

Properties of Decomposition And An Algorithm for BCNF Decomposition

By David Stillion

a presentation prepared for

Dr. Sin-Min Lee

Introduction to Database Management

Fall, 2005

Why Do We Care?

Because no amount of good SQL code will overcome bad table design.

It is very inefficient to do table joins in order to apply and evaluate integrity constraints.

The Theoretical Explanation of Schema Decomposition

Given $R = (R^*; F)$, where R^* is the set of attributes of the schema and F is its set of Functional Dependencies, is a collection of schemas

$$R_1=(R^*_1;F_1), R_2=(R^*_2 ;F_2),\dots, R_n=(R^*_n ;F_n)$$

These conditions must hold:

1. $R_i \neq R_j$, if $i \neq j$
2. $R^* = \bigcup_{i=1}^n R^*_i$
3. F entails F_i for every $i = 1,2,\dots,n$

The 2nd and 3rd conditions are the most interesting

- The second condition states that a decomposition should not introduce new attributes, and it should not drop attributes found in the original table.
- The third condition states that the decomposition should not introduce new functional dependences (but may drop some).

Schema Decomposition vs. Relation Decomposition

The decomposition of a schema naturally leads to the decomposition of relations over it.

The decomposition of a *relation* instance is NOT the same as the decomposition of its *schema*.

Schema Decomposition vs. Relation Decomposition cont.

Give *relation* r and our original *schema* R , a *relation decomposition* is a set of relations

$$r_1 = \pi_{R^*_1}(r), r_2 = \pi_{R^*_2}(r), \dots, r_n = \pi_{R^*_n}(r)$$

Relation r No Longer exists.

r has been decomposed as described in the previous slide and now exists as relations

r_1, r_2, \dots, r_n in the database.

HOWEVER,

the database must be able to reconstruct r from the projections r_1, r_2, \dots, r_n .

How to reconstruct?

- Use any method that works.
- Usually the best and most practical way to do it is through a natural join.

Natural Join must satisfy 3 conditions

1. It is based on an equality.
2. All common attributes participate in the join.
3. One set of common fields appears in the final result.

Thus,

$$r = r_1 \bowtie_X r_2 \bowtie_X \dots \bowtie_X r_n$$

Caveat

Reconstructibility must be a property of schema decomposition and NOT a particular instance over a schema.

Any transformation performed on a schema must guarantee that reconstructibility holds for all of its valid relations.

What is Lossless?

Given $R = (R^*; F)$, where R^* is the set of attributes of the schema and F is its set of Functional Dependencies, is a collection of schemas

$$R_1=(R^*_1;F_1), R_2=(R^*_2 ;F_2),\dots, R_n=(R^*_n ;F_n)$$

is *LOSSLESS*, if for every valid instance r of schema R ,

$$r = r_1 |x| r_2 |x| \dots |x| r_n$$

where

$$r_1 = \pi_{R^*_1}(r), r_2 = \pi_{R^*_2}(r), \dots, r_n = \pi_{R^*_n}(r)$$

otherwise the decomposition is lossy.

What is Losslessness?

Losslessness asserts the following

$$r \supseteq r_1 \mid x \mid r_2 \mid x \mid \dots \mid x \mid r_n$$

A lossy decomposition will return more tuples when joining its projections than in the original undecomposed relation.

The issue becomes what information is lost or which is accurate or complete or which is valid?

With no way to tell the results are meaningless.

Testing Decomposition

Two ways!

- There is a general test of an n schema decomposition that works but it is complex and it difficult to implement.
- Easier and more practical - Binary Decomposition(BD).

With BD, lossless decomposition can be checked as long as the original decomposition was done on a binary basis.

Binary Decomposition Requires

Once again: Given $R = (R^*; F)$, where R^* is the set of attributes of the schema and F is its set of Functional Dependencies, **is a collection of schemas**

$$R_1=(R^*_1;F_1), R_2=(R^*_2 ;F_2),\dots, R_n=(R^*_n ;F_n)$$

This decomposition is lossless iff either of the following is true:

$$(R^*_1 \cap R^*_2) \rightarrow R^*_1 \in F^+$$

$$(R^*_1 \cap R^*_2) \rightarrow R^*_2 \in F^+$$

Dependency-Preserving Decompositions

Consider the decomposition of $\mathbf{R} = (\mathbf{R}^*; \mathbf{F})$, such
that

$$\mathbf{R}_1 = (\mathbf{R}^*_1; \mathbf{F}_1), \mathbf{R}_2 = (\mathbf{R}^*_2; \mathbf{F}_2), \dots, \mathbf{R}_n = (\mathbf{R}^*_n; \mathbf{F}_n).$$

By definition, \mathbf{F} entails \mathbf{F}_i and thus \mathbf{F} entails $\bigcup_{i=1}^n \mathbf{F}_i$
but $\bigcup_{i=1}^n \mathbf{F}_i$ need not entail \mathbf{F} .

That is to say not all functional dependencies need to be
preserved for a decomposition to be dependency preserving.

Dependency-Preserving

If and only if

when the sets of F and $\bigcup_{i=1}^n F_i$ are equivalent is the decomposition of $R = (R^*; F)$ into

$$R_1 = (R^*_1; F_1), R_2 = (R^*_2; F_2), \dots, R_n = (R^*_n; F_n)$$

Dependency preserving

Projection of the set of F on S

Given $R=(R^*;F)$, a relation r , S (a decomposed schema from R) or $S \subseteq R$.

