Chapter 7: Relational Database Design

First Normal Form

- Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
  - A relational schema R is in first normal form if the domains of all attributes of R are atomic
  - Non-atomic values complicate storage and encourage redundant (repeated) storage of data
    - E.g. Set of accounts stored with each customer, and set of owners stored with each account
  - We assume all relations are in first normal form (revisit this in Chapter 9 on Object Relational Databases)

Pitfalls in Relational Database Design

- Relational database design requires that we find a “good” collection of relation schemas. A bad design may lead to
  - Repetition of information.
  - Inability to represent certain information.
- Design Goals:
  - Avoid redundant data
  - Ensure that relationships among attributes are represented
  - Facilitate the checking of updates for violation of database integrity constraints.

Decomposition

- Decompose the relation schema Lending-schema into:
  - Branch-schema = (branch-name, branch-city, assets)
  - Loan-info-schema = (customer-name, loan-number, amount)
  - All attributes of an original schema (R) must appear in the decomposition (R₁, R₂):
    - R = R₁ ∪ R₂
  - Lossless-join decomposition.
    - For all possible relations r on schema R
      - r = \( \Pi_{R_1}(t) \bowtie \Pi_{R_2}(t) \)

Example of Non Lossless-Join Decomposition

- Decomposition of R = (A, B)
  - \( R_2 = (A) \)
  - \( R_2 = (B) \)
  - \( A \)
  - \( A \)
  - \( B \)
  - \( \Pi_{B}(t) \)
  - \( \Pi_{A}(t) \)
  - \( \Pi_{A}(t) \bowtie \Pi_{B}(t) \)

Example

- Consider the relation schema:
  - Lending-schema = (branch-name, branch-city, assets, customer-name, loan-number, amount)

<table>
<thead>
<tr>
<th>branch-name</th>
<th>branch-city</th>
<th>assets</th>
<th>customer-name</th>
<th>loan-number</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>90000</td>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Redwood</td>
<td>Palos Heights</td>
<td>210000</td>
<td>Smith</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>Perrysville</td>
<td>Houston</td>
<td>170000</td>
<td>Hayes</td>
<td>L-15</td>
<td>1500</td>
</tr>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>900000</td>
<td>Jackson</td>
<td>L-14</td>
<td>1800</td>
</tr>
</tbody>
</table>

- Redundancy:
  - Data for branch-name, branch-city, assets are repeated for each loan that a branch makes
  - Wastes space
  - Complicates updating, introducing possibility of inconsistency of attributes
- Null values:
  - Cannot store information about a branch if no loans exist
  - Can use null values, but they are difficult to handle.
Goal — Devise a Theory for the Following

- Decide whether a particular relation \( R \) is in “good” form.
- In the case that a relation \( R \) is not in “good” form, decompose it into a set of relations \( \{ R_1, R_2, \ldots, R_n \} \) such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value of a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a key.

Use of Functional Dependencies

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
  - If a relation \( r \) is legal under a set \( F \) of functional dependencies, we say that \( r \) satisfies \( F \).
  - specify constraints on the set of legal relations.
    - We say that \( F \) holds on \( r \) if all legal relations on \( r \) satisfy the set of functional dependencies \( F \).
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of Loan-schema may, by chance, satisfy \( \text{loan-number} \rightarrow \text{customer-name} \).

Closure of a Set of Functional Dependencies

- Given a set \( F \) of functional dependencies, there are certain other functional dependencies that are logically implied by \( F \).
  - E.g. If \( A \rightarrow B \) and \( B \rightarrow C \), then we can infer that \( A \rightarrow C \)
- The set of all functional dependencies logically implied by \( F \) is the closure of \( F \).
- We denote the closure of \( F \) by \( F^* \).
- We can find all of \( F^* \) by applying Armstrong’s Axioms:
  - If \( \beta \subseteq \alpha \), then \( \alpha \rightarrow \beta \) (reflexivity)
  - If \( \alpha \rightarrow \beta \) and \( \gamma \rightarrow \beta \), then \( \alpha \rightarrow \gamma \) (augmentation)
  - If \( \alpha \rightarrow \beta \) and \( \beta \rightarrow \gamma \), then \( \alpha \rightarrow \gamma \) (transitivity)
- These rules are
  - sound (generate only functional dependencies that actually hold)
  - complete (generate all functional dependencies that hold).

