Lecture 3

Database design III

Normal Forms

1NF

- Atomic Attributes

ID_Number	Name	Address	#Phone
3344	Zlatan	Manchester	63345,22523

⇓

ID_Number	Name	Address	#Phone
3344	Zlatan	Manchester	63345
3344	Zlatan	Manchester	22523

2NF

Get rid of partial dependencies

- 1NF + Partial Dependencies

ID_Number	Name	Address	Item_ID	Description
3344	Zlatan	Manchester	12	T-shrit
3344	Zlatan	Manchester	14	pants
3345	Casillas	Porto	14	pants

⇓

ID_Number	Name	Address
3344	Zlatan	Manchester
3345	Casillas	Porto

Item_ID	ID_Number	Description
12	3344	T-shrit
14	3344	pants
14	3345	pants

3NF

- 2NF + FD's

ID_Number	Name	Address	Item_ID	Description
3344	Zlatan	Manchester	12	T-shrit
3344	Zlatan	Manchester	14	pants
3345	Casillas	Porto	14	pants

⇓

ID_Number	Name	Address
3344	Zlatan	Manchester
3345	Casillas	Porto

Item_ID	Description
12	T-shrit
14	pants

Item_ID	ID_Number
12	3344
14	3344
14	3345

Quiz time!

What's wrong with this schema?

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

code → name
code, period → teacher

{('TDA357', 2, 'Databases', 'Mickey'),
 ('TDA357', 4, 'Databases', 'Tweety')}
```

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)

{('TDA357', 2, 'Databases', 'Mickey'),
 ('TDA357', 4, 'Databases', 'Tweety')}

code → name
code, period → teacher
```

Decomposition

`Courses(code, period, 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?

Decomposition

`Courses(code, period, 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.

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

Boyce-Codd Normal Form

- A relation R is in BCNF if, whenever a nontrivial FD X → A holds on R, X is a superkey of R.
  - every non-trivial FD of R has a key of R as part of the LHS
  - Remember: nontrivial means A is not part of X
  - Remember: a superkey is any superset of a key (including the keys themselves).

```
Courses(code, name)
GivenCourses(code, period, teacher)
```

BCNF violations

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

Example: `code → name` violates BCNF for the relation

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

but `code, period → teacher` does not.

BCNF normalization

- Algorithm: Given a relation R and FDs F.
  1. Compute F^+, i.e. the closure of F.
  2. Look among the FDs in F^+ for a violation X → 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)`

$\boxed{\texttt{code → name}}$ Violates BCNF, so we will kick it out of the relation
`code, period → teacher`

`{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!

Recovery

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

code	*per*	*name*	*teacher*
TDA357	3	Databases	Mickey
TDA357	2	Databases	Tweety

code	*name*
TDA357	Databases

+

code	*per*	*teacher*
TDA357	3	Mickey
TDA357	2	Tweety

code	*per*	*name*	*teacher*
TDA357	3	Databases	Mickey
TDA357	2	Databases	Tweety

"Lossy join"

Let's try to split on non-existant **`code → teacher`**

code	*per*	*name*	*teacher*
TDA357	3	Databases	Mickey
TDA357	2	Databases	Tweety

code	*teacher*
TDA357	Mickey
TDA357	Tweety

+

code	*per*	*name*
TDA357	3	Databases
TDA357	2	Databases

What?

"Lossy join"

Let's try to split on non-existant **`code → teacher`**

code	*per*	*name*	*teacher*
TDA357	3	Databases	Mickey
TDA357	2	Databases	Tweety

code	*teacher*
TDA357	Mickey
TDA357	Tweety

+

code	*per*	*name*
TDA357	3	Databases
TDA357	2	Databases

code	*per*	*name*	*teacher*
TDA357	3	Databases	Mickey
TDA357	2	Databases	Mickey
TDA357	3	Databases	Tweety
TDA357	2	Databases	Tweety