The only FD's guaranteed to hold over $\pi_s(r)$ are $X \rightarrow Y \in F^+$ s.t. $X, Y \subseteq S$ gives rise to $\pi_s(F) = (X \rightarrow Y \mid X \rightarrow Y \in F^+ \text{ and } X \cup Y \subseteq S)$.

For F in $\pi_s(F)$ we must look at all FDs in F^+ not just in F .

Projection of Functional Dependencies...

..opens a way to construct decompositions.

If $R=(R^*;F)$ is a schema and R^*_1, \dots, R^*_n are subsets of attributes s.t. $R=U_{i=1}^n R_i$, then the schemas $(R_1; \pi_{R_1}(F_a)), \dots, (R_n; \pi_{R_n}(F_z))$ are a decomposition that preserves as many of the original FDs as possible.

Again, F must reflect the closure of F not just F .

BCNF and Decomposition

Properties of a relation in Boyce-Codd Normal Form -

Already in 3NF, 2NF and 1NF and
all determinants are Super Keys.

The goal of relations in BCNF is to eliminate redundancy due to functional dependencies.

The Algorithm for BCNF Decomposition

Input: $R = (R^*; F)$

Decomposition := /* Initially *Decomposition* consists of only one schema*/

while

there is a schema $S = (S^*; F')$ in *Decomposition* that is not in BCNF

do

/* Let $X \rightarrow Y$ be a FD in F s.t. $X, Y \subseteq S$ and it violates BCNF in S .

Decompose using this FD */

Replace S in *Decomposition* with schema $S_1 = (X, Y; F_1)$ and $S_2 = ((S^* - Y)$

$\cup X; F_2)$, where $F_1 = \pi_{XY}(F')$ and $F_2 = \pi_{((S^* - Y) \cup X)}(F')$

end

return *Decomposition*

The Algorithm is Very Clear...

Except

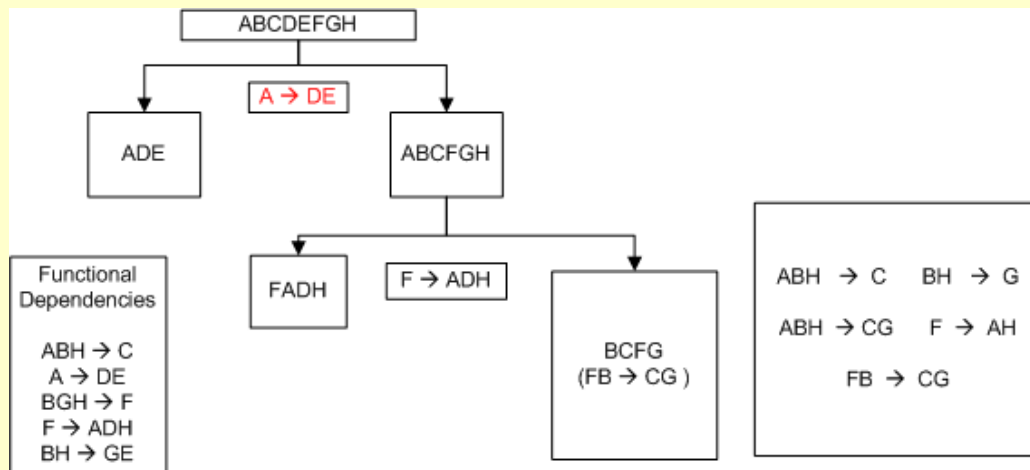
No part of the algorithm addresses how to apply Armstrong's Axioms - Reflexivity, Augmentation and Transitivity at each iteration to determine F^+ .

The algorithm simply says that we choose the next FD not in BCNF.

For Example

Consider schema $R=(R^*;F)$ where $R^*= ABCDEFGH$, and the set of F being $(ABH \rightarrow C, A \rightarrow DE, BGH \rightarrow F, F \rightarrow ADH, \text{ and } BH \rightarrow GE)$

When we start the *Decomposition*, $R=S$ and algorithm selects an FD not in BCNF



Our Resulting Relations

Relations and FD

R1 (ADE; $A \rightarrow DE$)

R2(FAH; $F \rightarrow AH$)

R3 (BCFG; $FB \rightarrow CG$)

Is this a lossless decomposition? - YES

Is this a dependency-preserving
decomposition? - NO

Why not? Because F and $\bigcup_{i=1}^n F_i$ are not
equivalent.

What Can we Count On?

1. A BCNF decomposition is always lossless.
2. BCNF decompositions do not need to be dependency preserving.
3. The BCNF decomposition is nondeterministic.

This means different decompositions occur depending on the order in which FDs are selected in the while loop.

The Database Designer Decides

The order of selection of FDs for the Decomposition may be a matter of taste.

Objective criteria be developed and and applied to achieve a particular decomposition.

For example, the decomposition may need to support a particular type or set of queries. So the decomposition would be done to support this group of pre-defined queries.

References

Database Systems - *An Application-Oriented Approach*, 2nd Edition. Michael Kifer, Arthur Bernstein and Philip M. Lewis. Pearson Education, 2006.