Design-phase 2

part1

Database Management Systems

Module 16: Relational Database Design/1

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar Himadri B G S
Bhuyan Gurunath Reddy M

Week 03 Recap

PPD

Module 11: Advanced SQL

- Accessing SQL From a Programming Language
- Functions and Procedural Constructs
- Triggers

Module 12: Formal Relational Query Languages

- Relational Algebra
- Tuple Relational Calculus (Overview only)
- Domain Relational Calculus (Overview only)
- Equivalence of Algebra and Calculus

Module 13: Entity-Relationship Model/1

- Design Process
- E-R Model

Module 14: Entity-Relationship Model/2

- E-R Diagram
- E-R Model to Relational Schema

Module 15: Entity-Relationship Model/3

- Extended E-R Features
- Design Issues

Module Objectives

PPD

To identify the features of good relational design

To familiarize with the First Normal Form

To Introduce Functional Dependencies

Module Outline

PPD

Features of Good Relational Design

Atomic Domains and First Normal Form

Functional Dependencies

FEATURES OF GOOD RELATIONAL DESIGN

Combine Schemas?

Anomalies: insert, delete, update

Suppose we combine *instructor* and *department* into *inst_dept*

(No connection to relationship set inst_dept)

Result is possible repetition of information (*building* and *budget* against *dept_name*)

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

A Combined Schema Without Repetition

Consider combining relations

sec_class(sec_id, building, room_number) and

section(course_id, sec_id, semester, year)

into one relation

section(course_id, sec_id, semester, year, building, room_number)

No repetition in this case

What About Smaller Schemas?

Suppose we had started with *inst_dept.* How would we know to split up (**decompose**) it into *instructor* and *department*?

Write a rule "if there were a schema (*dept_name, building, budget*), then *dept_name* would be a candidate key"

Denote as a **functional dependency**:

dept_name → *building, budget*

In *inst_dept*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated.

This indicates the need to decompose *inst_dept*

Not all decompositions are good. Suppose we decompose *employee(ID, name, street, city, salary)* into

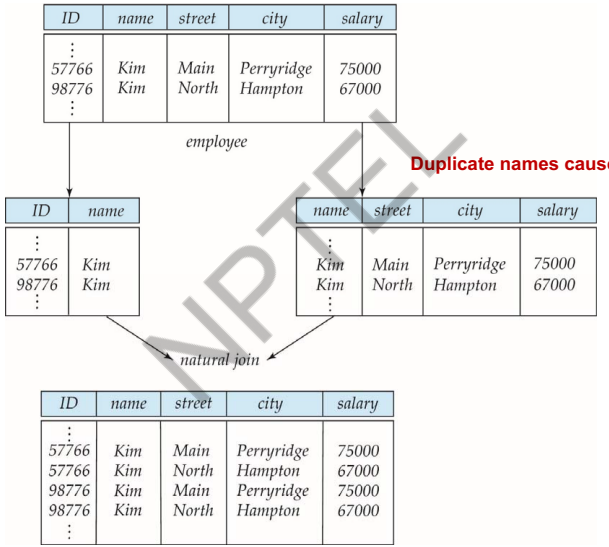
employee1 (*ID*, *name*)

employee2 (*name*, *street, city, salary*)

The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**.

A Lossy Decompositio

PPD



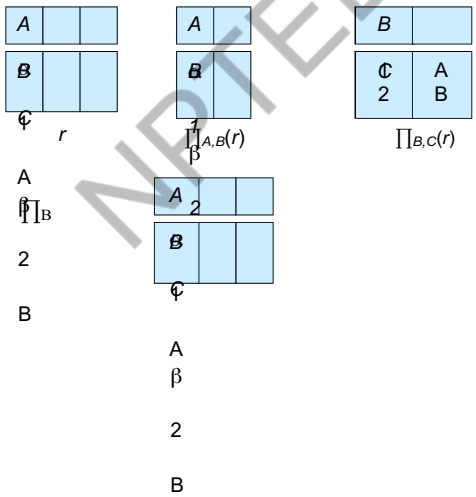
Duplicate names cause the problem here.