Example

- \( R = (A, B, C, G, H, I) \)
  - \( F = \{ A \rightarrow B, C \rightarrow G, H \rightarrow I, A \rightarrow C, C \rightarrow H, B \rightarrow H \} \)
  - some members of \( F^* \)
    - \( A \rightarrow H \)
      - by transitivity from \( A \rightarrow B \) and \( B \rightarrow H \)
    - \( AG \rightarrow I \)
      - by augmenting \( A \rightarrow C \) with \( G \) to get \( AG \rightarrow CG \) and then transitivity with \( CG \rightarrow I \)
    - \( CG \rightarrow HI \)
      - from \( CG \rightarrow H \) and \( CG \rightarrow I \)
        - “union rule” can be inferred from
          - definition of functional dependencies, or
          - augmentation of \( CG \rightarrow I \) to infer \( CG \rightarrow CGI \), augmented \( CG \rightarrow H \) to infer \( CGI \rightarrow HI \), and then transitivity
Procedure for Computing $F^+$

- To compute the closure of a set of functional dependencies $F$:
  
  
  $F^+ = F$
  
  repeat
  
  for each functional dependency $f$ in $F^+$
  
  apply reflexivity and augmentation rules on $f$
  
  add the resulting functional dependencies to $F^+$
  
  for each pair of functional dependencies $f_1$ and $f_2$ in $F^+$
  
  if $f_1$ and $f_2$ can be combined using transitivity
  
  then add the resulting functional dependency to $F^+$
  
  until $F^+$ does not change any further
  
  NOTE: We will see an alternative procedure for this task later.

Closure of Attribute Sets

- Given a set of attributes $\alpha$, define the closure of $\alpha$ under $F$ (denoted by $\alpha^*$) as the set of attributes that are functionally determined by $\alpha$ under $F$:
  
  $\alpha \rightarrow \beta$ in $F^+$ $\Rightarrow$ $\beta \subseteq \alpha^*$
  
  Algorithm to compute $\alpha^*$, the closure of $\alpha$ under $F$:
  
  result $\leftarrow \alpha$
  
  while changes to result do
  
  for each $\beta \rightarrow \gamma$ in $F$ do
  
  begin
  
  if $\beta \subseteq \text{result}$ then result $\leftarrow \text{result} \cup \gamma$
  
  end
  
  Procedure for Computing $F^+$

  1. $\alpha \rightarrow \beta$ holds, then $\alpha \rightarrow \beta^*$ holds (union)
  
  2. $\alpha \rightarrow \beta$, $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta \cap \gamma$ holds (decomposition)
  
  3. $\alpha \rightarrow \beta$, $\gamma \rightarrow \delta$ holds, then $\alpha \gamma \rightarrow \delta$ holds (pseudotransitivity)
  
  The above rules can be inferred from Armstrong’s axioms.

Uses of Attribute Closure

- Testing for superkey:
  
  To test if $\alpha$ is a superkey, we compute $\alpha^*$ and check if $\alpha^*$ contains all attributes of $R$.
  
  Testing functional dependencies:
  
  To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in $F^+$), just check if $\beta \subseteq \alpha^*$.
  
  That is, we compute $\alpha^*$ by using attribute closure, and then check if it contains $\beta$.
  
  Is a simple and cheap test, and very useful.
  
  Computing closure of $F$:
  
  For each $\gamma \subseteq R$, find the closure $\gamma^*$, and for each $S \subseteq \gamma^*$, we output a functional dependency $\gamma \rightarrow S$.

Extraneous Attributes

- Consider a set $F$ of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in $F$:
  
  Attribute $A$ is extraneous in $F$ if $A \subseteq \alpha$ and $F$ logically implies $F' = F \setminus \{\alpha \rightarrow \beta\}$.
  
  Attribute $A$ is extraneous in $F$ if $A \subseteq \beta$ and the set of functional dependencies $F' = F \setminus \{\alpha \rightarrow \beta\}$ logically implies $F$.
  
