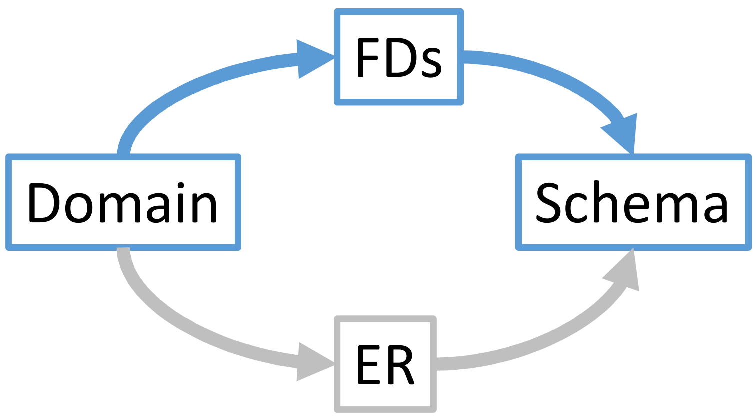
# TDA357/DIT621 – Databases

Lecture 6 – Design using Functional Dependencies and normal forms

Jonas Duregård

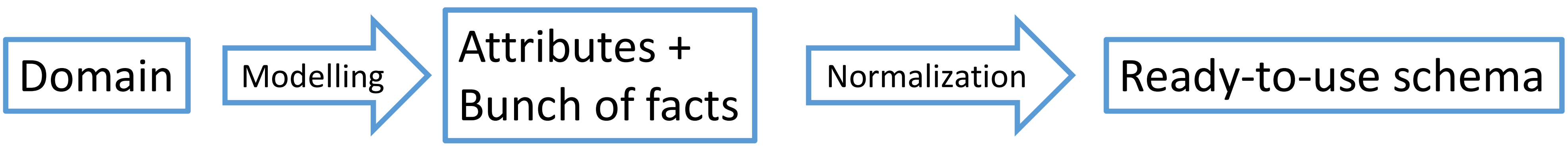
# Another high level design approach

- This week we will look at functional dependencies (FDs) and normal forms
- This is an alternative (and to some degree complementary) approach to ER that we studied last week
    - We start in a domain description and end in a database schema
- A single lecture, and Friday exercises as usual



# Normalisation in a nutshell

- Extract a bunch of formal statements from the domain description
- Compute a normalised database schema from the formal statements




- Highly systematic, almost mechanical process
- By a carefully constructed normalization algorithm, the normal form the schema ends up in will satisfy some important properties

# Functional Dependencies (FDs)

- A functional dependency is written as <set of attributes> → <attribute>
- Example: room time→course

Do not confuse with references!

- Pronounced "room and time determines course"
- It is a statement that can be true or false
- A few ways of understanding the meaning of the statement above:
  - If we know room and a time, we can uniquely determine course
  - There can be at most one course value for each (room,time)-pair
  - There exists a partial function *f* that takes a room and a time and yields a course
- In a domain it might have said something like "courses can book rooms at any free times"

# Three ways we can use functional dependencies

- Check if they hold for a specific data set
- Check if a design ensures they hold for all data sets
- Express desired properties of a design

- I will explain each of these in turn

# Functional dependency as a property of data

- One way of formally defining functional dependency $x_1\ x_2\ \ldots \rightarrow y$ :

  For $R(y\ x_1\ x_2\ \ldots)$, if two rows agree on $x_1$, $x_2$ … they must also agree on y
- In other words, there can not exist two rows where the left hand side attributes are the same, but the right hand side attribute differs
  - "$X \rightarrow y$ = rows that agree on X must agree on y"

Two rows "agreeing on x" just means the x-column(s) have the same value

- Which FDs hold for this data?