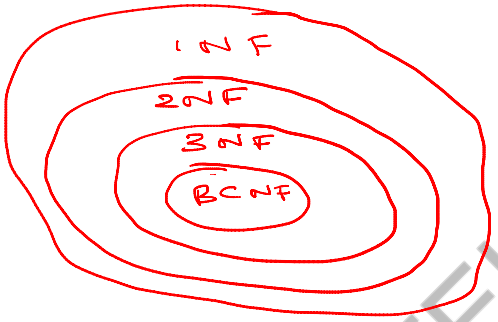
Example of Lossless-Join Decomposition

Lossless join decomposition

Decomposition of $R = (A, B, C)$

$R_1 = (A, B)$ $R_2 = (B, C)$





ATOMIC DOMAINS AND FIRST NORMAL FORM

First Normal Form (1NF)

PPD

Domain is **atomic** if its elements are considered to be indivisible units

Examples of non-atomic domains:

Set of names, composite attributes

Identification numbers like CS101 that can be broken up into parts

A relational schema R is in **first normal form** if the domains of all attributes of R are atomic

the value of each attribute contains only a single value from that domain

Non-atomic values complicate storage and encourage redundant (repeated) storage of data

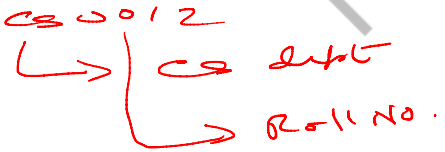
Example: Set of accounts stored with each customer, and set of owners stored with each account

We assume all relations are in first normal form

First Normal Form (Cont'd)

Atomicity is actually a property of how the elements of the domain are used

- Example: Strings would normally be considered indivisible
- Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
- If the first two characters are extracted to find the department, the domain of roll numbers is not atomic
- Doing so is a bad idea: leads to encoding of information in application program rather than in the database



First Normal Form (Cont'd)

PPD

The following is not in 1NF

Customer

Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633

A telephone number is composite

Telephone number is multi-valued

Source: https://en.wikipedia.org/wiki/First_normal_form

First Normal Form (Cont'd)

Consider:

Customer

Customer ID	First Name	Surname	Telephone Number1	Telephone Number2
123	Pooja	Singh	555-861-2025	192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53	182-929-2929
789	John	Doe	555-808-9633	

Is in 1NF if telephone number is not considered composite

However, conceptually, we have two attributes for the same concept

- Arbitrary and meaningless ordering of attributes
- How to search telephone numbers
- Why only two numbers?

Source: https://en.wikipedia.org/wiki/First_normal_form

First Normal Form (Cont'd)

PPD

Is the following in 1NF?

Customer

Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025
123	Pooja	Singh	192-122-1111
456	San	Zhang	182-929-2929
456	San	Zhang	(555) 403-1659 Ext. 53
789	John	Doe	555-808-9633

Duplicated information

ID is no more the key. Key is (ID, Telephone Number)

Source: https://en.wikipedia.org/wiki/First_normal_form

First Normal Form (Cont'd)

PPD

Better to have 2 relations:

Customer Name

Customer ID	First Name	Surname
123	Pooja	Singh
456	San	Zhang
789	John	Doe

Customer Telephone Number

Customer ID	Telephone Number
123	555-861-2025
123	192-122-1111
456	(555) 403-1659 Ext. 53
456	182-929-2929
789	555-808-9633

One-to-Many relationship between parent and child relations

Incidentally, satisfies 2NF and 3NF

Source: https://en.wikipedia.org/wiki/First_normal_form

FUNCTIONAL DEPENDENCIES

Goal — Devise a Theory for the Following

- Decide whether a particular relation R is in "good" form.
- In the case that a relation R is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

Functional Dependencies

Constraints on the set of legal relations

Require that the value for a certain set of attributes determines uniquely the value for another set of attributes

A functional dependency is a generalization of the notion of a *key*

Functional Dependencies (Cont.)

Let R be a relation schema

$\alpha \subseteq R$ and $\beta \subseteq R$

The **functional dependency**

$\alpha \rightarrow \beta$

holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \quad \Rightarrow \quad t_1[\beta] = t_2[\beta]$$

Example: Consider $r(A,B)$ with the following instance of r.

1	4
1	5
3	7

On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

Functional Dependencies (Cont.)

