# Relational Normalization Theory

CSE 532, Theory of Database Systems

Stony Brook University

http://www.cs.stonybrook.edu/~cse532

# 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
  - 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

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*)

# Functional Dependencies

- **Definition:** A *functional dependency* (FD) on a relation schema **R** is a constraint $\boldsymbol{X \rightarrow Y}$**,** where $X$ and $Y$ are subsets of attributes of **R.**
- **Definition**: An FD $\boldsymbol{X \rightarrow Y}$ 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*)
    - keys are (*ClientId, OfficeId*), (*AcctNum, ClientId*)
  - *Client, OfficeId* $\rightarrow$ *AcctNum*
  - *AcctNum* $\rightarrow$ *OfficeId*
    - Thus, attribute values need not depend only on key values

# Entailment, Closure, Equivalence

- **Definition**: If $\boldsymbol{F}$ is a set of FDs on schema **R** and $f$ is another FD on **R**, then $\boldsymbol{F}$ *entails* $f$ if every instance **r** of **R** that satisfies every FD in $\boldsymbol{F}$ 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 $\boldsymbol{F}$, denoted $\boldsymbol{F}^+$, is the set of all FDs entailed by $\boldsymbol{F}$
- **Definition**: $\boldsymbol{F}$ and $\boldsymbol{G}$ are *equivalent* if $\boldsymbol{F}$ entails $\boldsymbol{G}$ and $\boldsymbol{G}$ entails $\boldsymbol{F}$

# Entailment (cont.)

- Satisfaction, entailment, and equivalence are <u>*semantic*</u> concepts – defined in terms of the actual relations in the "real world."
  - They define <u>*what these notions are*</u>, **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* arities
  - **Solution**: find algorithmic, <u>*syntactic*</u> ways to compute these notions
    - <u>*Important*</u>: 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 $f$: $X \to Y$ 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 $X \to Y$ and $X \to Z$ 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, $X \to Y Z$ is satisfied in every relation where both $X \to Y$ and $X \to Z$ 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 $\boldsymbol{F}$ entails $f$, then $f$ can be derived from $\boldsymbol{F}$ using the axioms
- A consequence of completeness is the following (naïve) algorithm to determining if $\boldsymbol{F}$ 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 $F^+$

***F***

$AB \rightarrow C$

$A \rightarrow D \xrightarrow{aug} AB \rightarrow BD$

$D \rightarrow E \xrightarrow{aug} BCD \rightarrow BCDE$

$AB \rightarrow C,\ AB \rightarrow BD \xrightarrow{union} AB \rightarrow BCD$

$AB \rightarrow BCD,\ BCD \rightarrow BCDE \xrightarrow{trans} AB \rightarrow BCDE \xrightarrow{decomp} AB \rightarrow CDE$

Thus, $AB \rightarrow BD$, $AB \rightarrow BCD$, $AB \rightarrow BCDE$, and $AB \rightarrow CDE$ are all elements of $F^+$ (part-of, there are other FDs: $AC \rightarrow CD$, $AE \rightarrow ED$, *etc.*)

# 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 $X^+_{\boldsymbol{F}}$) is the set of all attributes, $A$, such that $X \rightarrow A$
  - $X^+_{\boldsymbol{F1}}$ is not necessarily the same as $X^+_{\boldsymbol{F2}}$ if $\boldsymbol{F1} \neq \boldsymbol{F2}$
- *Attribute closure and entailment*:
  - Algorithm: Given a set of FDs, $\boldsymbol{F}$, then $X \rightarrow Y$ if and only if $X^+_{\boldsymbol{F}} \supseteq Y$

# Example - Computing Attribute Closure

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

| $X$ | $X_{\boldsymbol{F}}^{+}$ |
|---|---|
| $A$ | $\{A, D, E\}$ |
| $AB$ | $\{A, B, C, D, E\}$ (Hence $AB$ is a key) |
| $B$ | $\{B\}$ |
| $D$ | $\{D, E\}$ |

