Lecture 2

Database Management Systems

Winter 2004

CMPUT 391: Database Design Theory

or Relational Normalization Theory

Dr. Osmar R. Zaïane

University of Alberta

Chapter 8 of Textbook

Based on slides by Lewis, Bernstein and Kifer.

Limitations of Relational Database Designs

- Provides a set of guidelines, does not result in a unique database schema
- Does not provide a way of evaluating alternative schemas
- Pitfalls:
  - Repetition of information
  - Inability to represent certain information
  - Loss of information

➢ Normalization theory provides a mechanism for analyzing and refining the schema produced by an E-R design

Redundancy

- Dependencies between attributes cause redundancy
  - Ex. 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
	…………………		

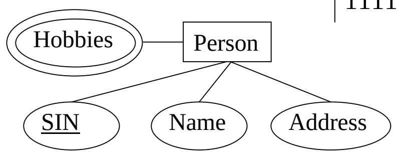
Redundancy

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

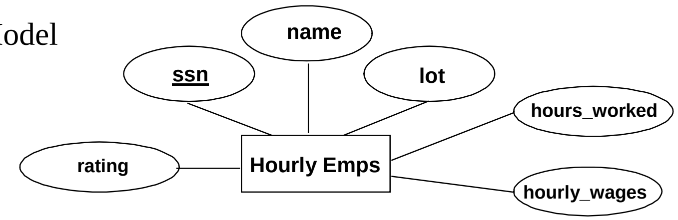
- **Solution**: use two relations to store Person information
  - Person1 (*SSN*, *Name*, *Address*)
  - Hobbies (*SSN*, *Hobby*)
- The decomposition is more general: people with hobbies can now be described
- No update anomalies:
  - Name and address stored once
  - A hobby can be separately supplied 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*)

Example

ER Model



Relational Model

- Hourly_Emps (<u>*ssn*</u>, *name*, *lot*, *rating*, *hrly_wages*, *hrs_worked*)
- Some functional dependencies on Hourly_Emps:
  - *ssn* is the key: S ⟶ SNLRWH
  - *rating* determines *hrly_wages*: R ⟶ W
- Are there anomalies?

Functional Dependencies

- **Definition:** A *functional dependency* (FD) on a relation schema **R** is a <u>constraint</u> ***X*** ⟶ ***Y*,** where *X* and *Y* are subsets of attributes of **R.**
- **Definition**: An FD ***X*** ⟶ ***Y*** is *satisfied* in an instance **r** of **R** if for <u>every</u> 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*
- **Definition**: A *constraint* on a relation schema R is a condition that has to be *satisfied in every allowable instance of* R.
  - FDs must be identified based on semantics of application.
  - Given a particular allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over the schema R!
  - A key constraint is a special kind of functional dependency: all attributes of relation occur on the right-hand side of the FD:
    - *SSN* ⟶ *SSN, Name, Address*

Functional Dependencies

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

Functional Dependency - Example

- Brokerage firm 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*)
    - keys are (*ClientId*, *OfficeId*), (*AcctNum*, *ClientId*)
  - *ClientId*, *OfficeId* ⟶ *AcctNum*
  - *AcctNum* ⟶ *OfficeId*
    - 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: $F = \{A \to B, B \to C\}$ and f is $A \to C$
    - If *Streetaddr* $\to$ *Town* and *Town* $\to$ *Zip* then *Streetaddr* $\to$ *Zip*
- **Definition**: The *closure* of F, denoted 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 in number for infinite domains
    - Even for finite domains, we have to look at relations of *all* arities
  - **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*

Armstrong's Axioms for FDs

- This is the *syntactic* way of computing/testing the various properties of FDs
- **Reflexivity**: If $Y \subseteq X$ then $X \to Y$ (trivial FD)
  - *Name*, *Address* $\to$ *Name*
- **Augmentation**: If $X \to Y$ then $XZ \to YZ$
  - If *Town* $\to$ *Zip* then *Town*, *Name* $\to$ *Zip*, *Name*
- **Transitivity**: If $X \to Y$ and $Y \to Z$ then $X \to Z$

Armstrong's Axioms for FDs (cont.)

