Normalization

Schema Decomposition, Normal Forms

Tamer Özsu

David R. Cheriton School of Computer Science
University of Waterloo

CS 640
Principles of Database Management and Use
Winter 2013

Notes

Outline

1. Schema Decomposition
   - Lossless-Join Decompositions
   - Dependency Preservation
2. Normal Forms based on FDs
   - Boyce-Codd Normal Form
   - Third Normal Form

Notes

Schema Decomposition

Definition (Schema Decomposition)

Let R be a set of attributes.
A **decomposition** of R is a set $\{R_1, R_2, \ldots, R_n\}$ such that:

$$R = R_1 \cup R_2 \cup \cdots \cup R_n.$$

A good decomposition does not

- lose information
- complicate checking of constraints
- contain anomalies (or at least contains fewer anomalies)

Notes

Lossless-Join Decompositions

We should be able to construct the instance of the original table from the instances of the tables in the decomposition

Example: Consider replacing

Marks

Student	Assignment	Group	Mark
Ann	A1	G1	80
Ann	A2	G3	60
Bob	A1	G2	60

by decomposing (i.e. projecting) into two tables:

SGM

Student	Group	Mark
Ann	G1	80
Ann	G3	60
Bob	G2	60

AM

Assignment	Mark
A1	80
A2	60
A1	60

Notes

Lossless-Join Decompositions (cont.)

But computing the natural join of SGM and AM produces

Student	Assignment	Group	Mark
Ann	A1	G1	80
Ann	A2	G3	60
Ann	A1	G3	60
Bob	A2	G2	60
Bob	A1	G2	60

. . . and we get extra data (spurious tuples). We would therefore lose information if we were to replace Marks by SGM and AM.

If re-joining SGM and AM would always produce exactly the tuples in Marks, then we call SGM and AM a **lossless-join decomposition**.

Notes

Lossless-Join Decompositions (cont.)

A decomposition $\{R_1, R_2\}$ of R is lossless if and only if the common attributes of R_1 and R_2 form a superkey for either schema, that is

$$R_1 \cap R_2 \to R_1 \quad \text{or} \quad R_1 \cap R_2 \to R_2$$

Example: In the previous example we had

$$R = \{\text{Student, Assignment, Group, Mark}\},$$
$$\Sigma = \{\{\text{Student, Assignment} \to \text{Group, Mark}\}\},$$

$$R_1 = \{\text{Student, Group, Mark}\},$$
$$R_2 = \{\text{Assignment, Mark}\}.$$

Decomposition $\{R_1, R_2\}$ is lossy because $R_1 \cap R_2 = \{\text{Mark}\}$ is not a superkey of either $\{\text{Student, Group, Mark}\}$ or $\{\text{Assignment, Mark}\}$.

Notes

Dependency Preservation

How do we test/enforce constraints on the decomposed schema?

Example: A table for a company database could be

R

Proj	Dept	Div

FD1: Proj $\rightarrow$ Dept,
FD2: Dept $\rightarrow$ Div, and
FD3: Proj $\rightarrow$ Div

and two decompositions

D_1 = {R1[Proj, Dept], R2[Dept, Div]}
D_2 = {R1[Proj, Dept], R3[Proj, Div]}

Both are lossless. (Why?)

Notes

Dependency Preservation (cont.)

Which decomposition is *better*?

- Decomposition D_1 lets us test FD1 on table R1 and FD2 on table R2; if they are both satisfied, FD3 is automatically satisfied.
- In decomposition D_2 we can test FD1 on table R1 and FD3 on table R3. Dependency FD2 is an **interrelational constraint**: testing it requires joining tables R1 and R3.

$\Rightarrow$ D_1 is better!

Let Σ be a set of functional dependencies over a set of attributes R. A decomposition $D = \{R_1, \ldots, R_n\}$ of R is **dependency preserving** if there is an equivalent set of functional dependencies Σ', none of which is interrelational in D.

Notes

Normal Forms

What is a "good" relational database schema?

Rule of thumb: Independent facts in separate tables:

"Each relation schema should consist of a primary key and a set of mutually independent attributes"

This is achieved by transforming a schema into a normal form.