Is $AB \rightarrow E$ entailed by $\boldsymbol{F}$? *Yes*
Is $D \rightarrow C$ entailed by $\boldsymbol{F}$? *No*

*Result*: $X_{\boldsymbol{F}}^{+}$ allows us to determine FDs of the form $X \rightarrow Y$ entailed by $\boldsymbol{F}$

# Computation of Attribute Closure $X^+_F$

$$
\begin{array}{l}
\mathit{closure} := X; \qquad\qquad\qquad // \text{ } \mathit{since} \ \ X \subseteq X^+_{\boldsymbol{F}} \\
\textbf{repeat} \\
\quad \mathit{old} := \mathit{closure}; \\
\quad \textbf{if} \text{ there is an FD } Z \rightarrow V \text{ in } \boldsymbol{F} \text{ such that} \\
\qquad\qquad Z \subseteq \mathit{closure} \ \textbf{and} \ V \not\subseteq \mathit{closure} \\
\qquad \textbf{then} \ \ \mathit{closure} := \mathit{closure} \cup V \\
\textbf{until} \ \mathit{old} = \mathit{closure}
\end{array}
$$

– If $T \subseteq \mathit{closure}$ 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 :

$$
\begin{aligned}
AB &\rightarrow C \quad &\text{(a)}\\
A &\rightarrow D \quad &\text{(b)}\\
D &\rightarrow E \quad &\text{(c)}\\
AC &\rightarrow B \quad &\text{(d)}
\end{aligned}
$$

**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) – a research lab accident; has no practical or theoretical value:
  - no non prime attribute is dependent on any proper subset of any candidate key of the table (where a non prime attribute of a table is an attribute that is not a part of any candidate key of the table): every non-prime attribute is either dependent on the whole of a candidate key, or on another non prime attribute.
- 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 $X \rightarrow Y$ associated with **R** either
  - $Y \subseteq X$ (i.e., the FD is trivial) or
  - $X$ is a superkey of **R**
    - Remember: a *superkey* is a combination of attributes that can be used to uniquely identify a database record. A table might have many superkeys.
    - Remember: a *candidate* key is a special subset of superkeys that do not have any extraneous information in them: it is a **minimal** superkey.
- **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 <u>not</u> satisfy requirements of BCNF
    - since **the key is (*SSN, Hobby*)**
- HasAccount (*AcctNum, ClientId, OfficeId*)
  - The FD *AcctNum*→ *OfficeId* does <u>not</u> satisfy BCNF requirements
    - since keys are (*ClientId, OfficeId*) and (*AcctNum, ClientId*); not *AcctNum*.

# 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); or
  - $X$ is a superkey of **R;** or
  - Every $A \in Y$ is part of some key of **R**
- 3NF is weaker than BCNF (every schema that is in BCNF is also in 3NF)

*BCNF conditions*

# 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 (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

# Decomposition

- Schema $\mathbf{R} = (R, \boldsymbol{F})$
  - $R$ is set a of attributes
  - $\boldsymbol{F}$ 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 $\mathbf{R}_i = (R_i, \boldsymbol{F}_i)$ 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*, $\mathbf{r}$, 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

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

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

can be decomposed into

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

$\boldsymbol{F}_1 = \{SSN \rightarrow Name, Address\}$

and

$R_2 = \{SSN, Hobby\}$

$\boldsymbol{F}_2 = \{ \}$

# 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:

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

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

# Lossy Decomposition

*The following is always the case:*

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

*But the following is* ***not always true****:*

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

*Example:* $\mathbf{r} \not\supseteq \mathbf{r}_1 \bowtie \mathbf{r}_2$

**r**

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

**r**$_1$

| *SSN* | *Name* |
|---|---|
| 1111 | Joe |
| 2222 | Alice |
| 3333 | Alice |

**r**$_2$

| *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 $\mathbf{R} = (R, \boldsymbol{F})$ 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

