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

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Town</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Joe</td>
<td>Stony Brook</td>
<td>11790</td>
</tr>
<tr>
<td>4321</td>
<td>Mary</td>
<td>Stony Brook</td>
<td>11790</td>
</tr>
<tr>
<td>5454</td>
<td>Tom</td>
<td>Stony Brook</td>
<td>11790</td>
</tr>
</tbody>
</table>

...
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**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td>{biking, hiking}</td>
</tr>
</tbody>
</table>

**Relational Model**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td>biking</td>
</tr>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td>hiking</td>
</tr>
</tbody>
</table>

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 \( X \rightarrow Y \), where \( X \) and \( Y \) are subsets of attributes of \( R \).

- **Definition:** An FD \( X \rightarrow Y \) 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 → 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
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 → AcctNum
  - AcctNum → 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: $F = \{A \rightarrow B, B \rightarrow C\}$ and $f$ is $A \rightarrow C$
    - If Town $\rightarrow$ Zip and Zip $\rightarrow$ AreaCode then Town $\rightarrow$ AreaCode

- **Definition:** The **closure** of $F$, denoted $F^+$, 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 \( X Z \rightarrow YZ \)
  - 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 $F$ using the axioms, then $f$ holds in every relation that satisfies every FD in $F$.

- Example: Given $X \rightarrow Y$ and $X \rightarrow Z$ then

  $X \rightarrow XY$  
  $YX \rightarrow YZ$  
  $X \rightarrow YZ$  

  *Augmentation by $X$*  
  *Augmentation by $Y$*  
  *Transitivity*

- 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 (naive) 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 $F^+$.
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 $F^+$

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 $X \rightarrow A$.
  - $X^+_{F_1}$ is not necessarily the same as $X^+_{F_2}$ if $F_1 \neq F_2$.

- *Attribute closure and entailment*:
  - **Algorithm**: Given a set of FDs, $F$, then
    $$X \rightarrow Y \text{ if and only if } X^+_F \supseteq Y$$
Example - Computing Attribute Closure

\[ F: AB \rightarrow C \]
\[ A \rightarrow D \]
\[ D \rightarrow E \]
\[ AC \rightarrow B \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( X_F^+ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{A, D, E}</td>
</tr>
<tr>
<td>AB</td>
<td>{A, B, C, D, E}</td>
</tr>
<tr>
<td>B</td>
<td>{B}</td>
</tr>
<tr>
<td>D</td>
<td>{D, E}</td>
</tr>
</tbody>
</table>

(Hence \( AB \) is a key)

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 $X^+_F$

\[
closure := X; \quad // \text{since } X \subseteq X^+_F
\]

repeat

\[
old := closure;
\]

if there is an FD $Z \rightarrow V$ in $F$ such that

\[
Z \subseteq closure \text{ and } V \not\subseteq closure
\]

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:
  
  \[
  \begin{align*}
  AB & \rightarrow C \quad (a) \\
  A & \rightarrow D \quad (b) \\
  D & \rightarrow E \quad (c) \\
  AC & \rightarrow B \quad (d)
  \end{align*}
  \]

- **Solution:**
  
  Initially \( \text{closure} = \{AB\} \)
  
  Using (a) \( \text{closure} = \{ABC\} \)
  
  Using (b) \( \text{closure} = \{ABCD\} \)
  
  Using (c) \( \text{closure} = \{ABCDE\} \)
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 dependency
- 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

Data sources

Users

Users’ requirements specification

Forms/reports that are used or generated by the enterprise

Sources describing the enterprise such as data dictionary and corporate data model

Use top-down approach such as ER modeling

ER model is mapped to a set of relations

Set of well-designed relations

Use normalization as a validation technique to check structure of relations. (This approach is described in Chapter 16, Step 2.2)

Approach 1

Use normalization as a bottom-up technique to create set of relations. (This approach is described in this chapter and the next)

Approach 2

Figure 13.1 How normalization can be used to support database design.
Relationship Between Normal Forms

- 1NF
- 2NF
- 3NF
- BCNF
- 4NF
- 5NF
- Higher normal forms
Process of Normalization

Data sources

- Users
- Users’ requirements specification
- Forms/reports that are used or generated by the enterprise (as described in this chapter and the next)
- Sources describing the enterprise such as data dictionary and corporate data model

Transfer attributes into table format

Unnormalized Form (UNF)

Remove repeating groups

First Normal Form (1NF)

Remove partial dependencies

Second Normal Form (2NF)

Remove transitive dependencies

Third Normal Form (3NF)
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.

