CSC 355 Database Systems Lecture 12

Eric J. Schwabe

School of Computing, DePaul University

Spring 2020

Today:

- Relational Database Design
  - BCNF decomposition algorithm
  - Dependency preservation
  - Lossless join

Relational Database Design

- Starting with $R(A_1, A_2, \ldots, A_n)$, the decomposition $D = \{R_1, R_2, \ldots, R_m\}$ should satisfy the following conditions:
  1. The union of the R_i's is R
  2. Redundancy has been removed from relations in D
  3. The functional dependencies in R are preserved
  4. The original relation R can be recovered from D
- We are formalizing conditions 2.-4. …

Functional Dependencies

- A set of attributes $Y = \{Y_1, Y_2, \ldots, Y_n\}$ is *functionally dependent* on a set of attributes $X = \{X_1, X_2, \ldots, X_m\}$ if and only if every pair of tuples that have the same values for X must also have the same values for Y
  - Also "X functionally determines Y" or "$X \rightarrow Y$"
  - X is called the *determinant*
- (Less formally, "the values of X uniquely determine the values of Y"…)

Keys

- A set of attributes X is a *superkey* of R if X determines all attributes of R
- A set of attributes X is a *candidate key* of R if X is a superkey, but no proper subset Y of X is a superkey
- An attribute is *prime* if it is contained in some candidate key (and is *non-prime* otherwise)

BCNF

- A relation R is in Boyce-Codd Normal Form (BCNF) if for every non-trivial functional dependency $X \rightarrow Y$ in R, X is a superkey
  - "Every determinant must contain a candidate key"
  - A relation in BCNF will not have any redundancy, since every functional dependency in the relation will have a superkey as its determinant

BCNF Decomposition Algorithm

Set D = {R}

While there is some Q in D that is not in BCNF:

Choose a Q that is not in BCNF

Find an X→Y in Q that violates BCNF

Replace Q with two relations:

Q – Y and (X union Y)

(When algorithm is done, all relations will be in BCNF)

Projections

- Suppose we have a set of functional dependencies F in the relation R
- For a relation R_i, consider all $X \rightarrow Y$ that can be derived from F where both X and Y are subsets of R_i
- This set is called the *projection of F on R_i*
  - It represents the set of all constraints that F puts on the attributes of R_i

Dependency Preservation Property

3. The union over all i in {1,…,m} of the projections of F on R_i is equivalent to F

- We want the set of all projections to be equivalent to F – that is, the decomposition neither destroys any functional dependencies in F nor introduces any new ones…
  - Not all decompositions have this property!

Restrictions

- The *restriction* of a relation state r to a set S is the set of distinct tuples obtained from r by including only the values of the attributes in S from each tuple
- Restrictions that overlap can be combined using a natural join on the attributes they share
  - Ideally, this will yield a result that is still a restriction of the original relation state r…

Lossless Join Property

4. For every relation state r of R, the natural join of the restrictions of r to the relations R_1, R_2, …, R_m in the decomposition the same as r

- That is, if we take the restrictions of any relation state and join them back together, we will get the original relation state – no *spurious tuples* are added
- Not all decompositions have this property!

Relational Database Design

- Starting with $R(A_1, A_2, \ldots, A_n)$, the decomposition $D = \{R_1, R_2, \ldots, R_m\}$ should satisfy the following conditions:
  1. The union of the R_i's is R
  2. Each of the R_i's is in BCNF
  3. D has the dependency preservation property
  4. D has the lossless join property

Binary Lossless Join Test

- $D = \{R_1, R_2\}$ has the lossless join property if and only if one or both of the following hold:
  1. $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ can be derived from F
  2. $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$ can be derived from F
  - That is, if and only if the intersection between the two sets of attributes is a superkey in one of the relations…

General Test for Lossless Join

1. Create a matrix S with a row i for each R_i and a column for j for each A_j
2. Set each S(i,j) = "b_{ij}"
3. For each entry (i,j)

   If relation R_i includes A_j, then set S(i,j) = "a_j"
4. Repeat the following loop until there are no changes to S:

   For each X→Y in F

   For all rows in S that have the same symbols in all columns in X,

   set all of the columns in Y in those rows to agree as follows:

   If any row has a_j for the columns, set all rows to that same a_j

   If not, choose one of the b_{ij}s and set all rows to that same b_{ij}
5. If any row has all a_j's, return true (D has the lossless join property);

   if not, return false (D does not have lossless join property).

Next:

- Finish lossless join
- Third Normal Form (3NF)
- Minimal basis
- Algorithm for 3NF decomposition