CSE 305 / CSE532

Lecture 08 (Chapter 6)

Relational Normalization Theory

Lecturer: Sael Lee

Slide adapted from the author's and Dr. Ilchul Yoon's slides.

Limitations of E-R Designs

- Provides a set of guidelines, does not result in a unique database schema
- Does not provide a way of evaluating alternative schemas
- Normalization theory provides a mechanism for analyzing and refining the schema produced by an E-R design

Redundancy

- Dependencies between attributes cause redundancy
  - e.g., all addresses in the same town have the same zip code

SSN	*Name*	*Town*	*Zip*
1234	Joe	Stony Brook	11790
4321	Mary	Stony Brook	11790
5454	Tom	Stony Brook	11790
	…………………		

Redundancy

Redundancy and Other Problems

- Set valued attributes in the E-R diagram result in multiple rows in corresponding table
- Example: Person (*SSN, Name, Address, Hobbies*)
  - A person entity with multiple hobbies yields multiple rows in table Person
    - Hence, the association between Name and Address for the same person is stored redundantly
  - *SSN* is key of entity set, but (*SSN, Hobby*) is key of corresponding relation
    - The relation Person can't describe people without hobbies

Example

ER Model

SSN	Name	Address	Hobby
1111	Joe	123 Main	{biking, hiking}

Relational Model

SSN	Name	Address	Hobby
1111	Joe	123 Main	biking
1111	Joe	123 Main	hiking

…………….

Redundancy

Anomalies

- Redundancy leads to anomalies:
  - **Update anomaly**: A change in *Address* must be made in several places
  - **Deletion anomaly**: Suppose a person gives up all hobbies. Do we:
    - Set *Hobby* attribute to null? No, since *Hobby* is part of key
    - Delete the entire row? No, since we lose other information in the row
  - **Insertion anomaly**: *Hobby* value must be supplied for any inserted row since *Hobby* is part of key

Decomposition

- **Solution**: use two relations to store Person information
  - Person1 (SSN, Name, Address)
  - Hobbies (SSN, Hobby)
- The decomposition is more general: people with/without hobbies can now be described
- No update anomalies:
  - Name and address stored once
  - A hobby can be separately supplied or deleted

Normalization Theory

- Result of E-R analysis need further refinement
- Appropriate decomposition can solve problems

- The underlying theory is referred to as ***normalization theory*** and is based on **functional dependencies** (and other kinds, like **multivalued dependencies**)

Functional Dependencies

- Definition: A **functional dependency** (FD) on a relation schema R is a constraint $\mathbf{X} \rightarrow \mathbf{Y}$, where **X** and **Y** are subsets of attributes of R.

- Definition: An FD $\mathbf{X} \rightarrow \mathbf{Y}$ is satisfied in an instance **r** of **R,** if for every pair of tuples, **t** and **s**: if **t** and **s** agree on all attributes in **X** then they must agree on all attributes in **Y**
  - Key constraint is a special kind of functional dependency: all attributes of relation occur on the right-hand side of the FD:
    - SSN $\rightarrow$ SSN, Name, Address

Functional Dependencies

- Address $\rightarrow$ ZipCode
  - Stony Brook's ZIP is 11733
- ArtistName $\rightarrow$ BirthYear
  - Picasso was born in 1881
- Autobrand $\rightarrow$ Manufacturer, Engine type
  - Pontiac is built by General Motors with gasoline engine
- Author, Title $\rightarrow$ PublDate
  - Shakespeare's Hamlet published in 1600

Functional Dependency - Example

- Consider a brokerage firm that allows multiple clients to share an account, but each account is managed from a single office and a client can have no more than one account in an office
  - HasAccount (*AcctNum, ClientId, OfficeId*)
    - FDs:

      $$ClientId, OfficeId \rightarrow AcctNum$$

      $$AcctNum \rightarrow OfficeId$$

    - keys:

      (*ClientId, OfficeId*)

      (*AcctNum, ClientId*)

  - *Thus, attribute values need not depend only on key values*

Entailment, Closure, Equivalence

- **Definition**: If ***F*** is a set of FDs on schema **R** and f is another FD on **R**, then ***F*** ***entails*** f if every instance **r** of **R** that satisfies every FD in ***F*** also satisfies f
  - Ex: $\boldsymbol{F} = \{A \rightarrow B, B \rightarrow C\}$ and f is $A \rightarrow C$
    - If *Town* $\rightarrow$ *Zip* and *Zip* $\rightarrow$ *AreaCode* then *Town* $\rightarrow$ *AreaCode*

- **Definition**: The *closure* of ***F***, denoted $\boldsymbol{F}^+$, is the set of all FDs entailed by ***F***

- **Definition**: ***F*** and ***G*** are *equivalent* if ***F*** entails ***G*** and ***G*** entails ***F***

Entailment (cont’d)

- Satisfaction, entailment, and equivalence are *semantic* concepts – defined in terms of the actual relations in the “real world.”
  - They define *what these notions are*, **not** how to compute them

- How to check if ***F*** entails *f* or if ***F*** and ***G*** are equivalent?
  - Apply the respective definitions for all possible relations?
    - *Bad idea*: might be infinite number for infinite domains
    - Even for finite domains, we have to look at relations of *all* antities
  - **Solution**: find algorithmic, *syntactic* ways to compute these notions
    - *Important*: The syntactic solution must be “correct” with respect to the semantic definitions
    - Correctness has two aspects: *soundness* and *completeness* – see later

Armstrong's Axioms for FDs

- This is the *syntactic* way of computing/testing the various properties of FDs

- **Reflexivity**: If $Y \subseteq X$ then $X \rightarrow Y$ (trivial FD)
  - $Name, Address \rightarrow Name$

- **Augmentation**: If $X \rightarrow Y$ then $XZ \rightarrow YZ$
  - If $Town \rightarrow Zip$ then $Town, Name \rightarrow Zip, Name$

- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

Soundness

- Axioms are *sound*: If an FD f: $X \rightarrow Y$ can be derived from a set of FDs $\boldsymbol{F}$ using the axioms, then f holds in every relation that satisfies every FD in $\boldsymbol{F}$.
- Example: Given $X \rightarrow Y$ and $X \rightarrow Z$ then

$$
\begin{array}{ll}
X \rightarrow XY & \textit{Augmentation by X} \\
YX \rightarrow YZ & \textit{Augmentation by Y} \\
X \rightarrow YZ & \textit{Transitivity}
\end{array}
$$

  - Thus, $X \rightarrow YZ$ is satisfied in every relation where both $X \rightarrow Y$ and $X \rightarrow Z$ are satisfied
    - Therefore, we have derived the *union rule* for FDs: we can take the union of the RHSs of FDs that have the same LHS

Completeness

- Axioms are *complete:* If ***F*** entails *f* , then *f* can be derived from ***F*** using the axioms

- A consequence of completeness is the following (naïve) algorithm to determining if ***F*** entails *f*:
  - *Algorithm: Use the axioms in all possible ways to generate* $\boldsymbol{F}^+$ *(the set of possible FD's is finite so this can be done) and see if* *f* *is in* $\boldsymbol{F}^+$

Correctness

- The notions of *soundness* and *completeness* link the syntax (Armstrong's axioms) with semantics (the definitions in terms of relational instances)

- This is a precise way of saying that the algorithm for entailment based on the axioms is "correct" with respect to the definitions

Generating F^+

$$\begin{tikzcd}[column sep=small]
\underline{F} & & & & \\
AB\to C \arrow[rr, dashed, dash] & & AB\to BCD \arrow[dr, dashed, dash] & & \\
A\to D \arrow[r, dashed, dash, "aug"] & AB\to BD \arrow[ur, dashed, dash, "union"'] & & AB\to BCDE \arrow[r, dashed, dash, "decomp"] & AB\to CDE \\
D\to E \arrow[r, dashed, dash, "aug"] & BCD\to BCDE \arrow[urr, dashed, dash, "trans"'] & & &
\end{tikzcd}$$

Thus, $AB\to BD$, $AB \to BCD$, $AB \to BCDE$, and $AB \to CDE$ are all elements of F^+

Attribute Closure

- Calculating *attribute closure* leads to a more efficient way of checking entailment

- The ***attribute closure*** of a set of attributes, X, with respect to a set of functional dependencies, $\boldsymbol{F}$, (denoted X^+_F) is the set of all attributes, A, such that $X \to A$
  - X^+_{F1} is not necessarily the same as X^+_{F2} if $\boldsymbol{F1} \neq \boldsymbol{F2}$

- *Attribute closure and entailment*:
  - Algorithm: Given a set of FDs, $\boldsymbol{F}$, then

$$X \to Y \text{ if and only if } X^+_F \supseteq Y$$

Example - Computing Attribute Closure

F: $AB \rightarrow C$
$A \rightarrow D$
$D \rightarrow E$
$AC \rightarrow B$

X	X_F^+
A	$\{A, D, E\}$
AB	$\{A, B, C, D, E\}$ (Hence AB is a key)
B	$\{B\}$
D	$\{D, E\}$

