Normalization

Features of Good Relational Designs

- Suppose we combine *instructor* and *department* into *in_dep,* which represents the natural join on the relations *instructor* and *department*

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

- There is repetition of information (redundancy)
- It may cause anomaly (possibility of certain data getting inconsistent): Updation, Insertion, Deletion
- Need to use null values (if we add a new department with no instructors)

A Combined Schema Without Repetition

- Not all combined schemas result in repetition of information

- 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

Decomposition

- Suppose we had started with the schema *ins_dept*
- How would we know to split up (**decompose**) it into *instructor* and *department*?
- Write a rule “if there were schema (*dept_name*, *building*, *budget*), then *dept_name* would be a candidate key”
- Denote as **functional dependency**

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

- In *inst_dept*, because *dept_name* is not a candidate key, the *building* and the *budget* of a *department* may have to be repeated
- The only way to avoid the repetition-of-information problem in the *in_dep* schema is to decompose it into two schemas – *instructor* and *department* schemas

Decomposition

- Not all decompositions are good
- Suppose we decompose

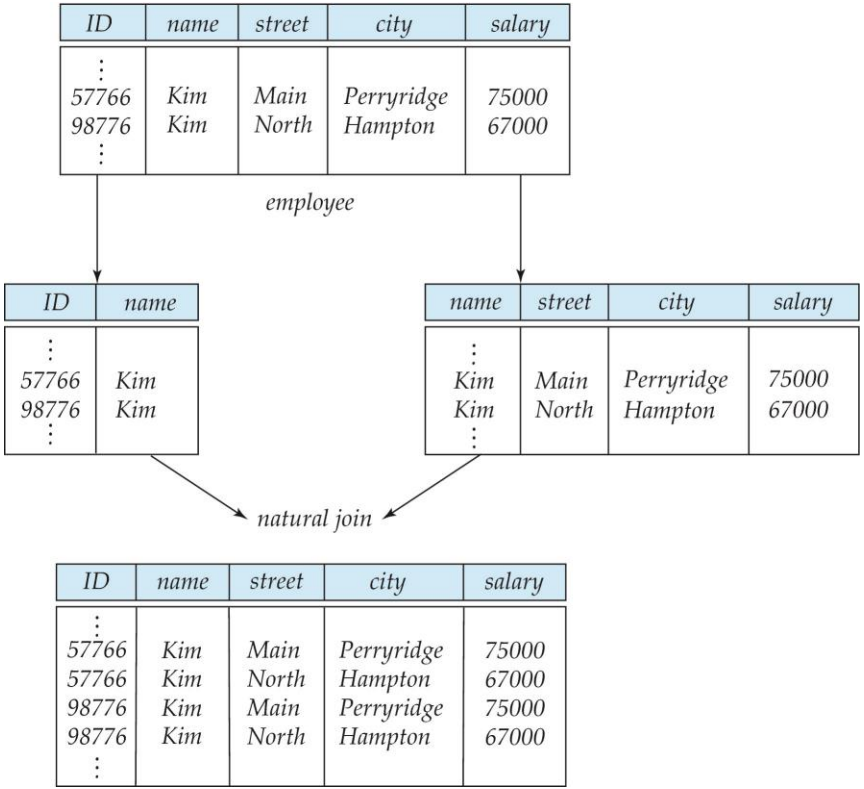
  employee(ID, name, street, city, salary) into

  employee1 (*ID*, *name*)

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

- The problem arises when we have two employees with the same name
- The next slide shows how we lose information- we cannot reconstruct the original *employee* relation- and so, this is a **lossy decomposition**

A Lossy Decomposition



A Lossy Decomposition

- Let *R* be a relation schema and let R_1 and R_2 form a decomposition of R
- That is R = R_1 U R_2
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas R_1 U R_2
- Formally,

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

- And, conversely a decomposition is lossy if

$$r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

Example of Lossless 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

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

$\Pi_A (r) \bowtie \Pi_B (r)$

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, 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

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

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

- E.g. 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

- The following is not in 1NF
  - *Telephone Number* is composite
  - *Telephone Number* is multivalued

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

First Normal Form

- 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 concepts
  - Arbitrary and meaningless ordering of attributes
  - How to search *Telephone Numbers*
  - Why only two numbers?

First Normal Form

- 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*)

First Normal Form

- Better to have two relations

Customer Name

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

Customer Telephone Number

Telephone Number ID	Customer ID	Telephone Number
1	123	555-861-2025
2	123	192-122-1111
3	456	(555) 403-1659 Ext. 53
4	456	182-929-2929
5	789	555-808-9633

  - One-to-many relationship between parent and child relations
  - Incidentally satisfies 2NF and 3NF

Normalization Theory

- 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 set of relations {R_1, R_2, ..., R_n} such that
  - Each relation is in good form
  - The decomposition is a lossless decomposition

- Our theory is based on:
  - Functional dependencies
  - Multivalued dependencies

Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world

- For example, some of the constraints that are expected to hold in a university database are:
  - Students and instructors are uniquely identified by their ID
  - Each student and instructor has only one name
  - Each instructor and student is (primarily) associated with only one department
  - Each department has only one value for its budget, and only one associated building

Functional Dependencies

- An instance of a relation that satisfies all such real-world constraints is called a **legal instance** of the relation
- A legal instance of a database is one where all the relation instances are legal instances
- 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 Definition

- Let *R* be a relation schema

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

- 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, $B \rightarrow A$ hold; $A \rightarrow B$ does **NOT** hold

Keys and Functional Dependencies

- *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:

  in_dep(*ID, name, salary, dept_name, building, budget*)
- We expect these functional dependencies to hold:

  $dept\_name \rightarrow building$

  $ID \rightarrow building$
- But would not expect the following to hold:

  $dept\_name \rightarrow salary$

Use of Functional Dependencies

- We use functional dependencies to:
  - 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*
  - To 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

$$name \rightarrow ID$$

Trivial Functional Dependencies

- 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 $\beta \subseteq \alpha$

Functional Dependencies Example

- Functional dependencies are:

Student ID	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 Example

- 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 Dependencies

- Given a set *F* set of functional dependencies, there are certain other functional dependencies that are logically implied by *F*
  - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
  - etc.

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

- We denote the *closure* of *F* by $\boldsymbol{F^+}$

Next Lecture

Normalization

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in