**Table: Bookings**

| courseCode | name | day | timeslot | room | seats |
|---|---|---|---|---|---|
| TDA357 | Databases | Tuesday | 0 | GD | 236 |
| TDA357 | Databases | Tuesday | 1 | GD | 236 |
| ERE033 | Reglerteknik | Tuesday | 0 | HB4 | 224 |
| ERE033 | Reglerteknik | Friday | 0 | GD | 236 |

courseCode → name?

- Yes! (TDA357 maps to Databases, ERE033 to Reglerteknik)

day → timeslot?

- No! (Tuesday maps to both 0 and 1)

LHS = Left hand side (of arrow)
RHS = Right hand side

day timeslot room → courseCode?

- Yes! (There are no rows where all three LHS columns match)

seats → room?

- Yes! 236 for GD, 224 for HB4,
- This might not be intentional given what we know of the domain…

# FDs as a properties of designs

- Knowing what it means for an FD to hold for a data set, we can determine if a design (schema) guarantees that it holds for all valid data sets
- Example: Does the schema below guarantee that ...

code → cname?

- Yes! (by primary key constraint in courses)

cname → code?

- No! (Counterexample: any two courses with the same name)

code → email

- Yes! (teacher is just another name for email)

code → tname

- Yes! (by primary key + reference)

Teachers(<u>email</u>, tname)
Courses(<u>code</u>, cname, teacher)
teacher -> Teachers.email

***Bookings(courseCode, name, day, timeslot, room, seats)***
***(day, timeslot, room) UNIQUE***

- Does the schema above guarantee …
- day timeslot room → courseCode
  - Yes (through UNIQUE constraint)
- day timeslot room coursCode → seats
  - Yes (through primary key and/or UNIQUE)
- room → seats
  - No ☹
- courseCode → name
  - No ☹

Counterexample of room → seats and courseCode → name

| **courseCode** | **name** | **day** | **timeslot** | **room** | **seats** |
|---|---|---|---|---|---|
| CC1 | N1 | Tuesday | 0 | R1 | 0 |
| CC1 | N2 | Tuesday | 1 | R1 | 1 |

Different timeslot, so no key violation

# FDs as intention for designs

- Since we can verify that an FD holds for a schema, we can also use them to specify desired properties of our schema
- This is what makes FDs a design tool
- A sentence like "every course has a teacher" can be modelled as the FD course → teacher (or whatever attributes we use)
  - If this FD does not hold for our design, maybe the design is bad?

# Formal properties of FDs

Warning: Things may get slightly mathsy from this point

- FDs have lots of interesting mathematical properties
- I will explain some of the more useful ones:
  - Transitivity
  - Augmentation
  - Reflexivity

- These three are commonly referred to as Armstrongs axioms and they can be formulated in a few different but equivalent ways*

*But the way I formulate them is -of course- the best way

# Side note: Single/Multiple FDs

Notation: I use lowercase x/y/z for single attributes and uppercase X/Y/Z for attribute sets

- It is common to have multiple attributes on the right hand side of FDs
  x y z → a b c
- This means exactly the same as these three FDs:
  x y z → a
  x y z → b
  x y z → c
- I find it most useful to think of the first as a convenient way of writing multiple FDs, rather than thinking of it as single FD with multiple attributes
- It is <u>not</u> the same with the left hand side! x y → a does not mean x → a!

Recall: $X \rightarrow y$ = rows that agree on X must agree on y

# Transitivity of functional dependencies

- Functional dependency is a transitive relation
  - This means that if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

- Note that Y is an attribute set here, so $X \rightarrow Y$ may be multiple FDs with the same LHS

- Proof sketch: Look at any rows that agree on X. Since $X \rightarrow Y$, they must also agree on Y, and since $Y \rightarrow Z$ they must further agree on Z.
  Thus $X \rightarrow Z$.