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

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

can be decomposed into

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

$\boldsymbol{F}_1$ = {$SSN \rightarrow Name, Address$}

and

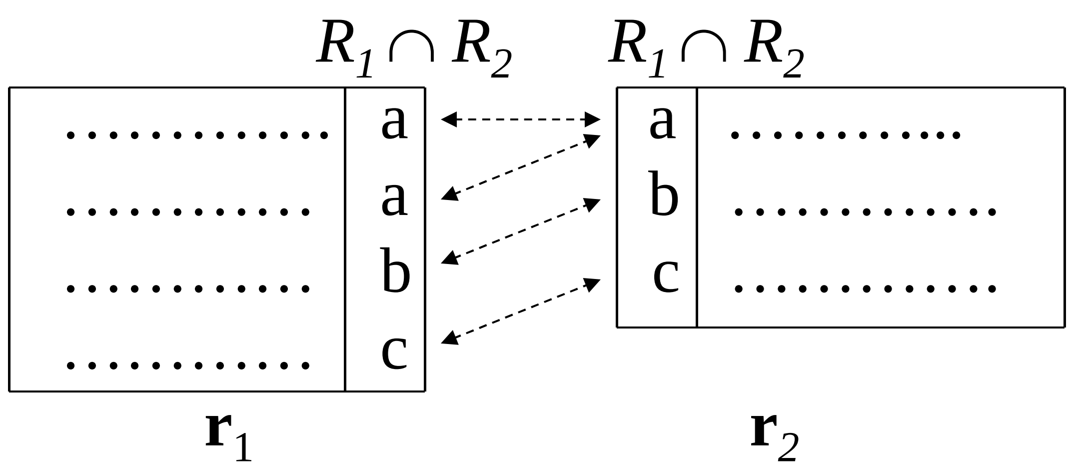
$R_2$ = {$SSN$, $Hobby$}

$\boldsymbol{F}_2$ = { }

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

# Intuition Behind the Test for Losslessness

- Suppose $R_1 \cap R_2 \to R_2$. 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)



# Proof of Lossless Condition

- $r \subseteq r_1 \bowtie r_2$ – *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

  $card\ (\mathbf{r}_1 \bowtie r_2) = card\ (\mathbf{r}_1)$

  *(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 $F_2$ 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.

# Example

Schema $(R, \boldsymbol{F})$ where

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

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

can be decomposed into

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

$$\boldsymbol{F}_1 = \{SSN \rightarrow Name, Address\}$$

and

$$R_2 = \{SSN, Hobby\}$$

$$\boldsymbol{F}_2 = \{\ \}$$

Since $\boldsymbol{F} = \boldsymbol{F}_1 \cup \boldsymbol{F}_2$ the decomposition is dependency preserving

# Example

- Schema: $(ABC;\ F)$ , $\boldsymbol{F} = \{A \rightarrow B, B \rightarrow C, C \rightarrow B\}$
- Decomposition:
  - $(AC, \boldsymbol{F}_1)$, $\boldsymbol{F}_1 = \{A \rightarrow C\}$
    - Note: $A \rightarrow C \notin \boldsymbol{F}$, 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 $AcctNum \rightarrow OfficeId$

- 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, eg, by enumeration)

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

# BCNF Decomposition Algorithm

***Input*:** $\mathbf{R} = (R; \boldsymbol{F})$

$Decomp := \mathbf{R}$

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

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

Replace $\mathbf{S}$ in $Decomp$ with $\mathbf{S_1} = (XY; \boldsymbol{F}_1)$, $\mathbf{S_2} = (S - (Y - X); \boldsymbol{F}_2)$

// $\boldsymbol{F}_1$ = *all FDs of* $\boldsymbol{F}'$ *involving only attributes of* $XY$

// $\boldsymbol{F}_2$ = *all FDs of* $\boldsymbol{F}'$ *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** = ($R$; $\boldsymbol{F}$) where $R = ABCDEGHK$ and

