Schema Refinement, Functional Dependencies and Normal Form (cont.)

EGCI 321: LECTURE12 (WEEK 9)

Outline

Introduction
- Problems due to Poor Designs

Functional Dependencies
- Logical Implications of FDs
- Attribute Closure

Schema Decomposition
- Lossless-Join Decompositions
- Dependency Preservation

Normal Forms based on FDs
- Boyce-Codd Normal Form
- Third Normal Form

Review: Functional Dependency

Relationship: 1-to-1 or many-to-1

- x→y : x determine y or
- y depends on X

Functional Dependency

- FDs : *determinant-attribute → dependency-attribute*
- FDs : PERSON_ID → PERSON_NAME

1 determinant : 1 dependency

- PERSON_ID → PERSON_NAME

1 determinant : many dependency

- PERSON_ID → FNAME, LNAME, ADDRESS, BIRTH_DATE, ISSUE_DATE

Many determinant : 1 dependency

- PRODUCT_LINE, ITEM_NO → USED_QTY

Review: Functional Dependency (cont.)

Example: Full Functional Dependency

D1 : *PERSON_ID* → ADDRESS

D2 : PERSON_ID, PERSON_NAME → ADDRESS

D3 *: PRODUCT_LINE, ITEM_NO* → USE_QTY

D4 : PRODUCT_LINE, ITEM_NO, MANAGER → USE_QTY

- D1 and D3 is Full FD (smallest determinant or smallest set of attributes that can determine a dependency)

Advantages of Normalization

- Reduce database space
- Remove redundancy
- Reduce error during updating
- Reduce anomaly when deletion and insertion
- Stability of database structure

Schema Decomposition

Definition (Schema Decomposition)

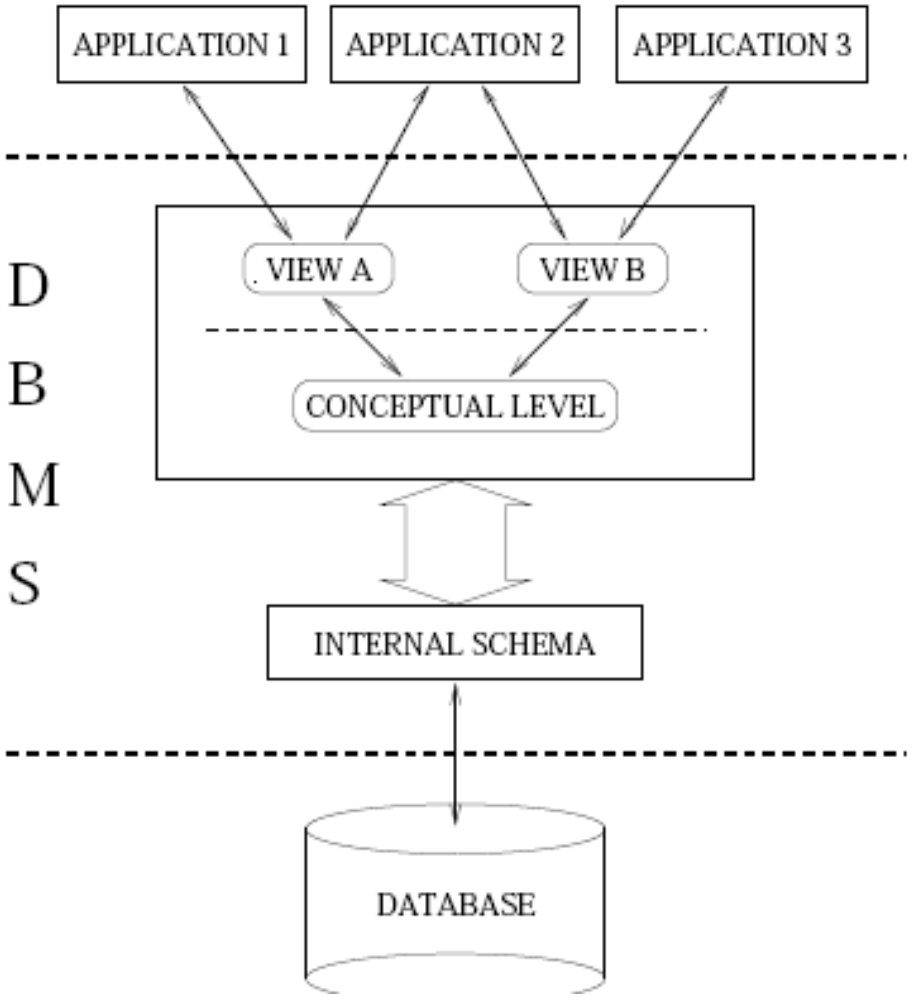
Let R be a relation schema (= set of attributes). The collection $\{R_1, \ldots, R_n\}$ of relation schemas is a decomposition of R if

$$R = R_1 \cup R_2 \cup \ldots \cup R_n$$

Good decomposition does not:

- Lose information
- Complicate checking of constraints
- Contain anomalies (or at least contains fewer anomalies)

Three-level Schema Architecture



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

Lossless-Join Decompositions (cont.)