Recall: $X \rightarrow y$ = rows that agree on X must agree on y

# Augmentation

- If $x_1\ x_2 \ldots \rightarrow y$, then for all z: $z\ x_1\ x_2 \ldots \rightarrow y$
- Intuitively: You can add any attributes you want to the LHS of a valid FD and still get a valid FD
  - Think: "knowing an extra attribute never prevent us from finding y"

- Proof sketch: Since all rows that agree on the xs must agree on y, then particularly all rows that agree on z as well as the xs must do so.

Recall: X → y = rows that agree on X must agree on y

# Reflexivity and trivial FDs

- For all x:  x → x
  (x determines itself)

- By augmentation, X → y whenever y ∈ X
  - Example: a b c → b
  - We call these depencies trivial
  - Rule of thumb: Ignore trivial dependencies

- Proof sketch: Any values that agree on x will agree on x ☺

# Example: Deriving functional dependencies

- For any attributes x, y, z, w, and q:
  $x \rightarrow y$, $z \rightarrow w$, and $y\ w \rightarrow q$ implies $x\ z \rightarrow q$
- Proof:
  $x \rightarrow y$ implies $x\ z \rightarrow y$ (by augmentation)
  $z \rightarrow w$ implies $x\ z \rightarrow w$ (by augmentation)
  $x\ z \rightarrow y\ w$ and $y\ w \rightarrow q$ implies $x\ z \rightarrow q$ (by transitivity)

$$\boxed{\frac{\begin{array}{r} x \rightarrow y \\ z \rightarrow w \\ y\ w \rightarrow q \end{array}}{x\ z \rightarrow q}}$$

- Note that in the third step we merge $x\ z \rightarrow y$ and $x\ z \rightarrow w$ into $x\ z \rightarrow y\ w$
  (See slide on Single/Multiple FDs)

# Minimal basis

- The minimal basis $F^-$ of a set of functional dependencies F is a set equivalent to F but with the following conditions:
  - F- has no trivial dependenices
  - No dependency in $F^-$ follow from other dependencies in $F^-$ through transitivity or augmentation

- Used for a lot of algorithms and to express a set of FDs in a compact form

# Minimal basis

- Suppose we are given this set of FDs (5 total), what is a minimal basis?
  a → b
  b → c
  a d → b c d

- a d → d is removed because it is trivial
- a d → b is removed because it is implied by a → b (augmentation)
- a d → c is removed because it is implied by a → b and b → c (transitivity and augmentation)
- Final set: a → b, b → c

# Transitive closure

- The transitive closure $X^+$ of a set of attributes X, is the set of attributes that can be functionally determined by X
- In other words $X^+$ = All attributes y such that $X \rightarrow y$
  - Includes ALL derived functional dependencies
  - Includes trivial dependencies
  - $X^+$ is closed in the sense that any FD from attributes in $X^+$ lead back to $X^+$
- Can be computed by a simple algorithm from any set of FDs:
  - Start with $X^+ = X$ (an under-approximation)
  - Repeat until done: For any FD $Y \rightarrow z$ such that $Y \subseteq X^+$, add z to $X^+$

# Transitive closure, example

$x \rightarrow y$
$y\ w \rightarrow q$
$z \rightarrow w$
$q \rightarrow x$
$r \rightarrow s$

- Given these FDs, compute the closure $\{x,z\}^+$
- Initially we know $\{x,z\} \subseteq \{x,z\}^+$ (from trivial FDs)
- Add y because $x \rightarrow y$ and $\{x\} \subseteq \{x,z\}^+$ $\qquad \{x,z,y\} \subseteq \{x,z\}^+$
- Add w because $z \rightarrow w$ and $\{x\} \subseteq \{x,z\}^+$ $\qquad \{x,z,y,w\} \subseteq \{x,z\}^+$
- Add q because $y\ w \rightarrow q$ and $\{y,w\} \subseteq \{x,z\}^+$ $\qquad \{x,z,y,w,q\} \subseteq \{x,z\}^+$
- No more FDs add attributes, so $\{x,z\}^+ = \{x,z,y,w,q\}$ is our result
- This proves all these non-trivial FDs:

  $x\ z \rightarrow y \qquad x\ z \rightarrow w \qquad x\ z \rightarrow q$

# Keys and superkeys

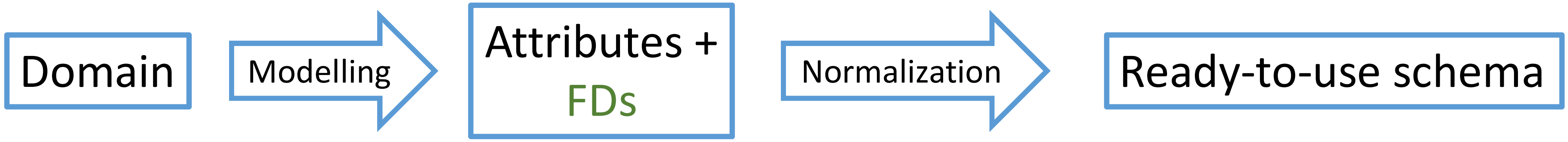
- We can define the property of being a key of a relation using FDs
- Intuitively: A set of attributes is a *superkey* if it determines all other attributes
- Formally: The attribute set X is a *superkey* of R if $X^+$ contains all attributes of R

- X is a (minimal) key if removing any attribute from X makes it a non-superkey
  - Saying only "key" usually means minimal key
  - Each superkey is a superset of at least one minimal key
  - Each key is a superkey (but not the other way around)
  - Adding any attribute to a superkey makes a new superkey

# Summary so far

- An FD X → y means any rows that agree on X also agree on Y
- We can extend a set of FDs with additional implied FDs using transitivity, augmentation, and reflexivity
- Conversely, we can reduce a set of FDs to a minimal basis by removing all implied FDs
- The closure $X^+$ is the set of all attributes that can be determined by X
- A superkey is a set of attributes that determine all other, keys are minimal superkeys
- To find a key: Start with all attributes (a superkey) and remove attributes until it is a key – finding all keys is more work though

# Normal forms and normalization



# Normal forms and normalization

- Normal form is a very important concept in database design
- Identify all the attributes in the domain and place them in one big relation D(x, y, z, …), collect FDS, then normalize D to get your design
- Normalizing is a recursive procedure, to normalize relation R:
  - Check if R is already a normal form, if it is we are done
  - Otherwise decompose R into relations $R_1$ and $R_2$ and normalize both
- Note: A normal form is not the same as a canonical form, there may be multiple normal forms derived from the same initial domain

# BCNF, the Boyce-Codd Normal Form

Arguably the most well known normal form

# BCNF Normalisation algorithm

To normalize relation R:

This FD is referred to as a BCNF-violation

Find a non-trivial FD $X \rightarrow y$ such that $X^+ \neq R$ (X is not a superkey)

If there is no such FD you are done

Otherwise decompose R into $R_1(X^+)$ and $R_2(X \cup (R - X^+))$ and normalize them

Note: R is replaced by $R_1$ and $R_2$ (so R is not present in the final schema)

# Example

1. Find violation
2. Decompose
3. Repeat

courseCode → name
room → seats
day timeslot courseCode → room
day timeslot room → courseCode

- Normalise this relation using the FDs above:

**~~R~~(courseCode, name, day, timeslot, room, seats)**

$R_1(X^+)$

$R_2(X \cup (R - X^+))$

Decompose on courseCode → name
courseCode$^+$ = {courseCode, name}

**$R_1$(courseCode, name)** **~~$R_2$~~(courseCode, day, timeslot, room, seats)**

$R_{21}(X^+)$

$R_{22}(X \cup (R - X^+))$

Decompose on room → seats
room$^+$ = {room, seats}

**$R_{21}$(room, seats)** **$R_{22}$(courseCode, day, timeslot, room)**

All of $R_1$, $R_{21}$, and $R_{22}$ are now BCNF!

