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

Normalisation 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

<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>

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

\[\text{Hourly Emps (ssn, name, lot, rating, hrly_wages, hrs_worked)}\]

Relational Model

- Hourly_Emps (ssn, name, lot, rating, hrly_wages, hrs_worked)
- Some functional dependencies on Hourly_Emps:
  - ssn is the key: S \rightarrow SNLRWH
  - rating determines hrly_wages: R \rightarrow W

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 \(Y\)
- **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 \(r_1\) 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:
    - \(SSN \rightarrow SSN, Name, Address\)

Functional Dependencies - Example

- **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

- 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 \rightarrow AcctNum**
  - **AcctNum \rightarrow 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: $F = \{A \rightarrow B, B \rightarrow C\}$ and $f$ is $A \rightarrow C$
  - If $Streetaddr \rightarrow Town$ and $Town \rightarrow Zip$ then $Streetaddr \rightarrow Zip$

- **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 $X \rightarrow Y$ (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 $X \rightarrow Z$

Armstrong’s Axioms for FDs (cont.)

- Two more rules (which can be derived from the axioms) can be useful:
  - **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$
  - **Decomposition**: If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$
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 $X \rightarrow Y$
  
  \[
  \begin{array}{c}
  \text{t}_1 = (a_1, b_1, c_1, d_1, e_1) \\
  \text{t}_2 = (a_2, b_2, c_2, d_2, e_2) \\
  \pi_X(t_1) = \pi_X(t_2) \Rightarrow \\
  a_1 = a_2, b_1 = b_2, c_1 = c_2, d_1 = d_2 \\
  \pi_Y(t_1) = \pi_Y(t_2) \Leftrightarrow
  \end{array}
  \]

Augmentation

• If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any $Z$
  
  \[
  \begin{array}{c}
  \text{t}_1 = (a_1, b_1, c_1, d_1, e_1) \\
  \text{t}_2 = (a_2, b_2, c_2, d_2, e_2) \\
  \pi_{XZ}(t_1) = \pi_{XZ}(t_2) \Rightarrow \\
  a_1 = a_2, b_1 = b_2, e_1 = e_2 \\
  \pi_{YZ}(t_1) = \pi_{YZ}(t_2)
  \end{array}
  \]

Transitivity

• If $X \rightarrow Y$, and $Y \rightarrow Z$ then $X \rightarrow Z$
  
  \[
  \begin{array}{c}
  \text{t}_1 = (a_1, b_1, c_1, d_1, e_1) \\
  \text{t}_2 = (a_2, b_2, c_2, d_2, e_2) \\
  \pi_X(t_1) = \pi_X(t_2) \Rightarrow \\
  a_1 = a_2, b_1 = b_2 \\
  \pi_Y(t_1) = \pi_Y(t_2) \\
  a_1 = a_2, b_1 = b_2, c_1 = c_2, d_1 = d_2 \\
  \pi_Y(t_1) = \pi_Y(t_2) \\
  \pi_Z(t_1) = \pi_Z(t_2)
  \end{array}
  \]
Generating $F^+$

<table>
<thead>
<tr>
<th>$F$</th>
<th>$\text{union}$</th>
<th>$\text{trans}$</th>
<th>$\text{decomp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AB \rightarrow C$</td>
<td>$AB \rightarrow BCD$</td>
<td>$AB \rightarrow BCDE$</td>
<td>$AB \rightarrow CDE$</td>
</tr>
<tr>
<td>$A \rightarrow D$</td>
<td>$\text{aug}$</td>
<td>$\text{aug}$</td>
<td>$\text{aug}$</td>
</tr>
<tr>
<td>$D \rightarrow E$</td>
<td>$BCD \rightarrow BCDE$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 $X \rightarrow A$
  - $X^+_F$ is not necessarily the same as $X^+_{F1}$ 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

<table>
<thead>
<tr>
<th>$X$</th>
<th>$X^+_F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$: $AB \rightarrow C$</td>
<td>$A \rightarrow {A, D, E}$</td>
</tr>
<tr>
<td>$A \rightarrow D$</td>
<td>$AB \rightarrow {A, B, C, D, E}$</td>
</tr>
<tr>
<td>$D \rightarrow E$</td>
<td>$\text{(Hence } AB \text{ is a key)}$</td>
</tr>
<tr>
<td>$AC \rightarrow B$</td>
<td>$B \rightarrow {B}$</td>
</tr>
<tr>
<td>$D \rightarrow E$</td>
<td>$D \rightarrow {D, E}$</td>
</tr>
</tbody>
</table>

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 $Z \subseteq \text{closure and } V \not\subseteq \text{closure}$
    then closure := closure $\cup$ $V$
until old = closure

- If $T \subseteq \text{closure}$ 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:

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

**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 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 $X \rightarrow Y$ associated with $R$ either
  - $Y \subseteq X$ (i.e., the FD is trivial) or
  - $X$ is a superkey of $R$

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

(non) BCNF Examples

- **Person** $(SSN, Name, Address, Hobby)$
  - The FD $SSN \rightarrow Name, Address$ does not satisfy requirements of BCNF
    - since the key is $(SSN, Hobby)$
- **HasAccount** $(AccountNumber, ClientId, OfficeId)$
  - The FD $AcctNum \rightarrow OfficeId$ 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 \( B \) (\( \Rightarrow \) redundancy, if the A-value repeats twice)

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

<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>stamps</td>
</tr>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td>coins</td>
</tr>
</tbody>
</table>

- 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)

3NF Example

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

3NF Exam

- **HasAccount** might store redundant data:

<table>
<thead>
<tr>
<th>ClientId</th>
<th>OfficeId</th>
<th>AcctNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Stony Brook</td>
<td>28315</td>
</tr>
<tr>
<td>2222</td>
<td>Stony Brook</td>
<td>28315</td>
</tr>
<tr>
<td>3333</td>
<td>Stony Brook</td>
<td>28315</td>
</tr>
</tbody>
</table>

- Decompose to eliminate redundancy:

<table>
<thead>
<tr>
<th>ClientId</th>
<th>AcctNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>28315</td>
</tr>
<tr>
<td>2222</td>
<td>28315</td>
</tr>
<tr>
<td>3333</td>
<td>28315</td>
</tr>
</tbody>
</table>

- **OfficeId** is key
  - FD: \( \text{AcctNum} \rightarrow \text{OfficeId} \)

BCNF: \( \text{AcctNum} \) is key
  - FD: \( \text{AcctNum} \rightarrow \text{OfficeId} \)

BCNF (only trivial FDs)
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 \(R = (R, F)\)
  - \(R\) 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* \(R\) is a collection of schemas \(R_i = (R_i, F_i)\) where
  - \(R = \bigcup_i R_i\) for all \(i\) (*no new attributes*).
  - \(F_i\) is a set of functional dependencies involving only attributes of \(R_i\).
  - \(F\) entails \(F_i\) for all \(i\) (*no new FDs*).
- The *decomposition of an instance*, \(r\), of \(R\) is a set of relations \(r_i = \pi_{R_i}(r)\) 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 \((R_1, \ldots, R_n)\) of a schema, \(R\), is *lossless* if every valid instance, \(r\), of \(R\) can be reconstructed from its components:
  \[ r = r_1 \times r_2 \times \ldots \times r_n \]
- 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:
\[
\begin{array}{ccc}
\text{SSN} & \text{Name} & \text{Address} \\
1111 & Joe & 1 \text{ Pine} \\
2222 & Alice & 2 \text{ Oak} \\
3333 & Alice & 3 \text{ Pine} \\
\end{array}
\]
\[
\begin{array}{ccc}
\text{SSN} & \text{Name} & \text{Address} \\
1111 & Joe & 1 \text{ Pine} \\
2222 & Alice & 2 \text{ Oak} \\
3333 & Alice & 3 \text{ Pine} \\
\end{array}
\]

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

Lossy Decompositions: What is Actually Lost?

- In the previous example, the tuples (2222, Alice, 3 Pine) and (3333, Alice, 2 Oak) were *gained*, not lost!
  - Why do we say that the decomposition was lossy?
- What was lost is information:
  - That 2222 lives at 2 Oak: *In the decomposition, 2222 can live at either 2 Oak or 3 Pine*
  - That 3333 lives at 3 Pine: *In the decomposition, 3333 can live at either 2 Oak or 3 Pine*

Testing for Losslessness

- A (binary) decomposition of \(R = (R, F)\) into \(R_1 = (R_1, F_1)\) 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 \(F^+\)

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
\[
R = \{\text{SSN, Name, Address, Hobby}\}
\]
\[
F = \{\text{SSN }\rightarrow\text{Name, Address}\}
\]
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 \(\text{SSN }\rightarrow\text{R}_1\) 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)

\[
\begin{array}{c|c|c|c}
R_1 \cap R_2 & R_1 \cap R_2 \\
\hline
\cdots & a & \cdots & \cdots \\
\cdots & a & \cdots & \cdots \\
\cdots & b & \cdots & \cdots \\
\cdots & c & \cdots & \cdots \\
\end{array}
\]

\(r_1\)

\(r_2\)

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 \(r_2\)

• 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: \(A \rightarrow C \not\in F\), but in \(F^+\)
  - \((BC, F_2)\), \(F_2 = \{B \rightarrow C, C \rightarrow B\}\)

- \(A \rightarrow B \not\in (F_1 \cup F_2)^\ast\), 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

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

Example

BCNF Decomposition Algorithm

\textbf{Input: } \(R = (R; F)\)