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

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

<table>
<thead>
<tr>
<th>Staff Property Inspection</th>
<th>propertyNo</th>
<th>iDate</th>
<th>iTime</th>
<th>pAddress</th>
<th>comments</th>
<th>staffNo</th>
<th>sName</th>
<th>carReg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG4</td>
<td>18-Oct-00</td>
<td>10.00</td>
<td>6 Lawrence St, Glasgow</td>
<td>Need to replace crockery</td>
<td>SG37</td>
<td>Ann Beech</td>
<td>M231 JGR</td>
</tr>
<tr>
<td></td>
<td>PG4</td>
<td>22-Apr-01</td>
<td>09.00</td>
<td>6 Lawrence St, Glasgow</td>
<td>In good order</td>
<td>SG14</td>
<td>David Ford</td>
<td>M533 HDR</td>
</tr>
<tr>
<td></td>
<td>PG4</td>
<td>1-Oct-01</td>
<td>12.00</td>
<td>6 Lawrence St, Glasgow</td>
<td>Damp rot in bathroom</td>
<td>SG14</td>
<td>David Ford</td>
<td>N721 HFR</td>
</tr>
<tr>
<td></td>
<td>PG16</td>
<td>22-Apr-01</td>
<td>13.00</td>
<td>5 Novar Dr, Glasgow</td>
<td>Replace living room carpet</td>
<td>SG14</td>
<td>David Ford</td>
<td>M533 HDR</td>
</tr>
<tr>
<td></td>
<td>PG16</td>
<td>24-Oct-01</td>
<td>14.00</td>
<td>5 Novar Dr, Glasgow</td>
<td>Good condition</td>
<td>SG37</td>
<td>Ann Beech</td>
<td>N721 HFR</td>
</tr>
</tbody>
</table>
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 → B and there is NO A → B or X → B

- 2nd Normal Form
  - A relation that does not have a FD, X → 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 \(\rightarrow\) iTime, staffNo, comments, sName, carReg
- Fd2: PropertyNo \(\rightarrow\) pAddress
- Fd3: staffNo \(\rightarrow\) sName
- Fd4: iDate, staffNo \(\rightarrow\) carReg
- Fd5: iDate, iTime, carReg \(\rightarrow\) all other attributes
- Fd6: iDate, iTime, staffNo \(\rightarrow\) all other attributes
1NF to 2NF

- Transformed into following two tables.
  - Property (propertyNo, pAddress)
  - PropertyInspection (propertyNo, iDate, 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** \((SSN, \text{Name}, \text{Address}, \text{Hobby})\)
  - The FD \(SSN \rightarrow \text{Name, Address}\) does **not** satisfy requirements of BCNF
  - since the key is \((SSN, \text{Hobby})\)

- **HasAccount** \((\text{AcctNum}, \text{ClientId}, \text{OfficeId})\)
  - The FD \(\text{AcctNum} \rightarrow \text{OfficeId}\) does **not** satisfy BCNF requirements
  - since keys are \((\text{ClientId}, \text{OfficeId})\) and \((\text{AcctNum}, \text{ClientId})\); not \(\text{AcctNum}\).
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 \( B \) (\( \rightarrow \) redundancy, if the \( A \)-value repeats twice)

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

“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** \((AcctNum, ClientId, OfficeId)\)
  - \(ClientId, OfficeId \rightarrow AcctNum\)
    - 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 \(AcctNum \rightarrow OfficeId\) (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 $\mathbf{R} = (R, F)$
  - $R$ is a set 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
  - $R = \bigcup_i R_i$ 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 $r_i = \pi_{R_i}(\mathbf{r})$ for all $i$
Example Decomposition