# Wait, why not split on day timeslot course → room?

$R_{22}$(courseCode, day, timeslot, room)

day timeslot courseCode → room

day timeslot room → courseCode

Recall: Find a non-trivial FD X → y such that $X^+ \neq R$ (X is no superkey)

{day, timeslot, courseCode}$^+$ = { day, timeslot, courseCode, room} = $R_{22}$

{day, timeslot, room}$^+$ = { day, timeslot, room, courseCode} = $R_{22}$

Both {day, timeslot, courseCode} and {day, timeslot, room} are keys!

courseCode → name
room → seats
day timeslot courseCode → room
day timeslot room → courseCode

# What about keys?

- Keys can be determined using FDs (and closures) after decomposing
- Much of it is already done as part of the algorithm (we found two keys for $R_{22}$ for instance)

**$R_1$(<u>courseCode</u>, name)**

**$R_{21}$(<u>room</u>, seats)**

**$R_{22}$(<u>courseCode</u>, <u>day</u>, <u>timeslot</u>, room)**
**(day, timeslot, room) UNIQUE**

Multiple keys: Use one as primary key, the other(s) UNIQUE

# What about references?

- In this case it's fairly easy to see that these are sensible references:

```
R1(courseCode, name)
R21(room, seats)
R22(courseCode, day, timeslot, room)
     (day, timeslot, room) UNIQUE
     courseCode -> R1.courseCode
     room -> R21.room
```

- General pattern: When decomposing R, add a reference X -> $R_1$.X to $R_2$
- This will not always work, particularly if $R_1$ or $R_2$ is later decomposed ☹

# Decomposition of data

No redundancy, no anomalies ☺

**Table: Bookings**

| courseCode | name | day | timeslot | room | seats |
|---|---|---|---|---|---|
| TDA357 | Databases | Tuesday | 0 | GD | 236 |
| TDA357 | Databases | Tuesday | 1 | GD | 236 |
| ERE033 | Reglerteknik | Tuesday | 0 | HB4 | 224 |
| ERE033 | Reglerteknik | Friday | 0 | GD | 236 |

**Table: $R_1$ (Courses)**

| courseCode | name |
|---|---|
| TDA357 | Databases |
| ERE033 | Reglerteknik |

**Table: $R_{22}$ (Bookings)**

| courseCode | day | timeslot | room |
|---|---|---|---|
| TDA357 | Tuesday | 0 | GD |
| TDA357 | Tuesday | 1 | GD |
| ERE033 | Tuesday | 0 | HB4 |
| ERE033 | Friday | 0 | GD |

**Table $R_{21}$ (Rooms)**

| room | seats |
|---|---|
| HB4 | 224 |
| GD | 236 |

# Lossless join

- Note that if we join along the references, we get the original table

**Table: $R_1$ (Courses)**

| **courseCode** | **name** |
|---|---|
| TDA357 | Databases |
| ERE033 | Reglerteknik |

**Table: $R_{22}$ (Bookings)**

| **courseCode** | **day** | **timeslot** | **room** |
|---|---|---|---|
| TDA357 | Tuesday | 0 | GD |
| TDA357 | Tuesday | 1 | GD |
| ERE033 | Tuesday | 0 | HB4 |
| ERE033 | Friday | 0 | GD |

**Table $R_{21}$ (Rooms)**

| **room** | **seats** |
|---|---|
| HB4 | 224 |
| GD | 236 |

Joins ON (courseCode) and ON (room)

**Query: $R_1$ NATURAL JOIN $R_{22}$ NATURAL JOIN $R_{21}$**

| **courseCode** | **name** | **day** | **timeslot** | **room** | **seats** |
|---|---|---|---|---|---|
| TDA357 | Databases | Tuesday | 0 | GD | 236 |
| TDA357 | Databases | Tuesday | 1 | GD | 236 |
| ERE033 | Reglerteknik | Tuesday | 0 | HB4 | 224 |
| ERE033 | Reglerteknik | Friday | 0 | GD | 236 |

