Chapter 8: Relational Database Design

Database System Concepts, 6^{th} Ed.

Edited by Radhika Sukapuram

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 (3NF)
  - Allows some redundancy (with resultant problems; we will see examples later)
  - But functional dependencies 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., $\beta \subseteq \alpha$)
  - 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.
- The third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).
- A relation in 3NF may have redundancies due to FDs

3NF Example

- Relation *dept_advisor*:
  - *dept_advisor* (*s_ID, i_ID, dept_name)*
    $F = \{s\_ID, dept\_name \rightarrow i\_ID,\ i\_ID \rightarrow dept\_name\}$
  - Two candidate keys: *s_ID, dept_name,* and *i_ID, s_ID*
  - *R* is in 3NF
    - $s\_ID, dept\_name \rightarrow i\_ID$
      - *s_ID, dept_name* is a superkey
    - $i\_ID \rightarrow dept\_name$
      - *dept_name* is contained in a candidate key

Redundancy in 3NF

- There is some redundancy in this schema
- 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

- repetition of information (e.g., the relationship l_1, k_1)
  - *(i_ID, dept_name)*
- need to use null values (e.g., to represent the tuple l_2, k_2 where there is no corresponding value for J)
  - Or do not represent this tuple at all

Testing for 3NF

- Optimization: Need to check only FDs in *F*, need not check all FDs in F^+.
- Use attribute closure to check (for each dependency $\alpha \rightarrow \beta$) whether α is a superkey.
- If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of *R*
  - this test is rather more expensive, since it involves finding candidate keys
  - testing for 3NF has been shown to be NP-hard
  - Interestingly, **decomposition** into third normal form (described shortly) can be done in polynomial time

3NF Decomposition Algorithm

Let F_c be a canonical cover for *F;*

  i := 0;
  for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**
   if none of the schemas R_j, $1 \leq j \leq i$ contains $\alpha\ \beta$
      then begin
         $i := i + 1$;
         $R_i := \alpha\ \beta$
       end
  if none of the schemas R_j, $1 \leq j \leq i$ contains a candidate key for R
   then begin
       $i := i + 1$;
       R_i := any candidate key for R;
     end
  /* Optionally, remove redundant relations */

  repeat
  if any schema R_j is contained in another schema R_k
     then /* delete R_j ***/**
       $R_j = R_i$;
       $i=i-1$;
  return $(R_1, R_2, ..., R_i)$

3NF Decomposition Algorithm (Cont.)

- Above algorithm ensures:
  - each relation schema R_i is in 3NF
  - decomposition is dependency preserving and lossless-join

3NF Decomposition: An Example

- Relation schema:

  cust_banker_branch = (*<u>customer_id, employee_id</u>*, *branch_name*, *type*)

- The functional dependencies for this relation schema are:

  1. *customer_id*, *employee_id* $\rightarrow$ *branch_name*, *type*
  2. *employee_id* $\rightarrow$ *branch_name*
  3. *customer_id*, *branch_name* $\rightarrow$ *employee_id*

- We first compute a canonical cover

  - ***Any extraneous attribute in any functional dependency?***

3NF Decomposition: An Example

- Relation schema:

  cust_banker_branch = (*<u>customer_id, employee_id</u>, branch_name, type*)

- The functional dependencies for this relation schema are:
  1. *customer_id, employee_id* $\rightarrow$ *branch_name, type*
  2. *employee_id* $\rightarrow$ *branch_name*
  3. *customer_id, branch_name* $\rightarrow$ *employee_id*
- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st dependency
    1. compute α^+ using only the dependencies in

       $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,
    2. check that α^+ contains *A;* if it does*, A* is extraneous in β
  - No other attribute is extraneous, so we get F_C =

    customer_id, employee_id $\rightarrow$ *type*
    employee_id $\rightarrow$ *branch_name*
    customer_id, branch_name $\rightarrow$ *employee_id*

3NF Decomposition Example (Cont.)

- The **for** loop generates following 3NF schema:

  (*customer_id, employee_id, type*)

  (*employee_id, branch_name*)

  (*customer_id, branch_name, employee_id)*

  - Observe that (*customer_id, employee_id, type*) contains a candidate key of the original schema, so no further relation schema needs be added
- At end of for loop, detect and delete schemas, such as (*employee_id, branch_name*), which are subsets of other schemas
  - result will not depend on the order in which FDs are considered
- The resultant 3NF schema is:

  (*customer_id, employee_id, type*)

  (*customer_id, branch_name, employee_id)*

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF :
  - such that the decomposition is lossless
  - the dependencies may not be preserved