K is a superkey for relation schema R if and only if $K \rightarrow R$

K is a candidate key for R if and only if

- $K \rightarrow R$, and
- for no $\alpha \subset K$, $\alpha \rightarrow R$

Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:
inst_dept (*ID, name, salary, dept_name, building, budget*). We expect these functional dependencies to hold:

dept_name → *building*

and *ID* → *building*

but would not expect the following to hold:

dept_name → *salary*

Use of Functional Dependencies

We use functional dependencies to:

- test relations to see if they are legal under a given set of functional dependencies.
  - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* **satisfies** *F*
- specify constraints on the set of legal relations
  - We say that *F* **holds on** *R* if all legal relations on *R* satisfy the set of functional dependencies *F*

Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances

- For example, a specific instance of *instructor* may, by chance, satisfy
  name → *ID*

Functional Dependencies (Cont.)

A functional dependency is **trivial** if it is satisfied by all instances of a relation

Example:

ID, name → ID

name → name

In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Functional Dependencies (Cont.)

PPD

Functional dependencies are:

StudentID	Semester	Lecture	TA
1234	6	Numerical Methods	John
1221	4	Numerical Methods	Smith
1234	6	Visual Computing	Bob
1201	2	Numerical Methods	Peter
1201	2	Physics II	Simon

- *StudentID → Semester*
- *{StudentID, Lecture} → TA*
- *{StudentID, Lecture} → {TA, Semester}*

Functional Dependencies (Cont.)

PPD

Check if fd holds on the Relation

Functional dependencies are:

Employee ID	Employee Name	Department ID	Department Name
0001	John Doe	1	Human Resources
0002	Jane Doe	2	Marketing
0003	John Smith	1	Human Resources
0004	Jane Goodall	3	Sales

- *Employee ID → Employee Name*
- *Employee ID → Department ID*
- *Department ID → Department Name*

Closure of a Set of Functional Dependencie

Given a set *F* of functional dependencies, there are certain other functional dependencies that are logically implied by *F*

For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$

The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*

We denote the *closure* of *F* by **F^+**

F^+ is a superset of *F*

$F = \{A \rightarrow B, B \rightarrow C\}$

$F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

Module Summary

Identified the features of good relational design

Familiarized with the First Normal Form

Introduced the notion of Functional Dependencies

Instructor and TAs

PPD

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".

Database Management Systems

Module 17: Relational Database Design/2

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar Himadri B G S
Bhuyan Gurunath Reddy M

Module Recap

PPD

Features of Good Relational Design

Atomic Domains and First Normal Form

Functional Dependencies

Module Objectives

PPD

To understand how a schema can be decomposed for a 'good' design using functional dependencies

To introduce the theory of functional dependencies

Module Outline

PPD

Decomposition Using Functional Dependencies

Functional Dependency Theory

Decomposition Using Functional
Dependencies
Functional
Dependency Theory

DECOMPOSITION USING FUNCTIONAL DEPENDENCIES

Boyce-Codd Normal Form

A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for R

Example schema *not* in BCNF:

instr_dept (*ID, name, salary, dept_name, building, budget*)

because *dept_name → building, budget* holds on *instr_dept,* but *dept_name* is not a superkey

Decomposing a Schema into BCNF

Suppose we have a schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF

We decompose R into:

$(\alpha \cup \beta)$

$(R - (\beta - \alpha))$

In our example,

α = *dept_name*

β = *building, budget*

dept_name → *building, budget* *inst_dept* is replaced by

$(\alpha \cup \beta)$ = (*dept_name, building, budget*)

dept_name → *building, budget*

$(R - (\beta - \alpha))$ = (*ID, name, salary, dept_name*)

ID → *name, salary, dept_name*

BCNF and Dependency Preservation

Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation

If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving.*

Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form.*

Third Normal Form

A relation schema R is in **third normal form (3NF)** if for all:

$\alpha \rightarrow \beta$ in F^+

at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for R
- Each attribute A in $\beta - \alpha$ is contained in a candidate key for R

(**NOTE**: each attribute may be in a different candidate key)

If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold)

Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later)

Goals of Normalization

Let R be a relation scheme with a set F of functional dependencies

Decide whether a relation scheme R is in "good" form

In the case that a relation scheme R is not in "good" form, decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that

- each relation scheme is in good form
- the decomposition is a lossless-join decomposition
- Preferably, the decomposition should be dependency preserving

all fd's should hold on decomposed new Relations

How good is BCNF?

There are database schemas in BCNF that do not seem to be sufficiently normalized

Consider a relation

inst_info (ID, child_name, phone)

where an instructor may have more than one phone and can have multiple children

ID	*child_name*	*phone*
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	Willian	512-555-4321

inst_info

How good is BCNF? (Cont.)