Is $AB \rightarrow E$ entailed by F?
Yes
Is $D \rightarrow C$ entailed by F?
No

Result: X_F^+ allows us to determine FDs of the form $X \rightarrow Y$ entailed by F

Computation of Attribute Closure X^+_F

$closure := X;$ $\quad\quad$ *// since* $X \subseteq X^+_F$

repeat

$\quad old := closure;$

$\quad$ **if** there is an FD $Z \rightarrow V$ in $\boldsymbol{F}$ such that

$\quad\quad\quad Z \subseteq closure$ **and** $V \not\subseteq closure$

$\quad\quad$ **then** $closure := closure \cup V$

until $old = closure$

– If $T \subseteq closure$ then $X \rightarrow T$ is entailed by $\boldsymbol{F}$

Example: Computation of Attribute Closure

- **Problem**: Compute the attribute closure of *AB* with respect to the set of FDs *:*

$$
\begin{array}{ll}
AB \to C & \text{(a)} \\
A \to D & \text{(b)} \\
D \to E & \text{(c)} \\
AC \to B & \text{(d)}
\end{array}
$$

- **Solution**:

Initially $closure = \{AB\}$
Using (a) $closure = \{ABC\}$
Using (b) $closure = \{ABCD\}$
Using (c) $closure = \{ABCDE\}$

Normal Forms

- Each normal form is a set of conditions on a schema that guarantees certain properties (relating to redundancy and update anomalies)
- First normal form (1NF) is the same as the definition of relational model (relations = sets of tuples; each tuple = sequence of atomic values)
- Second normal form (2NF) – no partial depenency
- The two commonly used normal forms are ***third normal form*** (3NF) and ***Boyce-Codd normal form*** (BCNF)
- *Normalization is a database design technique for producing a set of **suitable** relations that support the data requirements of an enterprise.*

How Normalization Supports Database Design

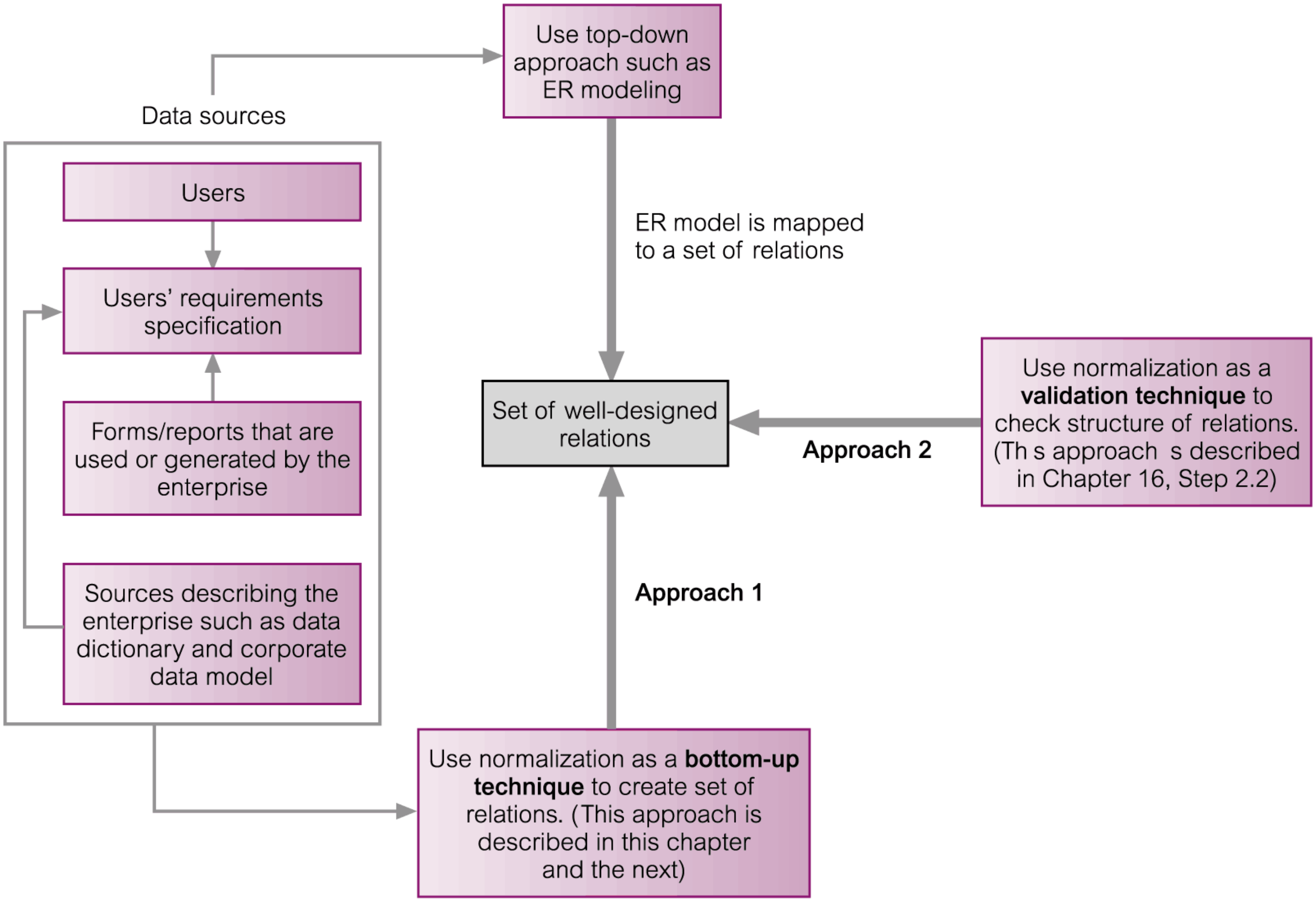
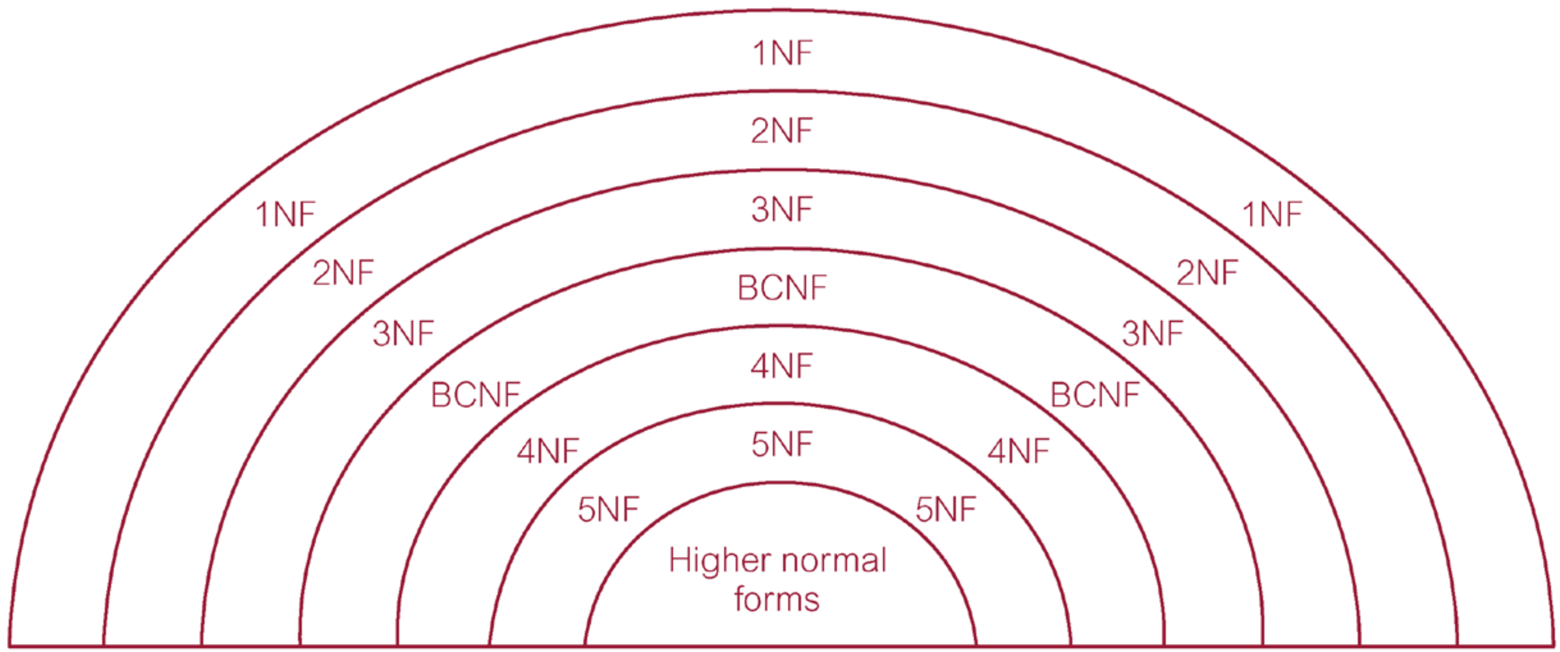
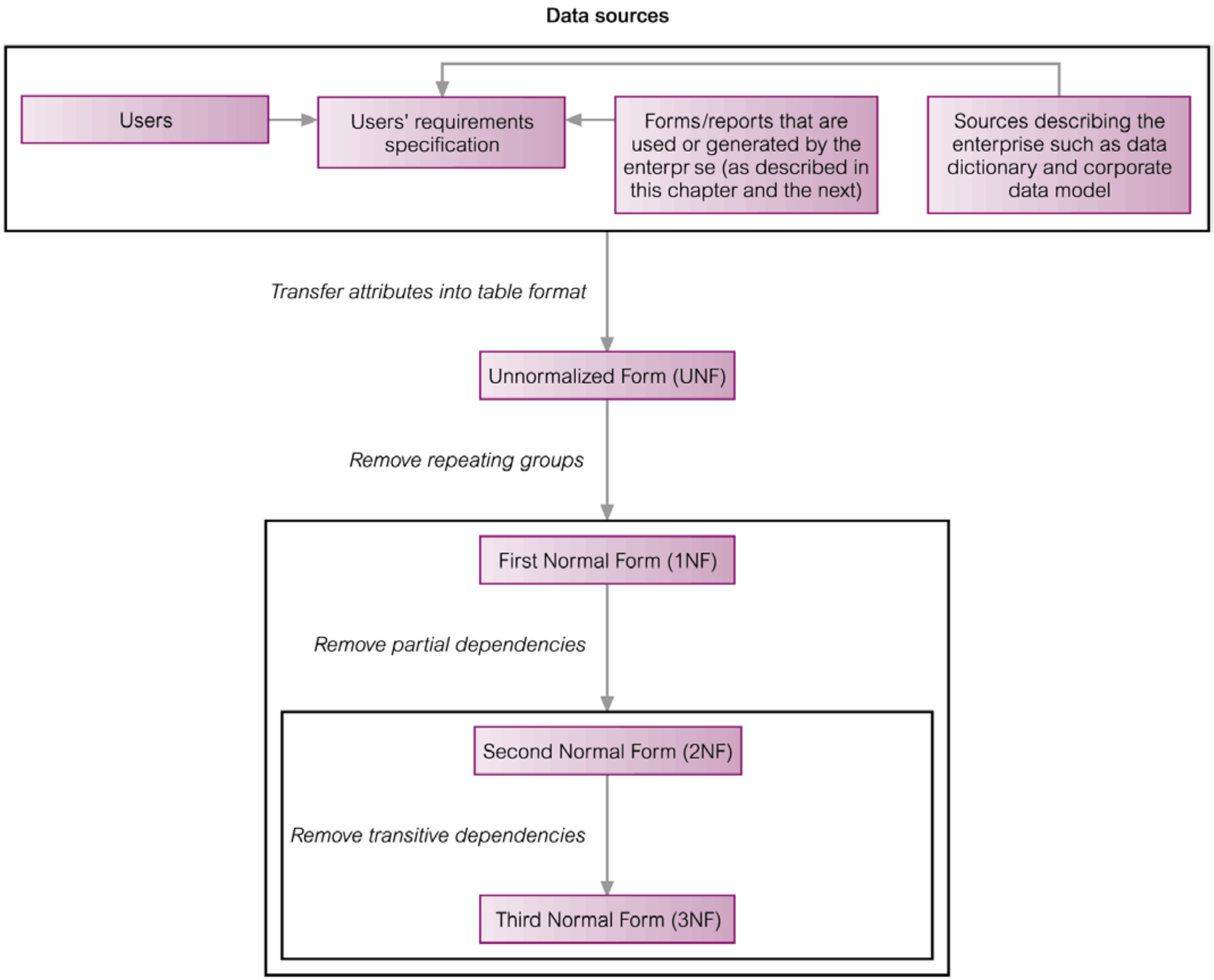