  Note: Implication in the opposite direction is trivial in each of the cases above, since a “stronger” functional dependency always implies a weaker one.
  
  Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  
  $B$ is extraneous in $AB \rightarrow C$ because $(A \rightarrow C, AB \rightarrow C)$ logically implies $A \rightarrow C$ (i.e., the result of dropping $B$ from $AB \rightarrow C$).
  
  Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  
  $C$ is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred after deleting $C$.

Testing if an Attribute is Extraneous

- Consider a set $F$ of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in $F$.
  
  To test if attribute $A \subseteq \alpha$ is extraneous in $\alpha$:
  
  1. compute $(\alpha \setminus A)^*$ using the dependencies in $F$
  
  2. check that $(\alpha \setminus A)^*$ contains $A$; if it does, $A$ is extraneous.
  
  To test if attribute $A \subseteq \beta$ is extraneous in $\beta$:
  
  1. compute $\beta^*$ using only the dependencies in $F' = (F \setminus \{\alpha \rightarrow \beta\}) \cup \{a \rightarrow \beta\}$.
  
  2. check that $A \subseteq \beta^*$ contains $A$; if it does, $A$ is extraneous.

Example of Attribute Set Closure

- $R = \{A, B, C, G, H, I\}$
  
  $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
  
  $(AG)^*$
  
  1. result $\leftarrow AG$
  
  2. result $\leftarrow ABCG$ ($A \rightarrow C$ and $A \rightarrow B$)
  
  3. result $\leftarrow ABCGH$ ($CG \rightarrow H$ and $CG \subseteq AGBC$)
  
  4. result $\leftarrow ABCGH$ ($CG \rightarrow I$ and $CG \subseteq AGBH$)
  
  Is $AG$ a candidate key?
  
  1. Is $AG$ a superkey?
  
     1. Does $AG \rightarrow R^+ = \text{Is} (AG)^* \supseteq R$
  
     2. Is any subset of $AG$ a superkey?
  
     1. Does $A \rightarrow R^+ = \text{Is} (A)^* \supseteq R$
  
     2. Does $G \rightarrow R^+ = \text{Is} (G)^* \supseteq R$

Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others:
  
  1. $A \rightarrow C$ is redundant in: $\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$
  
  2. Parts of a functional dependency may be redundant:
  
    3. $A \rightarrow C$ can be simplified to $A \rightarrow CB$.
  
    4. $A \rightarrow B$ can be simplified to $A \rightarrow B\rightarrow C$.
  
    5. $A \rightarrow B$ can be simplified to $A \rightarrow B\rightarrow C$.
  
  Intuitively, a canonical cover of $F$ is a “minimal” set of functional dependencies equivalent to $F$, having no redundant dependencies or redundant parts of dependencies.
Decomposition

- Decompose the relation schema 
  Lending-schema into: 
  Branch-schema = (branch-name, branch-city, assets) 
  Loan-info-schema = (customer-name, loan-number, branch-name, amount) 

  - All attributes of an original schema \( R \) must appear in the decomposition \( R_1, R_2, \ldots, R_n \). 
  - Lossless-join decomposition: 
    For all possible relations \( F \) of \( R \), 
    \[
    R = R_1 \cup R_2 \cup R_3 \cup \ldots \cup R_n 
    
    \text{A decomposition of } R \text{ into } R_1, R_2, R_3 \ldots R_n \text{ is lossless join if and only if:}
    \begin{align*}
    & \text{1. } R_1 \cap R_2 \rightarrow A \text{ and } R_1 \cap R_3 \rightarrow A \text{ and } \ldots \text{ and } R_1 \cap R_n \rightarrow A \\
    & \text{2. } R_2 \cap R_3 \rightarrow A \text{ and } R_2 \cap R_4 \rightarrow A \text{ and } \ldots \text{ and } R_2 \cap R_n \rightarrow A \\
    & \text{3. } R_3 \cap R_4 \rightarrow A \text{ and } R_3 \cap R_5 \rightarrow A \text{ and } \ldots \text{ and } R_3 \cap R_n \rightarrow A \\
    & \text{4. } \ldots \\
    & \text{5. } R_n \cap R_{n-1} \rightarrow A
    \end{align*}
  
  - A decomposition is dependency preserving, 
    otherwise checking updates for violation of functional dependencies may require computing joins, which is expensive.

