Relational Database Design _Part 1

Outline

Features of Good Relational Design

Atomic Domains and First Normal Form

Functional Dependencies

Normal Forms

Functional Dependency Theory

Decomposition using Functional Dependencies

Algorithms for Decomposition using Functional Dependencies

Decomposition Using Multivalued Dependencies

More Normal Form

Combine Schemas?

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

Is this a good design?

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

Anomalies :Problems identified with basic database operations

- Insertion
- Deletion
- Updation
- Need to split into smaller schemas

What about Smaller Schemas?

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**:

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

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*
- Instructor(ID,name, salary,dept_name) and
- Department(dept_name,building,budget)

What about Smaller Schemas?

Not all decompositions are good. Some cases we lose information

Suppose we decompose
employee(ID, name, street, city, salary)
into

employee1 (*ID, name*)

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

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

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
⋮				

98776	Kim	North	Hampton	67000
⋮				

Lossless-Join Decomposition

Lossless join decomposition

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

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

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

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

A	B	C
α	1	A
β	2	B

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

First Normal Form

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

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

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.

First Normal Form (Cont'd)

Atomic 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.

Goal :— Devise a Theory

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,

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

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 $ID \rightarrow building$

but would not expect the following to hold:

$$dept\_name \rightarrow salary$$

Functional Dependencies (Cont.)

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

- Example*:*
  - *ID, name* $\rightarrow$ *ID*
  - *name* $\rightarrow$ *name*
- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

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

$$name \rightarrow ID.$$

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.