Goals:

- Intuitive and straightforward transformation
- Anomaly-free/Nonredundant representation of data

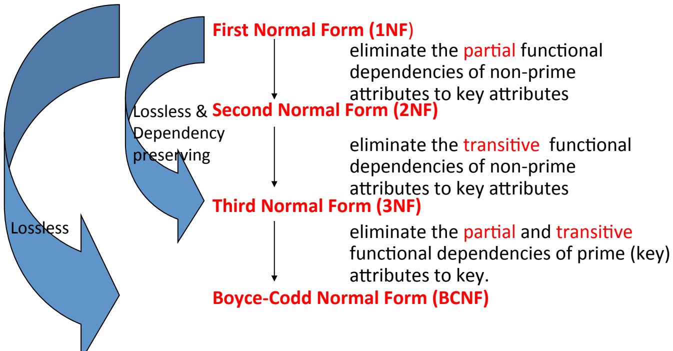
Normal Forms based on Functional Dependencies:

- Boyce-Codd Normal Form (BCNF)
- Third Normal Form (3NF)

Notes

Normal Forms Based on FDs

1NF eliminates relations within relations or relations as attributes of tuples



Notes

Boyce-Codd Normal Form (BCNF) - Informal

- BCNF formalizes the goal that in a good database schema, independent relationships are stored in separate tables.
- Given a database schema and a set of functional dependencies for the attributes in the schema, we can determine whether the schema is in BCNF. A database schema is in BCNF if each of its relation schemas is in BCNF.
- Informally, a relation schema is in BCNF if and only if any group of its attributes that functionally determines *any* others of its attributes functionally determines *all* others, i.e., that group of attributes is a superkey of the relation.

Notes

Formal Definition of BCNF

Let (R, Σ) be a relational schema (i.e. Σ are FDs over R).

This schema is in **BCNF** if and only if for each $(X \to Y) \in \Sigma^+$ it holds that either

- $(X \to Y)$ is trivial (i.e., $Y \subseteq X$), or
- X is a superkey of the schema.

A database schema is in BCNF if all of its relation schemas are in BCNF.

Notes

BCNF and Redundancy

- Why does BCNF avoid redundancy? Consider:

Supplied_Items

Sno	Sname	City	Pno	Pname	Price

- The following functional dependency holds:

$$\{\text{Sno}\} \rightarrow \{\text{Sname, City}\}$$

- Therefore, supplier name "Magna" and city "Ajax" must be repeated for each item supplied by supplier S1.
- Assume a relational schema in BCNF that includes the above FD. This implies that:
  - Sno is a superkey for this schema
  - each Sno value appears on one row only
  - no need to repeat Sname and City values

Notes

Lossless-Join BCNF Decomposition

function $DecomposeBCNF(R, \Sigma)$
begin
  Result := $\{R\}$;
  while some $R_i \in$ Result and $(X \rightarrow Y) \in \Sigma^+$
    violate the BCNF condition **do begin**
   Replace R_i by $R_i - (Y - X)$;
   Add $\{X, Y\}$ to Result;
  end;
  return Result;
end

Notes

Lossless-Join BCNF Decomposition

- No *efficient* procedure to do this exists.
- Results depend on sequence of FDs used to decompose the relations.
- It is possible that no lossless join dependency preserving BCNF decomposition exists
  - Consider $R = \{A, B, C\}$ and $\Sigma = \{AB \rightarrow C, C \rightarrow B\}$.

Notes

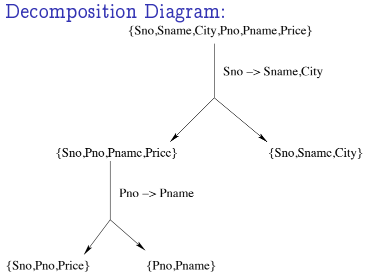
BCNF Decomposition - An Example

- $R = \{\text{Sno,Sname,City,Pno,Pname,Price}\}$
- Functional dependencies:
  Sno $\rightarrow$ Sname,City
  Pno $\rightarrow$ Pname
  Sno,Pno $\rightarrow$ Price
- This schema is not in BCNF because, for example, Sno determines Sname and City, but is not a superkey of R.

Notes

BCNF Decomposition - An Example (cont.)

Decomposition Diagram:



- The complete schema is now
  $R_1 = \{\text{Sno,Sname,City}\}$
  $R_2 = \{\text{Sno,Pno,Price}\}$
  $R_3 = \{\text{Pno,Pname}\}$
- This schema is a lossless-join, BCNF decomposition of the original schema R.

Notes

Third Normal Form (3NF)

Let (R, Σ) be a relational schema (i.e. Σ are FDs over R).

This schema is in **3NF** if and only if for each $(X \rightarrow Y) \in \Sigma^+$ it holds that either

- $(X \rightarrow Y)$ is trivial, or
- X is a superkey of the schema, or
- each attribute in $Y - X$ is contained in a candidate key of R.

A database schema is in 3NF if all of its relation schemas are in 3NF.