\textbf{Decomp} := \(R\)

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

Find \(X \rightarrow Y \in F'\) that violates BCNF // \(X\) 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 \(S - (Y - X)\)

\textbf{end}

\textbf{return} \text{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 \( A^+ = ADE \)

**step 2:** Split \( R \) into:
- \( R_1 = (ADE, \{ A \rightarrow DE \}) \)
- \( R_2 = (ABCFGH; \{ ABH \rightarrow C, BGH \rightarrow F, F \rightarrow AH, BH \rightarrow G \}) \)

Note 1: \( R_1 \) is in BCNF
Note 2: Decomposition is lossless since \( A \) is a key of \( R_1 \)
Note 3: FDs \( F \rightarrow D \) and \( BH \rightarrow E \) are not in \( T_1 \) or \( T_2 \).
   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 \( R = (R,F) \) 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 \( f \) 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 \( R_1 \) or \( R_2 \) either since \( X' \) is a superkey of \( R \) and hence also of \( R_1 \) and \( R_2 \).
- The decomposition is lossless
  - Since \( F_1 \cap F_2 = X \)

**Example (con’t)**

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

**step 1:** Find a FD that violates BCNF.
- Not \( ABH \rightarrow C \) or \( BGH \rightarrow F \), since \( BH \) is a key of \( R_2 \)
- \( F \rightarrow AH \) violates BCNF since \( F \) is not a superkey \( (F^+ = AH) \)

**step 2:** Split \( R_2 \) into:
- \( R_{21} = (FAH, \{ F \rightarrow AH \}) \)
- \( R_{22} = (BCFG; \{ \}) \)

Note 1: Both \( R_{21} \) and \( R_{22} \) are in BCNF.
Note 2: The decomposition is lossless (since \( F \) is a key of \( R_{21} \))
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 always possible (and, of course, lossless)
- 3NF decomposition is based on a minimal cover

Computing Minimal Cover

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

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 \( X \rightarrow A \) 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

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 \( X \rightarrow A \) then check if \( A \in X^+T_{-f} \)
    - Example: \( BGH \rightarrow F \) is entailed by \( E \rightarrow F, BH \rightarrow E \), 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 $R = (R, T)$

**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 $U_1, U_2, \ldots, U_n$ such that the LHS of all elements of $U_i$ are the same

- $U_1 = \{BH \rightarrow C, BH \rightarrow K\}, U_2 = \{A \rightarrow D\}$,
  
  $U_3 = \{C \rightarrow E\}, U_4 = \{F \rightarrow A\}, U_5 = \{E \rightarrow F\}$


Synthesizing a 3NF schema (con’t)

**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*

  - $R_1 = \{BHC; BH \rightarrow C, BH \rightarrow K\}$,
  
  $R_2 = \{AD; A \rightarrow D\}$,
  
  $R_3 = \{CE; C \rightarrow E\}$,
  
  $R_4 = \{FA; F \rightarrow A\}$,
  
  $R_5 = \{EF; E \rightarrow F\}$


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:
      
      $R_1 = \{AB; A \rightarrow B\}$,
      
      $R_2 = \{CD; C \rightarrow D\}$. Lossy! Need to add $(AC; \{\})$, for losslessness

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

- The resulting decomposition, $R_0, R_1, \ldots, R_n$, is
  - Dependency preserving (since every FD in $U$ is a FD of some schema)
  - Lossless (although this is not obvious)
  - In 3NF (although this is not obvious)
- Strategy for decomposing a relation
  - Use 3NF decomposition first to get lossless, dependency preserving decomposition
  - If any resulting schema is not in BCNF, split it using the BCNF algorithm (but this may yield a non-dependency preserving result)

**Normalization Drawbacks**

- By limiting redundancy, normalization helps maintain consistency and saves space
- But performance of querying can suffer because related information that was stored in a single relation is now distributed among several
- **Example:** A join is required to get the names and grades of all students taking CS305 in S2002.

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

**Denormalization**

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

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

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

**Fourth Normal Form**

- 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:
    $$ r = \pi_X (r) \Join \pi_Y (r) $$
    then a **multi-valued dependency**
    $$ R = \pi_X (R) \Join \pi_Y (R) $$
    holds in $r$

Ex: $\text{Person} = \pi_{\text{SSN,PhoneN}} (\text{Person}) \Join \pi_{\text{SSN,ChildSSN}} (\text{Person})$

Fourth Normal Form (4NF)

- A schema is in **fourth normal form** (4NF) if for every non-trivial multi-valued dependency:
  $$ R = X \Join Y $$
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,
  - $R_1 = XY$, $R_2 = R-Y$ is a lossless decomposition of $R$
  - Thus $R$ has the multi-valued dependency:
    $$ R = R_1 \Join R_2 $$
  - 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