Figure 13.1 How normalization can be used to support database design.

Relationship Between Normal Forms



Process of Normalization



Un-Normalized Form (UNF)

- A table that contains one or more repeating groups.
- To create an unnormalized table:
  - Transform data from information source (e.g. form) into table format with columns and rows.

StaffPropertyInspection

propertyNo	pAddress	iDate	iTime	comments	staffNo	sName	carReg
PG4	6 Lawrence St, Glasgow	18-Oct-00 22-Apr-01 1-Oct-01	10.00 09.00 12.00	Need to replace crockery In good order Damp rot in bathroom	SG37 SG14 SG14	Ann Beech David Ford David Ford	M231 JGR M533 HDR N721 HFR
PG16	5 Novar Dr, Glasgow	22-Apr-01 24-Oct-01	13.00 14.00	Replace living room carpet Good condition	SG14 SG37	David Ford Ann Beech	M533 HDR N721 HFR

First Normal Form (1NF)

- *A relation in which intersection of each row and column contains one and only one (atomic) value.*

StaffPropertyInspection

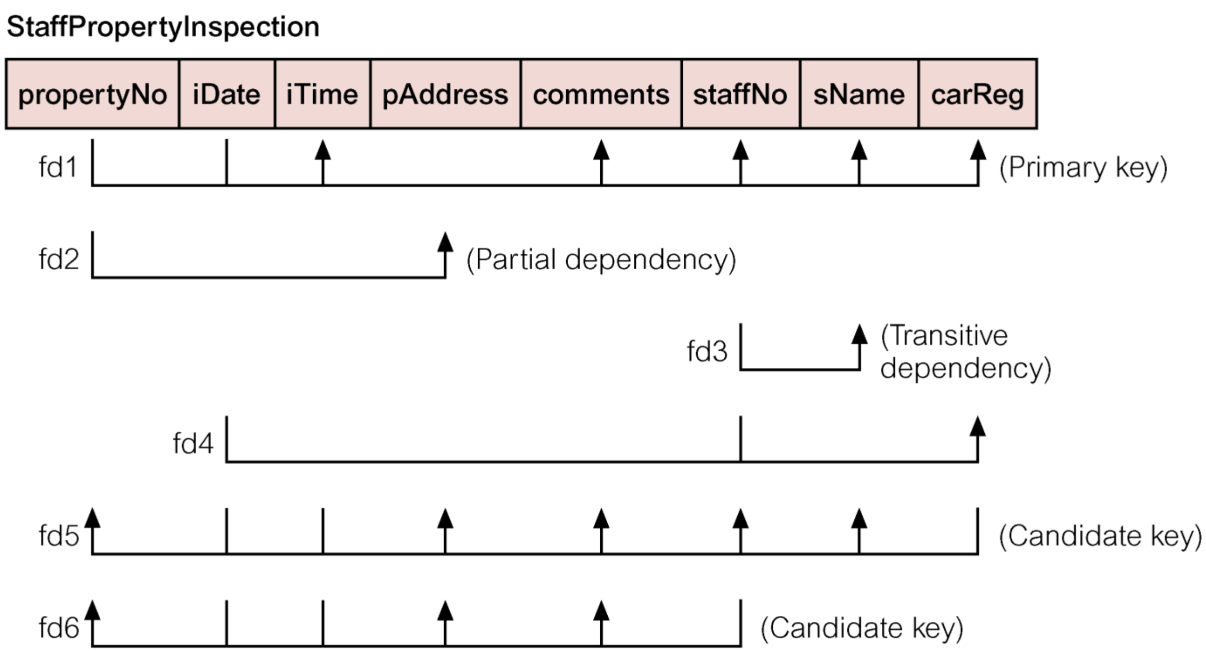
propertyNo	iDate	iTime	pAddress	comments	staffNo	sName	carReg
PG4	18-Oct-00	10.00	6 Lawrence St, Glasgow	Need to replace crockery	SG37	Ann Beech	M231 JGR
PG4	22-Apr-01	09.00	6 Lawrence St, Glasgow	In good order	SG14	David Ford	M533 HDR
PG4	1-Oct-01	12.00	6 Lawrence St, Glasgow	Damp rot in bathroom	SG14	David Ford	N721 HFR
PG16	22-Apr-01	13.00	5 Novar Dr, Glasgow	Replace living room carpet	SG14	David Ford	M533 HDR
PG16	24-Oct-01	14.00	5 Novar Dr, Glasgow	Good condition	SG37	Ann Beech	N721 HFR

Second Normal Form (2NF)