Schema \( R, F \) where
\[
R = \{ \text{SSN, Name, Address, Hobby} \}
\]
\[
F = \{ \text{SSN} \rightarrow \text{Name, Address} \}
\]
can be decomposed into:
\[
R_1 = \{ \text{SSN, Name, Address} \}
\]
\[
F_1 = \{ \text{SSN} \rightarrow \text{Name, Address} \}
\]
and
\[
R_2 = \{ \text{SSN, Hobby} \}
\]
\[
F_2 = \{ \} \]
Lossless Schema Decomposition

- A decomposition should not lose information.

- A decomposition \((R_1, \ldots, R_n)\) of a schema, \(R\), is *lossless* if every valid instance, \(r\), of \(R\) can be reconstructed from its components:

\[
 r = r_1 \land r_2 \land \ldots \land r_n
\]

where each \( r_i = \pi_{R_i}(r) \)
Lossy Decomposition

- **The following is always the case (Think why?):**
  \[ r \subseteq r_1 \times r_2 \times \ldots \times r_n \]

- **But the following is not always true:**
  \[ r \supseteq r_1 \times r_2 \times \ldots \times r_n \]

- **Example**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Joe</td>
<td>1 Pine</td>
</tr>
<tr>
<td>2222</td>
<td>Alice</td>
<td>2 Oak</td>
</tr>
<tr>
<td>3333</td>
<td>Alice</td>
<td>3 Pine</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Joe</td>
</tr>
<tr>
<td>2222</td>
<td>Alice</td>
</tr>
<tr>
<td>3333</td>
<td>Alice</td>
</tr>
</tbody>
</table>

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*
A (binary) decomposition of $R = (R, F)$ into $R_1 = (R_1, F_1)$ and $R_2 = (R_2, F_2)$ 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 $F^+$
Example

Schema \((R, F)\) where

\[ R = \{ \text{SSN, Name, Address, Hobby} \} \]
\[ F = \{ \text{SSN} \rightarrow \text{Name, Address} \} \]

can be decomposed into:

\[ R_1 = \{ \text{SSN, Name, Address} \} \]
\[ F_1 = \{ \text{SSN} \rightarrow \text{Name, Address} \} \]

and

\[ R_2 = \{ \text{SSN, Hobby} \} \]
\[ F_2 = \{ \} \]

Since \( R_1 \cap R_2 = \text{SSN} \) and \( \text{SSN} \rightarrow R_1 \) the decomposition is lossless.
Intuition Behind the Test for Losslessness

- Suppose $R_1 \cap R_2 \rightarrow R_2$. Then a row of $r_1$ can combine with exactly one row of $r_2$ in the natural join (since in $r_2$ a particular set of values for the attributes in $R_1 \cap R_2$ defines a unique row)

\[
\begin{array}{c|c}
\ldots \ldots \ldots & a \\
\ldots \ldots & a \\
\ldots \ldots & b \\
\ldots \ldots & c \\
\hline
r_1
\end{array}
\quad
\quad
\begin{array}{c|c}
\ldots \ldots & \\
\ldots \ldots & a \\
\ldots \ldots & b \\
\ldots \ldots & c \\
\hline
r_2
\end{array}
\]
Tuple Structure in a Lossless Binary Decomposition

**Figure 6.6** Tuple structure in a lossless binary decomposition: a row of $r_1$ combines with exactly one row of $r_2$. 
Proof of Lossless Condition

- $r \subseteq r_1 \Join r_2$ — this is true for any decomposition

- $r \supseteq r_1 \Join r_2$

If $R_1 \cap R_2 \rightarrow R_2$ then

$$\text{card } (r_1 \Join r_2) = \text{card } (r_1)$$

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

But $\text{card } (r) \geq \text{card } (r_1)$ (since $r_1$ is a projection of $r$)

and therefore $\text{card } (r) \geq \text{card } (r_1 \Join r_2)$

Hence $r = r_1 \Join r_2$
Dependency Preservation

Consider a decomposition of \( R = (R, F) \) into \( R_1 = (R_1, F_1) \) and \( R_2 = (R_2, F_2) \)

