for a relation, then all possible values of Y for that X must be combined with all possible combinations of values for ”all other attributes” for that X.
  - If X → A, there is only one possible value of A for that X, and it will appear in all tuples where X appears. Thus it will be combined with all combinations of values that exist for that X for the rest of the attributes.

Example:

code	name	room	teacher
TDA357	Databases	VR	Niklas Broberg
TDA357	Databases	VR	Graham Kemp
TDA357	Databases	HC1	Niklas Broberg
TDA357	Databases	HC1	Graham Kemp

`code ↠ name`	There are four possible combinations of values for the attributes **`room`** and **`teacher`**, and the only possible value for the **`name`** attribute, ”Databases”, appears in combination with all of them.
`code ↠ teacher`	There are two possible combinations of values for the attributes **`name`** and **`room`**, and all possible values of the attribute **`teacher`** appear with both of these combinations.
`code ↠ room`	There are two possible combinations of values for the attributes **`name`** and **`teacher`**, and all possible values of the attribute **`room`** appear with both of these combinations.

IND rules ≠ FD rules

- Complementation
  - If X ↠ Y, and Z is all other attributes, then X ↠ Z.
- Splitting doesn't hold!!
  - **`code ↠ room, #seats`**
    - **`code ↠ room`** does not hold, since **`room`** and **`#seats`** are not independent!
- None of the other rules for FDs hold either.

Example:

code	name	room	#seats	teacher
TDA357	Databases	VR	216	Niklas Broberg
TDA357	Databases	VR	216	Graham Kemp
TDA357	Databases	HC1	126	Niklas Broberg
TDA357	Databases	HC1	126	Graham Kemp

`code ↠ room, #seats`

We cannot freely swap values in the #seats and room columns, so neither

`code ↠ room`

or

`code ↠ #seats`

holds.

Fourth Normal Form (4NF)

- The redundancy that comes from IND's is not removable by putting the database schema in BCNF.
- There is a stronger normal form, called 4NF, that (intuitively) treats IND's as FD's when it comes to decomposition, but not when determining keys of the relation.

Fourth Normal Form

- 4NF is a strengthening of BCNF to handle redundancy that comes from independence.
  - An IND X ↠ Y is trivial for R if
    - Y is a subset of X
    - X and Y together = R
  - Non-trivial X → A violates BCNF for a relation R if X is not a superkey.
  - Non-trivial X ↠ Y violates 4NF for a relation R if X is not a superkey.
    - Note that what is a superkey or not is still determined by FDs only.

BCNF Versus 4NF

- Remember that every FD X → Y is also a IND, X ↠Y.
- Thus, if R is in 4NF, it is certainly in BCNF.
  - Because any BCNF violation is a 4NF violation.
- But R could be in BCNF and not 4NF, because IND's are "invisible" to BCNF.

INDs for validation

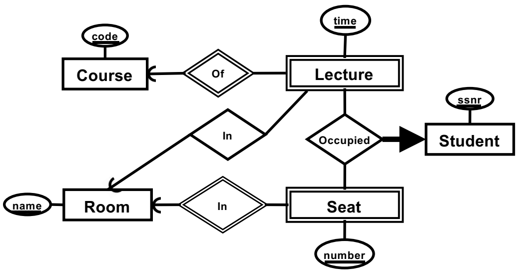
- Remember that FDs can:
  - Allow you to validate your schema.
  - Find "extra" constraints that the basic structure doesn't capture.
- INDs ONLY validate your schema.
  - No extra dependencies to be found.
  - If your E-R diagram and translation are correct, INDs don't matter.

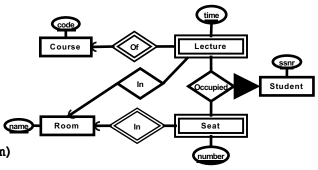
Example

```
R(code, name, period, room, seats, teacher)
  code → name
  code, period → room, teacher
  room → seats
  code, period ↠ room, seats
  code, period ↠ teacher
```

(on blackboard)

Example:
E-R does not imply BCNF





```
Students(ssnr)
Courses(code)
Rooms(name)
Lectures(course,time,room)
  course -> Courses.code
  room   -> Rooms.name
Seats(room,number)
  room   -> Rooms.name
Occupied(course,time,room,number,student)
  (course,time) -> Lectures.(course,time)
  (room,number) -> Seats.(room,number)
  student       -> Students.ssnr
```

Redundancy!

Quiz: What just went wrong?

Fix attempt #1

```
Students(ssnr)
Courses(code)
Rooms(name)
Lectures(course,time,room)
  course -> Courses.code
  room   -> Rooms.name
Seats(room,number)
  room   -> Rooms.name
Occupied(course,time,number,student)
  (course,time)  -> Lectures.(course,time)
  student        -> Students.ssnr

   (room,number)  -> Seats.(room,number)  ??
```

We broke the reference! Now we could (in theory) book seats that don't exist in the room where the lecture is given!

Fix attempt #2

```
Students(ssnr)
Courses(code)
Rooms(name)
Lectures(course,time,room)
  course -> Courses.code
  room   -> Rooms.name
Seats(room,number)
  room   -> Rooms.name
Occupied(course,time,number,room,student)
  (course,time) -> Lectures.(course,time)
  (room,number) -> Seats.(room,number)
  student       -> Students.ssnr
```

Same? We can't say!

No longer part of key!

No guarantee that the room where the seat is booked is the same room that the lecture is in!

… and redundancy (3NF solution)

Fix attempt #3

```
Students(ssnr)
Courses(code)
Rooms(name)
Lectures(course,time,room)
  course -> Courses.code
  room   -> Rooms.name
Seats(room,number)
  room   -> Rooms.name
Occupied(course,time,number,room,student)
  (course,time,room) ->
          Lectures.(course,time,room)
  (room,number)  -> Seats.(room,number)
  student        -> Students.ssnr
```

Same!

Still redundancy though (3NF solution). Possibly the best we can do though.

Next time, Lecture 5

Database Construction –

SQL Data Definition Language