Relational Normalization Theory

Chapter 6

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

<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)
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

- $Address \rightarrow ZipCode$
  - 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 → AcctNum
  - AcctNum → OfficeId
    - Thus, attribute values need not depend only on key values

Entailment, Closure, Equivalence

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

Armstrong’s Axioms for FDs

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

  - **Reflexivity**: If $Y \subseteq X$ then $X \rightarrow Y$ (trivial FD)
    - *Name, Address* $\rightarrow$ *Name*
  - **Augmentation**: If $X \rightarrow Y$ then $XZ \rightarrow YZ$
    - If *Town* $\rightarrow$ *Zip* then *Town, Name* $\rightarrow$ *Zip, Name*
  - **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$
Soundness

• Axioms are sound: If an FD $f: X \rightarrow Y$ can be derived from a set of FDs $F$ 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

$$
X \rightarrow XY \quad \text{Augmentation by } X
$$

$$
YX \rightarrow YZ \quad \text{Augmentation by } Y
$$

$$
X \rightarrow YZ \quad \text{Transitivity}
$$

– 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 (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 $F^+$
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 $X \rightarrow Y$ if and only if $X^+_F \supseteq Y$

Example - Computing Attribute Closure

<table>
<thead>
<tr>
<th>$F$: $AB \rightarrow C$</th>
<th>$A \rightarrow D$</th>
<th>$D \rightarrow E$</th>
<th>$AC \rightarrow B$</th>
</tr>
</thead>
</table>

<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>${A, B, C, D, E}$ (Hence $AB$ is a key)</td>
</tr>
<tr>
<td>$B$</td>
<td>${B}$</td>
</tr>
<tr>
<td>$D$</td>
<td>${D, E}$</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 \( X^+_F \)

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

\textbf{repeat}

\textit{old} := \text{closure};

\textbf{if} there is an FD \( Z \rightarrow V \) in \( F \) such that

\( Z \subseteq \text{closure and } V \notin \text{closure} \)

\textbf{then} \( \text{closure} := \text{closure} \cup V \)

\textbf{until} \( \text{old} = \text{closure} \)

- 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) – a research lab accident; 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:** 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 \(SSN \rightarrow Name, Address\) does **not** satisfy requirements of BCNF
    - since the key is \((SSN, Hobby)\)
- **HasAccount** \((AcctNum, ClientId, OfficeId)\)
  - The FD \(AcctNum \rightarrow OfficeId\) does **not** 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 \(B\) (\(\Rightarrow\) redundancy, if the \(A\)-value repeats twice)

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Office</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** ($AcctNum$, $ClientId$, $OfficeId$)
  - $ClientId$, $OfficeId \rightarrow AcctNum$
    - 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 \((SSN, \text{Name}, \text{Address}, \text{Hobby})\)
  - \((SSN, \text{Hobby})\) is the only key.
  - \(SSN \rightarrow \text{Name}\) violates 3NF conditions since \text{Name}\ is not part of a key and \text{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:
  \[ r = r_1 \Join r_2 \Join \ldots \Join r_n \]
- where each \( r_i = \pi_{R_i}(r) \)

Lossy Decomposition

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

\[ r \subseteq r_1 \Join r_2 \Join \ldots \Join r_n \]

*But the following is not always true:*

\[ r \supseteq r_1 \Join r_2 \Join \ldots \Join r_n \]

*Example:*

\[
\begin{array}{ccc}
\text{SSN} & \text{Name} & \text{Address} \\
1111 & Joe & 1 Pine \\
2222 & Alice & 2 Oak \\
3333 & Alice & 3 Pine \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{SSN} & \text{Name} & \text{Address} \\
1111 & Joe & 1 Pine \\
2222 & Alice & 2 Oak \\
3333 & Alice & 3 Pine \\
\end{array}
\]

*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 $F^+$
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).
Proof of Lossless Condition

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

If $R_1 \cap R_2 \rightarrow R_2$ then
\[\text{card} \ (r_1 \bowtie r_2) = \text{card} \ (r_1)\]
(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 $\text{card} \ (r) \geq \text{card} \ (r_1 \bowtie r_2)$

Hence $r = r_1 \bowtie r_2$

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.