- Means we did not loose any data in the decomposition

# Finding all FDs

- Consider this simple situation with four attributes R(x,y,z,w) and two functional dependencies: x → z and y z → w
- When normalizing R it may be important to know that there is another FD that can be derived from these: x y → w
- In principle, you should consider all non-trivial derived FDs but sometimes this a large set and it is easy to miss FDs
  - Essentially you have to consider every LHS and compute closures
  - There are some clever tricks you can use, but we will not have time for those today

# A flaw of BCNF

- Same example as before: R(x,y,z,w) where x → z and y z → w
- If we decompose on x → z ($\{x\}^+ = \{x,z\}$) we get
  - $R_1(\underline{x},z)$ {x} is the only key
  - $R_2(\underline{x},\underline{y},w)$ {x,y} is the only key

Because x y → w

- Both of these relations are in BCNF w.r.t. the given FDs
- But now y z → w is not guaranteed by the schema ☹
- There is a weaker normal form called third normal form (3NF) that does not have this problem, but it has other problems instead…
  - There is no "silver bullet" for design work

# Yet another issue with BCNF

- This relation has no non-trivial functional dependencies, so is in BCNF:

**Table: Courses**

| course | book | author | teacher |
|---|---|---|---|
| Databases | DTCB | Ullman | Jonas |
| Databases | DTCB | Ullman | Aarne |
| Reglerteknik | RTB 1 | Author1 | Teacher3 |
| Reglerteknik | RTB 2 | Author2 | Teacher3 |

Deletion anomaly: Deleting all course books also deletes all teachers

Update anomaly: Changing some value can cause inconsistencies

- The domain said something like "each course has a number of teachers and a number of books with one or more authors" (no FDs at all!)
- The data above says Databases has one book and two teachers, and Reglerteknik has two books and one teacher
- Clearly there is redundancy here, and potential for anomalies

# Looks like we need another normal form!

- This one is called the fourth normal form (4NF)
- Since the problematic table had no FDs at all, this form will need some additional source of facts
- We call these facts multivalued dependencies (MVDs)*
- We write $x_1\ x_2\ x_3\ \ldots \twoheadrightarrow y_1\ y_2\ y_3\ \ldots$
  - Note that both sides are sets of values and we can <u>not</u> split the RHS

*The term multivalued dependency is really quite unfortunate, but it is what it is

# Multivalued dependencies, informally

- For our example, we would say course ↠ teacher
- This means that if we fix a course value, the teacher value is independent from all other values (author and book)

**Table: Courses**

| course | book | author | teacher |
|---|---|---|---|
| Databases | DTCB | Ullman | Jonas |
| Databases | DTCB | Ullman | Aarne |
| Reglerteknik | RTB 1 | Author1 | Teacher3 |
| Reglerteknik | RTB 2 | Author2 | Teacher3 |

- This is exactly the same as saying course ↠ book author

# Multivalued dependencies, formally

- The claim that X↠Y holds for relation R means:

For every pair of rows row t and u in R that agree on X we can find a row v s.t:

v agrees with both t and u on X

v agrees with t on Y

v agrees with u on R – X – Y (all attributes not in the MVD)

- Example: course ↠ teacher

If we remove any row, the MVD won't hold

**Table: Courses**

| course | book | author | teacher |
|---|---|---|---|
| Databases | DTCB | Ullman | Jonas |
| Databases | DTCB | Ullman | Aarne |
| Databases | DTCB | Widom | Jonas |
| Databases | DTCB | Widom | Aarne |

Row t

Row u

Row v:
v.(book,author)=u.(book,author)
v.teacher = t.teacher

# Verifying MVDs on data is hard

- To check if an FD holds: Just group values up by the LHS and check that all rows in each group have the same value for the RHS
- To check if an MVD holds: Check every individual pair of values with identical LHS and search for a row with correct values
- I find a more intuitive way of thinking is this: For $X \twoheadrightarrow Y$, every X needs to have every possible combination of Y and other attributes (R–X–Y)
  - Essentially the rows for a given X must be a cartesian product!
  - If teacher Jonas occurs with one book/autor, it must occur with all book/author combinations for that course
  - This is what makes (book,author) independent from teacher