Design Goals

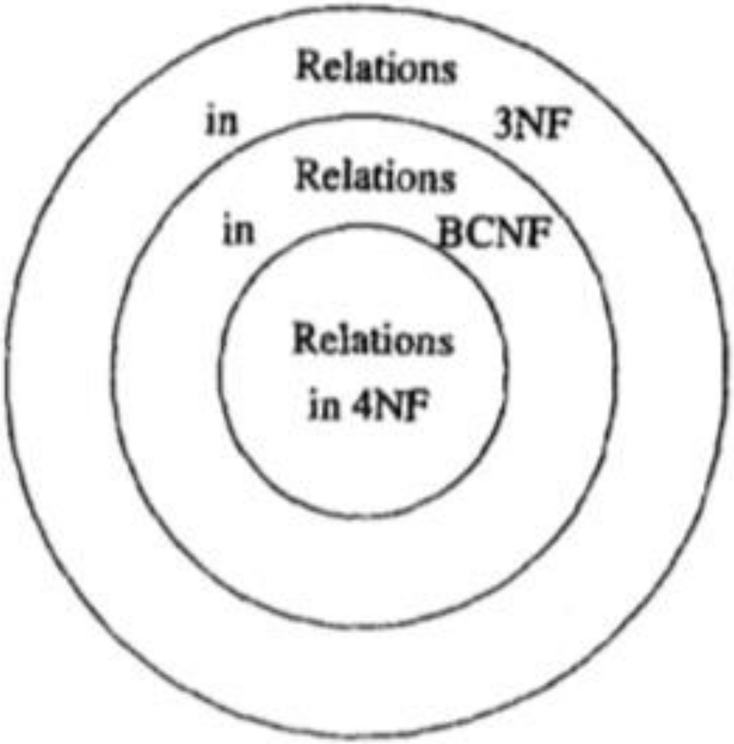
- Goal for a relational database design is:
  - Relations that are in BCNF.
  - Relations that have a Lossless join.
  - Dependency preservation in relations.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- SQL does not provide a direct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

  Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- 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.

Properties preserved by normal forms

Property	3NF	BCNF	4NF
Eliminates redundancy due to FD's	Most	Yes	Yes
Eliminates redundancy due to MVD's	No	No	Yes
Preserves FD's	Yes	Maybe	Maybe
Preserves MVD's	Maybe	Maybe	Maybe

4NF implies BCNF implies 3NF



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**).
  - *R* could have been the result of some ad hoc design of relations, which we then test/convert to normal form.
  - Normalization breaks R into smaller relations

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an *employee* entity with attributes
    department_name and *building*,
    and a functional dependency
    $department\_name \rightarrow building$
  - Good design would have made department an entity
- Functional dependencies from non-key attributes of a relationship set possible, but rare

Denormalization for Performance

- May want to use non-normalized schema for performance
- For example,
  - *course (course_id, title)* and *prereq(course_id, pre)*
  - displaying *prereqs* along with *course_id,* and *title* requires join of *course* with *prereq*

Denormalization for Performance cont.

- Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes (info for course is repeated for every prereq)
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: use a materialized view defined as

  course *prereq*

  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:

  Instead of *earnings* (*company_id, year, amount*), use

  - *earnings_2004, earnings_2005, earnings_2006*, etc., all on the schema (*company_id, earnings*).
    - Above are in BCNF, but make querying across years difficult and needs new table each year
  - *company_year* (*company_id, earnings_2004, earnings_2005, earnings_2006*)
    - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
    - Is an example of a **crosstab**, where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools

End of Chapter

Database System Concepts, 6th Ed.

Edited by Radhika Sukapuram

Proof of Correctness of 3NF Decomposition Algorithm

Database System Concepts, 6^{th} Ed.

Edited by Radhika Sukapuram

Correctness of 3NF Decomposition Algorithm

- 3NF decomposition algorithm is dependency preserving (since there is a relation for every FD in F_c)
- Decomposition is lossless
  - A candidate key (C) is in one of the relations R_i in decomposition
  - Closure of candidate key under F_c must contain all attributes in R.
  - Follow the steps of attribute closure algorithm to show there is only one tuple in the join result for each tuple in R_i

Correctness of 3NF Decomposition Algorithm (Cont'd.)

Claim: if a relation R_i is in the decomposition generated by the above algorithm, then R_i satisfies 3NF.

- Let R_i be generated from the dependency $\alpha \rightarrow \beta$
- Let $\gamma \rightarrow$ B be any non-trivial functional dependency on R_i. (We need only consider FDs whose right-hand side is a single attribute.)
- Now, B can be in either β or α but not in both. Consider each case separately.

Correctness of 3NF Decomposition (Cont'd.)

- Case 1: If B in β:
  - If γ is a superkey, the 2nd condition of 3NF is satisfied
  - Otherwise α must contain some attribute not in γ
  - Since $\gamma \rightarrow B$ is in F^+ it must be derivable from F_c, by using attribute closure on γ.
  - Attribute closure not have used $\alpha \rightarrow \beta$. If it had been used, α must be contained in the attribute closure of γ, which is not possible, since we assumed γ is not a superkey.
  - Now, using $\alpha \rightarrow (\beta - \{B\})$ and $\gamma \rightarrow B$, we can derive $\alpha \rightarrow B$

    (since $\gamma \subseteq \alpha\beta$, and $B \notin \gamma$ since $\gamma \rightarrow B$ is non-trivial)
  - Then, B is extraneous in the right-hand side of $\alpha \rightarrow \beta$; which is not possible since $\alpha \rightarrow \beta$ is in F_c.
  - Thus, if B is in β then γ must be a superkey, and the second condition of 3NF must be satisfied.

Correctness of 3NF Decomposition (Cont'd.)

- Case 2: *B* is in α.
  - Since α is a candidate key, the third alternative in the definition of 3NF is trivially satisfied.
  - In fact, we cannot show that γ is a superkey.
  - This shows exactly why the third alternative is present in the definition of 3NF.

Q.E.D.