- 3NF is looser than BCNF
  - allows more redundancy
  - e.g. $R = \{A, B, C\}$ and $\Sigma = \{AB \rightarrow C, C \rightarrow B\}$.
- lossless-join, dependency-preserving decomposition into 3NF relation schemas always exists.

Notes

Minimal Cover

Definition: Two sets of functional dependencies Σ and Γ (over the same set of attributes) are **equivalent** if and only if $\Sigma^+ = \Gamma^+$.

There are different sets of functional dependencies that have the same logical implications. Simple sets are desirable.

Definition: A set of functional dependencies Σ is **minimal** if

1. every right-hand side of an FD in Σ is a single attribute, and
2. for no $X \to A$ is the set $\Sigma - \{X \to A\}$ equivalent to Σ, and
3. for no $X \to A$ and Z a proper subset of X is the set $\Sigma - \{X \to A\} \cup \{Z \to A\}$ equivalent to Σ.

Theorem: For every set of functional dependencies Σ there exists an equivalent minimal set of functional dependencies (**minimal cover**).

Notes

Finding Minimal Covers

A minimal cover for Σ can be computed in three steps. Note that each step must be repeated until it no longer succeeds in updating Σ.

Step 1.
Replace $X \to YZ$ with the pair $X \to Y$ and $X \to Z$.

Step 2.
Remove A from the left-hand-side of $X \to B$ in Σ if
B is in $\mathit{ComputeAttrClosure}(X - \{A\}, \Sigma)$.

Step 3.
Remove $X \to A$ from Σ if
$A \in \mathit{ComputeAttrClosure}(X, \Sigma - \{X \to A\})$.

Notes

Dependency-Preserving 3NF Decomposition

Idea: Decompose into 3NF relations and then "repair"

```
function Decompose3NF(R, Σ)
begin
    Result := {R};
    while some R_i ∈ Result and (X → Y) ∈ Σ^+
            violate the 3NF condition do begin
        Replace R_i by R_i − (Y − X);
        Add {X, Y} to Result;
    end;
    N := (a minimal cover for Σ) − (⋃_i Σ_i)^+
    for each (X → Y) ∈ N do
        Add {X, Y} to Result;
    end;
    return Result;
end
```


Notes

Dep-Preserving 3NF Decomposition - An Example

- $R = \{$Sno,Sname,City,Pno,Pname,Price$\}$
- Functional dependencies:
  Sno $\rightarrow$ Sname,City   Pno $\rightarrow$ Pname
  Sno,Pno $\rightarrow$ Price   Sno, Pname $\rightarrow$ Price
- Following same decomposition tree as BCNF example:
  $R_1 = \{$Sno,Sname,City$\}$
  $R_2 = \{$Sno,Pno,Price$\}$
  $R_3 = \{$Pno,Pname$\}$
- Minimal cover:
  Sno $\rightarrow$ Sname   Pno $\rightarrow$ Pname
  Sno $\rightarrow$ City   Sno, Pname $\rightarrow$ Price
- Add relation to preserve missing dependency
  $R_4 = \{$Sno, Pname, Price$\}$

Notes

3NF Synthesis

A lossless-join 3NF decomposition that is dependency preserving can be efficiently computed

```
function Synthesize3NF(R, Σ)
begin
    Result := ∅;
    Δ := a minimal cover for Σ;
    for each (X → Y) ∈ Δ do
        Result := Result ∪ {XY};
    if there is no R_i ∈ Result such that
        R_i contains a candidate key for R then begin
          compute a candidate key K for R;
          Result := Result ∪ {K};
    end;
    return Result;
end
```


Notes

3NF Synthesis - An Example

- $R = \{$Sno,Sname,City,Pno,Pname,Price$\}$
- Functional dependencies:
  Sno $\rightarrow$ Sname,City   Pno $\rightarrow$ Pname
  Sno,Pno $\rightarrow$ Price   Sno, Pname $\rightarrow$ Price
- Minimal cover:
  Sno $\rightarrow$ Sname $R_1 = \{$Sno, Sname$\}$
  Sno $\rightarrow$ City $R_2 = \{$Sno, City$\}$
  Pno $\rightarrow$ Pname $R_3 = \{$Pno, Pname$\}$
  Sno, Pname $\rightarrow$ Price $R_4 = \{$Sno, Pname, Price$\}$
- Add relation for candidate key $R_5 = \{$Sno, Pno$\}$
- Optimization: combine relations R_1 and R_2 (same key)

Notes

Summary

- Functional dependencies provide clues towards elimination of (some) *redundancies* in a relational schema.
- Goals: to decompose relational schemas in such a way that the decomposition is

  (1) lossless-join
  (2) dependency preserving
  (3) BCNF (and if we fail here, at least 3NF)

Notes

Notes

Notes