- Based on concept of full functional dependency:
  - A, X and B are attributes of a relation,
  - B is **fully dependent** on (A, X) if B is functionally dependent on (A,X) but not on any proper subset of (A,X) such as (A) or (X).
  - A, X $\rightarrow$ B and there is NO A $\rightarrow$ B or X $\rightarrow$ B
- *2nd Normal Form*
  - *A relation that does not have a FD, X $\rightarrow$ Y, where X is a strict subset of that schema's key and Y has attributes that do not occur in any of the schema's keys.*

1NF to 2NF (Functional Dependencies)

- Fd1: PropertyNo, iDate → iTime, staffNo, comments, sName, carReg
- **Fd2: PropertyNo → pAddress**
- Fd3: staffNo → sName
- Fd4: iDate, staffNo → carReg
- Fd5: iDate, iTime, carReg → all other attributes
- Fd6: iDate, iTime, staffNo → all other attributes



1NF to 2NF

- Transformed into following two tables.
  - Property (<u>propertyNo</u>, pAddress)
  - PropertyInspection (<u>propertyNo, iDate</u>, iTime, comments, staffNo, sName, carReg)

Boyce-Codd Normal Form (BCNF)

- **Definition**: A relation schema **R** is in BCNF if for every FD $X \rightarrow Y$ associated with **R** either
  - $Y \subseteq X$ (i.e., the FD is **trivial**) or
  - X is a superkey of **R**

- **Example**: **Person1** (*SSN, Name, Address*)
  - The only FD is $SSN \rightarrow Name, Address$
  - Since SSN is a key, Person1 is in BCNF

(non) BCNF Examples

- Person (*SSN, Name, Address, Hobby*)
  - The FD *SSN* → *Name, Address* does not satisfy requirements of BCNF
  - since the key is (*SSN, Hobby*)
- HasAccount (*AcctNum, ClientId, OfficeId*)
  - The FD *AcctNum* → *OfficeId* does not satisfy BCNF requirements
  - since keys are (*ClientId, OfficeId*) and (*AcctNum, ClientId*); not *AcctNum*.

HasAccount (***AcctNum, ClientId, OfficeId***)

FDs:

Client, OfficeId → ***AcctNum***

AcctNum → ***OfficeId***

keys:

(***ClientId, OfficeId***)

(***AcctNum, ClientId***)

Redundancy

- Suppose **R** has a FD $A \rightarrow B$, and A is not a superkey. If an instance has 2 rows with same value in A, they must also have same value in B (=> redundancy, if the *A-value* repeats twice)

redundancy

$$SSN \rightarrow Name, Address$$

SSN	*Name*	*Address*	*Hobby*
1111	Joe	123 Main	stamps
1111	Joe	123 Main	coins

- If A is a superkey, there cannot be two rows with same value of A
  - Hence, BCNF eliminates redundancy

Third Normal Form

- A relational schema **R** is in 3NF if for every FD $X \rightarrow Y$ associated with R either:
  - $Y \subseteq X$ (i.e., the FD is trivial); <u>or</u>
  - X is a superkey of **R**; <u>or</u>
  - Every $A \in Y$ is part of some key of **R**

BCNF conditions

- 3NF is weaker than BCNF (every schema that is in BCNF is also in 3NF)
- "for each nontrivial FD, either the left side is a superkey or the right side consist of prime attributes only."
- Prime : attribute that is a member of some key

3NF Example

- HasAccount (*AcctNum, ClientId, OfficeId*)
  - *ClientId, OfficeId* → *AcctNum*
    - OK since LHS contains a key
  - *AcctNum* → *OfficeId*
    - OK since RHS is part of a key

HasAccount (***AcctNum, ClientId, OfficeId***)
FDs:
Client, OfficeId → ***AcctNum***
AcctNum → ***OfficeId***
keys:
(***ClientId, OfficeId***)
(***AcctNum, ClientId***)

- HasAccount is in 3NF but it might still contain redundant information due to *AcctNum* → *OfficeId* (which is not allowed by BCNF)

3NF (Non) Example

- Person (*SSN, Name, Address, Hobby*)
  - (*SSN, Hobby*) is the only key.
  - *SSN* $\rightarrow$ *Name* violates 3NF conditions since *Name* is not part of a key and *SSN* is not a superkey
- If we decompose Person into
  - Person1 (SSN, Name, Addr)
  - Hobby(SSN, Hobby)
- Then, these are 3NF and BCNF

Decompositions

- **Goal**: Eliminate redundancy by decomposing a relation into several relations in a higher normal form

- Decomposition must be *lossless*: it must be possible to reconstruct the original relation from the relations in the decomposition

Decomposition

- Schema $\mathbf{R} = (R, \boldsymbol{F})$
  - R is a set of attributes
  - $\boldsymbol{F}$ is a set of functional dependencies over R
    - Each key is described by a FD

- The *decomposition of schema* **R** is a collection of schemas $\mathbf{R}_i = (R_i, \boldsymbol{F}_i)$ where
  - $R = \cup_i R_i$ for all i (*no new attributes*)
  - $\boldsymbol{F}_i$ is a set of functional dependences involving only attributes of R_i
  - $\boldsymbol{F}$ entails $\boldsymbol{F}_i$ for all i (*no new FDs*)
- The *decomposition of an instance*, **r**, of **R** is a set of relations $\mathbf{r}_i = \pi_{R_i}(\mathbf{r})$ for all i

Example Decomposition

Schema (R, F) where

R = {SSN, $Name$, $Address$, $Hobby$}

F = {$SSN \rightarrow Name$, $Address$}

can be decomposed into:

R_1 = {SSN, $Name$, $Address$}

F_1 = {$SSN \rightarrow Name$, $Address$}

and

R_2 = {SSN, $Hobby$}

F_2 = { }

Lossless Schema Decomposition

- A decomposition should not lose information

- A decomposition ($\mathbf{R}_1, \ldots, \mathbf{R}_n$) of a schema, **R**, is *lossless* if every valid instance, **r**, of **R** can be reconstructed from its components:

$$\mathbf{r} = \mathbf{r}_1 \bowtie \mathbf{r}_2 \bowtie \cdots\cdots \bowtie \mathbf{r}_n$$

where each $\mathbf{r}_i = \pi_{\mathbf{R}_i}(\mathbf{r})$

Lossy Decomposition

- *The following is always the case* (Think why?):

$$\mathbf{r} \subseteq \mathbf{r}_1 \bowtie \mathbf{r}_2 \bowtie \ldots \bowtie \mathbf{r}_n$$

- *But the following is not always true*:

$$\mathbf{r} \supseteq \mathbf{r}_1 \bowtie \mathbf{r}_2 \bowtie \ldots \bowtie \mathbf{r}_n$$

- *Example*

$$\mathbf{r} \not\supseteq \mathbf{r}_1 \bowtie \mathbf{r}_2$$

SSN	*Name*	*Address*
1111	Joe	1 Pine
2222	Alice	2 Oak
3333	Alice	3 Pine

SSN	*Name*
1111	Joe
2222	Alice
3333	Alice

Name	*Address*
Joe	1 Pine
Alice	2 Oak
Alice	3 Pine

The tuples (2222, Alice, 3 Pine) and (3333, Alice, 2 Oak) *are in the join, but not in the original*

Lossy Decompositions: *What is Actually Lost*?

- In the previous example, the tuples (2222, Alice, 3 Pine) and (3333, Alice, 2 Oak) were *gained*, not lost!
  - Why do we say that the decomposition was lossy?

- What was lost is ***information***:
  - That 2222 lives at 2 Oak: *In the decomposition, 2222 can live at either 2 Oak or 3 Pine*
  - That 3333 lives at 3 Pine: *In the decomposition, 3333 can live at either 2 Oak or 3 Pine*

Testing for Losslessness

- A (binary) decomposition of **R** = (R, ***F***) into $\mathbf{R}_1 = (R_1, \boldsymbol{F}_1)$ and $\mathbf{R}_2 = (R_2, \boldsymbol{F}_2)$ is lossless *if and only if* :
  - either the FD
    - $(R_1 \cap R_2) \rightarrow R_1$ is in $\boldsymbol{F}^+$
  - or the FD
    - $(R_1 \cap R_2) \rightarrow R_2$ is in $\boldsymbol{F}^+$

Example

Schema (R, F) where

R = {SSN, $Name$, $Address$, $Hobby$}

$F = \{SSN \rightarrow Name, Address\}$

can be decomposed into:

R_1 = {SSN, $Name$, $Address$}

$F_1 = \{SSN \rightarrow Name, Address\}$

and

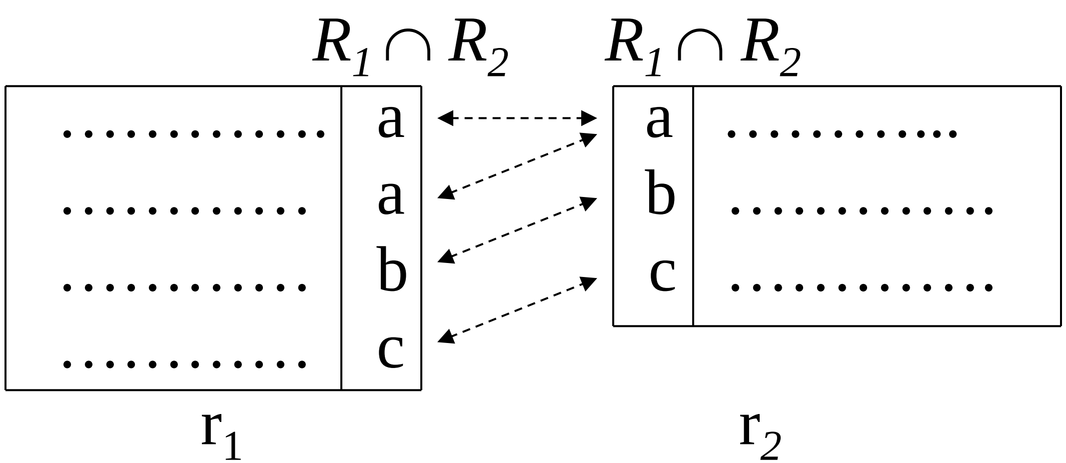
R_2 = {SSN, $Hobby$}

$F_2 = \{ \}$

Since $R_1 \cap R_2 = SSN$ and $SSN \rightarrow R_1$ the decomposition is lossless

Intuition Behind the Test for Losslessness

- Suppose $R_1 \cap R_2 \rightarrow R_2$. Then a row of $\mathbf{r}_1$ can combine with exactly one row of $\mathbf{r}_2$ in the natural join (since in $\mathbf{r}_2$ a particular set of values for the attributes in $R_1 \cap R_2$ defines a unique row)



Tuple Structure in a Lossless Binary Decomposition

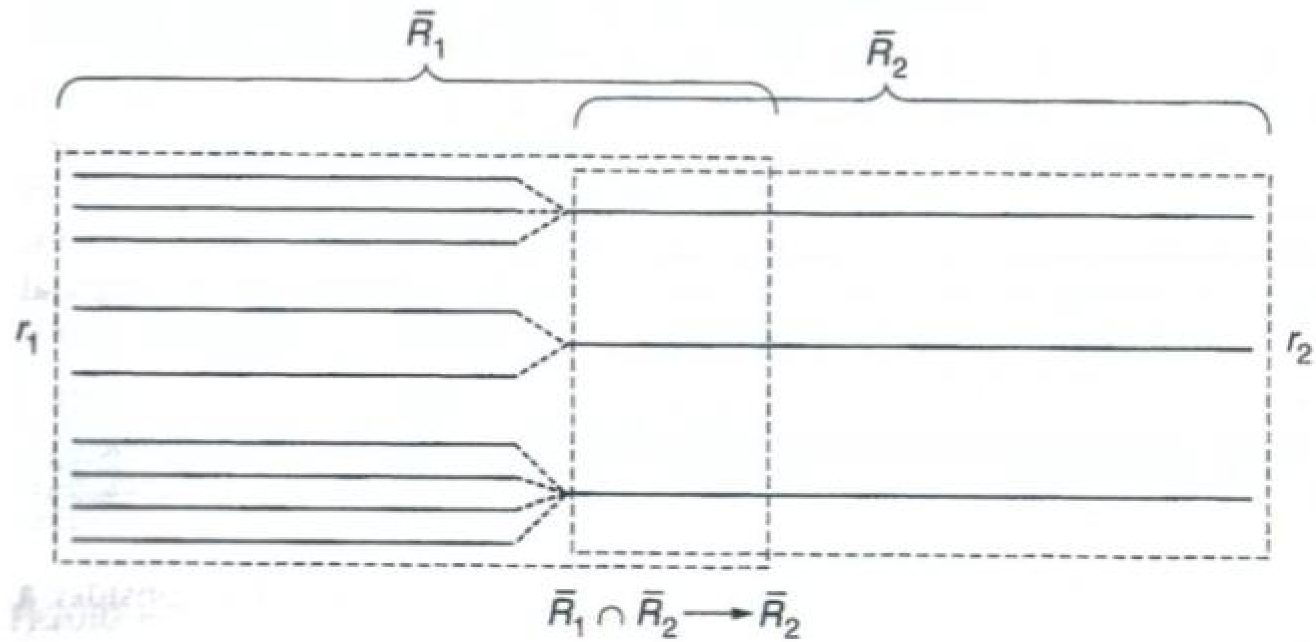


FIGURE 6.6 Tuple structure in a lossless binary decomposition: a row of $\mathbf{r}_1$ combines with exactly one row of $\mathbf{r}_2$.

Proof of Lossless Condition

- $\mathbf{r} \subseteq \mathbf{r}_1 \bowtie \mathbf{r}_2$ – *this is true for any decomposition*
- $\mathbf{r} \supseteq \mathbf{r}_1 \bowtie \mathbf{r}_2$

If $R_1 \cap R_2 \rightarrow R_2$ **then**

$card\,(\mathbf{r}_1 \bowtie \mathbf{r}_2) = card\,(\mathbf{r}_1)$

(since each row of r_1 joins with exactly one row of r_2)

But $card\,(\mathbf{r}) \geq card\,(\mathbf{r}_1)$ (*since* $\mathbf{r}_1$ *is a projection of* $\mathbf{r}$)

and therefore $card\,(\mathbf{r}) \geq card\,(\mathbf{r}_1 \bowtie \mathbf{r}_2)$

Hence $\mathbf{r} = \mathbf{r}_1 \bowtie \mathbf{r}_2$

Dependency Preservation

- Consider a decomposition of $\mathbf{R} = (R, \boldsymbol{F})$ into $\mathbf{R}_1 = (R_1, \boldsymbol{F}_1)$ and $\mathbf{R}_2 = (R_2, \boldsymbol{F}_2)$
  - An FD $X \rightarrow Y$ of $\boldsymbol{F}^+$ is in $\boldsymbol{F}_i$ iff $X \cup Y \subseteq R_i$
  - An FD, $f \in \boldsymbol{F}^+$ may be in neither $\boldsymbol{F}_1$, nor $\boldsymbol{F}_2$, nor even $(\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$
    - Checking that f is true in $\mathbf{r}_1$ or $\mathbf{r}_2$ is (relatively) easy
    - Checking f in $\mathbf{r}_1 \bowtie \mathbf{r}_2$ is harder – requires a join
    - *Ideally*: want to check FDs locally, in $\mathbf{r}_1$ and $\mathbf{r}_2$, and have a guarantee that every $f \in F$ holds in $\mathbf{r}_1 \bowtie \mathbf{r}_2$
- The decomposition is ***dependency preserving*** iff the sets $\boldsymbol{F}$ and $\boldsymbol{F}_1 \cup \boldsymbol{F}_2$ are equivalent: $\boldsymbol{F}^+ = (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$
  - Then checking all FDs in $\boldsymbol{F}$, as $\mathbf{r}_1$ and $\mathbf{r}_2$ are updated, can be done by checking $\boldsymbol{F}_1$ in $\mathbf{r}_1$ and $\boldsymbol{F}_2$ in $\mathbf{r}_2$

Dependency Preservation

- If f is an FD in $\boldsymbol{F}$, but f is not in $\boldsymbol{F}_1 \cup \boldsymbol{F}_2$, there are two possibilities:
  - $f \in (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$
    - If the constraints in $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$ are maintained, f will be maintained automatically.
  - $f \notin (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$
    - f can be checked only by first taking the join of $\mathbf{r}_1$ and $\mathbf{r}_2$. This is costly.
    - Incur additional runtime overhead of constraint maintenance

Example

Schema (R, F) where

$R = \{SSN, Name, Address, Hobby\}$

$F = \{SSN \rightarrow Name, Address\}$

can be decomposed into:

$R_1 = \{SSN, Name, Address\}$

$F_1 = \{SSN \rightarrow Name, Address\}$

and

$R_2 = \{SSN, Hobby\}$

$F_2 = \{\ \}$

Since $F = F_1 \cup F_2$ the decomposition is dependency preserving

SUNY Korea

Example

- Schema: $(ABC;\ \boldsymbol{F})$, $\boldsymbol{F} = \{A \rightarrow B, B \rightarrow C, C \rightarrow B\}$
- Decomposition:
  - $(AC, \boldsymbol{F}_1)$, $\boldsymbol{F}_1 = \{A \rightarrow C\}$
    - Note: A$\rightarrow$C $\notin$ ***F***, but in $\boldsymbol{F}^+$
  - $(BC, \boldsymbol{F}_2)$, $\boldsymbol{F}_2 = \{B \rightarrow C, C \rightarrow B\}$
- $A \rightarrow B \notin (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)$, **but** $A \rightarrow B \in (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$.
  - So $\boldsymbol{F}^+ = (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$ and thus the decompositions is still dependency preserving

Example

- HasAccount (*AcctNum, ClientId, OfficeId*)

  f_1: *AcctNum* $\rightarrow$ *OfficeId*

  f_2: *ClientId, OfficeId* $\rightarrow$ *AcctNum*

- Decomposition:

  R_1 = (*AcctNum, OfficeId*; {*AcctNum* $\rightarrow$ *OfficeId*})

  R_2 = (*AcctNum, ClientId*; {})

- Decomposition is lossless:

  $R_1 \cap R_2$= {*AcctNum*} and *AcctNum* $\rightarrow$ *OfficeId* (*i.e.* R_1)

- In BCNF

- Not dependency preserving: $f_2 \notin (\boldsymbol{F_1} \cup \boldsymbol{F_2})^+$

- HasAccount *does not* have BCNF decompositions that are both lossless and dependency preserving! (check by enumeration)

- Hence: "BCNF + lossless + dependency preserving" decompositions are not always achievable!

