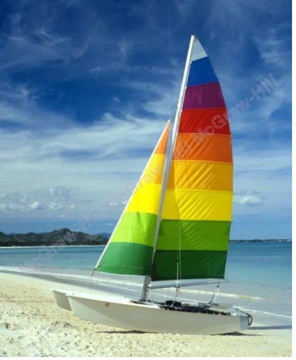

Chapter 8: Relational Database Design

Database System Concepts, 6th Ed.

Edited by Radhika Sukapuram

Question

- How to design a database schema for a relational database ?
  - Without redundancy
  - Allows us to retrieve information easily
- The above imply we need to design schemas in an appropriate **normal form**. How to do this formally ?

Combine Schemas?

- Suppose instead of *instructor* and *department* we combine them into *inst_dept*
  - *instructor (ID, name, salary); department (dept_name, building, budget)*
  - (No connection to relationship set inst_dept)
- Result is possible repetition of information

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

Combine Schemas ? Cont.

- Possible null values
  - A new department is created, no instructors yet. (Null required for id, name, salary)
  - The schema *department* can handle this

  - Question: Given a schema, how to recognize that there will be repetitions and hence it should be split ?

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**:

  $$dept\_name \rightarrow building,\ budget$$

  dept_name **determines** *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*

Lossy decompositions

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

  employee1 (*ID*, *name*)

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

What is the consequence ?

A Lossy Decomposition

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

name	street	city	salary
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Lossy decomposition cont.

- We lose information
  - we cannot reconstruct the original *employee* relation
  - so, this is a **lossy decomposition**.

Example of Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of $R = (A, B, C)$

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

A	B	C
α	1	A
β	2	B

r

A	B
α	*1*
β	*2*

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_{A,B}(r) \bowtie \Pi_{B,C}(r)$

A	B	C
α	1	A
β	2	B

First Normal Form

- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names (children of an employee), composite attributes
    - Employee Identification numbers like CS0022 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
- 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 as separate relations
    - If an account is to be added, needs addition in two places
    - Enough if only one relation is stored
      - but which one ?
      - Complicates queries if only one is stored

First Normal Form cont.

- 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 programs rather than in the database.
- We assume all relations are in first normal form

Goal — Devise a Theory for the Following

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

Functional Dependencies

- An instance of a relation that satisfies all real-life constraints is a **legal relation**
- 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.
  - Generalization of the notion of a *key.*

Functional Dependencies (Cont.)

- Let r(*R*) be a relation schema (R denotes the set of attributes)

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

- The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on *R*

if and only if for any legal relation *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,

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

- 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:

  $$dept\_name \rightarrow building$$

  and $$dept\_name \rightarrow budget$$

  but would not expect the following to hold:

  $$dept\_name \rightarrow salary$$

Use of Functional Dependencies

- We use functional dependencies to:
  - test relation instances 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 $name \rightarrow ID.$

Functional Dependencies (Cont.)

- *A* functional dependency $\alpha \rightarrow \beta$ is **trivial** if $\beta \subseteq \alpha$
- Examples*:*
  - *ID, name* $\rightarrow$ *ID*
  - *name* $\rightarrow$ *name*

Closure of a Set of Functional Dependencies

- 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 $\mathbf{F^+}$.
- F^+ is a superset of *F*.

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

In other words: “R is in BCNF if the only non-trivial FDs over R are key constraints.”

Example schema *not* in BCNF:

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

because *dept_name* $\rightarrow$ *building, budget*
holds on *instr_dept,* but *dept_name* is not a superkey

What does BCNF achieve ?

- A relation in BCNF has no redundancy due to FDs

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

- SNLRWH has FDs $S \rightarrow SNLRWH$ and $R \rightarrow W$
- Q: Is this relation in BCNF?