Database Management Systems (LIX022B05)

Instructor: Çağrı Çöltekin
c.coltekin@rug.nl

Information science/Informatiekunde

September 23, 2013

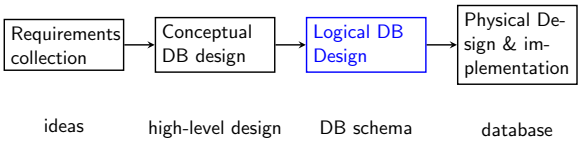
Summary of last week

- A conceptual design using E-R data model allows us to
  - think about the DB requirements systematically and formalize the ideas from the requirement analysis,
  - communicate the overall design of the database using a graphical representation.
- E-R constructs can be reduced to a database schema.
- Conceptual modeling is helpful, however, it does not guarantee correct relational database design.

Conceptual (E-R) design: things to remember

- Entity / entity set
- Relationship / relationship set
- Attribute
  - Simple
  - Composite
  - Multi-valued
- Weak entity
- one-to-one, one-to-many, many-to-many relationships
- total or partial participation
- binary or n-ary relationship sets
- recursive relationship sets
- Primary keys, foreign keys
- Converting E-R diagrams to table schemas and SQL statements

Database Design Process



- DB design is generally part of a bigger software design process.
- These steps reflect the idealized case. Typically, you may need to re-iterate over some of the steps multiple times.
- In some cases 'conceptual design' step is skipped.
- This week, we are interested in the third step.

What can go wrong (1)

Is anything wrong with this table?

title	*author*	*genre*
I, Robot & Foundation	Isaac Asimov	sci-fi
A Wizard of Earthsea	Ursula K. LeGuin	fantasy

title	*author*	*genre*
I, Robot & Foundation	Isaac Asimov	sci-fi
A Wizard of Earthsea	Ursula K. LeGuin	fantasy

Problem: The title column is not atomic.

- Try to write a query that finds the books in a certain genre.
- What if an author writes in multiple genres? (or worse, a book is in multiple genres)
- What happens if a typo in an application replaces the separator '&' with another character?
- Solution is to use atomic domains.

What can go wrong (2)

How about this one?

title	*author*	*author_phone*	*genre*
I, Robot	Isaac Asimov	1234	sci-fi
Foundation	Isaac Asimov	1234	sci-fi
A Wizard of Earthsea	Ursula K. LeGuin	2345	fantasy

More problems:

- Some fields are repeated unnecessary: waste of storage.
- If one updates phone number of an author on one row but not on the other(s).
- What happens to information about all books of an author is deleted from the system?
- Can someone insert a new book for an author whose phone number (s)he does not know?

Solution is ... to split the table into smaller tables (decomposition)

What do we want to avoid?

- Inconsistent database.
- Redundant repeated storage of information.

- Update anomalies, where we update the same information in one place but not the other.
- Deletion anomalies, where we have to delete unwanted data together with the data we want to delete.
- Insertion anomalies, where it is not possible to store a certain information without inserting additional unnecessary/unrelated data.

Do we really have to?

A good E-R design should prevent most of these anomalies. However,

- E-R design includes many subjective choices, and does not guarantee a database without anomalies.
- Even if you have a good E-R design, there are cases where anomalies can occur.
- You do not always start database design from conceptual level. Sometimes you need to start with already existing tables.

The solution

The solution is to make sure that your tables meet certain formal criteria: normal forms.

- The normal forms are achieved by decomposing (splitting) the tables that do not conform into multiple tables such that the new tables are in the desired normal form.
- Being in a certain normal form is not enough, you need to make sure that you keep the same information and same constrains using the new tables.
- There still are some loose ends: for example, which normal form to pick, and whether violate some of the requirements intentionally.

What comes next is a rather 'light' introduction to a highly theoretical subject.

First normal form

First Normal Form (1NF): Domains of all attributes should be atomic.

title	*author*	*genre*
I, Robot & Foundation	Isaac Asimov	sci-fi
A Wizard of Earthsea	Ursula K. LeGuin	fantasy

⇓

title	*author*	*genre*
I, Robot	Isaac Asimov	sci-fi
Foundation	Isaac Asimov	sci-fi
A Wizard of Earthsea	Ursula K. LeGuin	fantasy

First normal form (2)

title	*author*	*genre*
I, Robot	Isaac Asimov	sci-fi
Foundation	Isaac Asimov	sci-fi
A Wizard of Earthsea	Ursula K. LeGuin	fantasy

- 1NF does not guarantee a good design, it is just a beginning.
- 1NF is typically assumed by any database design process.
- The definition of 'atomic', and as a result 1NF, is somewhat unclear and application dependent:
  - Is 'author' field above atomic? (probably not)
  - How about an 'email' field? (maybe not)
  - How about a field such as ISBN? (certainly atomic?)