BCNF Decomposition Algorithm

Input: R = (*R; F*)

Decomp := R

while there is **S** = (*S;* ***F'***) $\in$ *Decomp* and **S** not in BCNF **do**

Find $X \to Y \in F'$ that violates BCNF // i.e., *X isn't a superkey in* S

Replace S in *Decomp* with $S_1 = (XY; F_1)$, $S_2 = (S - (Y - X); F_2)$

// *F_1 = all FDs of F' involving only attributes of XY*

// *F_2 = all FDs of F' involving only attributes of S - (Y - X)*

end

return *Decomp*

Simple Example

- HasAccount:

$(ClientId,\ OfficeId,\ AcctNum)$

$ClientId, OfficeId \rightarrow AcctNum$
$AcctNum \rightarrow OfficeId$

- Decompose using $AcctNum \rightarrow OfficeId$:

$(OfficeId,\ AcctNum)$

BCNF: *AcctNum* is key
FD: $AcctNum \rightarrow OfficeId$

$(ClientId\ ,\ AcctNum)$

BCNF (only trivial FDs)

A Larger Example

Given: R = (R; F) where $R = ABCDEGHK$ and

$F = \{ABH \rightarrow C, A \rightarrow DE, BGH \rightarrow K, K \rightarrow ADH, BH \rightarrow GE\}$

Step 1: Find a FD that violates BCNF

Not $ABH \rightarrow C$ since $(ABH)^+$ includes all attributes

(BH is a key)

$A \rightarrow DE$ violates BCNF since A is not a superkey ($A^+ = ADE$)

Step 2: Split R into:

$R_1 = (ADE, F_1 = \{A \rightarrow DE\})$

$R_2 = (ABCGHK; F_2 = \{ABH \rightarrow C, BGH \rightarrow K, K \rightarrow AH, BH \rightarrow G\})$

Note 1: R_1 is in BCNF

Note 2: Decomposition is *lossless* since A is a key of R_1.

Note 3: FDs $K \rightarrow D$ and $BH \rightarrow E$ are not in F_1 or F_2. But both can be derived from $F_1 \cup F_2$

(*E.g.*, $K \rightarrow A$ and $A \rightarrow D$ implies $K \rightarrow D$)

Hence, decomposition *is dependency preserving*.

Example (con't)

Given: $R_2 = (ABCGHK; \{ABH \rightarrow C, BGH \rightarrow K, K \rightarrow AH, BH \rightarrow G\})$

step 1: Find a FD that violates BCNF.

Not $ABH \rightarrow C$ or $BGH \rightarrow K$, since BH is a key of R_2

$K \rightarrow AH$ violates BCNF since K is not a superkey ($K^+ = AHK$)

step 2: Split R_2 into:

$R_{21} = (KAH, F_{21} = \{K \rightarrow AH\})$

$R_{22} = (BCGK; F_{22} = \{\})$

Note 1: Both R_{21} and R_{22} are in BCNF.

Note 2: The decomposition is *lossless* (since K is a key of R_{21})

Note 3: FDs $ABH \rightarrow C$, $BGH \rightarrow K$, $BH \rightarrow G$ are not in F_{21} or F_{22}, and they can't be derived from $F_1 \cup F_{21} \cup F_{22}$.

Hence the decomposition is *not* dependency-preserving

Properties of BCNF Decomposition Algorithm

Let $X \rightarrow Y$ violate BCNF in $\mathbf{R} = (R, \boldsymbol{F})$ and $\mathbf{R_1} = (R_1, \boldsymbol{F}_1)$, $\mathbf{R_2} = (R_2, \boldsymbol{F}_2)$ is the resulting decomposition. Then:

- There are *fewer violations* of BCNF in $\mathbf{R_1}$ and $\mathbf{R_2}$ than there were in $\mathbf{R}$
  - $X \rightarrow Y$ implies X is a key of $\mathbf{R_1}$
  - Hence $X \rightarrow Y \in \boldsymbol{F}_1$ does not violate BCNF in $\mathbf{R_1}$ and, since $X \rightarrow Y \notin \boldsymbol{F}_2$, does not violate BCNF in $\mathbf{R_2}$ either
  - Suppose $f: X' \rightarrow Y' \in \boldsymbol{F}$ doesn't violate BCNF in **R**. If $f \in \boldsymbol{F}_1$ or $\boldsymbol{F}_2$ it does not violate BCNF in $\mathbf{R_1}$ or $\mathbf{R_2}$ either since X' is a superkey of **R** and hence also of $\mathbf{R_1}$ and $\mathbf{R_2}$.

Properties of BCNF Decomposition Algorithm

- A BCNF decomposition is *not necessarily* dependency preserving
- But *always* lossless:

  since $R_1 \cap R_2 = X, \quad X \rightarrow Y,$ and $\quad R_1 = XY$
- BCNF+lossless+dependency preserving is sometimes unachievable (recall HasAccount)

Third Normal Form

- A relational schema **R** is in 3NF if for every FD $X \rightarrow Y$ associated with *R* either:
  - $Y \subseteq X$ (i.e., the FD is trivial); or
  - *X* is a superkey of **R**; or
  - Every $A \in Y$ is part of some key of **R**

BCNF conditions

- 3NF is weaker than BCNF (every schema that is in BCNF is also in 3NF)
  - Compromise – Not all redundancy removed, but dependency preserving decompositions are always possible (and, of course, lossless)
- 3NF decomposition is based on a *minimal cover*

Minimal Cover

- A *minimal cover* of a set of dependencies, ***F,*** is a set of dependencies, ***U,*** such that:
  - *U* is equivalent to ***F*** ($\boldsymbol{F^+ = U^+}$)
  - All FDs in ***U*** have the form $X \rightarrow A$ where *A* is a single attribute
  - It is not possible to make ***U*** smaller (while preserving equivalence) by
    - Deleting an FD
    - Deleting an attribute from an FD (either from LHS or RHS)
  - FDs and attributes that can be deleted in this way are called *redundant FD*
  - *Redundant attributes* can be defined similarly.

Computing Minimal Cover

- **Example**: $\boldsymbol{F} = \{ABH \rightarrow CK,\ A \rightarrow D,\ C \rightarrow E,$
  $BGH \rightarrow L,\ L \rightarrow AD,\ E \rightarrow L,\ BH \rightarrow E\}$

- **Step 1**: Make RHS of each FD into a single attribute
  - *Algorithm*: Use the decomposition inference rule for FDs
  - Example: $L \rightarrow AD$ replaced by $L \rightarrow A,\ L \rightarrow D$; $ABH \rightarrow CK$ by $ABH \rightarrow C,\ ABH \rightarrow K$
- **Step 2**: Eliminate redundant attributes from LHS.
  - *Algorithm*: If FD $XB \rightarrow A \in \boldsymbol{F}$ (where B is a single attribute) and $X \rightarrow A$ is entailed by $\boldsymbol{F}$, then B was unnecessary
  - Example: Can an attribute be deleted from $ABH \rightarrow C$?
    - Compute $AB^+_{\boldsymbol{F}}$, $AH^+_{\boldsymbol{F}}$, $BH^+_{\boldsymbol{F}}$.
    - Since $C \in (BH)^+_{\boldsymbol{F}}$, $BH \rightarrow C$ is entailed by $\boldsymbol{F}$ and A is redundant in $ABH \rightarrow C$.

Computing Minimal Cover (con't)

- **Example (con'd)**:
  - $\boldsymbol{F} = \{BH \to C, BH \to K, A \to D, C \to E, BGH \to L,$
    $L \to A, L \to D, E \to L, BH \to E\}$

- **Step 3**: Delete redundant FDs from ***F***
  - *Algorithm*: If $\boldsymbol{F} - \{f\}$ entails f, then f is redundant
    - If f is $X \to A$ then check if A $\in X^{+}_{\boldsymbol{F}-\{f\}}$
  - Example: $BGH \to L$ is entailed by $BH \to E$ and $E \to L$, so it is redundant

- *Note*: The order of steps 2 and 3 cannot be interchanged!!