- 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 \( r_1 \) or \( r_2 \) is (relatively) easy
  - Checking \( f \) in \( r_1 \bowtie r_2 \) is harder – requires a join
  - Ideally: want to check FDs locally, in \( r_1 \) and \( r_2 \), and have a guarantee that every \( f \in F \) holds in \( r_1 \bowtie r_2 \)

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 \( r_1 \) and \( r_2 \) are updated, can be done by checking \( F_1 \) in \( r_1 \) and \( F_2 \) in \( r_2 \)
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.
    - Incur additional runtime overhead of constraint maintenance
Example

Schema \((R, F)\) where

\[
R = \{\text{SSN, Name, Address, Hobby}\}
\]

\[
F = \{\text{SSN }\rightarrow \text{Name, Address}\}
\]

can be decomposed into:

\[
R_1 = \{\text{SSN, Name, Address}\}
\]

\[
F_1 = \{\text{SSN }\rightarrow \text{Name, Address}\}
\]

and

\[
R_2 = \{\text{SSN, Hobby}\}
\]

\[
F_2 = \{\}
\]

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** \((\text{AcctNum}, \text{ClientId}, \text{Officeld})\)
  
  \[ f_1: \text{AcctNum} \rightarrow \text{Officeld} \]
  
  \[ f_2: \text{ClientId}, \text{Officeld} \rightarrow \text{AcctNum} \]

- **Decomposition:**
  
  \[ R_1 = (\text{AcctNum}, \text{Officeld}; \{\text{AcctNum} \rightarrow \text{Officeld}\}) \]
  
  \[ R_2 = (\text{AcctNum}, \text{ClientId}; \{\}) \]

- **Decomposition is lossless:**
  
  \[ R_1 \cap R_2 = \{\text{AcctNum}\} \text{ and } \text{AcctNum} \rightarrow \text{Officeld} \text{ (i.e. } R_1) \]

- **In BCNF**

- **Not dependency preserving:** \( f_2 \not\in (F_1 \cup F_2)^+ \)

- **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:** \( R = (R; F) \)

\[ \text{Decomp} := R \]

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

- Find \( X \rightarrow Y \in F' \) that violates BCNF // i.e., \( X \) isn’t a superkey in \( S \)
- Replace \( S \) in \( \text{Decomp} \) 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** \( \text{Decomp} \)
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
\((BH \text{ is a key})\)
\( A \rightarrow DE \) violates BCNF since \( A \) is not a superkey \((A^+ = ADE)\)

Step 2:  Split \( R \) into:
\[
R_1 = (ADE, F_1 = \{A \rightarrow DE\})
\]
\[
R_2 = (ABCDEFGHK; F_2 = \{ABH \rightarrow C, BGH \rightarrow K, K \rightarrow AH, BH \rightarrow G\})
\]
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 \( F_1 \cup F_2 \)
\((E.g., K \rightarrow A \text{ and } A \rightarrow D \text{ implies } K \rightarrow D)\)
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 $R = (R,F)$ and $R_1 = (R_1,F_1)$, $R_2 = (R_2,F_2)$ is the resulting decomposition. Then:

- There are *fewer violations* of BCNF in $R_1$ and $R_2$ than there were in $R$
  - $X \rightarrow Y$ implies $X$ is a key of $R_1$
  - Hence $X \rightarrow Y \in F_1$ does not violate BCNF in $R_1$ and, since $X \rightarrow Y \notin F_2$, does not violate BCNF in $R_2$ either
  - Suppose $f : X' \rightarrow Y' \in F$ doesn’t violate BCNF in $R$. If $f \in F_1$ or $F_2$ it does not violate BCNF in $R_1$ or $R_2$ either since $X'$ is a superkey of $R$ and hence also of $R_1$ and $R_2$.
Properties of BCNF Decomposition Algorithm