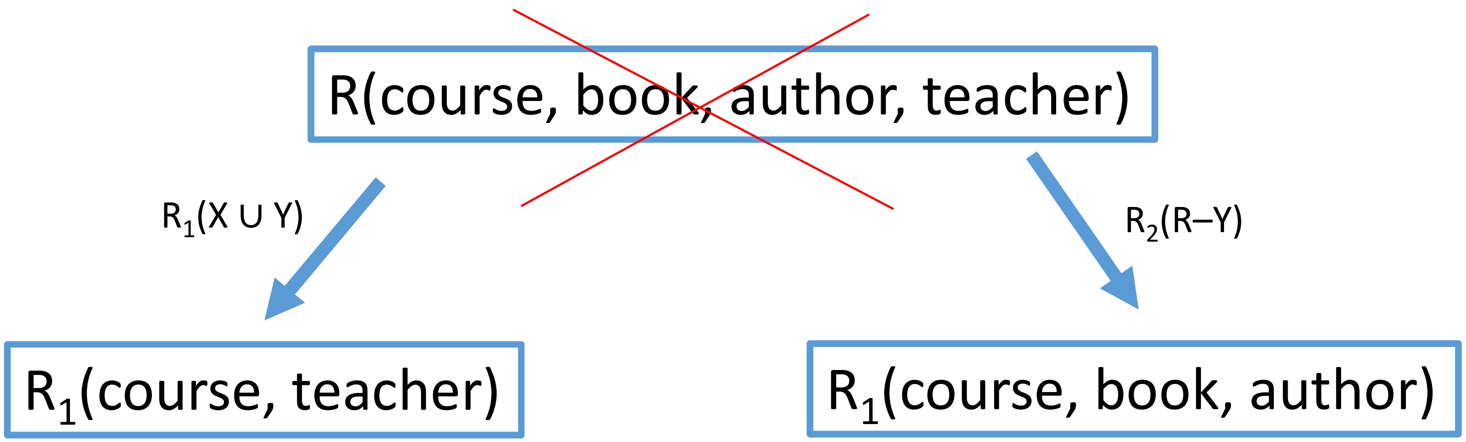
# Fourth normal form

- For a relation R to be in fourth normal:
  - R must be in BCNF
  - For all non-trivial MVDs $X \twoheadrightarrow Y$ on R, X is a superkey of R
- If $X \twoheadrightarrow Y$ and X is not a superkey, we say $X \twoheadrightarrow Y$ is a 4NF violation

- To normalize: Find a violation $X \twoheadrightarrow Y$ and break R into
  - $R_1(X \cup Y)$ (”every attribute in the MVD”)
  - $R_2(R - Y)$ (”LHS and every attribute <u>not</u> in the MVD”)
  - Then normalize both $R_1$ and $R_2$

# 4NF normalisation

- Normalizing R(course, book, author, teacher) on course $\twoheadrightarrow$ teacher



# Normalising the data

**Table: Courses**

| **course** | **book** | **author** | **teacher** |
|---|---|---|---|
| Databases | DTCB | Ullman | Jonas |
| Databases | DTCB | Ullman | Aarne |
| Databases | DTCB | Widom | Jonas |
| Databases | DTCB | Widom | Aarne |
| Reglerteknik | RTB 1 | AuthorX | TeacherX |
| Reglerteknik | RTB 2 | AuthorX | TeacherX |

Exercise: Find another MVD here?