- Two more rules (which can be derived from the axioms) can be useful:
  - **Union**: If $X \to Y$ and $X \to Z$ then $X \to YZ$
  - **Decomposition**: If $X \to YZ$ then $X \to Y$ and $X \to Z$

Soundness and Completeness

- Axioms are *sound*: If an FD $f: X \rightarrow Y$ can be derived from a set of FDs F using the axioms, then f holds in every relation that satisfies every FD in F.
- 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 F^+ (the set of possible FD's is finite so this can be done) and see if f is in F^+

Reflexivity

- If $Y \subseteq X$, then $X \rightarrow Y$
- $R=(\overbrace{A,B,\underbrace{C,D}_{Y}}^{X},E)$

$t_1 =(a_1,b_1,c_1,d_1,e_1)$
$t_2 =(a_2,b_2,c_2,d_2,e_2)$
$\pi_X(t_1)= \pi_X(t_2) \Rightarrow$
$a_1 = a_2, b_1 = b_2, \underbrace{c_1 = c_2, d_1 = d_2}$

$\pi_Y(t_1)= \pi_Y(t_2) \Leftarrow$

Augmentation

- If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
- $R=(\overbrace{A,B}^{X},\underbrace{C,D}_{Y},\overbrace{E}^{Z})$

$t_1 =(a_1,b_1,c_1,d_1,e_1)$
$t_2 =(a_2,b_2,c_2,d_2,e_2)$
$\pi_{XZ}(t_1)= \pi_{XZ}(t_2) \Rightarrow$
$a_1 = a_2, b_1 = b_2, e_1 = e_2$
Since $X \rightarrow Y$ and $e_1 = e_2$
then $c_1 = c_2, d_1 = d_2, e_1 = e_2$
$\pi_{YZ}(t_1)= \pi_{YZ}(t_2)$

Transitivity

- If $X \rightarrow Y$, and $Y \rightarrow Z$ then $X \rightarrow Z$
- $R=(\overbrace{A,B}^{X},\underbrace{C,D}_{Y},\overbrace{E}^{Z})$

$t_1 =(a_1,b_1,c_1,d_1,e_1)$
$t_2 =(a_2,b_2,c_2,d_2,e_2)$
assume $X \rightarrow Y$ and $Y \rightarrow Z$
$\pi_X(t_1)= \pi_X(t_2) \Rightarrow$
$a_1 = a_2, b_1 = b_2$
Since $X \rightarrow Y$ then $c_1 = c_2, d_1 = d_2$
$\Rightarrow \pi_Y(t_1)= \pi_Y(t_2)$
Since $Y \rightarrow Y$ then $e_1 = e_2$
$\Rightarrow \pi_Z(t_1)= \pi_Z(t_2)$

Generating F^+

$\underline{F}$

$AB \rightarrow C$ ⋯ *union* ⋯ $AB \rightarrow BCD$ ⋯ *trans* ⋯ $AB \rightarrow BCDE$ ⋯ *decomp* ⋯ $AB \rightarrow CDE$

$A \rightarrow D$ ⋯ *aug* ⋯ $AB \rightarrow BD$

$D \rightarrow E$ ⋯ *aug* ⋯ $BCD \rightarrow BCDE$

Thus, $AB \rightarrow BD$, $AB \rightarrow BCD$, $AB \rightarrow BCDE$, and $AB \rightarrow CDE$ 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, F, (denoted X^+_F) is the set of all attributes, A, such that $X \rightarrow A$
  - X^+_{F1} is not necessarily the same as X^+_{F2} if $F1 \neq F2$
- *Attribute closure and entailment*:
  - Algorithm: Given a set of FDs, F, then $X \rightarrow Y$ 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; // *since* $X \subseteq X^+_F$
repeat
  old := *closure*;
  if there is an FD $Z \rightarrow V$ in F such that
    $Z \subseteq$ *closure* **and** $V \not\subseteq$ *closure*
   then *closure* := *closure* $\cup$ V
until *old* = *closure*

