Schema Refinement: Dependencies and Normal Forms

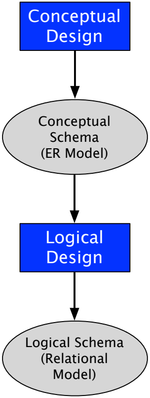
Grant Weddell

Cheriton School of Computer Science
University of Waterloo

CS 348
Introduction to Database Management
Winter 2017

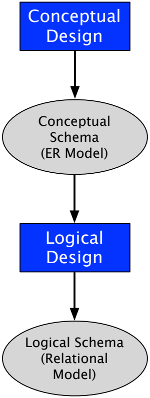
Outline

Design Process – Where are we?



Step 1 – ER-to-relational mapping: obtaining an initial design

Design Process – Where are we?



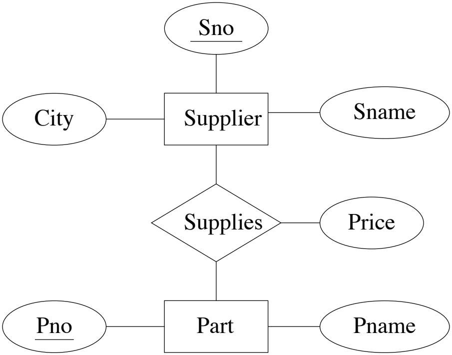
Step 1 – ER-to-relational mapping: obtaining an initial design
Step 2 – Normalization: diagnosing and improving a design

Relational Design Principles

- Relations should have semantic unity
- Information repetition should be avoided
  - Anomalies: insertion, deletion, modification
- Avoid null values as much as possible
  - Certainly avoid excessive null values
- Avoid spurious joins

A Parts/Suppliers Database Example

- Description of a parts/suppliers database:
    - Each type of part has a name and an identifying number, and may be supplied by zero or more suppliers. Each supplier may offer the part at a different price.
    - Each supplier has an identifying number, a name, and a contact location for ordering parts.



Parts/Suppliers Example (cont.)

Suppliers

Sno	Sname	City
S1	Magna	Ajax
S2	Budd	Hull

Parts

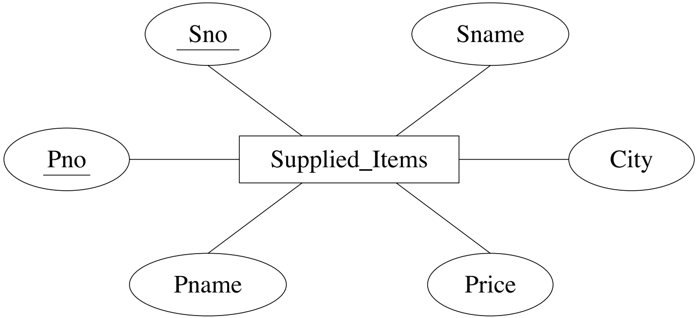
Pno	Pname
P1	Bolt
P2	Nut
P3	Screw

Supplies

Sno	Pno	Price
S1	P1	0.50
S1	P2	0.25
S1	P3	0.30
S2	P3	0.40

An instance of the parts/suppliers database.

Alternative Parts/Suppliers Database



An alternative E-R model for the parts/suppliers database.

Alternative Example (cont.)

Supplied_Items

Sno	Sname	City	Pno	Pname	Price
S1	Magna	Ajax	P1	Bolt	0.50
S1	Magna	Ajax	P2	Nut	0.25
S1	Magna	Ajax	P3	Screw	0.30
S2	Budd	Hull	P3	Screw	0.40

A database instance corresponding to the alternative E-R model.

Change Anomalies

Consider

- Is one schema better than the other?
- What does it mean for a schema to be good?

Change Anomalies

Consider

- Is one schema better than the other?
- What does it mean for a schema to be good?

