Database design III

Functional dependencies cont.

BCNF and 3NF

MVDs and 4NF

Quiz time!

What's wrong with this schema?

Courses(<u>code</u>, <u>period</u>, 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?

Decomposition

Courses(<u>code</u>, <u>period</u>, name, teacher)

code → name
code, period → teacher

- Fix the problem by decomposing Courses:
  - Create one relation with the attributes from the offending FD, in this case **code** and **name**.
  - Keep the original relation, but remove all attributes from the RHS of the FD. Insert a reference from the LHS in this relation, to the key in the first.

What?

Boyce-Codd Normal Form

- A relation R is in Boyce-Codd Normal Form (BCNF) if, whenever a nontrivial FD $X \rightarrow A$ holds on R, X is a superkey of R.
  - Remember: nontrivial means A is not part of X
  - Remember: a superkey is any superset of a key (including the keys themselves).

Courses(<u>code</u>, name)
CoursePeriods(<u>code</u>, <u>period</u>, teacher)

BCNF violations

- We say that an FD X → A *violates* BCNF with respect to relation R if X → A holds on R, but X is not a superkey of R.

Example:

Courses(<u>code</u>, <u>period</u>, name, teacher)

code → name	violates BCNF
code, period → teacher	does not.

BCNF normalization

- Algorithm: Given a relation R and FDs F.
  1. Identify new FDs using the transitive rule, and add these to F.
  2. Look among the FDs in F for a violation $X \rightarrow A$ of BCNF w.r.t. R.
  3. Decompose R into two relations
     - One relation RX containing all the attributes in X^+.
     - The original relation R, except the values in X^+ that are not also in X (i.e. $R - X^+ + X$), and with a reference from X to X in RX.
  4. Repeat from 2 for the two new relations until there are no more violations.

Quiz!

Decompose Courses into BCNF.

Courses(code, period, name, teacher)

code → name

code, period → teacher

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

Courses(code, name)

CoursePeriods(course, period, teacher)

course -> Courses.code

No BCNF violations left, so we're done!

Quiz again!

Why not use BCNF decomposition for designing database schemas? Why go via E-R diagrams?

- Decomposition doesn't handle all situations gracefully. E.g.
  - Self-relationships
  - Many-to-one vs. many-to-"exactly one"
  - Subclasses
  - Single-attribute entities
- E-R diagrams are graphical, hence easier to sell than some mathematical formulae.

Recovery

- We must be able to recover the original data after decomposition.

code	*per*	*name*	*teacher*
TDA357	2	Databases	Niklas Broberg
TDA357	4	Databases	Rogardt Heldal

code	*name*
TDA357	Databases

+

code	*per*	*teacher*
TDA357	2	Niklas Broberg
TDA357	4	Rogardt Heldal

code	*per*	*name*	*teacher*
TDA357	2	Databases	Niklas Broberg
TDA357	4	Databases	Rogardt Heldal

”Lossy join”

Let’s try to split on non-existent **`code`** → **`teacher`**

code	*per*	*name*	*teacher*
TDA357	2	Databases	Niklas Broberg
TDA357	4	Databases	Rogardt Heldal

code	*teacher*
TDA357	Niklas Broberg
TDA357	Rogardt Heldal

+

code	*per*	*name*
TDA357	2	Databases
TDA357	4	Databases

What?

Lossless join

- Only if we decompose on proper dependencies can we guarantee that no facts are lost.
  - Schemas from proper translation of E-R diagrams get this ”for free”.
  - The BCNF decomposition algorithm guarantees lossless join.
- A decompositon that does not give lossless join is bad.

Example of BCNF decomposition:

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

course, period → teacher
teacher → course
```

Violation!

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

Decompose:

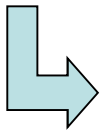
```
Teaches(teacher, course)
  course  -> Courses.code
CoursePeriods(period, teacher)
  teacher -> Teaches.teacher
```

Quiz: What just went wrong?

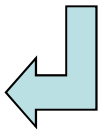
```
Teaches(teacher, course)
  course  -> Courses.code
CoursePeriods(period, teacher)
  teacher -> Teaches.teacher
```

teacher	*course*
Niklas Broberg	TDA357
Rogardt Heldal	TDA357

per	*teacher*
2	Niklas Broberg
2	Rogardt Heldal



course	*per*	*teacher*
TDA357	2	Niklas Broberg
TDA357	2	Rogardt Heldal



course, period → teacher ??

Problem with BCNF

- Some structures cause problems for decomposition.
  - $AB \rightarrow C$, $C \rightarrow B$
  - Decomposing w.r.t. $C \rightarrow B$ gives two relations, containing {C,B} and {A,C} respectively. This means we can no longer enforce $AB \rightarrow C$!
  - Intuitively, the cause of the problem is that we must split the LHS of $AB \rightarrow C$ over two different relations.
    - Not quite the full truth, but good enough.

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.
  - Non-trivial $X \rightarrow A$ violates BCNF for R if X is not a superkey of R.
  - Non-trivial $X \rightarrow A$ violates 3NF for R if X is not a superkey or R, *and A is not prime in R.*

Third Normal Form (3NF)

”*A nonkey field must provide a fact about the key, the whole key and nothing but the key, so help me Codd*”

Edgar F. (Ted) Codd was the inventor of the relational data model.

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.
  - 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

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

Decompose:

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

CoursePeriods contains a key for the original Courses relation, so we have finished.

Earlier example revisited:

```
CoursePeriods(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 CoursePeriods is in 3NF.

Quiz: What's the problem now then?

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.

Almost?

Example:

Courses(<u>code</u>, <u>name</u>, <u>room</u>, <u>teacher</u>)

code → name

code	*name*
TDA357	Databases

code	*room*	*teacher*
TDA357	VR	Niklas Broberg
TDA357	VR	Rogardt Heldal
TDA357	HC1	Niklas Broberg
TDA357	HC1	Rogardt Heldal

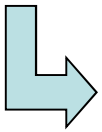
These two relations are in BCNF, but there's lots of redundancy!

Quiz: Why?

Let’s start from the bottom…

code	*room*
TDA357	HC1
TDA357	VR

code	*teacher*
TDA357	Niklas Broberg
TDA357	Rogardt Heldal



code	*room*	*teacher*
TDA357	VR	Niklas Broberg
TDA357	VR	Rogardt Heldal
TDA357	HC1	Niklas Broberg
TDA357	HC1	Rogardt Heldal

- No redundancy before join the two independent relations
- The two starting relations are what we really want to have

Independent sets of attributes

- Partition the sets of attributes in relation R into three sets: X, Y and Z.
- If when we fix the values for one set of attributes, X, the values of another set of attributes Y are independent of the values of all other attributes Z, then we can write:

  $X \twoheadrightarrow Y$

  and, by symmetry, $X \twoheadrightarrow Z$
- This kind of statement is a *multivalued dependency* (abbreviated MVD).

An example

code	*room*	*teacher*
TDA357	VR	Niklas Broberg
TDA357	VR	Rogardt Heldal
TDA357	HC1	Niklas Broberg
TDA357	HC1	Rogardt Heldal

code $\twoheadrightarrow$ room

code $\twoheadrightarrow$ teacher

- room and teacher are independent multivalued attributes.
- the rooms a course uses is *independent* of the teachers on the course.
- X=code, Y=room, Z=teacher

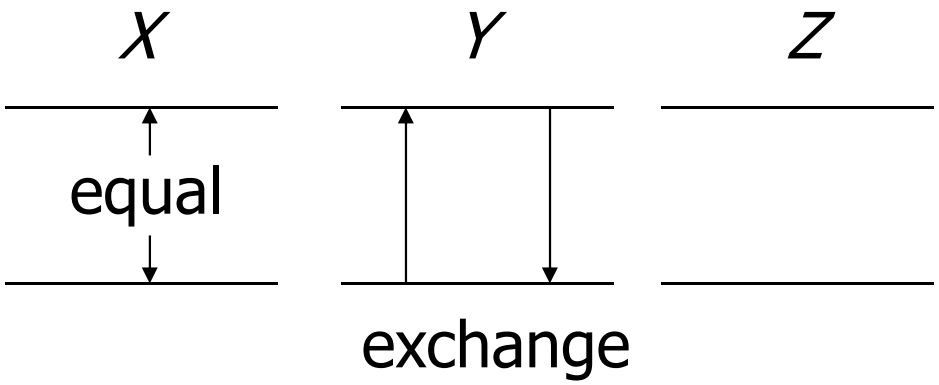
The name: multivalued dependency

- The concept that you've seen on the previous slides was given the name *multivalued dependency* by Ronald Fagin (IBM Research Laboratory) in 1977.
- The concept is about the independence of sets of multivalued attributes.
- In the VT2009 version of this course, the teacher decided to refer to this concept as "independency" (with abbreviation "IND") to emphasize this independence, and used "X ↠ Y | Z" to show that concept relates three sets of attributes.
- I think that was a good idea! I'm happy for you to use either MVD or IND in this course.

Intuitive Definition of MVD

- An MVD $X \twoheadrightarrow Y$ 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.