– If $T \subseteq$ *closure* then $X \rightarrow T$ is entailed by F

Example: Computation of Attribute Closure

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

$AB \to C$ (a)
$A \to D$ (b)
$D \to E$ (c)
$AC \to B$ (d)

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 non-key attribute is dependent on part of a key*; has no practical or theoretical value – won't discuss
- The two commonly used normal forms are *third normal form* (3NF) and *Boyce-Codd normal form* (BCNF)

BCNF

- **Definition**: A relation schema **R** is in BCNF if for every FD $X \to Y$ 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* $\to$ *Name*, *Address*
  - Since *SSN* is a key, Person1 is in BCNF

(non) BCNF Examples

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

Redundancy

- Suppose **R** has a FD $A \rightarrow B$. 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); 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)

3NF Example

- HasAccount (*AcctNum, ClientId, OfficeId*)
  - *ClientId, OfficeId* $\rightarrow$ *AcctNum*
    - OK since LHS contains a key
  - *AcctNum* $\rightarrow$ *OfficeId*
    - OK since RHS is part of a key
- HasAccount is in 3NF but it might still contain redundant information due to *AcctNum* → *OfficeId* (which is not allowed by BCNF)

3NF Example

- HasAccount might store redundant data:

ClientId	*OfficeId*	*AcctNum*
1111	Stony Brook	28315
2222	Stony Brook	28315
3333	Stony Brook	28315

3NF: *OfficeId* part of key
FD: *AcctNum* $\rightarrow$ *OfficeId*

redundancy

- Decompose to eliminate redundancy:

ClientId	*AcctNum*
1111	28315
2222	28315
3333	28315

BCNF (only trivial FDs)

OfficeId	*AcctNum*
Stony Brook	28315

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

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

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
  - We will see why

Decomposition

- Schema $\mathbf{R} = (R, F)$
  - R is set a of attributes
  - F is a set of functional dependencies over R
    - Each key is described by a FD
- The ***decomposition of schema*** $\mathbf{R}$ is a collection of schemas $\mathbf{R}_i = (R_i, F_i)$ where
  - $R = \cup_i R_i$ for all i (*no new attributes*)
  - F_i is a set of functional dependences involving only attributes of R_i
  - F entails F_i for all i (*no new FDs*)
- The ***decomposition of an instance***, $\mathbf{r}$, of $\mathbf{R}$ 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, $\mathbf{R}$, is *lossless* if every valid instance, $\mathbf{r}$, of $\mathbf{R}$ can be reconstructed from its components:

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

- 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 $\mathbf{R} = (R, F)$ into $\mathbf{R}_1 = (R_1, F_1)$ and $\mathbf{R}_2 = (R_2, F_2)$ is lossless *if and only if* :
  - either the FD
    - $(R_1 \cap R_2) \rightarrow R_1$ is in F^+
  - or the FD
    - $(R_1 \cap R_2) \rightarrow R_2$ is in F^+

Intuitively: the attributes common to R_1 and R_2 must contain a key for either R_1 or R_2.

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

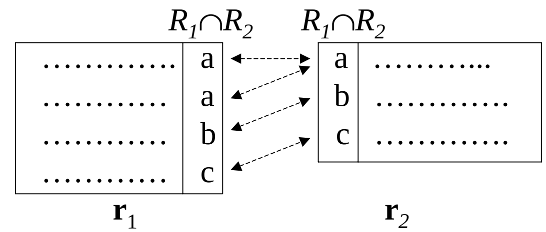
$R_2 = \{SSN, Hobby\}$

$F_2 = \{\ \}$

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)



Dependency Preservation

- Consider a decomposition of $\mathbf{R} = (R, F)$ into $\mathbf{R}_1 = (R_1, F_1)$ and $\mathbf{R}_2 = (R_2, F_2)$
  – An FD $X \rightarrow Y$ of F is in F_i iff $X \cup Y \subseteq R_i$
  – An FD, $f \in F$ may be in neither F_1, nor F_2, nor even $(F_1 \cup F_2)^+$
    - 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 <u>locally</u>, 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 F and $F_1 \cup F_2$ are equivalent: $F^+ = (F_1 \cup F_2)^+$
  – Then checking all FDs in F, as $\mathbf{r}_1$ and $\mathbf{r}_2$ are updated, can be done by checking F_1 in $\mathbf{r}_1$ and F_2 in $\mathbf{r}_2$