Testing for Dependency Preservation

- To check if a dependency \( \alpha \rightarrow \beta \) is preserved in a decomposition of \( R \) into \( R_1, R_2, \ldots, R_n \), we apply the following simplified test (with attribute closure done w.r.t. \( F \)). 
  - result = \( \alpha \) 
  - while (changes to result) do 
    for each \( R_i \) in the decomposition 
    \[
    t = \text{result} \cap R_i 
    
    \text{result = result} \cup t
    \]
  - If result contains all attributes in \( \beta \), then the functional dependency \( \alpha \rightarrow \beta \) is preserved.

  - We apply the test on all dependencies in \( F \) to check if a decomposition is dependency preserving.

  - This procedure takes polynomial time, instead of the exponential time required to compute \( F \) and \( (F_1 \cup F_2 \cup \ldots \cup F_n)^+ \).
Boyce-Codd Normal Form

A relation schema $R$ is in BCNF with respect to a set $F$ of functional dependencies if for all functional dependencies in $F'$ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- $\alpha$ is a superkey for $R$

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  1. compute $\alpha^+$ (the attribute closure of $\alpha$), and
  2. verify that it includes all attributes of $R$, that is, it is a superkey of $R$.

Simplified test: To check if a relation schema $R$ is in BCNF, it suffices to check only the dependencies in the given set $F$ for violation of BCNF, rather than checking all dependencies in $P$.

- If none of the dependencies in $F$ causes a violation of BCNF, then none of the dependencies in $P$ will cause a violation of BCNF either.

However, using only $F$ is incorrect when testing a relation in a decomposition of $R$

- E.g. Consider $R (A, B, C, D)$ with $F = \{ A \rightarrow B, B \rightarrow C \}$
  - Decompose $R$ into $R_1 (A,B)$ and $R_2 (A,C,D)$
  - Neither of the dependencies in $F$ contain only attributes from $R_1 (A,C,D)$ so we might be mislead into thinking $R_2$ satisfies BCNF.
  - In fact, dependency $A \rightarrow C$ in $F$ shows $R_2$ is not in BCNF.

Example of BCNF Decomposition

- $R = (\text{branch-name}, \text{branch-city}, \text{assets}, \text{customer-name, loan-number, amount})$
- $F = \{ \text{branch-name} \rightarrow \text{assets}, \text{branch-city} \rightarrow \text{loan-number} \rightarrow \text{amount} \}$
- Key = (loan-number, customer-name)

Decomposition

- $R_1 = (\text{branch-name}, \text{branch-city}, \text{assets})$
- $R_2 = (\text{branch-name}, \text{customer-name}, \text{loan-number, amount})$
- $R_3 = (\text{branch-name}, \text{loan-number, amount})$
- $R_4 = (\text{customer-name, loan-number})$

Final decomposition $R_1, R_2, R_3, R_4$

BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- $R = (J, K, L)$
- $F = \{ JK \rightarrow L \}$
- Two candidate keys = JK and JL
- $R$ is not in BCNF
- Any decomposition of $R$ will fail to preserve $JK \rightarrow L$

Example

- $R = (A, B, C)$
- $F = \{ A \rightarrow B \}$
- Key = $\{ A \}$
- $R$ is not in BCNF
- Decomposition $R_1 = (A, B), R_2 = (B, C)$
  - $R_1$ and $R_2$ in BCNF
  - Lossless-join decomposition
  - Dependency preserving

Example of BCNF Decomposition

- $R = (J, K, L)$
- $F = \{ JK \rightarrow L \}$
- Two candidate keys = JK and JL
- $R$ is not in BCNF
- Any decomposition of $R$ will fail to preserve $JK \rightarrow L$

BCNF and Dependency Preservation

- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important

Solution: define a weaker normal form, called Third Normal Form.