Lossless join

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

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
        teacher, period, weekday, hour → room
```

Done on blackboard.

Quiz result

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

Same as what we got by translating our E-R diagram (lecture 2), plus the extra uniqueness constraint!

`Quiz: teacher, period, weekday, hour → room ?`

Quiz again!

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

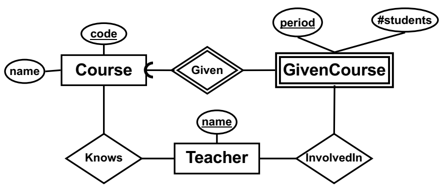
- Decomposition doesn't handle all situations gracefully. E.g.
  - Self-relationships
  - Many-to-one vs. many-to-"exactly one"
  - Subclasses
  - Single-attribute entities
- E-R diagrams are graphical, hence easier to sell than some "mathematical formulae".

Quiz again!

Why use FDs and decomposition at all? Why not just go via E-R diagrams?

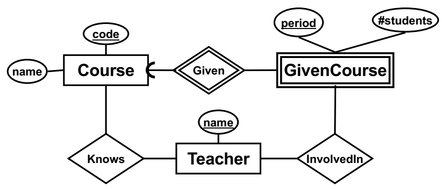
- Some constraints ("physical reality") are not captured by E-R modelling.
- FDs/BCNF decomposition allows you to:
  - *Prove* that your design is free from redundancy (or discover that it isn't!).
  - Spot dependency constraints that are not captured (e.g. `teacher, period, weekday, hour → room`), and do something sensible about them.
  - Discover errors in your E-R model or translation to relations.

Example



Quiz: What's the problem?

Example



We probably want to ensure that a teacher can only be involved in giving a course that they know. We have no formal syntax or theory for such "extra" constraints.

Example

```
Courses(code, name)
GivenCourses(course, period, #students, teacher)
  course -> Courses.code
Teachers(name)
Knows(teacher, course)
  teacher -> Teachers.name
  course  -> Courses.code
InvolvedIn(teacher, course, period)
  teacher -> Teachers.name
  (course, period) -> GivenCourses.(course, period)
```

Quiz: How can we fix the problem?

Example

```
Courses(code, name)
GivenCourses(course, period, #students, teacher)
  course -> Courses.code
Teachers(name)
Knows(teacher, course)
  teacher -> Teachers.name
  course  -> Courses.code
InvolvedIn(teacher, course, period)
  teacher -> Teachers.name
  (course, period) -> GivenCourses.(course, period)
```

Insert an extra reference!

```
(teacher, course) -> Knows(teacher, course)
```

Equality constraints

- FDs don't always give the full story.
- Equality constraints over circular relationship paths are relatively common.
  – Can sometimes – but not always – be captured via extra references.
  – Extra attributes may be needed – more on that later…

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*
Mickey	TDA357
Tweety	TDA357

per	*teacher*
2	Mickey
2	Tweety

course	*per*	*teacher*
TDA357	2	Mickey
TDA357	2	Tweety

`course, period → teacher` ??

Problem with BCNF

- Some structures cause problems for decomposition.
  – Ex: AB → C, C → B
  – Decomposing w.r.t. C → B gets us two relations, containing {C,B} and {A,C} respectively. This means we can no longer enforce AB → C!
  – Intuitively, the cause of the problem is that we must split the LHS of AB → C over two different relations.
    - Not quite the full truth, but good enough.
  – (This is exactly what happened earlier with `teacher, period, weekday, hour → room` !)

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.

X → A is in **BCNF** iff either:	X → A is in **3NF** iff either:
• X → A is a trivial FD	• X → A is a trivial FD
• X is a superkey	• 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}

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 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, room, teacher)
```

code → name

code	*name*
TDA357	Databases

code	*room*	*teacher*
TDA357	VR	Mickey
TDA357	VR	Tweety
TDA357	HC1	Mickey
TDA357	HC1	Tweety

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

Quiz: Why?

Next time, Lecture 4

Independencies and 4NF