There are no non-trivial functional dependencies and therefore the relation is in BCNF

Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples

(99999, David, 981-992-3443)
(99999, William, 981-992-3443)

How good is BCNF? (Cont.)

Therefore, it is better to decompose *inst_info* into:

inst_child

ID	*child_name*
99999	David
99999	David
99999	William
99999	Willian

inst_phone

ID	*phone*
99999	512-555-1234
99999	512-555-4321
99999	512-555-1234
99999	512-555-4321

This suggests the need for higher normal forms, such as Fourth Normal Form (4NF)

FUNCTIONAL DEPENDENCY THEORY

Functional-Dependency Theory

We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies

We then develop algorithms to generate lossless decompositions into BCNF and 3NF

We then develop algorithms to test if a decomposition is dependency-preserving

Closure of a Set of Functional Dependencie

Given a set *F* set of functional dependencies, there are certain other functional dependencies that are logically implied by *F*

For e.g. If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$

The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*

We denote the *closure* of *F* by F^+

Closure of a Set of Functional Dependencie

We can find F^+, the closure of F, by repeatedly applying **Armstrong's Axioms:**

- if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (**reflexivity**)
- if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ (**augmentation**)
- if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (**transitivity**)

These rules are

- **sound** (generate only functional dependencies that actually hold), and
- **complete** (generate all functional dependencies that hold)

Example

$R = (A, B, C, G, H, I)$ $F = \{ A \rightarrow B$
$A \rightarrow C$ $CG \rightarrow H$ $CG \rightarrow I$
$B \rightarrow H\}$

Some members of F^+

$A \rightarrow H$

by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$

$AG \rightarrow I$

by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$

$CG \rightarrow HI$

by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$, and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

Procedure for Computing F^+

To compute the closure of a set of functional dependencies F:

$F^+ = F$
repeat
    for each functional dependency *f* in F^+
        apply reflexivity and augmentation rules on *f*
        add the resulting functional dependencies to F^+
    for each pair of functional dependencies f_1 and f_2 in F^+
        if f_1 and f_2 can be combined using transitivity
            then add the resulting functional dependency to F^+
until F^+ does not change any further

NOTE: We shall see an alternative procedure for this task later

Closure of Functional Dependencies (Cont.)

Additional rules:

- If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\,\gamma$ holds (**union**)
- If $\alpha \rightarrow \beta\,\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (**decomposition**)
- If $\alpha \rightarrow \beta$ holds and $\gamma\,\beta \rightarrow \delta$ holds, then $\alpha\,\gamma \rightarrow \delta$ holds (**pseudotransitivity**)

The above rules can be inferred from Armstrong's axioms

Closure of Attribute Sets

Given a set of attributes α, define the ***closure*** of α **under** *F* (denoted by α^+) as the set of attributes that are functionally determined by α under *F*

Algorithm to compute α^+, the closure of α under *F* *result* := α;

while (changes to *result*) **do** **for each** $\beta \rightarrow \gamma$ **in** *F* **do**
begin
if $\beta \subseteq$ *result* **then** *result* := *result* $\cup$ γ
end

Example of Attribute Set Closure

$R = (A, B, C, G, H, I)$

$F = \{A \rightarrow B$

$A \rightarrow C \quad CG \rightarrow H \quad CG \rightarrow I \quad B \rightarrow H\}$

$(AG)^+$

1. *result = AG*
2. *result = ABCG* $(A \rightarrow C$ and $A \rightarrow B)$
3. *result = ABCGH* $(CG \rightarrow H$ and $CG \subseteq AGBC)$
4. *result = ABCGHI* $(CG \rightarrow I$ and $CG \subseteq AGBCH)$

Is *AG* a candidate key?

5. Is AG a super key?
   1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$
   2. Is any subset of AG a superkey?
      1. Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$
      2. Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

Uses of Attribute Closure

- There are several uses of the attribute closure algorithm:

Testing for superkey:

To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of R.

Testing functional dependencies

To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if β

- $\subseteq \alpha^+$.

That is, we compute α^+ by using attribute closure, and then check if it contains β.

Is a simple and cheap test, and very useful

Computing closure of F

For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

Module Summary

Discussed issues in 'good' design in the context of functional dependencies

Introduced the theory of functional dependencies

Instructor and TAs

PPD

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with “PPD”.