- Allows some redundancy (with resultant problems; we will see examples later)
- But FDs can be checked on individual relations without computing a join.
- There is always a lossless-join, dependency-preserving decomposition into 3NF.
A relation schema \( R \) is in third normal form (3NF) if for all:

\[ \alpha \rightarrow \beta \text{ in } F^+ \]

at least one of the following holds:

- \( \alpha \rightarrow \beta \) is trivial (i.e., \( \beta \subseteq \alpha \))
- \( \alpha \) is a superkey for \( R \)
- Each attribute \( A \) in \( \beta \setminus \alpha \) is contained in a candidate key for \( R \).

(NOTE: each attribute may be in a different candidate key)

If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).

Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).

Example

\( R = (J, K, L) \)

\( F = \{JK \rightarrow L, L \rightarrow K\} \)

Two candidate keys: \( JK \) and \( JL \)

\( R \) is in 3NF

\( JK \rightarrow L \)

\( L \rightarrow K \)

\( JK \) is a superkey

\( L \) is contained in a candidate key

BCNF decomposition has \( (JK) \) and \( (LK) \)

Testing for \( JK \rightarrow L \) requires a join

There is some redundancy in this schema

Equivalent to example in book:

Banker-schema = (branch-name, customer-name, banker-name)

banker-name \( \rightarrow \) branch-name

branch-name customer-name \( \rightarrow \) banker-name

Optimization: Need to check only FDs in \( F \), need not check all FDs in \( F^+ \).

Use attribute closure to check for each dependency \( \alpha \rightarrow \beta \), if \( \alpha \) is a superkey.

If \( \alpha \) is not a superkey, we have to verify if each attribute in \( \beta \) is contained in a candidate key of \( R \)

this test is rather more expensive, since it involve finding candidate keys

testing for 3NF has been shown to be NP-hard

Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time

Above algorithm ensures:

- each relation schema \( R_i \) is in 3NF
- decomposition is dependency preserving and lossless-join
- Proof of correctness is at end of this file [link here]

Relation schema:

Banker-info-schema = (branch-name, customer-name, banker-name, office-number)

The functional dependencies for this relation schema are:

banker-name \( \rightarrow \) branch-name office-number

customer-name branch-name \( \rightarrow \) banker-name

The key is:

\{customer-name, branch-name\}

The for loop in the algorithm causes us to include the following schemas in our decomposition:

Banker-office-schema = (banker-name, branch-name, office-number)

Banker-schema = (customer-name, branch-name, banker-name)

Since Banker-schema contains a candidate key for Banker-info-schema, we are done with the decomposition process.

It is always possible to decompose a relation into relations in 3NF and

- the decomposition is lossless
- the dependencies are preserved

It is always possible to decompose a relation into relations in BCNF and

- the decomposition is lossless
- it may not be possible to preserve dependencies.
### Comparison of BCNF and 3NF (Cont.)

- Example of problems due to redundancy in 3NF
  \[ R = (J, K, L) \]
  \[ F = \{ J \rightarrow K, L \rightarrow K \} \]

<table>
<thead>
<tr>
<th>J</th>
<th>L</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>j1</td>
<td>r1</td>
<td>k1</td>
</tr>
<tr>
<td>j2</td>
<td>r2</td>
<td>k2</td>
</tr>
<tr>
<td>j3</td>
<td>r3</td>
<td>k3</td>
</tr>
<tr>
<td>null</td>
<td>rnull</td>
<td>knull</td>
</tr>
</tbody>
</table>

A schema that is in 3NF but not in BCNF has the problems of:
- Repetition of information (e.g., the relationship \( t_1, k_1 \))
- Need to use null values (e.g., to represent the relationship \( t_2, k_2 \) where there is no corresponding value for \( J_2 \)).

### Testing for FDs Across Relations

If decomposition is not dependency preserving, we can have an extra materialized view for each dependency \( \alpha \rightarrow \beta \) in \( F \), that is not preserved in the decomposition.

- The materialized view is defined as a projection on \( \alpha \cap \beta \) of the join of the relations in the decomposition.
- Many newer database systems support materialized views and database system maintains the view when the relations are updated.
- No extra coding effort for programmer.
- The functional dependency \( \alpha \rightarrow \beta \) is expressed by declaring \( \alpha \) as a candidate key on the materialized view.
- Checking for candidate key cheaper than checking \( \alpha \rightarrow \beta \).