Functional dependencies: formal definition

A set of attributes $B = \beta_1 \cdots \beta_M$ is functionally dependent on another set of attributes $A = \alpha_1 \cdots \alpha_N$ if for all possible tuples in the relation, value of A on a certain tuple determine the value of B in the same tuple.

We note this functional dependency as

$$\alpha_1\, \alpha_2\, \cdots\, \alpha_N \rightarrow\, \beta_1\, \beta_2\, \cdots\, \beta_M$$

and if this is a valid for a particular relation (table), we say that the functional dependency holds for that particular relation.

In other words: the functional dependency $A \rightarrow B$ means that 'if two tuples (rows) have identical value(s) for A then they have to have identical value(s) for B'.

Functional dependencies by example

title	*author*	*author_phone*	*genre*	*pages*
I, Robot	Isaac Asimov	1234	sci-fi	256
Foundation	Isaac Asimov	1234	sci-fi	272
A Wizard of Earthsea	Ursula K. LeGuin	2345	fantasy	320

Do following functional dependencies hold?

- author → phone ✔
- author → pages ✗
- author title → pages ✔
- title → pages ✔
- author → genre ✗
- genre → author ✗
- pages → title ✗
- author title → pages phone ✔
- title → title ✔
- title author → title ✔

Trivial functional dependency

A functional dependency is called a trivial functional dependency if all attributes on the right side also appear on the left side.
Examples:

- title → title
- title pages → title
- title pages → title pages

Trivial functional dependencies hold regardless of the choice of attributes.

Inferring FDs from others

For normalization, we typically need to consider all possible functional dependencies. Some rules help us reduce the FDs that we need to consider.

Armstrong's axioms: for sets of attributes A, B, and C

1. If B ⊆ A, then A → B holds (consider trivial FDs).
   Example: *Course ECTS* → *ECTS* holds.
2. If A → B holds, A C → B C holds.
   Example: If *title* → *pages*,
   then *title genre* → *pages genre*
3. If A → B and B → C holds, then A → C also holds.
   Example: If *title* → *pages* and *pages* → *weight*, then *title* → *weight*

More rules for inferring FDs from others

More rules (can be derived from Armstrong's axioms):

- If A → B and A → C hold, then A → B C also holds.
  Example: If *author* → *genre* and *author* → *phone*,
  then *author* → *phone genre*.
- If A → B C then, A → B and A → C also hold.
  Example: If *author* → *genre phone*,
  then *author* → *genre* and *author* → *phone*.
- If A → B and B C → D hold, then A C → D also holds.
  Example: If *author* → *genre* and *genre title* → *pages*, then *author title* → *pages*.

Why should you care?

- Certain normal forms (that prevent anomalies) depend on functional dependencies.
- The conditions for normal forms typically require you to consider all functional dependencies.
- All functional dependencies for a relation is an exponentially growing set of FDs.
- Knowing rules to infer one FD from others helps us by reducing the number of dependencies that we need to consider.

  For example, if we are looking for functional dependencies that does not hold, we can easily eliminate all trivial FDs (*title pages* $\rightarrow$ *titel*), or if we know *author* $\rightarrow$ *phone genre*, we do not need to consider *author* $\rightarrow$ *phone* and *author* $\rightarrow$ *genre*.

Keys (again)

Superkey is a set of attributes, that uniquely identify a record for a given relation.

Candidate key (or simply 'key') is a minimal set of attributes, that uniquely identify a record for a given relation. A key is the set of attributes form a superkey where unnecessary attributes removed.

Primary key is a key which is chosen by the DB designer.

For the schema addres(street_addr , postcode, city):

- Is {*street_addr, postcode, city*} a superkey(✔)/key(✗)
- Is {*street_addr, postcode*} a superkey(✔)/key(✔)
- Is {*street_addr, city*} a superkey(✔)/key(✔)
- Is {*postcode*} a superkey(✗)/key(✗)

Functional dependencies and keys

Where do the keys come from?

A set of attributes K is a key, if for all attributes A, functional dependency $K \rightarrow A$ holds, and K is a minimal set of attributes with this property.

Where do the functional dependencies come from?

We assert them based on our knowledge about entities/relationships that we are modeling.

Boyce-Codd normal form (BCNF)

A relation is in Boyce-Codd normal form if for any non-trivial functional dependency $A \rightarrow B$, A is a superkey.