- The single-table schema suffers from several kinds of problems:
  - Update problems (e.g. changing name of supplier)
  - Insert problems (e.g. add a new item)
  - Delete problems (e.g. Budd no longer supplies screws)
  - Likely increase in space requirements
- The multi-table schema does not have these problems.

Another Alternative Parts/Supplier Database

Is more tables always better?

Snos

Sno
S1
S2

Snames

Sname
Magna
Budd

Cities

City
Ajax
Hull

Inums

Inum
I1
I2
I3

Inames

Iname
Bolt
Nut
Screw

Prices

Price
0.50
0.25
0.30
0.40

Another Alternative Parts/Supplier Database

Is more tables always better?

Snos

Sno
S1
S2

Snames

Sname
Magna
Budd

Cities

City
Ajax
Hull

Inums

Inum
I1
I2
I3

Inames

Iname
Bolt
Nut
Screw

Prices

Price
0.50
0.25
0.30
0.40

Information about relationships is lost!

Designing Good Databases

Goals

- A methodology for evaluating schemas (detecting anomalies).
- A methodology for transforming bad schemas into good schemas (repairing anomalies).

Designing Good Databases

Goals

- A methodology for evaluating schemas (detecting anomalies).
- A methodology for transforming bad schemas into good schemas (repairing anomalies).

- How do we know an anomaly exists?
  - Certain types of *integrity constraints* reveal regularities in database instances that lead to anomalies.

Designing Good Databases

Goals

- A methodology for evaluating schemas (detecting anomalies).
- A methodology for transforming bad schemas into good schemas (repairing anomalies).

- How do we know an anomaly exists?
  - Certain types of *integrity constraints* reveal regularities in database instances that lead to anomalies.
- What should we do if an anomaly exists?
  - Certain *schema decompositions* can avoid anomalies while retaining all information in the instances

Functional Dependencies (FDs)

Idea: Express the fact that in a relation **schema** (values of) a set of attributes uniquely **determine** (values of) another set of attributes.

Functional Dependencies (FDs)

Idea: Express the fact that in a relation **schema** (values of) a set of attributes uniquely **determine** (values of) another set of attributes.

Definition (Functional Dependency)

Let R be a relation schema, and $X, Y \subseteq R$ sets of attributes. The **functional dependency**

$$X \rightarrow Y$$

holds on R if whenever an instance of R contains two tuples t and u such that $t[X] = u[X]$ then it is also true that $t[Y] = u[Y]$.

We say that X *functionally determines* Y (in R).

Notation: $t[A_1, \ldots, A_k]$ means projection of tuple t onto the attributes $A_1, \ldots, A_k$. In other words, $(t.A_1, \ldots, t.A_k)$.

Examples of Functional Dependencies

Consider the following relation schema:

EmpProj

<u>SIN</u>	<u>PNum</u>	Hours	EName	PName	PLoc	Allowance

- SIN determines employee name

  SIN $\rightarrow$ EName

- project number determines project name and location

  PNum $\rightarrow$ PName, PLoc

- allowances are always the same for the same number of hours at the same location

  PLoc, Hours $\rightarrow$ Allowance

Functional Dependencies and Keys

- Keys (as defined previously):
  - A **superkey** is a set of attributes such that no two tuples (in an instance) agree on their values for those attributes.
  - A **candidate key** is a *minimal* superkey.
  - A **primary key** is a candidate key chosen by the DBA

Functional Dependencies and Keys

- Keys (as defined previously):
  - A **superkey** is a set of attributes such that no two tuples (in an instance) agree on their values for those attributes.
  - A **candidate key** is a *minimal* superkey.
  - A **primary key** is a candidate key chosen by the DBA
- Relating keys and FDs:
  - If $K \subseteq R$ is a **superkey** for relation schema R, then dependency $K \to R$ holds on R.
  - If dependency $K \to R$ holds on R *and we assume that R does not contain duplicate tuples (i.e. relational model)* then $K \subseteq R$ is a **superkey** for relation schema R

Closure of FD Sets

How do we know what additional FDs hold in a schema?