Picture of MVD $X \twoheadrightarrow Y$ (or IND $X \twoheadrightarrow Y \mid Z$)



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(<u>code</u>, name, <u>room</u>, <u>teacher</u>)

code → name

code ↠ room
code ↠ teacher

If we have:

<u>code</u>	*name*	*<u>room</u>*	*<u>teacher</u>*
TDA357	Databases	VR	Niklas Broberg
TDA357	Databases	HC1	Rogardt Heldal

we must also have:

TDA357	Databases	VR	Rogardt Heldal
TDA357	Databases	HC1	Niklas Broberg

otherwise room and teacher would not be independent!

FDs are MVDs

- Every FD is an MVD (but of course not the other way around).
  - If $X \twoheadrightarrow Y$ 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 $X \rightarrow A$, there is only one possible value of A for that X, and it will appear in all tuples with X. 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	Rogardt Heldal
TDA357	Databases	HC1	Niklas Broberg
TDA357	Databases	HC1	Rogardt Heldal

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.

MVD 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	Rogardt Heldal
TDA357	Databases	HC1	126	Niklas Broberg
TDA357	Databases	HC1	126	Rogardt Heldal

$$\mathbf{code} \twoheadrightarrow \mathbf{room,\ \#seats}$$

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

$$\mathbf{code} \twoheadrightarrow \mathbf{room}$$

or

$$\mathbf{code} \twoheadrightarrow \mathbf{\#seats}$$

holds.

Fourth Normal Form

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

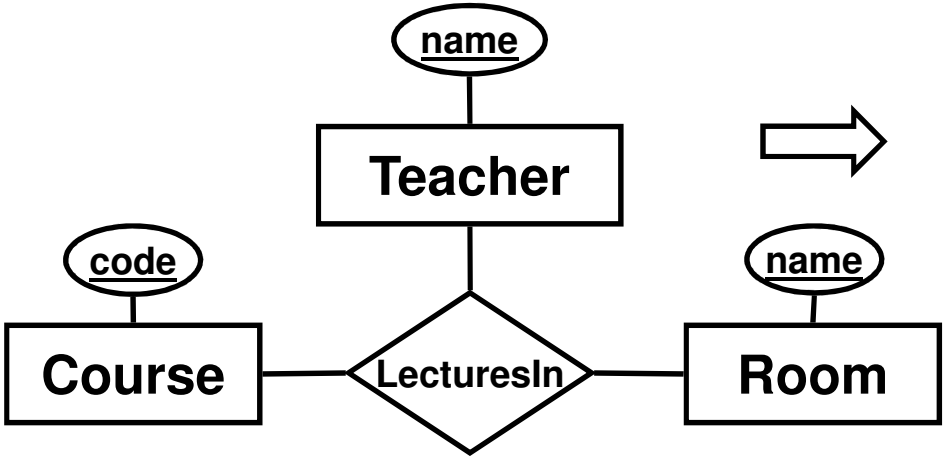
Fourth Normal Form (4NF)

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

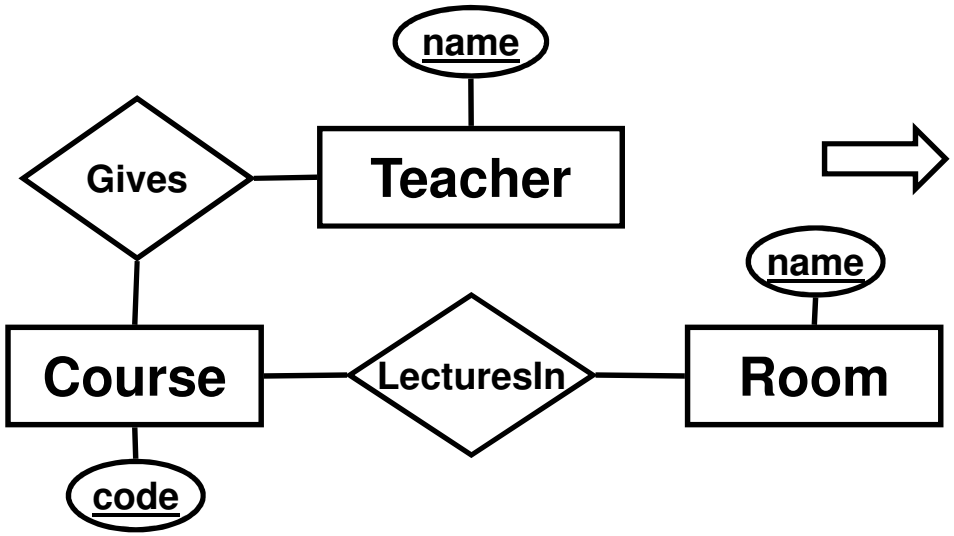
BCNF Versus 4NF

- Remember that every FD $X \rightarrow Y$ is also an MVD, $X \twoheadrightarrow Y$.
- Thus, if R is in 4NF, it is certainly in BCNF.
  - This is because any BCNF violation is a 4NF violation.
- But R could be in BCNF and not 4NF, because MVDs are “invisible” to BCNF.