- A BCNF decomposition is *not necessarily* dependency preserving
- But *always* lossless:
  \[
  \text{since } R_1 \cap R_2 = X, \quad X \rightarrow Y, \quad \text{and} \quad R_1 = XY
  \]
- 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 $R$

- 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 $F$ ($F^+ = U^+$)
  - 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:** \( F = \{ \text{ABH} \rightarrow \text{CK}, \text{A} \rightarrow \text{D}, \text{C} \rightarrow \text{E}, \text{BGH} \rightarrow \text{L}, \text{L} \rightarrow \text{AD}, \text{E} \rightarrow \text{L}, \text{BH} \rightarrow \text{E} \} \)

**Step 1:** Make RHS of each FD into a single attribute
- *Algorithm:* Use the decomposition inference rule for FDs
- Example: \( L \rightarrow \text{AD} \) replaced by \( L \rightarrow \text{A}, L \rightarrow \text{D} \); \( \text{ABH} \rightarrow \text{CK} \) by \( \text{ABH} \rightarrow \text{C}, \text{ABH} \rightarrow \text{K} \)

**Step 2:** Eliminate redundant attributes from LHS.
- *Algorithm:* If FD \( XB \rightarrow A \in F \) (where \( B \) is a single attribute) and \( X \rightarrow A \) is entailed by \( F \), then \( B \) was unnecessary
- Example: Can an attribute be deleted from \( \text{ABH} \rightarrow \text{C} \)?
  - Compute \( \text{AB}^+_F, \text{AH}^+_F, \text{BH}^+_F \).
  - Since \( C \in (\text{BH})^+_F, \text{BH} \rightarrow \text{C} \) is entailed by \( F \) and \( A \) is redundant in \( \text{ABH} \rightarrow \text{C} \).
Computing Minimal Cover (con’t)

- **Example (con’d):**
  - \( F = \{BH \rightarrow C, BH \rightarrow K, A \rightarrow D, C \rightarrow E, BGH \rightarrow L, L \rightarrow A, L \rightarrow D, E \rightarrow L, BH \rightarrow E\} \)

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

- **Note:** The order of steps 2 and 3 cannot be interchanged!!
Synthesizing a 3NF Schema

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

- **Step 1**: Compute a minimal cover, $\mathbf{U}$, of $\mathbf{F}$.
  - The decomposition is based on $\mathbf{U}$, but since $\mathbf{U}^+ = \mathbf{F}^+$ the same functional dependencies will hold.
  - A minimal cover for
    $$\mathbf{F} = \{ABH \rightarrow CK, A \rightarrow D, C \rightarrow E, BGH \rightarrow L, L \rightarrow AD, E \rightarrow L, BH \rightarrow E\}$$
    is
    $$\mathbf{U} = \{BH \rightarrow C, BH \rightarrow K, A \rightarrow D, C \rightarrow E, L \rightarrow A, E \rightarrow L\}$$
Synthesizing a 3NF schema (con’t)

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

- **Step 3**: For each $U_i$, form schema $R_i = (R_i, U_i)$, where $R_i$ is the set of all attributes mentioned in $U_i$
  - Each FD of $U$ will be in some $R_i$. Hence the decomposition is *dependency preserving*
  - $R_1 = (BHCK; BH \rightarrow C, BH \rightarrow K)$, $R_2 = (AD; A \rightarrow D)$,
    - $R_3 = (CE; C \rightarrow E)$, $R_4 = (AL; L \rightarrow A)$, $R_5 = (EL; E \rightarrow L)$
Synthesizing a 3NF schema (con’t)

- **Step 4:** If no $R_i$ is a superkey of $R$, add schema $R_0 = (R_0, \{\})$ where $R_0$ is a key of $R$.

