Chapter 7. Relational Database Design. Part 2.

- Review part 1:
  - Our problems.
  - Our goals.
  - First Normal Form
  - Lossless join decomposition
  - Functional Dependencies
  - Closure of a set of functional dependencies
- Plan part 2:
  - Normalization Using Functional Dependencies
  - Boyce-Codd Normal Form (BCNF)
  - Third Normal Form

Normalization Using Functional Dependencies

- When we decompose a relation schema R with a set of functional dependencies F into R_1, R_2,.., R_n we want
  - **Lossless-join decomposition:** Otherwise decomposition would result in information loss.
  - **No redundancy:** The relations R_i preferably should be in either Boyce-Codd Normal Form or Third Normal Form.
  - **Dependency preservation:** Let F_i be the set of dependencies F^+ that include only attributes in R_i.
    - Preferably the decomposition should be **dependency preserving**, that is, $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
    - Otherwise, checking updates for violation of functional dependencies may require computing joins, which is expensive.

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B)$, $R_2 = (B, C)$
  - Lossless-join decomposition:
    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Lossless-join decomposition:
    $R_1 \cap R_2 = \{A\}$ 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

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $B \rightarrow C\}$
  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

Testing for BCNF

- To check if a relation schema R with a given set of functional dependencies F is in BCNF, it suffices to check only the dependencies in the given set F for violation of BCNF, rather than checking all dependencies in F^+.
  - We can show that if none of the dependencies in F causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either.
- However, using only F is **incorrect** when testing a relation in a decomposition of R
  - E.g. Consider R (A, B, C, D), with $F = \{ A \rightarrow B, B \rightarrow C\}$
    - Decompose R into $R_1(A,B)$ and $R_2(A,C,D)$
    - Neither of the dependencies in F contain only attributes from (A,C,D) so we might be mislead into thinking R_2 satisfies BCNF.
    - In fact, dependency $A \rightarrow C$ in F^+ shows R_2 is not in BCNF.

BCNF Decomposition Algorithm

Let's R be a schema that is not in BCNF. Then there is at least one non-trivial functional dependency $\alpha \rightarrow \beta$ such that α is not a superkey for R. We replace R in our design with two schemas:

$(\alpha \cup \beta)$

$(R - (\beta - \alpha))$

It may be that one or more of the resulting schemas are not in BCNF. Then we continue decomposition.

Example of BCNF Decomposition

- *R = (branch-name, branch-city, assets, customer-name, loan-number, amount)*
  F = {branch-name → assets branch-city loan-number → amount branch-name}
  Key = *{loan-number, customer-name}*
- Decomposition, 1st step
  - R_1 = (*branch-name, branch-city, assets)*
  - R_2 = *(branch-name, customer-name, loan-number, amount)*
- *Decomposition, 2nd step,* R_2 is not in BCNF
  - 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

- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $L \rightarrow K\}$
  Two candidate keys = JK and JL
- R is not in BCNF
- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

Third Normal Form: Motivation

- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form.
  - Allows some redundancy (with resultant problems; we will see examples later)
  - But FDs can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

Third Normal Form

- A relation schema R is in third normal form (3NF) if for all:

  $$\alpha \rightarrow \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

    (NOTE: each attribute may be in a different candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).

3NF (Cont.)

- Example
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$
  - Two candidate keys: JK and JL
  - R is in 3NF

    $JK \rightarrow L$ JK is a superkey
    $L \rightarrow K$ K is contained in a candidate key
  - BCNF decomposition has (JL) and (LK)
    - Testing for $JK \rightarrow L$ requires a join
- There is some redundancy in this schema
- Equivalent to example in book:

  Banker-schema = (branch-name, customer-name, banker-name)

  banker-name → branch name

  branch-name, customer-name → banker-name

Testing for 3NF

Need to check only FDs in *F*, need not check all FDs in F^+.

Is the following schema in 3NF? in BCNF?

(employee, department_number, department_address)

employee →department_number

department_number → department_address

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

Comparison of BCNF and 3NF (Cont.)

- Example of problems due to redundancy in 3NF
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

A schema that is in 3NF but not in BCNF has the problems of

- repetition of information (e.g., the relationship l_1, k_1)
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).

Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.

  Can specify FDs using assertions, but they are expensive to test
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key.

Overall Database Design Process

- We have assumed schema *R* is given
  - *R* could have been generated when converting E-R diagram to a set of tables.
  - *R* could have been a single relation containing *all* attributes that are of interest (called **universal relation**).
  - Normalization breaks *R* into smaller relations.
  - *R* could have been the result of some ad hoc design of relations, which we then test/convert to normal form.
- In some cases we perform denormalization, returning to the version with redundancy, but avoiding some costly joins.