Lecture 18

Algorithms for 3NF and BCNF

Week 12

Overview

- 3NF algorithm

  Generating dependency preserving lossless join decompositions (synthesis algorithm)

- BCNF algorithm

  Lossless join decomposition

Generating dependency preserving lossless join decomposition into 3NF (algorithm 15.4 textbook)

- 3NF is always achievable
- This is a synthesis algorithm, creating a decomposition of relation R from the FDs on R rather then through splitting R step by step
- The aim is to generate a decomposition where all FDs have superkeys on the LHS (except possibly FDs with prime attribute on the RHS)

Example for algorithm 15.4

R (<u>A</u> B C D <u>E</u>)

with the FD set:

F={AC→B, A→C, B →C, CB→D, C→D}

Candidate keys: there is only one {AE}

R is not in 3NF, e.g.consider A→C

- $\{A\}_F^+$ = {ACBD} which does not include E, and
- C is not prime attribute in A→C

(A is not a superkey)

Algorithm 15.4

Step 1. Create minimal cover of F

$$\{A \rightarrow B \; A \rightarrow C \; B \rightarrow C \; C \rightarrow D\}$$

Step 2. Group FDs with identical LHSs

$A \rightarrow B$ $\quad$ $B \rightarrow C$ $\quad$ $C \rightarrow D$

$A \rightarrow C$

and create a relation from each group:

$R_1(\underline{A}BC)$ $\quad$ $R_2(\underline{B}C)$ $\quad$ $R_3(\underline{C}D)$

Algorithm 15.4 (cont'd)

Step 3. If no candidate key of R is part of any R_i thus generated, then create a relation from one of the candidate keys

$$R_4(\underline{AE})$$

Explanation

- Step 2 ensures that in any generated R_i the LHS of each functional dependency determines all attributes (or the RHS is a prime attribute) - hence the result is in 3NF
- Step 3 ensures the lossless join property

Just in case...

- Just in case you wondered: does this algorithm ensure that all attributes of R are represented in at least one relation?

- It does, since the only attributes that are not represented in R_i after step 2 are those which were not part of either the LHS nor the RHS of any FD -- which means that they must be part of any candidate key! So if no candidate key is in any R_i a candidate key will be included, in Step 3 at least.

Generating lossless join decomposition into BCNF (algorithm 15.3 E&N)

- Might lose a FD, but it keeps lossless join property (which is essential)
- Even if an FD is lost, the BCNF decomposition may still be better then the 3NF decomposition, should the cost of redundancy be too high and should updates be infrequent (thus, testing at insertion time the violation of the lost FD might be tolerable)

Algorithm 15.3

- **Step 1.** For each relation R_i in a decomposition R_i and a minimal cover of FDs F, which is true of R_i:

  if R_i is not in BCNF then take a FD $X \rightarrow Y$ from F which violates BCNF (where X is not a superkey in R_i) and make it the 'culprit' (go to **Step 2**)

Algorithm 15.3 (cont'd)

- **Step 2.** Split R_i into two parts
  $R_{i1}=\{R_i\}\backslash\{Y\}$ and $R_{i2}=\{XY\}$
- **Step 3.** Try to impose FDs in F on the decomposition. May need to generate an equivalent set of FDs instead of using F, so as to avoid losing an FD (not always possible)
- **Repeat** steps 1, 2 and 3 until all parts are in BCNF

Why Algorithm 15.3 works

- The algorithm obviously terminates after a finite number of steps
- The algorithm eliminates from all relations those FDs which violate BCNF (thus the result must finally be in BCNF)
- The result has the lossless join property, because in R_{i2} X is always a key (since $X \rightarrow Y$). Therefore the intersection of R_{i1} and R_{i2} is key in at least one of the components.

Example for algorithm 15.3

R (A B D E)

FDs F={A→B, A→D, D→A)

Candidate keys:
{AE} and {DE}

Prime attributes:
A, E, D

Step 1. A→B is culprit
(A not superkey)

Steps 2. and 3. Split into two and impose FDs:

R_1 (A<u>DE</u>) {D→A, A→D}

R_2 (<u>A</u>B) {A→B}

Example (cont'd)

- R_2 is in BCNF
- **Step1.** R_1 (A<u>DE</u>) not in BCNF, because D→A violates BCNF (D is not superkey in R_1).
  Make D→A the culprit
- **Step 2.** Split R_1 into R_{11} and R_{12}
  R_{11} = (<u>DE</u>) - is BCNF
  R_{12} = (<u>D</u>A) {D→A, A→D} - is BCNF

Example (cont'd)

The resulting BCNF decomposition of R is:

R_{11} = (<u>DE</u>)		- is in BCNF
R_{12} = (<u>D</u>A)	{D→A, A→D}	- is in BCNF
R_2 (<u>A</u>B)	{A→B}	- is in BCNF

Notes

- Interestingly, the 3NF algorithm often (but not always) creates a decomposition which is also in BCNF
- Refer to tutorial for examples where some FD is lost

Higher normal forms

- 4NF is defined on the basis of so called ‘multivalued dependencies’. These arise when a relation is storing the combination of two independent facts (thus it must store them in all possible combinations)

- We do not study this since DB design should be based on ER schemas and our mapping algorithm never generates such a situation. However, legacy systems may have such problems (refer textbook)

Conclusion

- The goal of relational database design should be to create BCNF (or if not possible in a lossless way, then 3NF) decompositions
- Usual way to progress:
  - create ER schema
  - map to relation schema
  - identify any additional FDs (in addition to known dependencies of attributes on candidate keys)
  - decompose to BCNF (or 3NF)

The end