Dependency Preservation

- If f is an FD in F, but f is not in $F_1 \cup F_2$, there are two possibilities:
  – $f \in (F_1 \cup F_2)^+$
    - If the constraints in F_1 and F_2 are maintained, f will be maintained automatically.
  – $f \notin (F_1 \cup F_2)^+$
    - f can be checked only by first taking the join of r_1 and r_2. This is costly.

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

Example

- Schema: $(ABC;\ F)$, $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow B\}$
- Decomposition:
  - (AC, F_1), $F_1 = \{A \rightarrow C\}$
    - Note: $A \rightarrow C \notin F$, but in F^+
  - (BC, F_2), $F_2 = \{B \rightarrow C, C \rightarrow B\}$

- $A \rightarrow B \notin (F_1 \cup F_2)$, **but** $A \rightarrow B \in (F_1 \cup F_2)^+$.
  - So $F^+ = (F_1 \cup F_2)^+$ and thus the decompositions is still dependency preserving

Example

- HasAccount (*AccountNumber, ClientId, OfficeId*)
  f_1: *AccountNumber* $\rightarrow$ *OfficeId*
  f_2: *ClientId, OfficeId* $\rightarrow$ *AccountNumber*
- Decomposition:
  AcctOffice = (*AccountNumber, OfficeId*; {*AccountNumber* $\rightarrow$ *OfficeId*})
  AcctClient = (*AccountNumber, ClientId*; {})
- Decomposition is lossless: $R_1 \cap R_2 = \{AccountNumber\}$ and *AccountNumber* $\rightarrow$ *OfficeId*
- In BCNF
- Not dependency preserving: $f_2 \notin (F_1 \cup F_2)^+$
- HasAccount *does not* have BCNF decompositions that are both lossless and dependency preserving! (Check, eg, by enumeration)
- Hence: BCNF+lossless+dependency preserving decompositions are not always possible!

BCNF Decomposition Algorithm

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

Decomp := **R**

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

Find $X \rightarrow Y \in F'$ that violates BCNF // *X isn't a superkey in* **S**

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

// 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*

Example

Given: **R** = (*R; T*) where *R* = *ABCDEFGH* and

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

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:

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

$\mathbf{R_2} = (ABCFGH; \{ABH \rightarrow C, BGH \rightarrow F, F \rightarrow AH, BH \rightarrow G\})$

Note 1: $\mathbf{R_1}$ is in BCNF

Note 2: Decomposition is *lossless* since *A* is a key of $\mathbf{R_1}$.

Note 3: FDs $F \rightarrow D$ and $BH \rightarrow E$ are not in T_1 or T_2. But both can be derived from $T_1 \cup T_2$

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

Hence, decomposition is *dependency preserving*.

Properties of BCNF Decomposition Algorithm

Let $X \rightarrow Y$ violate BCNF in $\mathbf{R} = (R,F)$ and $\mathbf{R_1} = (R_1,F_1)$, $\mathbf{R_2} = (R_2,F_2)$ is the resulting decomposition. Then:

- There are *fewer violations* of BCNF in $\mathbf{R_1}$ and $\mathbf{R_2}$ than there were in **R**
  - $X \rightarrow Y$ implies *X* is a key of $\mathbf{R_1}$
  - Hence $X \rightarrow Y \in F_1$ does not violate BCNF in $\mathbf{R_1}$ and, since $X \rightarrow Y \notin F_2$, does not violate BCNF in $\mathbf{R_2}$ either
  - Suppose *f* is $X' \rightarrow Y'$ and $f \in F$ doesn't violate BCNF in **R**. If $f \in F_1$ or 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 $\mathbf{R_2}$.
- The decomposition is *lossless*
  - Since $F_1 \cap F_2 = X$

Example (con't)

Given: $\mathbf{R_2} = (ABCFGH; \{ABH \rightarrow C, BGH \rightarrow F, F \rightarrow AH, BH \rightarrow G\})$

step 1: Find a FD that violates BCNF.