- The **closure** of the set of functional dependencies F (denoted F^+) is the set of all functional dependencies that are satisfied by every relational instance that satisfies F.
- Informally, F^+ includes all of the dependencies in F, plus any dependencies they imply.

Reasoning About FDs

Logical implications can be derived by using inference rules called **Armstrong's axioms**

- (reflexivity) $Y \subseteq X \Rightarrow X \rightarrow Y$
- (augmentation) $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
- (transitivity) $X \rightarrow Y,\ Y \rightarrow Z \Rightarrow X \rightarrow Z$

The axioms are

- sound (anything derived from F is in F^+)
- complete (anything in F^+ can be derived)

Reasoning About FDs

Logical implications can be derived by using inference rules called **Armstrong's axioms**

- (reflexivity) $Y \subseteq X \Rightarrow X \to Y$
- (augmentation) $X \to Y \Rightarrow XZ \to YZ$
- (transitivity) $X \to Y, Y \to Z \Rightarrow X \to Z$

The axioms are

- sound (anything derived from F is in F^+)
- complete (anything in F^+ can be derived)

Additional rules can be derived

- (union) $X \to Y, X \to Z \Rightarrow X \to YZ$
- (decomposition) $X \to YZ \Rightarrow X \to Y$

Reasoning About FDs (example)

Example: $F = \{$ SIN, PNum $\to$ Hours
SIN $\to$ EName
PNum $\to$ PName, PLoc
PLoc, Hours $\to$ Allowance $\}$

A derivation of SIN, PNum $\to$ Allowance:

1. SIN, PNum $\to$ Hours ($\in F$)
2. PNum $\to$ PName, PLoc ($\in F$)
3. PLoc, Hours $\to$ Allowance ($\in F$)
4. SIN, PNum $\to$ PNum (reflexivity)
5. SIN, PNum $\to$ PName, PLoc (transitivity, 4 and 2)
6. SIN, PNum $\to$ PLoc (decomposition, 5)
7. SIN, PNum $\to$ PLoc, Hours (union, 6, 1)
8. SIN, PNum $\to$ Allowance (transitivity, 7 and 3)

Computing Attribute Closures

- There is a more efficient way of using Armstrong's axioms, if we only want to derive the maximal set of attributes functionally determined by some X (called the **attribute closure of** X).

> **function** $ComputeX^+(X, F)$
> **begin**
> $X^+ := X$;
>   **while true do**
>   if there exists $(Y \to Z) \in F$ such that
>   (1) $Y \subseteq X^+$, and
>   (2) $Z \not\subseteq X^+$
>   **then** $X^+ := X^+ \cup Z$
>   **else exit**;
>   **return** X^+;
> **end**

Computing Attribute Closures (cont'd)

Let R be a relational schema and F a set of functional dependencies on R. Then

Theorem: X is a superkey of R if and only if

$$ComputeX^{+}(X, F) = R$$

Theorem: $X \to Y \in F^{+}$ if and only if

$$Y \subseteq ComputeX^{+}(X, F)$$

Attribute Closure Example

Example: $F = \{$ SIN $\rightarrow$ EName
PNum $\rightarrow$ PName, PLoc
PLoc, Hours $\rightarrow$ Allowance $\}$

`Compute`X^+`({Pnum,Hours},`F`):`

FD	X^+
initial	Pnum,Hours
Pnum$\rightarrow$Pname,Ploc	Pnum,Hours,Pname,Ploc
PLoc,Hours$\rightarrow$Allowance	Pnum,Hours,Pname,Ploc,Allowance

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

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

A good decomposition does not

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

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.)

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.

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**.

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

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

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

$$\begin{aligned} R &= \{Student,\ Assignment,\ Group,\ Mark\}\ , \\ F &= \{(Student,\ Assignment \to Group,\ Mark)\}\ , \end{aligned}$$

$$\begin{aligned} R_1 &= \{Student,\ Group,\ Mark\}\ , \\ R_2 &= \{Assignment,\ Mark\} \end{aligned}$$

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 test/enforce constraints on the decomposed schema?

