Chapter 7: Relational Database Design. Part 1.

1. Pitfalls in Relational Database Design

- Relational database design requires that we find a "good" collection of relation schemas. A bad design may lead to
  - Repetition of Information.
  - Inability to represent certain information.
- Design Goals:
  - Avoid redundant data
  - Ensure that relationships among attributes are represented
  - Facilitate the checking of updates for violation of database integrity constraints.

Example

- Consider the relation schema:
  Lending-schema = (*branch-name, branch-city, assets, customer-name, loan-number, amount)*

branch-name	branch-city	assets	customer-name	loan-number	amount
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

Problems:

- Redundancy:
  - Data for *branch-name, branch-city, assets* are repeated for each loan that a branch makes
  - Wastes space
  - Complicates updating, introducing possibility of inconsistency of *assets* value
- Null values
  - Cannot store information about a branch if no loans exist
  - Can use null values, but they are difficult to handle.

2. Decomposition

- Decompose the relation schema *Lending-schema* into:

Branch-schema = (branch-name, branch-city,assets)
Loan-info-schema = (customer-name, loan-number,branch-name, amount)

- All attributes of an original schema (*R*) must appear in the decomposition (R_1, R_2):
  $R = R_1 \cup R_2$
- **Lossless-join decomposition.**
  For all possible relations *r* on schema *R*
  $r = \prod_{R1}(r)\ |x|\ \prod_{R2}(r)$

3. A Lossy Decomposition. Example.

employee_id	*employee_name*	*telephone_number*	*start_date*
⋮			
123-45-6789	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
⋮			

employee

employee_id	*employee_name*
⋮	
123-45-6789	Kim
987-65-4321	Kim
⋮	

employee_name	*telephone_number*	*start_date*
⋮		
Kim	882-0000	1984-03-29
Kim	869-9999	1981-01-16
⋮		

$\bowtie$

employee_id	*employee_name*	*telephone_number*	*start_date*
⋮			
123-45-6789	Kim	882-0000	1984-03-29
123-45-6789	Kim	869-9999	1981-01-16
987-65-4321	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
⋮			

Problem: How to distinguish?

4. 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

5. Review: domain.

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

- Examples of non-atomic domains: set of names, composite attributes
- In relational schema all domains are usually assumed to be atomic

6. First Normal Form

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

  We assume all relations are in first normal form

If a relational schema is in first normal form, can we say that it is a "good" scheme. No. Let's go on.

7. Functional Dependencies

- We'll build our theory on functional dependencies.
- **Functional Dependency** is a constraint that requires 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.*

8. Review: keys.

- 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:

Loan-info-schema = (customer-name, loan-number, branch-name, amount).

We expect this set of functional dependencies to **hold** (be satisfied):

loan-number $\rightarrow$ *amount*

loan-number $\rightarrow$ *branch-name*

but would not expect the following to hold:

loan-number $\rightarrow$ *customer-name*

9. 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 *Loan-schema* may, by chance, satisfy
loan-number → *customer-name*

10. Functional Dependencies (Cont.)

- *A* functional dependency is **trivial** if it is satisfied by all instances of a relation
  - *E.g.*
    - *customer-name, loan-number* → *customer-name*
    - *customer-name* → *customer-name*

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

11. 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*.
  - 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 $\mathbf{F^+}$.
- We can find all of F^+ by 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**
- $R = (A, B, C, G, H, I)$

  $F = \{ \quad A \rightarrow B$

  $A \rightarrow C$

  $CG \rightarrow H$

  $CG \rightarrow I$

  $B \rightarrow H\}$
- 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 $CG \rightarrow I$
  - $CG \rightarrow HI$
    - from $CG \rightarrow H$ *and* $CG \rightarrow I$: "union rule" can be inferred from definition of functional dependencies, or:

    (1) augmentation of $CG \rightarrow I$ to infer $CG \rightarrow CGI$,
    (2) augmentation of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then
    (3) transitivity.

12. Closure of Functional Dependencies (Cont.)

- We can further simplify manual computation of F^+ by using the following 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 *a*nd $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds **(pseudotransitivity)**

  The above rules can be inferred from Armstrong's axioms.

13. Look back at our goal: lossless-join decomposition (#4 and #2).

- All attributes of an original schema (*R*) must appear in the decomposition (R_1, R_2):

  $R = R_1 \cup R_2$
- Lossless-join decomposition.

  For all possible relations *r* on schema *R*

  $r = \prod_{R1}(r) \; |x| \; \prod_{R2}(r)$

14. How do we know that decomposition is a lossless join?

. A decomposition of R into R_1 and R_2 is lossless join if and only if

. **at least one of** the following dependencies is in F^+:

$$R_1 \cap R_2 \rightarrow R_1$$

$$R_1 \cap R_2 \rightarrow R_2$$

Example: Lossless join.

Decompose the relation schema *Lending-schema* into:

Branch-schema = (branch-name, branch-city,assets)

Loan-info-schema = (customer-name, loan-number, branch-name, amount)

Example: A Lossy Decomposition (Kim example)

Decompose the relation schema *Employee-schema=(employee_id, employee_name, telephone_number, start_date)* into:

Employee-id-schema = (employee_id, employee_name)

Employee-info-schema = (employee_name, telephone_number, start_date)