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** *(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: \(f_2 \notin (F_1 \cup F_2)^+\)
- **HasAccount** *does not* have BCNF decompositions that are both lossless and dependency preserving! (Check, eg, by enumeration)
- Hence: BCNF+lossless+dependency preserving decompositions are not always achievable!
BCNF Decomposition Algorithm

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

\( Decomp := R \)

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

\hspace{1em} Find \( X \rightarrow Y \in F' \) that violates BCNF \hspace{1em} // \( X \) isn’t a superkey in \( S \)

\hspace{1em} Replace \( S \) in \( Decomp \) with \( S_1 = (XY; F_1), \ S_2 = (S - (Y - X); F_2) \)

\hspace{1em} // \( F_1 = \) all FDs of \( F' \) involving only attributes of \( XY \)

\hspace{1em} // \( F_2 = \) all FDs of \( F' \) involving only attributes of \( S - (Y - X) \)

\textbf{end}

\textbf{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

  \( FD: AcctNum \rightarrow OfficeId \)

  \( BCNF \) (only trivial FDs)
A Larger Example

**Given:** \( R = (R; F) \) where \( R = ABCDEGHK \) and

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

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

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

(\( BH \) is a key)

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

**step 2:** Split \( R \) into:

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

\( R_2 = (ABCGHK; F_1 = \{ ABH \rightarrow C, BGH \rightarrow K, K \rightarrow AH, BH \rightarrow G \}) \)

Note 1: \( R_1 \) is in BCNF

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

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

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

Hence, decomposition is *dependency preserving.*

---

Example (con’t)

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

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

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

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

**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 $R_1 = (R,F_1)$, $R_2 = (R,F_2)$ 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 $R_2$.

Properties of BCNF Decomposition Algorithm

- A BCNF decomposition is *not necessarily* dependency preserving
- But *always* lossless:
  
  $R_1 \cap R_2 = X$, $X \rightarrow Y$. and $R_1 = XY$

- 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, $F$, 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
Computing Minimal Cover

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

- **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 \( XB \rightarrow A \in F \) (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 \( ABH \rightarrow C \) ?
    - Compute \( AB^+_F, AH^+_F, BH^+_F \).
    - Since \( C \in (BH)^+_F \), \( BH \rightarrow C \) is entailed by \( F \) and \( A \) is redundant in \( ABH \rightarrow C \).

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

Computing Minimal Cover (con’t)

- **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 \( E \rightarrow L, BH \rightarrow E \), 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 \( R = (R, F) \)

- **step 1**: Compute a minimal cover, \( U \), of \( F \). The decomposition is based on \( U \), but since \( U^+ = F^+ \) the same functional dependencies will hold
  
  - A minimal cover for
  
    \[ F = \{ ABH \rightarrow CK, A \rightarrow D, C \rightarrow E, BGH \rightarrow L, L \rightarrow AD, \]
    
    \[ E \rightarrow L, BH \rightarrow E \} \]
    
    \[
    \]
    
    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 \( U \) into sets \( U_1, U_2, \ldots, U_n \) such that the LHS of all elements of \( U_i \) are the same

  - \( U_1 = \{ BH \rightarrow C, BH \rightarrow K \}, U_2 = \{ A \rightarrow D \}, \)
  
    \[ U_3 = \{ C \rightarrow E \}, U_4 = \{ L \rightarrow A \}, U_5 = \{ E \rightarrow L \} \]
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*
  
  – $R_1 = (BHCK; BH \rightarrow C, BH \rightarrow K)$, $R_2 = (AD; A \rightarrow D)$, $R_3 = (CE; C \rightarrow E)$, $R_4 = (AL; L \rightarrow A)$, $R_5 = (EL; E \rightarrow L)$

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: $R_1 = (AB; \{A \rightarrow B\})$, $R_2 = (CD; \{C \rightarrow D\})$.
    
    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

```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*

---

**Fourth Normal Form**

<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>

- 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:

    \[ r = \pi_X(r) \bowtie \pi_Y(r) \]

    then a *multi-valued dependency* holds in \( r \)

    \[ R = \pi_X(R) \bowtie \pi_Y(R) \]

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

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.
  - \( R_1 = XY, \ R_2 = R - Y \) 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