Synthesizing a 3NF Schema

- Starting with a schema **R** = (*R*, ***F***)

- **Step 1**: Compute a minimal cover, ***U***, of ***F***.
  - The decomposition is based on ***U***, but since $U^+ = F^+$ the same functional dependencies will hold
  - A minimal cover for

    $F=\{ABH \to CK, A \to D, C \to E, BGH \to L, L \to AD, E \to L, BH \to E\}$

    is

    $U=\{BH \to C, BH \to K, A \to D, C \to E, L \to A, E \to L\}$

Synthesizing a 3NF schema (con't)

- **Step 2**: Partition $\boldsymbol{U}$ into sets $\boldsymbol{U_1}, \boldsymbol{U_2}, \ldots \boldsymbol{U_n}$ such that the LHS of all elements of $\boldsymbol{U_i}$ are the same
  - $\boldsymbol{U_1} = \{BH \rightarrow C, BH \rightarrow K\}$, $U_2 = \{A \rightarrow D\}$, $\boldsymbol{U_3} = \{C \rightarrow E\}$, $U_4 = \{L \rightarrow A\}$, $U_5 = \{E \rightarrow L\}$

- **Step 3**: For each $\boldsymbol{U_i}$, form schema $\mathbf{R_i} = (R_i, \boldsymbol{U_i})$, where R_i is the set of all attributes mentioned in $\boldsymbol{U_i}$
  - Each FD of $\boldsymbol{U}$ will be in some $\mathbf{R_i}$. Hence the decomposition is *dependency preserving*
  - $\mathbf{R_1} = (BHCK;\ BH \rightarrow C, BH \rightarrow K)$, $\mathbf{R_2} = (AD;\ A \rightarrow D)$, $\mathbf{R_3} = (CE;\ C \rightarrow E)$, $\mathbf{R_4} = (AL;\ L \rightarrow A)$, $\mathbf{R_5} = (EL;\ E \rightarrow L)$

Synthesizing a 3NF schema (con't)

- **Step 4**: If no R_i is a superkey of **R**, add schema $\mathbf{R_0} = (R_0,\{\})$ where R_0 is a key of **R**.
  - $\mathbf{R_0} = (BGH, \{\})$
    - $\mathbf{R_0}$ might be needed when not all attributes are necessarily contained in $R_1 \cup R_2 \ldots \cup R_n$
      - a missing attribute, *A*, must be part of all keys (since it's not in any FD of U, deriving a key constraint from U involves the augmentation axiom)
    - $\mathbf{R_0}$ might be needed even if all attributes are accounted for in $R_1 \cup R_2 \ldots \cup Rn$
      - Example: (ABCD; {A→B, C→D}).
      - Step 3 decomposition: R1 = (AB; {A→B}), R2 = (CD; {C→D}). Lossy! Need to add (AC; { }), for losslessness
  - *Step 4 guarantees lossless decomposition.*

BCNF Design Strategy

- The resulting decomposition, $\mathbf{R_0, R_1, \ldots R_n}$, is
  - Dependency preserving (since every FD in *U* is a FD of some schema)
  - Lossless (although this is not obvious)
  - In 3NF (although this is not obvious)

- Strategy for decomposing a relation
  - Use 3NF decomposition first to get lossless, dependency preserving decomposition
  - If any resulting schema is not in BCNF, split it using the BCNF algorithm (but this may yield a non-dependency preserving result)

Normalization Drawbacks

- By limiting redundancy, normalization helps maintain consistency and saves space
- But performance of querying can suffer because related information that was stored in a single relation is now distributed among several

- **Example**: A join is required to get the names and grades of all students taking CS305 in S2002.

```
SELECT  S.Name, T.Grade
FROM  Student S, Transcript T
WHERE  S.Id = T.StudId  AND
        T.CrsCode = 'CS305'  AND  T.Semester = 'S2002'
```

Denormalization

- **Tradeoff**: Judiciously introduce redundancy to improve performance of certain queries
- **Example**: Add attribute Name to Transcript

```
SELECT  T.Name, T.Grade
FROM    Transcript'  T
WHERE   T.CrsCode = 'CS305'  AND  T.Semester = 'S2002'
```

- Join is avoided
- If queries are asked more frequently than Transcript is modified, added redundancy might improve average performance
- But, Transcript' is no longer in BCNF since key is (StudId, CrsCode, Semester) and StudId $\rightarrow$ Name

Additional note on BCNF and 3NF Synthesis

- Pitfalls : Relations $\mathbf{R_i}$ with FDs G_i from 3NF synthesis are also in BCNF
  - Tempted because FDs used for creating each relation are based on super keys
  - However, $\mathbf{R_i}$ can only guarantee the FDs in G_i , and cannot entail all FDs in G^+
  - Example
    - **R** = { *AcctNum, ClientId, OfficeId, DateOpened* }
    - F = { *ClientId, OfficeId* → *AcctNum*, *AcctNum* → *OfficeId, DateOpened* }
    - Through 3NF synthesis, we get

      Not in BCNF
      - R_1 = (*{ClientId, OfficeId, AcctNum}, {ClientId, OfficeId* → *AcctNum}*)
      - R_2 = (*{AcctNum, OfficeId, DateOpened}, {AcctNum* → *OfficeId, DateOpened}*)
  - Need to compute $\pi_{R_i}(G)$ and look for the violators there!!!

BCNF Decomposition from 3NF Synthesis

- Attributes
  - *St* (student), *C* (course), *Sem* (semester), *P* (professor), *T* (time), *R* (room)
- FDs
  - $St\ C\ Sem \rightarrow P$
  - $P\ Sem \rightarrow C$
  - $C\ Sem\ T \rightarrow P$
  - $P\ Sem\ T \rightarrow C\ R$
  - $P\ Sem\ C\ T \rightarrow R$
  - $P\ Sem\ T \rightarrow C$

BCNF Decomposition from 3NF Synthesis

- Minimal Cover Step 1.
  - *St C Sem → P*
  - *P Sem → C*
  - *C Sem T → P*
  - ~~*P Sem T → C R*~~
    - *P Sem T → C (decomposition)*
    - *P Sem T → R (decomposition)*
  - *P Sem C T → R*
  - ~~*P Sem T → C (duplicate)*~~
- Let *F* denote this set.

BCNF Decomposition from 3NF Synthesis

- Minimal Cover Step 2.
  - *FD1. St C Sem → P*
  - *FD2. P Sem → C*
  - *FD3. C Sem T → P*
  - ~~*P Sem T → C R*~~
    - *FD4. P Sem T → C (decomposition)*
    - *FD5. P Sem T → R (decomposition)*
  - ~~***P Sem C T → R***~~
    - ~~*P Sem T → R (reduced and this is duplicate. So, discard)*~~
  - ~~*P Sem T → C (duplicate)*~~
- e.g., check for the first FD, *(St C)*$^+$*, (St Sem)*$^+$*, (C Sem)*$^+$
  - no redundant attribute in the first FD
  - ***(P Sem T)***$^+$ ***= P Sem C T R***

BCNF Decomposition from 3NF Synthesis

- Minimal Cover Step 3.
  - *FD1. St C Sem → P*
  - ***FD2. P Sem → C***
  - *FD3. C Sem T → P*
  - ~~*FD4. P Sem T → C (decomposition)*~~
  - *FD5. P Sem T → R (decomposition)*
- *Search for Removable redundant FDs*
  - $(St\ C\ Sem)^{+}_{\{F-FD1\}} = (St\ C\ Sem)$
    - *So, FD1 cannot be removed.*
    - *Nor for FD 2,3,5*
  - *FD4 is redundant (because of FD2)*

BCNF Decomposition from 3NF Synthesis

- 3NF decomposition from the minimal Cover
  - *(St C Sem P; St C Sem → P)   ; include P Sem C*
  - *(P Sem C; P Sem → C)*
  - *(C Sem T P; C Sem T → P)   ; include P Sem C*
  - *(P Sem T R; P Sem T → R)*
- *Super key in any of above? No*
  - *Add R_0 = (St T Sem P; {}) ← this is one possibility*
- *Are these all in BCNF?*
  - *First and third are not because of the FD "P Sem → C" in the second.*
  - *Remember that we have to check all the dependencies over the attributes of R_i that are implied by the original set of dependencies G.* i.e.,$\pi_{R_i}(G)$
  - *First is decomposed into: (P Sem C; P Sem → C) , (P Sem St; {})   : St C Sem → P is not preserved*
  - *Third is decomposed into: (P Sem C; P Sem → C) , (P Sem T; {}) : C Sem T → P is not preserved.*