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

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

Both are lossless. (Why?)

Dependency Preservation (cont.)

Which decomposition is *better*?

- Decomposition D_1 lets us test FD1 on table 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.

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!

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!

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 in D.

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.

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

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)

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.

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 $(X \rightarrow Y) \in F^+$ and $XY \subseteq R$, then either

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

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

BCNF and Redundancy

- Why does BCNF avoid redundancy? Consider:

Supplied_Items

<u>Sno</u>	Sname	City	<u>Pno</u>	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.

BCNF and Redundancy

- Why does BCNF avoid redundancy? Consider:

Supplied_Items

<u>Sno</u>	Sname	City	<u>Pno</u>	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 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* $R_i \in$ *Result and* $(X \to Y) \in F^+$
    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

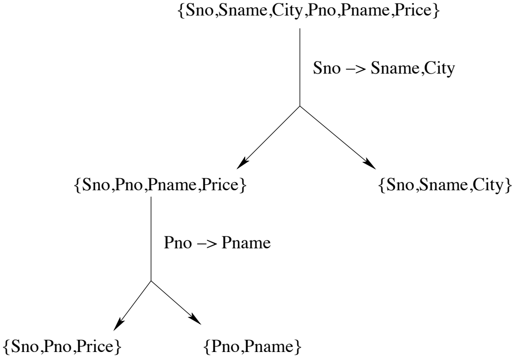
- No *efficient* procedure to do this exists.
- Results depend on sequence of FDs used to decompose the relations.
- It is possible that no lossless join dependency preserving BCNF decomposition exists
  - Consider $R = \{\text{A, B, C}\}$ and $F = \{\text{AB} \rightarrow \text{C, C} \rightarrow \text{B}\}$.

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 determines Sname and City, but is not a superkey of R.

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.

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 in $Y - X$ is 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.

Third Normal Form (3NF)

Schema R is in **3NF** (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, or
- X is a superkey of R, or
- each attribute in $Y - X$ is 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 looser than BCNF
  - allows more redundancy
  - e.g. $R = \{\mathrm{A, B, C}\}$ and $F = \{\mathrm{AB} \to \mathrm{C}, \mathrm{C} \to \mathrm{B}\}$.

Third Normal Form (3NF)

Schema R is in **3NF** (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, or
- X is a superkey of R, or
- each attribute in $Y - X$ is 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 looser than BCNF
  - allows more redundancy
  - e.g. $R = \{\text{A, B, C}\}$ and $F = \{\text{AB} \to \text{C}, \text{C} \to \text{B}\}$.
- 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.

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 $X \to A$ is the set $F - \{X \to A\}$ equivalent to F.
3. for no $X \to A$ and Z a proper subset of X is the set $F - \{X \to A\} \cup \{Z \to A\}$ equivalent to F.

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 $X \to A$ is the set $F - \{X \to A\}$ equivalent to F.
3. for no $X \to A$ and Z a proper subset of X is the set $F - \{X \to A\} \cup \{Z \to A\}$ 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.

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

Step 3.
Remove $X \to A$ from F if $A \in ComputeX^+(X, F - \{X \to A\})$.

Dependency-Preserving 3NF Decomposition

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

function *Decompose3NF*(R, F)
begin
  Result $:= \{R\}$;
  while *some* $R_i \in$ *Result and* $(X \to Y) \in F^+$
    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* F) $- (\bigcup_i F_i)^+$
  for each $(X \to Y) \in N$ **do**
      Add $\{X, Y\}$ *to Result*;
  end;
  return *Result*;
end

Dep-Preserving 3NF Decomposition - An Example

- $R = \{\text{Sno,Sname,City,Pno,Pname,Price}\}$
- Functional dependencies:

  Sno $\rightarrow$ Sname,City   Pno $\rightarrow$ Pname

  Sno,Pno $\rightarrow$ Price   Sno, Pname $\rightarrow$ Price

Dep-Preserving 3NF Decomposition - An Example

- $R = \{\text{Sno,Sname,City,Pno,Pname,Price}\}$
- Functional dependencies:

  Sno $\rightarrow$ Sname,City   Pno $\rightarrow$ Pname

  Sno,Pno $\rightarrow$ Price   Sno, Pname $\rightarrow$ Price
- Following same decomposition tree as BCNF example:

$$R_1 = \{\text{Sno,Sname,City}\}$$
$$R_2 = \{\text{Sno,Pno,Price}\}$$
$$R_3 = \{\text{Pno,Pname}\}$$

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:

$$R_1 = \{\text{Sno,Sname,City}\}$$
$$R_2 = \{\text{Sno,Pno,Price}\}$$
$$R_3 = \{\text{Pno,Pname}\}$$

- Minimal cover:
  Sno $\rightarrow$ Sname   Pno $\rightarrow$ Pname
  Sno $\rightarrow$ City   Sno, Pname $\rightarrow$ Price

Dep-Preserving 3NF Decomposition - An Example

- $R = \{\text{Sno,Sname,City,Pno,Pname,Price}\}$
- Functional dependencies:

  Sno $\rightarrow$ Sname,City   Pno $\rightarrow$ Pname

  Sno,Pno $\rightarrow$ Price   Sno, Pname $\rightarrow$ Price

- Following same decomposition tree as BCNF example:

$$R_1 = \{\text{Sno,Sname,City}\}$$
$$R_2 = \{\text{Sno,Pno,Price}\}$$
$$R_3 = \{\text{Pno,Pname}\}$$

- Minimal cover:

  Sno $\rightarrow$ Sname   Pno $\rightarrow$ Pname

  Sno $\rightarrow$ City   Sno, Pname $\rightarrow$ Price

- Add relation to preserve missing dependency

$$R_4 = \{\text{Sno, Pname, Price}\}$$

3NF Synthesis

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

function $Synthesize3NF(R, F)$
begin
    $Result := \emptyset;$
    $F' :=$ *a minimal cover for* F;
    for each $(X \rightarrow Y) \in F'$ **do**
        $Result := Result \cup \{XY\};$
    if *there is no* $R_i \in$ *Result such that*
        R_i *contains a candidate key for* R **then begin**
            compute a candidate key K *for* R;
            $Result := Result \cup \{K\};$
    end;
    return *Result*;
end

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

3NF Synthesis - An Example

- $R = \{\text{Sno,Sname,City,Pno,Pname,Price}\}$
- Functional dependencies:

  Sno $\rightarrow$ Sname,City   Pno $\rightarrow$ Pname

  Sno,Pno $\rightarrow$ Price   Sno, Pname $\rightarrow$ Price
- Minimal cover:

Sno $\rightarrow$ Sname	$R_1 = \{\text{Sno, Sname}\}$
Sno $\rightarrow$ City	$R_2 = \{\text{Sno, City}\}$
Pno $\rightarrow$ Pname	$R_3 = \{\text{Pno, Pname}\}$
Sno, Pname $\rightarrow$ Price	$R_4 = \{\text{Sno, Pname, Price}\}$

3NF Synthesis - An Example

- $R = \{\text{Sno,Sname,City,Pno,Pname,Price}\}$
- Functional dependencies:

  Sno → Sname,City   Pno → Pname

  Sno,Pno → Price   Sno, Pname → Price
- Minimal cover:

Sno → Sname	$R_1 = \{\text{Sno, Sname}\}$
Sno → City	$R_2 = \{\text{Sno, City}\}$
Pno → Pname	$R_3 = \{\text{Pno, Pname}\}$
Sno, Pname → Price	$R_4 = \{\text{Sno, Pname, Price}\}$

- Add relation for candidate key $R_5 = \{\text{Sno, Pno}\}$

3NF Synthesis - An Example

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

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)