**Table: $R_1$ (a.k.a. CourseTeacher)**

| **course** | **teacher** |
|---|---|
| Databases | Jonas |
| Databases | Aarne |
| Reglerteknik | Teacher3 |

**Table: R2 (a.k.a. CourseBooks)**

| **course** | **book** | **author** |
|---|---|---|
| Databases | DTCB | Ullman |
| Databases | DTCB | Widom |
| Reglerteknik | RTB 1 | AuthorX |
| Reglerteknik | RTB 2 | AuthorX |

# Lossless join

Note that if we join the two tables using course …

**Table: $R_1$ (a.k.a. CourseTeacher)**

| course | teacher |
|---|---|
| Databases | Jonas |
| Databases | Aarne |
| Reglerteknik | TeacherX |

**Table: R2 (a.k.a. CourseBooks)**

| course | book | author |
|---|---|---|
| Databases | DTCB | Ullman |
| Databases | DTCB | Widom |
| Reglerteknik | RTB 1 | AuthorX |
| Reglerteknik | RTB 2 | AuthorX |

NATURAL JOIN

| course | book | author | teacher |
|---|---|---|---|
| Databases | DTCB | Ullman | Jonas |
| Databases | DTCB | Ullman | Aarne |
| Databases | DTCB | Widom | Jonas |
| Databases | DTCB | Widom | Aarne |
| Reglerteknik | RTB 1 | AuthorX | TeacherX |
| Reglerteknik | RTB 2 | AuthorX | TeacherX |

We get the original table back!

Sanity check:
We did not loose any information

# Functional dependencies vs. ER-design

- FDs can find some things that ER can not find
- ER can find a lot of things that FDs can not find
  - Most many-to-many relationships can not be expressed using FDs
  - Sentences like ”students can register for courses” do not express any FDs (but possibly some MVDs?)
- The two approaches complement eachother, and confirm eachother (or sometimes contradict eachother which may indicate a problem)
- So doing both an ER-design and a FD analysis may be useful
  - This is what you will do in Task 2

# Practical use of FDs combined with ER

- FDs can be used to verify the correctness of an ER-design
  - Is the result in BCNF w.r.t. the dependencies you have identified?
  - Are the primary keys you identified sensible from your FDs?
  - If not there may be an error in your ER-translation or your understanding/modelling of the domain
- Sometimes FDs can be used to patch things up in your ER-design, particularly they are useful for finding secondary keys (UNIQUE constraints)
  - Every (minimal) key of each relation should be either the primary key or unique

# Finding functional dependencies

- Determine all attributes
- Discover FD's either by looking at each attribute and ask "what do i need to know to determine this?" or by looking at each fact in the domain description and asking "does this express a dependency?"
- You can find multiple FDs determining the same attribute

# Mining attributes (and FDs) from ER-design

- If you already have an ER-design, that may help you determine a useful set of attributes
- Looking at the relational schema is less helpful, because it contains multiple attributes that have different names but are conceptually the same (i.e. because of references)
- You can also extract some FDs by studying the diagram/schema, but that sort of misses the point of finding them since you will never find any FDs that can improve your design
  - We want to find FDs that express things our ER-design is missing
  - We should look for FDs in the domain description

# Other normal forms

- There is a whole little hierarchy of normal forms

Higly simplified:

- 1NF: basically means "only has actual tables"
- 2NF: 1NF + has valid primary key
- 3NF: 2NF + no FDs between attributes not in keys
- BCNF a.k.a. 3½NF: 3NF + attributes depend only on keys
- 4NF: 3NF + No violating MVDs
- 5NF, 6NF, DK/NF ...: Outside the scope of this course
- I expect you to know how to normalize to BCNF and 4NF

# I somehow doubt I will actually reach this slide

- So who cares what I write here?
- If a tree falls in the forest and nobody hears, does it make a sound? If a slide is the 50th slide of a 40 slide lecture, does it even exist?
- Well, there's the course page I guess, but who uses Canvas, am I right?

- Live and learn people: Don't make slides late at night or things are bound to get silly towards the end.