Fourth Normal Form

redundancy

SSN	*PhoneN*	*ChildSSN*
111111	123-4444	222222
111111	123-4444	333333
111111	321-5555	222222
111111	321-5555	333333
222222	987-6666	444444
222222	777-7777	444444
222222	987-6666	555555
222222	777-7777	555555

Person

- Relation has redundant data
- In BCNF (since there are no non-trivial FDs)
- Redundancy is due to *set valued attributes* (in the E-R sense), not because of the FDs

Multi-Valued Dependency

- **Problem**: multi-valued (or binary join) dependency
  - **Definition**: If every instance of schema **R** can be (losslessly) decomposed using attribute sets (*X, Y*) such that:

$$\mathbf{r} = \pi_X(\mathbf{r}) \bowtie \pi_Y(\mathbf{r})$$

- then a *multi-valued dependency*

$$\mathbf{R} = \pi_X(\mathbf{R}) \bowtie \pi_Y(\mathbf{R})$$ holds in **r**

- Ex: Person= $\pi_{SSN,PhoneN}$(Person) $\bowtie$ $\pi_{SSN,ChildSSN}$(Person)

Fourth Normal Form (4NF)

- A schema is in ***fourth normal form*** (4NF), if for every MVD $R = X \bowtie Y$

  in that schema is either:

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

Fourth Normal Form (Cont'd)

- *Intuition*: if $X \cap Y \rightarrow R$, there is a unique row in relation **r** for each value of $X \cap Y$ (hence no redundancy)
  - Ex: *SSN* does not uniquely determine *PhoneN* or *ChildSSN*, thus Person is not in 4NF.

- *Solution*: Decompose R into X and Y
  - Decomposition is lossless – but not necessarily dependency preserving (since 4NF implies BCNF – next)

4NF Implies BCNF

- Suppose R is in 4NF and $X \rightarrow Y$ is a FD.
  - *Assume X and Y are disjoint*
  - $R_1 = XY$, $R_2 = R - Y$ is a lossless decomposition of R
  - Thus R has the MVD: $R = R_1 \bowtie R_2$

- Since R is in 4NF, one of the following must hold :
  - $XY \subseteq R - Y$
    - (an impossibility)
  - $R - Y \subseteq XY$
    - (i.e., $R = XY$ and X is a superkey)
  - $XY \cap R - Y$ $(= X)$ is a superkey
- Hence, $X \rightarrow Y$ satisfies BCNF condition

4NF Decomposition Algorithm

For simplicity, assume A and B are disjoint for FDs A→B in R

Input: R $= (\overline{R};\ \mathcal{D})$ /* $\mathcal{D}$ is a set of FDs and MVDs; FDs are treated as MVDs */
Output: A lossless decomposition of R where each schema is in 4NF.

Decomposition := {**R**} /* Initially decomposition consists of only one schema */
while there is a schema $\mathbf{S} = (\overline{S};\ \mathcal{D}')$ in *Decomposition* that is not in 4NF **do**
  /* Let $\overline{X} \bowtie \overline{Y}$ be an MVD in $\mathcal{D}^+$ such that $\overline{X}\,\overline{Y} \subseteq \overline{S}$ and
   it violates 4NF in **S**. Decompose using this MVD */
  Replace **S** in *Decomposition* with schemas $\mathbf{S}_1 = (\overline{X}\,\overline{Y};\ \mathcal{D}'_1)$ and
  $\mathbf{S}_2 = ((\overline{S} - \overline{Y}) \cup \overline{X};\ \mathcal{D}'_2)$, where $\mathcal{D}'_1 = \pi_{\overline{X}\,\overline{Y}}(\mathcal{D}')$ and $\mathcal{D}'_2 = \pi_{(\overline{S}-\overline{Y})\cup\overline{X}}(\mathcal{D}')$
end
return *Decomposition*

The algorithm is not correct. S1 and S2 should be
S1 = (X; D1')
S2 = (Y; D2);

Otherwise, X join Y should be replaced to X->>Y. (See slide 88)
If X ->> Y, R = XY join X(R-Y)

Projection of MVD on a Set of Attributes

- Projection of MVD $R = V \bowtie W$ on a set of attributes X
  - $X = (X \cap V) \bowtie (X \cap W)$, if $V \cap W \subseteq X$
  - Undefined, otherwise.

- Example
  - Projection of MVD: $ABCD = AB \bowtie BCD$ *on* ABC
    - $AB \cap BCD = B \subseteq ABC$. So, the projection is $AB \bowtie BC$
  - Projection of MVD: $ABCD = ACD \bowtie BD$ *on* ABC
    - $ACD \cap BD = D \not\subseteq ABC$. So, the projection is undefined.

4NF Decomposition Example

- Example
  - Attributes = {*ABCD*}
  - MVDs
    - MVD1. $ABCD = AB \bowtie BCD$
    - MVD2. $ABCD = ACD \bowtie BD$
    - MVD3. $ABCD = ABC \bowtie BCD$
  - From MVD1, decomposed to *AB, BCD*
    - Projection of remaining MVDs on *AB* is not defined
    - Projection of remaining MVDs on *BCD* is:
      - For MVD2, $BCD = CD \bowtie BD$
      - For MVD3, $BCD = BC \bowtie BCD$ (trivial)
  - Finally, *AB, BD, CD*

3NF Synthesis, BNCF, and 4NF Decomposition

- Example
  - Attributes = {$ABCDEFG$}
  - FDs = {$AB \rightarrow C$, $C \rightarrow B$, $BC \rightarrow DE$, $E \rightarrow FG$}
  - MVDs: R = BC $\bowtie$ ABDEFG, R=EF $\bowtie$ FGABCD
  - 3NF Synthesis result
    - $R_1 = (ABC; \{AB \rightarrow C, \underline{C \rightarrow B}\})$
    - $R_2 = (CBDE; \{C \rightarrow BDE\})$
    - $R_3 = (EFG; \{E \rightarrow FG\})$
  - R_1 is not in BCNF due to $C \rightarrow B$
    - $R_{11} = (BC; \{C \rightarrow B\})$, $R_{12} = (AC; \{\})$

3NF Synthesis & 4NF Decomposition (cont')

- Example
  - BCNF Synthesis result
    - $R_{11} = (AC; \{\})$, $R_{12} = (BC; \{C \rightarrow B\})$
    - $R_2 = (CBDE; \{C \rightarrow BDE\})$, $R_3 = (EFG; \{E \rightarrow FG\})$
  - MVDs: R = BC $\bowtie$ ABDEFG, R=EF $\bowtie$ FGABCD
  - The first MVD can be projected to R_2 (here, B = V $\cap$ W $\subseteq$ CBDE)
    - then, "*projected R_2*" = *BC* $\bowtie$ *BDE*. Is R_2 in 4NF?
    - No! because *BC* $\cap$ *BDE* = *B* and *B* is not the key
    - $R_{21} = (BC; \{C \rightarrow B\})$, $R_{22} = (BDE; \{\})$
  - Similarly, the second MVD can be projected to R_3

    (here, F = V $\cap$ W $\subseteq$ EFG)
    - then, "*projected R_3*" = EF $\bowtie$ FG. Is R_3 in 4NF?
    - No! because EF $\cap$ *FG* = *F* and *F* is not the key
    - $R_{31} = (EF; \{E \rightarrow F\})$, $R_{22} = (GF; \{\})$

SUNY Korea

Customary Representation of MVDs

- Customary representation of MVDs
  - MVD R = V ⋈ W over R = (R; D), where
    - X = V ∩ W
    - X ∪ Y = V or X ∪ Y = W

    are represented as X ↠ Y
  - i.e., R = XY ⋈ X(R-Y)

- Another way of defining MVD in a relation
  - X ↠ Y then,
    - $\forall\ tuple\ t, u\ \in R : t[X] = u[X].\ then\ \exists\ tuple\ v \in R$ where
      - $v[X] = t[X]\ and$
      - $v[Y] = t[Y]\ and$
      - $v[rest] = u[rest]$

Examples

- Apply (SSN, college, hobby)
  - SSN $\twoheadrightarrow$ college

- Apply (SSN, college, date, major)
  - Requirements
    - Apply once to each college
    - May apply to multiple majors
  - We can derive…
    - SSN, college → date, major / date → college
    - SSN $\twoheadrightarrow$ college, date
      - What is the real world constraint encoded by the MVD above?
      - A student must apply to the same set of majors at all colleges.

4NF Decomposition Algorithm (Rewritten)

Input: relation R + FDs for R + MVDs for R

Output: decomposition of R into 4NF relations with "lossless join"

Compute keys for R

Repeat until all relations are in 4NF:

- Pick any R' with nontrivial $A \twoheadrightarrow B$ that violates 4NF
- Decompose R' into $R_1(A, B)$ and $R_2(A, rest)$
- Compute FDs and MVDs for R_1 and R_2
- Compute keys for R_1 and R_2