**BUT:**
- Space overhead: for storing the materialized view.
- Time overhead: Need to keep materialized view up to date when relations are updated.
- Database system may not support key declarations on materialized views.

### Multivalued Dependencies (Cont.)

<table>
<thead>
<tr>
<th>course</th>
<th>teacher</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>Avi</td>
<td>DB Concepts</td>
</tr>
<tr>
<td>database</td>
<td>Avi</td>
<td>Ullman</td>
</tr>
<tr>
<td>database</td>
<td>Hank</td>
<td>DB Concepts</td>
</tr>
<tr>
<td>database</td>
<td>Hank</td>
<td>Ullman</td>
</tr>
<tr>
<td>database</td>
<td>Sudarshan</td>
<td>DB Concepts</td>
</tr>
<tr>
<td>database</td>
<td>Sudarshan</td>
<td>Ullman</td>
</tr>
<tr>
<td>operating systems</td>
<td>Avi</td>
<td>OS Concepts</td>
</tr>
<tr>
<td>operating systems</td>
<td>Avi</td>
<td>Shaw</td>
</tr>
<tr>
<td>operating systems</td>
<td>Jim</td>
<td>OS Concepts</td>
</tr>
<tr>
<td>operating systems</td>
<td>Jim</td>
<td>Shaw</td>
</tr>
</tbody>
</table>

There are no non-trivial functional dependencies and therefore the relation is in BCNF.

- Insertion anomalies – i.e., if Sara is a new teacher that can teach database, two tuples need to be inserted:
  - (database, Sara, DB Concepts)
  - (database, Sara, Ullman)

### Multivalued Dependencies (MVDs)

- Let \( R \) be a relation schema and let \( \alpha \subseteq R \) and \( \beta \subseteq R \).
- The multivalued dependency
  \[ \alpha 

\[ \beta \]

holds on \( R \) if in any legal relation \( r(R) \), for all pairs for tuples \( t_1 \) and \( t_2 \) in \( r \) such that \( t_1[\alpha] = t_2[\alpha] \), there exist tuples \( t_1 \) and \( t_2 \) in \( r \) such that:

\[ t_1[\alpha] = t_2[\alpha] = t_3[\alpha] \]
\[ t_1[\beta] = t_2[\beta] = t_3[\beta] \]
\[ t_1[R - \beta] = t_2[R - \beta] \]
\[ t_3[R - \beta] = t_4[R - \beta] \]

### Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of:
  - Lack of dependency preservation.
  - Redundancy due to use of 3NF.
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.

### Multivalued Dependencies

There are database schemas in BCNF that do not seem to be sufficiently normalized.
- Consider a database
  \[ \text{classes}(\text{course}, \text{teacher}, \text{book}) \]
  such that \( (c, t, b) \in \text{classes} \) means that \( t \) is qualified to teach \( c \) and \( b \) is a required textbook for \( c \).

The database is supposed to list for each course the set of teachers any one of which can be the course’s instructor, and the set of books, all of which are required for the course (no matter who teaches it).

### MVD (Cont.)

- Tabular representation of \( \alpha \rightarrow \beta \):

\[
\begin{array}{ccc}
\alpha & \beta & R - \alpha - \beta \\
\hline
\{1\} & \{a_1 \ldots a_i\} & \{a_{i+1} \ldots a_j\} \\
\{2\} & \{a_1 \ldots a_i\} & \{b_j \ldots b_l\} \\
\{3\} & \{a_1 \ldots a_i\} & \{b_{j+1} \ldots b_{l+1}\} \\
\{4\} & \{a_1 \ldots a_i\} & \{b_{j+1} \ldots b_{l+1}\} \\
\end{array}
\]
Example

Let $R$ be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.

$Y, Z, W$

We say that $Y \rightarrow\rightarrow Z$ ($Y$ multidetermines $Z$) if and only if for all possible relations $r(R)$

$< y_1, z_1, w_1 > \in r \land < y_2, z_2, w_2 > \in r$

then

$< y_1, z_1, w_1 > \in r \land < y_2, z_2, w_2 > \in r$

Note that since the behavior of $Z$ and $W$ are identical it follows that $Y \rightarrow\rightarrow W$

Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:
  1. To test relations to determine whether they are legal under a given set of functional and multivalued dependencies
  2. To specify constraints on the set of legal relations. We shall thus concern ourselves only with relations that satisfy a given set of functional and multivalued dependencies.