Compare with E/R




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




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

Normal forms

1 NF – Only simple values allowed (definition).

Problems with nonkey attributes:
2NF – (A step towards 3NF)
3NF – All nonkey attributes only depends on the whole key.

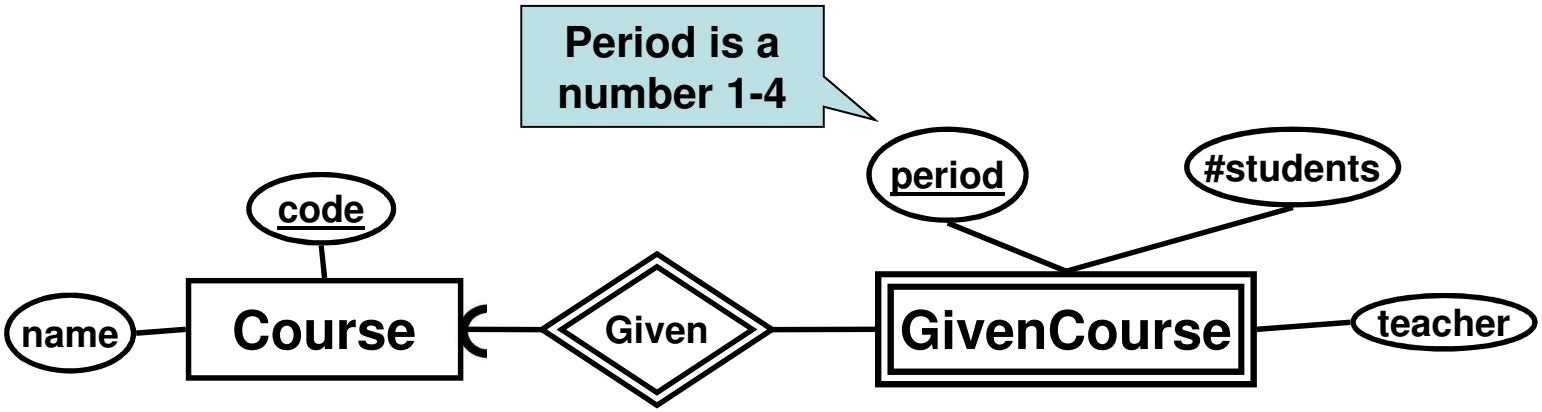
Problems within key attributes (key > 2):
4NF – Multivalued dependencies eliminated.
5NF – Other possible dependencies elimated.

Problems from nonkey attribute to key.
BCNF – Dependency from nonkey attribute to key eliminated

Constraints

- We have different kinds of constraints:
  - Dependency constraints ($X \rightarrow A$)
    - Table structure, keys, uniqueness
  - Referential constraints
    - References (a.k.a. foreign keys)
  - Value constraints
    - E.g. a room must have a positive number of seats
  - Cardinality constraints
    - E.g. no teacher may hold more than 2 courses at the same time.

Extra constraints in E-R



The point is that the diagram should be easy to understand, and easy to implement!

Extra constraints in schemas

- No formal syntax exists. Don’t let that stop you!

GivenCourses(course, period, teacher)
1 ≤ period ≤ 4

Goals of database design

- ”Map” the domain, find out what the database is intended to model.
  - The database should accept all data that is possible in reality.
  - The database should agree with reality and not accept impossible or unwanted data.
- We accomplish this by making sure that our database captures all the constraints of the domain.

The whole point of design

- The result of design should be a database schema that:
  - correctly models the domain and its constraints.
  - is easy to understand.
  - can be implemented directly in a DBMS!
    …even by someone else than the designer

Course Objectives – Design

When the course is through, you should

- Given a domain, know how to design a database that correctly models the domain and its constraints.

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

Exam – FDs and NFs

”A car rental company has the following, not very successful, database. They want your help to improve it. …”

- Identify all functional dependencies you expect to hold in the domain.
- Indicate which of those dependencies violate BCNF with respect to the relations in the database.
- Do a complete decomposition of the database so that the resulting relations are in BCNF.

Quiz!

Decompose Schedules into BCNF.

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

Next Lecture

Database Construction –
SQL Data Definition Language