Not $ABH \rightarrow C$ or $BGH \rightarrow F$, since *BH* is a key of $\mathbf{R_2}$

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

step 2: Split $\mathbf{R_2}$ into:

$\mathbf{R_{21}} = (FAH, \{F \rightarrow AH\})$

$\mathbf{R_{22}} = (BCFG; \{\})$

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

Note 2: The decomposition is *lossless* (since *F* is a key of $\mathbf{R_{21}}$)

Note 3: FDs $ABH \rightarrow C$, $BGH \rightarrow F$, $BH \rightarrow G$ are not in T_{21} or T_{22}, and they can't be derived from $T_1 \cup T_{21} \cup T_{22}$. Hence the decomposition is *not* dependency-preserving

Properties of BCNF Decomposition Algorithm

- A BCNF decomposition is *not necessarily* dependency preserving
- But *always* lossless
- BCNF+lossless+dependency preserving is sometimes unachievable (recall HasAccount)

Third Normal Form

- Compromise – Not all redundancy removed, but dependency preserving decompositions are <u>always</u> possible (and, of course, lossless)
- 3NF decomposition is based on a *minimal cover*

Minimal Cover

- A *minimal cover* of a set of dependencies, T, is a set of dependencies, U, such that:
  - U is equivalent to T $(T^+ = U^+)$
  - All FDs in U have the form $X \to A$ 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*

Computing Minimal Cover

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

Computing Minimal Cover (con't)

- **step 3**: Delete redundant FDs from T
  - *Algorithm*: If T - $\{f\}$ entails f, then f is redundant
    - If f is $X \to A$ then check if $A \in X^+_{T-\{f\}}$
  - Example: $BGH \to F$ is entailed by $E \to F$, $BH \to E$, so it is redundant
- *Note*: Steps 2 and 3 cannot be reversed!! See the textbook for a counterexample

Synthesizing a 3NF Schema

Starting with a schema $\mathbf{R} = (R, T)$

- **step 1**: Compute a minimal cover, U, of T. The decomposition is based on U, but since $U^+ = T^+$ the same functional dependencies will hold
  - A minimal cover for $T=\{ABH\rightarrow CK, A\rightarrow D, C\rightarrow E, BGH\rightarrow F, F\rightarrow AD, E\rightarrow F, BH \rightarrow E\}$

    is

    $U=\{BH\rightarrow C, BH\rightarrow K, A\rightarrow D, C\rightarrow E, F\rightarrow A, E\rightarrow F\}$

Synthesizing a 3NF schema (con't)

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

Synthesizing a 3NF schema (con't)

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

Synthesizing a 3NF schema (con't)

- **step 4**: If no R_i is a superkey of $\mathbf{R}$, add schema $\mathbf{R_0} = (R_0,\{\})$ where R_0 is a key of $\mathbf{R}$.
  - $\mathbf{R_0} = (BGH, \{\})$
    - $\mathbf{R_0}$ might be needed when not all attributes are necessarily contained in $R_1 \cup R_2 \dots \cup R_n$
      - 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 $R_1 \cup R_2 \dots \cup R_n$
      - Example: $(ABCD; \{A\rightarrow B, C\rightarrow D\})$. Step 3 decomposition: $R_1 = (AB; \{A\rightarrow B\})$, $R_2 = (CD; \{C\rightarrow D\})$. 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* → *Name*

Fourth Normal Form

redundancy

SSN	*PhoneN*	*ChildSSN*
111111	123-4444	222222
111111	123-4444	333333
222222	987-6666	444444
222222	555-5555	444444

Person

- Relation has redundant data
- Yet it is 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 non-trivial multi-valued dependency:

$$R = X \bowtie Y$$

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 an FD.
  - $R1 = XY$, $R2 = R\text{-}Y$ is a lossless decomposition of R
  - Thus R has the multi-valued dependency:

$$R = R_1 \bowtie R_2$$

  - 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