- If a relation $r$ fails to satisfy a given multivalued dependency, we can construct a relations $r'$ that does satisfy the multivalued dependency by adding tuples to $r$.

Fourth Normal Form

- A relation schema $R$ is in 4NF with respect to a set $D$ of functional and multivalued dependencies if for all multivalued dependencies in $D'$ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
  1. $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  2. $\alpha$ is a superkey for schema $R$

- If a relation is in 4NF it is in BCNF

4NF Decomposition Algorithm

```
result = \{R_1\};
done = false;
compute D';
Let D_i denote the restriction of D' to R_i
while (not done)
  if (there is a schema R in result that is not in 4NF) then begin
    let $\alpha \rightarrow\rightarrow \beta$ be a nontrivial multivalued dependency that holds on $R_i$ such that $\alpha \rightarrow R_i$ is not in $D_i$, and $\alpha \cup \beta = \psi$;
    result := (result + $\{R_i, \beta\}$) \cup ($\alpha$, $\beta$);
  end
  else done := true;
Note: each $R_i$ is in 4NF, and decomposition is lossless-join
```

Theory of MVDs

- From the definition of multivalued dependency, we can derive the following rule:
  1. If $\alpha \rightarrow \beta$, then $\alpha \rightarrow\rightarrow \beta$

- That is, every functional dependency is also a multivalued dependency

- The closure $D'$ of $D$ is the set of all functional and multivalued dependencies logically implied by $D$.

- We can compute $D'$ from $D$, using the formal definitions of functional dependencies and multivalued dependencies.

- We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice

- For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules (see Appendix C).

Restriction of Multivalued Dependencies

- The restriction of $D$ to $R_i$ is the set $D_i$ consisting of
  1. All functional dependencies in $D'$ that include only attributes of $R_i$
  2. All multivalued dependencies of the form $\alpha \rightarrow\rightarrow (\beta \cap R_i)$

- where $\alpha \subseteq R_i$ and $\alpha \rightarrow\rightarrow \beta$ in $D'$

Example

- In our example:
  - course $\rightarrow$ teacher
  - course $\rightarrow$ book

- The above formal definition is supposed to formalize the notion that given a particular value of $Y$ (course) it has associated with it a set of values of $Z$ (teacher) and a set of values of $W$ (book), and these two sets are in some sense independent of each other.

- Note:
  1. If $Y \rightarrow Z$ then $Y \rightarrow\rightarrow Z$
  2. Indeed we have (in above notation) $Z_i = Z_j$

  The claim follows.
Further Normal Forms

- Join dependencies generalize multivalued dependencies
  - Lead to project-join normal form (PJNF) (also called fifth normal form)
- A class of even more general constraints, leads to a normal form called domain-key normal form.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
- However, in a real (imperfect) design there can be FDs from non-key attributes of an entity to other attributes of the entity
- E.g. employee entity with attributes department-number and department-address, and an FD department-number → department-address
  - Good design would have made department an entity
- FDs from non-key attributes of a relationship set possible, but rare --- most relationships are binary

Universal Relation Approach

- Dangling tuples may occur in practical database applications.
- They represent incomplete information
- E.g. may want to break up information about loans into:
  - (branch-name, loan-number)
  - (loan-number, amount)
  - (loan-number, customer-name)
- Universal relation would require null values, and have dangling tuples

Universal Relation Approach (Contd.)