Computing the natural join of SGM and AM produces and we get extra data (spurious tuples). We would lose information if we were to replace Marks by SGM and AM.

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

Not a lossless-Join Decomposition

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

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

$$R_1 \cap R_2 \rightarrow R_1 \text{ or } R_1 \cap R_2 \rightarrow R_2$$

Example: In the previous example we had

R = {Student, Assignment, Group, Mark} ,

F = {(Student, Assignment, Group, Mark)} ,

R_1 = {Student, Group, Mark} ,

R_2 = {Assignment, Mark}

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

Dependency Preservation

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

Example: A table for a company database could be:

R

Proj	Dept	Div

FD1: Proj → Dept,
FD2: Dept → Div, and
FD3: Proj → Div

Two decompositions

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

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

Both are lossless (Why?)

Dependency Preservation (cont.)

Which decomposition is better?

- Decomposition D_1 lets us test *FD1* on table R_1, and *FD2* on table R_2; if they are both satisfied, *FD3* is automatically satisfied. *(transitive functional dependency)*
- In decomposition D_2 we can test *FD1* on table R_1 and *FD3* on table R_3. Dependency D_2 is an interrelational constraint: testing it requires joining tables R_1 and R_3

$\Rightarrow D_1$ is better

Given a schema R and a set of functional dependencies F, decomposition $D = \{R_1, \ldots, R_n\}$ of R is dependency preserving if there is an equivalent set of functional dependencies F', none of which is interrelational constraint.

Normal Forms

What is a "good" relational database schema?

- 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)

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 that ***functionally determined*** *all* others
  - i.e., that group of attributes is a superkey of the relation

Formal Definition of BCNF

Let R be a relation schema and F a set of functional dependencies

Schema R is in **BCNF** (w.r.t. F) if and only if whenever $(X \to Y) \in F^+$ and $XY \subseteq R$, then either

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

Database schema $\{ R_1, \ldots, R_n \}$ is in BCNF if each relation schema R_i is in BCNF

Functional dependency

- Functional dependency FD: X → Y is called trivial if Y is a subset of X.

- Boyce-Codd normal form (or BCNF)
  - A table is in BCNF if and only if for every one of its non-trivial functional dependencies *X* → *Y*, *X* is a superkey—that is, *X* is either a candidate key or a superset thereof.
  - (Y is not included in X)

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 the above FD holds over a schema *R* that is in BCNF. This is implies that:
  - Sno is a superkey for *R*
  - Each Sno value appears on one row only
  - No need to repeat Sname and City values

Lossless-Join BCNF Decomposition

```
function DecomposeBCNF(R, F)
begin
    Result := {R};
    while some R_i ∈ Result and (X → Y) ∈ F+
            violate the BCNF condition do begin
        Replace R_i by R_i − (Y − X);
        Add {X, Y} to Result;
    end;
    return Result;
end
```

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 F = { AB $\rightarrow$ C, C $\rightarrow$ B}.

BCNF Decomposition – An Example

R = { Sno, Sname, City, Pno, Pname, Price}

Functional dependencies:

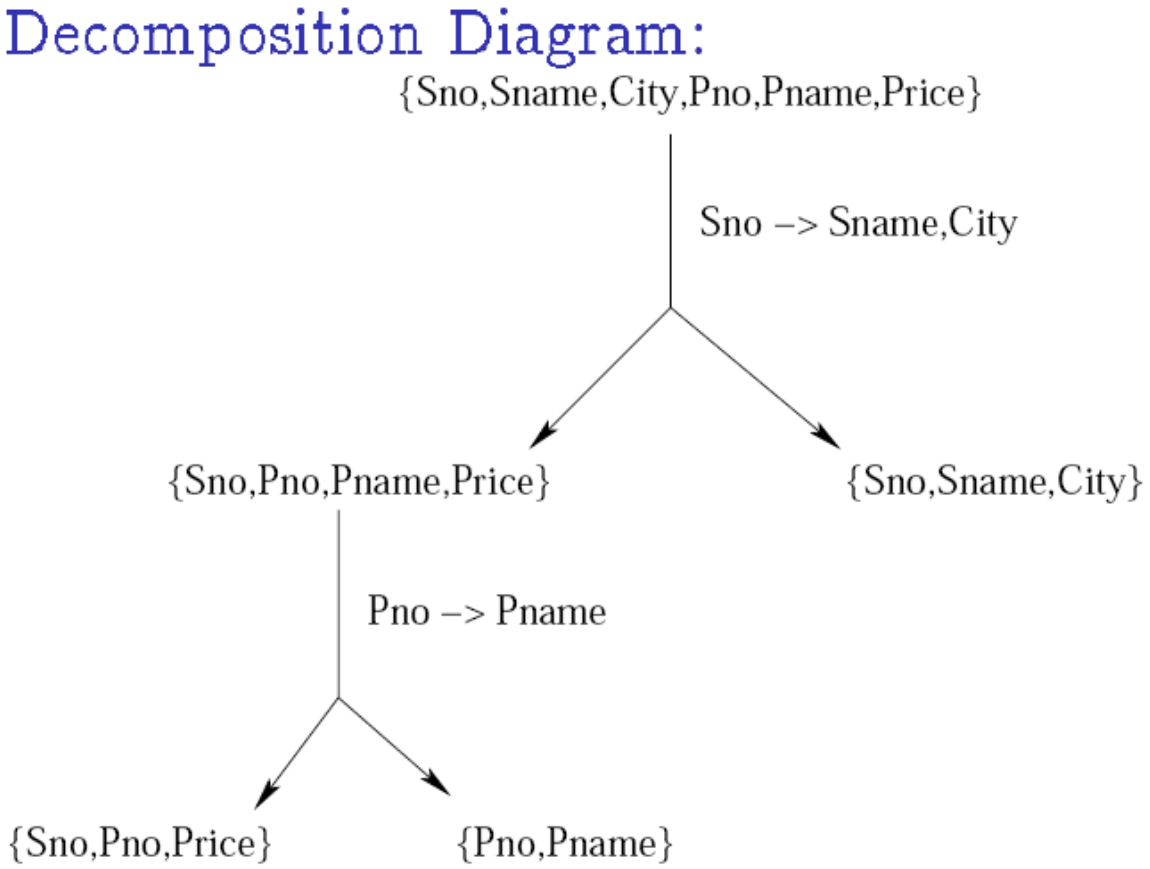
Sno $\rightarrow$ Sname, City

Pno $\rightarrow$ Pname

Sno, Pno $\rightarrow$ Price

This schema is not in BCNF because, for example, Sno determined Sname and City, but is not a superkey of *R*

BCNF Decomposition – An Example (cont.)



The complete schema is now

R_1 = { Sno, Sname, City }

R_2 = { Sno, Pno, Price }

R_3 = { Pno, Pname }

This schema is a lossless-join, BCNF decomposition of the original schema *R*.

Third Normal Form (3NF)

- Schema R is in 3NF (w.r.t. F) if and only if whenever $(X \rightarrow Y) \in F^+$ and $XY \subseteq R$, then either
  - ($X \rightarrow Y$) is trivial, or
  - X is a superkey of R, or
  - Each attribute of Y contained in a candidate key of R

A database schema { $R_1, \ldots, R_n$ } is in 3NF if each relation schema R_i is in 3NF.

3NF is losser than BCNF

- Allows more redundancy
- E.g. R = { A, B, C } and F = { AB $\rightarrow$ C, C $\rightarrow$ B }.

Lossless-join, dependency-preserving *decomposition* into 3NF relation schemas always exists.

Minimal Cover

Definition: Two sets of dependencies F and G are **equivalent** iff $F^+ = G^+$

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

Definition: A set of dependencies *G* is **minimal** if

1. Every right-hand side of an dependency in *F* is a single attribute.
2. For no $X \rightarrow A$ is the set $F - \{X \rightarrow A\}$ equivalent to *F*.
3. For no $X \rightarrow A$ and *Z* a proper subset of X is the set

   $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ equivalent to *F*.

Theorem: for every set of dependencies F there is an equivalent minimal set of dependencies **(minimal cover)** .

Finding Minimal Covers

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

Step1.

Replace $X \rightarrow YZ$ with the pair $X \rightarrow Y$ and $X \rightarrow Z$

Step 2.

Remove *A* from the left-hand-side of $X \rightarrow B$ in *F*

if *B* is in $ComputeX^{+}\ (X - \{ A \}, F)$.

Step 3.

Remove $X \rightarrow A$ from *F* if $A \in ComputeX^{+}\ (X, F - \{ X \rightarrow A \})$

Minimal Cover (cont.)

1. Put the FDs in a standard from:
   - Obtain a collection G of equivalent FDs with a single attribute on the rhs.
2. Minimise the lhs of each FD:
   - For each FD in G, check each attribute in the lhs to see if it can be deleted while preserving equivalence of F+.
3. Delete redundant FDs:
   - Check each remaining FD to see if G can be deleted while preserving equivalence to F+.

Minimal Cover: Example

- Let F be the FDs:

  A->B, ABCD->E, EF->G, EF->H, ACDF->EG.

- Make rhs single attributes:

  ACDF->E, ACDF->G

- The dependency ACDF->G is implied by the following FDs, so we can delete it:

  A->B, ABCD->E, EF->G

- Similarly we can delete ACDF-> E.

- Since A->B, ABCD->E,
  - We can replace the second FD by ACD->E.

- Thus the minimal cover is

  A->B, ACD->E, EF->G, EF->H.

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)

Reference

1. Ramakrishnan R, Gehrke J., Database management systems, 3rd ed., New York (NY): McGraw-Hill, 2003.