Chapter 7: Relational Database Design

- Pitfalls in Relational Database Design
- Decomposition
- Normalization Using Functional Dependencies
- Normalization Using Multivalued Dependencies
- Normalization Using Join Dependencies
- Domain-Key Normal Form
- Alternative Approaches to Database Design

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*)

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

Decomposition

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

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

Customer-loan-schema = (*customer-name, loan-number, 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

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

Example of a Non Lossless-Join Decomposition

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

  $R_1 = (A) \qquad R_2 = (B)$

A	B
α	1
α	2
β	1

r

A
α
β

$\Pi_A(r)$

B
1
2

$\Pi_{B(r)}$

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

A	B
α	1
α	2
β	1
β	2

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

Normalization Using Functional Dependencies

When we decompose a relation schema R with a set of functional dependencies F into R_1 and R_2 we want:

- Lossless-join decomposition: At least one of the following dependencies is in F+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- No redundancy: The relations R_1 and R_2 preferably should be in either Boyce-Codd Normal Form or Third Normal Form.
- Dependency preservation: Let F_i be the set of dependencies in F^+ that include only attributes in R_i. Test to see if:
  - $(F_1 \cup F_2)^+ = F^+$

  Otherwise, checking updates for violation of functional dependencies is expensive.

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:
  $$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$
  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:
  $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

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

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $\quad\quad B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF
- Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF
  - Lossless-join decomposition
  - Dependency preserving

BCNF Decomposition Algorithm

result := $\{R\}$;
done := false;
compute F^+;
while (**not** *done*) **do**
  if (there is a schema R_i in *result* that is not in BCNF)
    then begin
      let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+, and $\alpha \cap \beta = \emptyset$;
      result := (*result* $-$ R_i) $\cup$($R_i - \beta$) $\cup$(α, β);
    end
  else *done* := true;

Note: each R_i is in BCNF, and decomposition is lossless-join.

Example of BCNF Decomposition

- R = (*branch-name, branch-city, assets,*
  customer-name, loan-number, amount)

  F = {*branch-name* $\rightarrow$ *assets branch-city*
  loan-number $\rightarrow$ *amount branch-name*}

  Key = {*loan-number, customer-name*}

- Decomposition
  - R_1 = (*branch-name, branch-city, assets*)
  - R_2 = (*branch-name, customer-name, loan-number, amount*)
  - R_3 = (*branch-name, loan-number, amount*)
  - R_4 = (*customer-name, loan-number*)

- Final decomposition

$$R_1, R_3, R_4$$

BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- $R = (J, K, L)$

  $$F = \{JK \rightarrow L, \quad L \rightarrow K\}$$

  Two candidate keys = JK and JL

- R is not in BCNF

- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$