- A particular decomposition defines a restricted form of incomplete information that is acceptable in our database.
  - Above decomposition requires at least one of customer-name, branch-name or amount in order to enter a loan number without using null values
  - Rules out storing of customer-name, amount without an appropriate loan-number (since it is a key, it can't be null either!)
- Universal relation requires unique attribute names unique role assumption
  - E.g. customer-name, branch-name
- Reuse of attribute names is natural in SQL since relation names can be prefixed to disambiguate names

Denormalization for Performance

- May want to use non-normalized schema for performance
- E.g. displaying customer-name along with account-number and balance requires join of account with depositor
- Alternative 1: Use denormalized relation containing attributes of account as well as depositor with all above attributes
  - Faster lookup
  - Extra space and extra execution time for updates
  - Extra coding work for programmer and possibility of error in extra code
- Alternative 2: Use a materialized view defined as a new account → depositor
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:
  - Above are in BCNF, but make querying across years difficult and needs new table each year
  - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
  - Is an example of a crosstab, where values for one attribute become column names
  - Used in spreadsheets, and in data analysis tools
Proof of Correctness of 3NF Decomposition Algorithm

Correctness of 3NF Decomposition Algorithm (Contd.)

Claim: if a relation Ri is in the decomposition generated by the above algorithm, then Ri satisfies 3NF.

Let R be generated from the dependency α → β

Let γ → B be any non-trivial functional dependency on R. (We need only consider FDs whose right-hand side is a single attribute.)

Now, B can be in either β or α but not in both. Consider each case separately.

Correctness of 3NF Decomposition (Contd.)

Case 1: B in β:

If γ is a superkey, the 2nd condition of 3NF is satisfied

Otherwise α must contain some attribute not in γ

Since γ → B is in F* it must be derivable from Fc by using attribute closure on γ.

Attribute closure not have used α → β - if it had been used, α must be contained in the attribute closure of γ, which is not possible, since we assumed γ is not a superkey.

Now, using α → (β - {B}) and γ → B, we can derive α → B (since γ ⊆ αβ, and B ∉ γ since γ → B is non-trivial)

Then, B is extraneous in the right-hand side of α → β; which is not possible since α → β is in Fc.

Thus, if B is in β then γ must be a superkey, and the second condition of 3NF must be satisfied.

Correctness of 3NF Decomposition (Contd.)

Case 2: B is in α.

Since α is a candidate key, the third alternative in the definition of 3NF is trivially satisfied.

In fact, we cannot show that γ is a superkey.

This shows exactly why the third alternative is present in the definition of 3NF.

Q.E.D.

End of Chapter

Sample lending Relation

<table>
<thead>
<tr>
<th>branch-name</th>
<th>branch-city</th>
<th>assets</th>
<th>customer-name</th>
<th>loan-number</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>900000</td>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Redwood</td>
<td>Palo Alto</td>
<td>2100000</td>
<td>Smith</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>Perryridge</td>
<td>Honesneck</td>
<td>1700000</td>
<td>Hayes</td>
<td>L-15</td>
<td>1500</td>
</tr>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>9000000</td>
<td>Jackson</td>
<td>L-14</td>
<td>1500</td>
</tr>
<tr>
<td>Miami</td>
<td>Honesneck</td>
<td>4000000</td>
<td>Jones</td>
<td>L-93</td>
<td>500</td>
</tr>
<tr>
<td>Round Hill</td>
<td>Honesneck</td>
<td>8000000</td>
<td>Turner</td>
<td>L-11</td>
<td>900</td>
</tr>
<tr>
<td>Pownal</td>
<td>Btrington</td>
<td>3000000</td>
<td>Williams</td>
<td>L-29</td>
<td>1200</td>
</tr>
<tr>
<td>North Town</td>
<td>Rye</td>
<td>3700000</td>
<td>Hayes</td>
<td>L-16</td>
<td>1300</td>
</tr>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>9000000</td>
<td>Johnson</td>
<td>L-18</td>
<td>2000</td>
</tr>
<tr>
<td>Perryridge</td>
<td>Honesneck</td>
<td>1700000</td>
<td>Glenn</td>
<td>L-25</td>
<td>2500</td>
</tr>
<tr>
<td>Brighton</td>
<td>Brooklyn</td>
<td>7100000</td>
<td>Brooks</td>
<td>L-10</td>
<td>2200</td>
</tr>
</tbody>
</table>

Sample Relation r

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>d3</td>
</tr>
<tr>
<td>a3</td>
<td>b3</td>
<td>c2</td>
<td>d4</td>
</tr>
</tbody>
</table>