- In other words, the BCNF says that all functional dependencies should involve keys.
- If you have functional dependencies that violate BCNF, then there is a sub-structure in the relation.
- It does not solve all problems of DB design, but the BCNF eliminates most sources of redundancy.
- The BCNF is one of the most common normal forms DB design practice aims to achieve.

Is this table in BCNF?

title	*author*	*author_phone*	*genre*	*pages*
I, Robot	Isaac Asimov	1234	sci-fi	256
Foundation	Isaac Asimov	1234	sci-fi	272
A Wizard of Earthsea	Ursula K. LeGuin	2345	fantasy	320

Let's consider a few functional dependencies.

- author genre title pages genre $\rightarrow$ genre . . . holds, but says nothing: left side is a superkey.
- author title genre $\rightarrow$ phone . . . holds, but says nothing: left side is a superkey.
- author $\rightarrow$ phone . . . holds, and proves that table is not in BCNF: left side is not a superkey.

BCNF: decomposition

Is this decomposition good?

author	*author_phone*
Isaac Asimov	1234
Ursula K. LeGuin	2345

- *author* $\rightarrow$ *phone* holds, author is a (super)key.
- *phone* $\rightarrow$ *author* doesn't hold.

$\rightarrow$ relation is in BCNF.

title	*genre*	*pages*
I, Robot	sci-fi	256
Foundation	sci-fi	272
A Wizard of Earthsea	fantasy	320

- *title* $\rightarrow$ *genre* holds, {title} is a (super)key.
- *genre* $\rightarrow$ *pages* does not hold, but {genre} is not a (super)key.
- . . .

$\rightarrow$ relation is in BCNF.

BCNF: a trivia

Any relation (table) with two attributes (columns) is in BCNF.

Consider a relation *r(A,B)*, all possible non-trivial functional dependencies of concern are: $A \rightarrow B$, $B \rightarrow A$ (why?)

	Key			
	AB	A	B	A & B
$A \rightarrow B$	cannot hold	has to hold	cannot hold	has to hold
$B \rightarrow A$	cannot hold	cannot hold	has to holds	has to hold

Rules for decomposition

Why not decomposing everything to two-row tables?

Aside form not upsetting people who use the database, we want our decomposition to,

- produce tables that are in the desired normal form, for example, BCNF.
- allow lossless join: we should be able to recover the original table by joining the new tables.
- be dependency preserving: the functional dependencies that exist in the original table should be present in the resulting tables.

An example decomposition (bad)

title	*author*	*author_phone*	*genre*	*pages*
I, Robot	Isaac Asimov	1234	sci-fi	256
Foundation	Isaac Asimov	1234	sci-fi	272
A Wizard of Earthsea	Ursula K. LeGuin	2345	fantasy	320

⇓

author	*author_phone*
Isaac Asimov	1234
Ursula K. LeGuin	2345

title	*genre*	*pages*
I, Robot	sci-fi	256
Foundation	sci-fi	272
A Wizard of Earthsea	fantasy	320

Both tables are in BCNF, what is wrong with this decomposition?

Another example decomposition

Is this table in BCNF?

`address(street_addr, postcode, city)`

We identify the following FDs:
street_addr city → *postcode* *postcode* → *city*
! *postcode is not a superkey*

Let's decompose:

`addr1(street_addr, postcode) & addr2(postcode, city)`

This is (trivially) in BCNF, but what happened to FD:
street_addr city → *postcode* ?

BCNF decomposition

It is possible to decompose any table into multiple tables such that the resulting tables are in BCNF, and decomposition lossless.

An algorithm:

1. Find an FD $A \rightarrow \beta$ that violates BCNF, where A is a set of attributes and β is a single attribute.
2. Decompose the relation into $A\beta$ and C, where C consists of the all attributes except β.
3. Test the resulting tables for BCNF, repeat the above steps for the new tables that are not in BCNF.

Note that there is no guarantee that the result will be dependency preserving.

Dependency preservation is problematic if there are multiple overlapping keys.

BCNF: a summary

The formal definition:

A relation is in Boyce-Codd normal form if for any non-trivial functional dependency $A \rightarrow B$, A is a superkey.

A more intuitive definition (due to Bill Kent/Chris Date):

Each attribute must represent a fact about the key, the complete key, and nothing but the key.

- BCNF eliminates most (but not all!) causes of redundancy/inconsistency.
- A table can be split into multiple tables in a lossless way. However, there is no guarantee of dependency preservation.

Third normal form: one step back

A relation is in third normal form (3NF) if for any non-trivial FD $A \rightarrow B$ one of the following holds.

1. A is a superkey.
2. B-A (attributes in B but not in A) is part of a key.

The intuitive definition of BCNF3NF:

Each non-key attribute must represent a fact about the key, the complete key, and nothing but the key.