  - $R_0 = (BGH, \{\})$

    - $R_0$ 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)

    - $R_0$ 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: $R1 = (AB; \{A \rightarrow B\}), \ R2 = (CD; \{C \rightarrow D\})$. Lossy! Need to add $(AC; \{\})$, for losslessness

- **Step 4 guarantees lossless decomposition.**
BCNF Design Strategy

- The resulting decomposition, \( R_0, R_1, \ldots R_n \), 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.

```sql
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

```sql
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
Additional note on BCNF and 3NF Synthesis

- Pitfalls: Relations $R_i$ 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, $R_i$ can only guarantee the FDs in $G_i$, and cannot entail all FDs in $G^+$
- Example
  - $R = \{\text{AcctNum, ClientId, OfficeId, DateOpened}\}$
  - $F = \{\text{ClientId, OfficeId }\rightarrow\text{AcctNum, AcctNum }\rightarrow\text{OfficeId, DateOpened}\}$
  - Through 3NF synthesis, we get
    - $R_1 = (\{\text{ClientId, OfficeId, AcctNum}\}, \{\text{ClientId, OfficeId }\rightarrow\text{AcctNum}\})$
    - $R_2 = (\{\text{AcctNum, OfficeId, DateOpened}\}, \{\text{AcctNum }\rightarrow\text{OfficeId, DateOpened}\})$
  - Need to compute $\pi_{R_i}(G)$ and look for the violators there!!!

Not in BCNF
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.
  - $St \ C \ Sem \rightarrow P$
  - $P \ Sem \rightarrow C$
  - $C \ Sem \ T \rightarrow P$
  - $P \ Sem \ T \rightarrow C \ R$
    - $P \ Sem \ T \rightarrow C$ (decomposition)
    - $P \ Sem \ T \rightarrow R$ (decomposition)
  - $P \ Sem \ C \ T \rightarrow R$
  - $P \ Sem \ T \rightarrow C$ (duplicate)

- Let $F$ denote this set.
BCNF Decomposition from 3NF Synthesis

- Minimal Cover Step 2.
  - FD1. St C Sem $\rightarrow$ P
  - FD2. P Sem $\rightarrow$ C
  - FD3. C Sem T $\rightarrow$ P
  - $P$ Sem T $\rightarrow$ C R
  - $P$ Sem C T R
    - FD4. P Sem T $\rightarrow$ C (decomposition)
    - FD5. P Sem T $\rightarrow$ R (decomposition)
  - $P$ Sem C T $\rightarrow$ R
    - $P$ Sem T $\rightarrow$ R (reduced and this is duplicate. So, discard)
    - $P$ Sem T $\rightarrow$ 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.**
  1. **FD1. St C Sem → P**
  2. **FD2. P Sem → C**
  3. **FD3. C Sem T → P**
  5. **FD5. P Sem T → R (decomposition)**

- **Search for Removable redundant FDs**
  1. \((\text{St C Sem})^+\)\(_{\{F-FD1\}}\) = \((\text{St C Sem})\)
    - 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
- \((St \ C \ Sem \ P; St \ C \ Sem \rightarrow P)\) ; include \(P \ Sem \ C\)
- \((P \ Sem \ C; P \ Sem \rightarrow C)\)
- \((C \ Sem \ T \ P; C \ Sem \ T \rightarrow P)\) ; include \(P \ Sem \ C\)
- \((P \ Sem \ T \ R; P \ Sem \ T \rightarrow R)\)

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 “\(P \ Sem \rightarrow C\)” 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: \((P \ Sem \ C; P \ Sem \rightarrow C)\) , \((P \ Sem \ St; \{\})\) : \(St \ C \ Sem \rightarrow P\) is not preserved
- Third is decomposed into: \((P \ Sem \ C; P \ Sem \rightarrow C)\) , \((P \ Sem \ T; \{\})\) : \(C \ Sem \ T \rightarrow P\) is not preserved.
Fourth Normal Form

