Database design II

Functional Dependencies

BCNF

Functional dependencies (FDs)

- $X \rightarrow A$
  - ”X determines A”, ”X gives A”
  - ”A depends on X”
- X is a set of attributes, A is a single attribute
- Examples:
  - **`code`** $\rightarrow$ **`name`**
  - **`code, period`** $\rightarrow$ **`teacher`**

Assertions on a schema

- X → 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`** → **`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 → 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.
  - 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`** → **`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 → 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`** → **`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 same relation, e.g. in relation Courses we have FD:

  `code` → `name`

- A **reference** exists between attributes in two different 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 → A,B
  - Short for X → A and X → B
  - If we have both X → A and X → B, we can combine them to X → A,B.
  - **`code, period → teacher, #students`**
- Multiple attributes on LHS can be crucial!
  - **`code, period → 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	3	87	Niklas Broberg	HC1	126	Monday	15:15
TDA357	Databases	3	87	Niklas Broberg	HC2	94	Thursday	10:00
TDA357	Databases	2	93	Graham Kemp	HC4	216	Tuesday	10:00
TDA357	Databases	2	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!

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.

Trivial FDs

- A FD is *trivial* if the attribute on the RHS is 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 → room
  - room → #seats

  ⇒ course, period, weekday → #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.

- The closure of X is written 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^+

- Simple, exponential algorithm
  1. For each set of attributes X, 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 $X \rightarrow A$.
- A simplifying trick
  - If $X^+ = Y$, then for any superset Z of X, where Z is a subset of Y, $Z^+ = Y$ and no new FDs will be found.
    - 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.

Summary – FDs so far

- $X \rightarrow A$
  - X ”determines” A, A ”depends on” X
- Constraints in domain
- Trivial FDs, combining RHSs

- Computing closures
  - Attribute closures: X^+
  - FD set closures: F^+

Lab Assignment

- Write a ”student portal” application in Java
  - Part I: Design
    - Given a domain description, design a database schema using an E-R diagram.
  - Part II: Design
    - Given a domain description, find and act on the functional dependencies of the domain to fix the schema from Part I.
  - Part III: Construction and Usage
    - Implement the schema from Part II in Oracle.
    - Insert relevant data.
    - Create views to support key operations.
  - Part IV: Construction
    - Create triggers to support key operations.
  - Part V: Interfacing from external Application
    - Write a Java application that uses the database from Part III.

Lab Assignment

- Each task must be completed and approved before the next can be started.
  - Submit in good time!
- Work in groups of exactly two.
  - If you have compelling reasons for not being two in a group, come talk to me.

Part I – E-R Modelling

- Model the database by drawing an E-R diagram of the domain.
- Generate a database schema by translating your diagram to relations.

Part I – E-R Modelling

- Hand in:
  - a diagram
  - a database schema

- Submission deadline: Fri, Jan 31 (23:59)

Finding keys

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

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

```
{('TDA356', 2, 'Databases', 'Niklas Broberg'),
 ('TDA356', 4, 'Databases', 'Rogardt Heldal')}
```

code → name
code, period → teacher

Decomposition

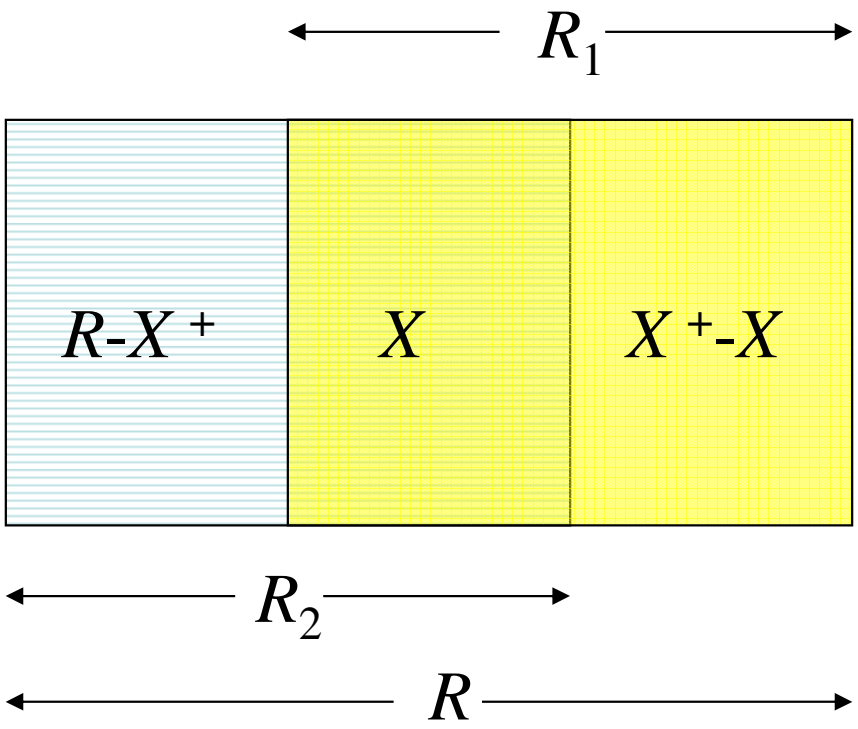
Courses(<u>code</u>, <u>period</u>, name, teacher)

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.

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

Decomposition Picture



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)

Next Lecture

BCNF decomposition

3NF