- 3NF is less strict than BCNF.
- It may be desirable in cases where BCNF decomposition is not dependency preserving.

3NF example

Is this table in BCNF/3NF?

`address(street_addr, postcode, city)`

Note that {`street_addr, postcode`} and {`street_addr, city`} are keys.

FD	**BCNF**	**3NF**
street_addr city → *postcode*	OK	OK
postcode → *city*	not OK	OK

Normal forms: an interim summary

- We studied three normal forms, 1NF, BCNF and 3NF, that set rules about good database design.
- 1NF: domains of attributes should be atomic.
- 3NF: Each non-key attribute must represent a fact about the key, the complete key, and nothing but the key.
- BCNF: Each attribute must represent a fact about the key, the complete key, and nothing but the key.
- What happened to 2NF? It is a relaxed form of 3NF, it is mostly considered to be a historical artifact.
- Are we done? No, even BCNF does not prevent all forms of redundancy/inconsistencies.

Do we need more than BCNF?

Is there anything wrong with this table?

author	*phone*	*publisher*
Isaac Asimov	1234	Spectra
Isaac Asimov	3456	Spectra
Ursula K. LeGuin	2345	HMH
Ursula K. LeGuin	3456	HMH

- This table is in BCNF? (why?)
- It clearly replicates data, we solve the problem by decomposing it into `author1(name, publisher)` and `author2(name, phone)`.
- But we do not have a principled way of detecting the anomaly.
- Fourth normal form (4NF) is the principled solution we are looking for.

Multivalued dependencies

For set of attributes that X, Y, and Z that form a relation, A multivalued dependency (MVD)

$$X \twoheadrightarrow Y$$

means that given a set of values for X, Y can have multiple values, but the values of Y are independent of values of Z. (see the textbook for the formal definition) Given the relation

`author(name, publisher, phone)`

- `name` $\rightarrow$ `publisher` ✗
- `name` $\rightarrow$ `phone` ✗
- `publisher` $\rightarrow$ `author` ✗

- `name` $\twoheadrightarrow$ `publisher` ✓
- `name` $\twoheadrightarrow$ `phone` ✓
- `publisher` $\twoheadrightarrow$ `author` ✗

Note: every FD is an MVD, but not every MVD is an FD.

Fourth normal form (4NF)

A relation is in fourth normal form (4NF) if for any non-trivial multivalued dependency $A \twoheadrightarrow B$, A is a superkey.

Back to

`author(name, publisher, phone)`

- We know that it is in BCNF.
- Is it in 4NF?
  - `name` $\twoheadrightarrow$ `phone`
  - but `name` is not a superkey

  $\Rightarrow$ the relation is not in 4NF.

Note: every relation that is in 4NF is also in BCNF, but the reverse is (obviously) not true.

Normal forms: the list so far

- 1NF domains of attributes should be atomic.
- 3NF Each non-key attribute must represent a fact about the key, the complete key, and nothing but the key.
- BCNF Each attribute must represent a fact about the key, the complete key, and nothing but the key.
- 4NF is a further restriction over the BCNF which eliminates more cases of redundancy/inconsistency.

Are we done with the normal forms?
We are, but there are higher (more strict) normal forms that we will not study.

Normal forms: a summary

- Normal forms state (formal) rules that a table should meet so that it is guarded against certain forms of anomalies.
- Once we detect a violation of a normal form, we decompose tables into smaller tables until all conform to the normal form.
- While splitting the tables, we seek lossless and dependency preserving decomposition.
- Trying to achieve 3NF, BCNF or 4NF is common in practice.
- Sometimes normal forms are intentionally violated for reasons of performance, which is called denormalization. (We will return to this in our discussion of SQL.)
- Higher forms exist, but they cover rather rare problematic cases and complicated to understand and apply.

Why do we need normal forms?

- A good conceptual (E-R) design eliminates most problems of redundancy/inconsistency, but not all.
  - There are many subjective decision in E-R design that can go wrong. Checking result of E-R design for normal forms may discover poor E-R choices.
  - Even a good E-R design may result in a poor DB schema. Things to watch out: many-to-many relationship sets and multivalued attributes.
- Sometimes you need to start from the data, a design process from the start is not available.

Normal forms: what do you need to know

- Identify functional dependencies and multivalued dependencies that exist in a table schema.
- Identify whether a table is in 1NF, 3NF, BCNF or 4NF.
- Be able to do simple decompositions to meet the requirements of one of these normal forms.

What is next?

- Reading for next week: Introduction to SQL (Chapter 3).
- Assignment 2: will be posted today, due before September 27.