- 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

<table>
<thead>
<tr>
<th>SSN</th>
<th>PhoneN</th>
<th>ChildSSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>111111</td>
<td>123-4444</td>
<td>222222</td>
</tr>
<tr>
<td>111111</td>
<td>123-4444</td>
<td>333333</td>
</tr>
<tr>
<td>111111</td>
<td>321-5555</td>
<td>222222</td>
</tr>
<tr>
<td>111111</td>
<td>321-5555</td>
<td>333333</td>
</tr>
<tr>
<td>222222</td>
<td>987-6666</td>
<td>444444</td>
</tr>
<tr>
<td>222222</td>
<td>777-7777</td>
<td>444444</td>
</tr>
<tr>
<td>222222</td>
<td>987-6666</td>
<td>555555</td>
</tr>
<tr>
<td>222222</td>
<td>777-7777</td>
<td>555555</td>
</tr>
</tbody>
</table>
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:

$$ r = \pi_X(r) \bowtie \pi_Y(r) $$

- then a *multi-valued dependency*

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

- **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 \( \text{PhoneN} \) or \( \text{ChildSSN} \), 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: \( R = R_1 \Join R_2 \)

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

\[ \text{Decomposition} := \{ R \} \]  /* Initially decomposition consists of only one schema */
while there is a schema \( S = (\bar{S}; D') \) in Decomposition that is not in 4NF do
  /* Let \( \bar{X} \bowtie \bar{Y} \) be an MVD in \( D^+ \) such that \( \bar{X} \bar{Y} \subseteq \bar{S} \) and 
    it violates 4NF in \( S \). Decompose using this MVD */
  Replace \( S \) in Decomposition with schemas \( S_1 = (\bar{X} \bar{Y}; D'_1) \) and 
  \( S_2 = ((\bar{S} - \bar{Y}) \cup \bar{X}; D'_2) \), where \( D'_1 = \pi_{\bar{X} \bar{Y}}(D') \) and \( D'_2 = \pi_{(\bar{S} - \bar{Y}) \cup \bar{X}}(D') \)
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-$$\Rightarrow$$Y. (See slide 88)
If X -$$\Rightarrow$$ Y, \( R = XY \) join X(R-Y)
Projection of MVD on a Set of Attributes

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

- Example
  - Projection of MVD: $ABCD = AB \Join BCD$ on $ABC$
    - $AB \cap BCD = B \subseteq ABC$. So, the projection is $AB \Join BC$
  
  - Projection of MVD: $ABCD = ACD \Join BD$ on $ABC$
    - $ACD \cap BD = D \subseteq ABC$. So, the projection is undefined.
4NF Decomposition Example

- **Example**
  - **Attributes** = \{ABCD\}
  - **MVDs**
    - MVD1. \(ABCD = AB \Join BCD\)
    - MVD2. \(ABCD = ACD \Join BD\)
    - MVD3. \(ABCD = ABC \Join BCD\)
  - From MVD1, decomposed to \(AB, BCD\)
    - Projection of remaining MVDs on \(AB\) is not defined
    - Projection of remaining MVDs on \(BCD\) is:
      - For MVD2, \(BCD = CD \Join BD\)
      - For MVD3, \(BCD = BC \Join BCD\) (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
  - \( R_1 = (ABC; \{AB \rightarrow C, C \rightarrow B\}) \)
  - \( R_2 = (CBDE; \{C \rightarrow BDE\}) \)
  - \( R_3 = (EFG; \{E \rightarrow FG\}) \)
- \( R_1 \) 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
  - $R_{31} = (EF; \{E \rightarrow F\})$, $R_{22} = (GF; \{\})$
Customary Representation of MVDs

- Customary representation of MVDs
  - MVD $R = V \Join W$ over $R = (R; D)$, where
    - $X = V \cap W$
    - $X \cup Y = V$ or $X \cup Y = W$
      are represented as $X \rightarrow Y$
    - i.e., $R = XY \Join X(R-Y)$

- Another way of defining MVD in a relation
  - $X \rightarrow Y$ 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 $\rightarrow$ college

- **Apply (SSN, college, date, major)**
  - **Requirements**
    - Apply once to each college
    - May apply to multiple majors
  - **We can derive...**
    - SSN, college $\rightarrow$ date, major / date $\rightarrow$ college
    - SSN $\rightarrow$ 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.

from slides by Prof. Widom
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 $A \rightarrow B$ that violates 4NF

Decompose $R'$ into $R_1(A, B)$ and $R_2(A, \text{rest})$

Compute FDs and MVDs for $R_1$ and $R_2$

Compute keys for $R_1$ and $R_2$