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

**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:

$\mathbf{R_1} = (ADE, \boldsymbol{F}_1 = \{A \rightarrow DE\})$

$\mathbf{R_2} = (ABCGHK; \boldsymbol{F}_1 = \{ABH \rightarrow C, BGH \rightarrow K, K \rightarrow AH, BH \rightarrow G\})$

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

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

Note 3: FDs $K \rightarrow D$ and $BH \rightarrow E$ are not in $\boldsymbol{F}_1$ or $\boldsymbol{F}_2$. But

both can be derived from $\boldsymbol{F}_1 \cup \boldsymbol{F}_2$

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

Hence, decomposition <u>*is*</u> *dependency preserving*.

# Example (con't)

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

**step 1:** Find a FD that violates BCNF.

Not $ABH \to C$ or $BGH \to K$, since $BH$ is a key of $\mathbf{R_2}$

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

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

$\mathbf{R_{21}} = (KAH,\ \boldsymbol{F}_{21} = \{K \to AH\})$

$\mathbf{R_{22}} = (BCGK;\ \boldsymbol{F}_{22} = \{\})$

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

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

Note 3: FDs $ABH \to C$, $BGH \to K$, $BH \to G$ are not in $\boldsymbol{F}_{21}$ or $\boldsymbol{F}_{22}$, and they can't be derived from $\boldsymbol{F}_1 \cup \boldsymbol{F}_{21} \cup \boldsymbol{F}_{22}$.
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 $f$ is $X' \rightarrow Y'$ and $f \in \boldsymbol{F}$ doesn't violate BCNF in $\mathbf{R}$. 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 $\mathbf{R}$ 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

- 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, $\boldsymbol{F}$, is a set of dependencies, $\boldsymbol{U}$, such that:
  - $U$ is equivalent to $\boldsymbol{F}$ $(\boldsymbol{F}^+ = \boldsymbol{U}^+)$
  - All FDs in $\boldsymbol{U}$ have the form $X \rightarrow A$ where $A$ is a single attribute
  - It is not possible to make $\boldsymbol{U}$ 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**: $\boldsymbol{F} = \{ABH \rightarrow CK, A \rightarrow D, C \rightarrow E,$

  $$BGH \rightarrow L, L \rightarrow AD, E \rightarrow L, BH \rightarrow E\}$$

- **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)

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

- *Note*: The order of steps 2 and 3 cannot be interchanged!! See the textbook for a counterexample

# Synthesizing a 3NF Schema

## Starting with a schema **R** = ($R$, $\boldsymbol{F}$)

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

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

is

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

# Synthesizing a 3NF schema (con't)

- **step 2**: Partition $\boldsymbol{U}$ into sets $\boldsymbol{U}_1$, $\boldsymbol{U}_2$, … $\boldsymbol{U}_n$ such that the LHS of all elements of $\boldsymbol{U}_i$ are the same

$$\boldsymbol{U}_1 = \{BH \to C, BH \to K\}, U_2 = \{A \to D\},$$

$$\boldsymbol{U}_3 = \{C \to E\}, U_4 = \{L \to A\}, U_5 = \{E \to L\}$$

# Synthesizing a 3NF schema (con't)

- **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*

$$\mathbf{R_1} = (BHCK;\ BH \to C, BH \to K),$$

$$\mathbf{R_2} = (AD;\ A \to D),$$

$$\mathbf{R_3} = (CE;\ C \to E),$$

$$\mathbf{R_4} = (AL;\ L \to A),$$

$$\mathbf{R_5} = (EL;\ E \to L)$$

# 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 $R_1 \cup R_2 \ldots \cup R_n$
      - 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

| *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

*redundancy*

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

# Fourth Normal Form (4NF)

- A schema is in *fourth normal form* (4NF) if for every multi-valued dependency

$$R = X \bowtie Y$$

in that schema, 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.
  - $R1 = XY,\ R2 = R - Y$ 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