FUNCTIONAL DEPENDENCY THEORY II
Last Time: Canonical Cover

- Last time, introduced concept of canonical cover
- A canonical cover $F_c$ for $F$ is a set of functional dependencies such that:
  - $F$ logically implies all dependencies in $F_c$
  - $F_c$ logically implies all dependencies in $F$
  - Can’t infer any functional dependency in $F_c$ from other dependencies in $F_c$
  - No functional dependency in $F_c$ contains an extraneous attribute
  - Left side of all functional dependencies in $F_c$ are unique
    - There are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in $F_c$ such that $\alpha_1 = \alpha_2$
Extraneous Attributes

- Given a set $F$ of functional dependencies
  - An attribute in a functional dependency is extraneous if it can be removed from $F$ without affecting closure of $F$

- Formally: given $F$, and $\alpha \rightarrow \beta$
  - If $A \in \alpha$, and $F$ logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$, then $A$ is extraneous
  - If $A \in \beta$, and $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies $F$, then $A$ is extraneous
    - i.e. generate a new set of functional dependencies $F'$ by replacing $\alpha \rightarrow \beta$ with $\alpha \rightarrow (\beta - A)$
    - See if $F'$ logically implies $F$
Testing Extraneous Attributes

- Given relation schema $R$, and a set $F$ of functional dependencies that hold on $R$
- Attribute $A$ in $\alpha \rightarrow \beta$
- If $A \in \alpha$ (i.e. $A$ is on left side of the dependency), then let $\gamma = \alpha - \{A\}$
  - See if $\gamma \rightarrow \beta$ can be inferred from $F$
  - Compute $\gamma^+$ under $F$
  - If $\beta \subseteq \gamma^+$ then $A$ is extraneous in $\alpha$
Testing Extraneous Attributes (2)

- Given relation schema $R$, and a set $F$ of functional dependencies that hold on $R$
- Attribute $A$ in $\alpha \rightarrow \beta$
- If $A \in \beta$ (on right side of the dependency), then try the altered set $F'$
  - $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$
  - See if $\alpha \rightarrow A$ can be inferred from $F'$
  - Compute $\alpha^+$ under $F'$
  - If $\alpha^+$ includes $A$ then $A$ is extraneous in $\beta$
Computing Canonical Cover

- A simple way to compute the canonical cover of $F$

\[
F_c = F
\]

repeat

- apply union rule to replace dependencies in $F_c$ of form $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$
- find a functional dependency $\alpha \rightarrow \beta$ in $F_c$ with an extraneous attribute

/* Use $F_c$ for the extraneous attribute test, not $F$ !!! */

if an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

until $F_c$ stops changing
Canonical Cover Example

- Functional dependencies $F$ on schema $(A, B, C)$
  - $F = \{ A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C \}$
  - Find $F_c$
- Apply union rule to $A \rightarrow BC$ and $A \rightarrow B$
  - Left with: $\{ A \rightarrow BC, B \rightarrow C, AB \rightarrow C \}$
- $A$ is extraneous in $AB \rightarrow C$
  - $B \rightarrow C$ is logically implied by $F$ (obvious)
  - Left with: $\{ A \rightarrow BC, B \rightarrow C \}$
- $C$ is extraneous in $A \rightarrow BC$
  - Logically implied by $A \rightarrow B, B \rightarrow C$
- $F_c = \{ A \rightarrow B, B \rightarrow C \}$
A set of functional dependencies can have multiple canonical covers

Example:

\[ F = \{ A \rightarrow BC, B \rightarrow AC, C \rightarrow AB \} \]

Has several canonical covers:

- \( F_c = \{ A \rightarrow B, B \rightarrow C, C \rightarrow A \} \)
- \( F_c = \{ A \rightarrow B, B \rightarrow AC, C \rightarrow B \} \)
- \( F_c = \{ A \rightarrow C, C \rightarrow B, B \rightarrow A \} \)
- \( F_c = \{ A \rightarrow C, B \rightarrow C, C \rightarrow AB \} \)
- \( F_c = \{ A \rightarrow BC, B \rightarrow A, C \rightarrow A \} \)
Another Example

- Functional dependencies $F$ on schema $(A, B, C, D)$
  - $F = \{ A \rightarrow B, BC \rightarrow D, AC \rightarrow D \}$
  - Find $F_c$

- In this case, it may look like $F_c = F$…

- However, can infer $AC \rightarrow D$ from $A \rightarrow B, BC \rightarrow D$ (pseudotransitivity), so $AC \rightarrow D$ is extraneous in $F$
  - Therefore, $F_c = \{ A \rightarrow B, BC \rightarrow D \}$

- Alternately, can argue that $D$ is extraneous in $AC \rightarrow D$
  - With $F' = \{ A \rightarrow B, BC \rightarrow D \}$, we see that $\{AC\}^+ = ACD$, so $D$ is extraneous in $AC \rightarrow D$
  - (If you eliminate the entire RHS of a functional dependency, it goes away)
Some schema decompositions lose information

Example:

\texttt{employee(emp\_id, emp\_name, phone, title, salary, start\_date)}

- Decomposed into:
  
  \texttt{emp\_ids(emp\_id, emp\_name)}
  \texttt{emp\_details(emp\_name, phone, title, salary, start\_date)}

Problem:

- \texttt{emp\_name} doesn’t uniquely identify employees
- This is a lossy decomposition
Lossless Decompositions

- Given:
  - Relation schema $R$, relation $r(R)$
  - Set of functional dependencies $F$
- Let $R_1$ and $R_2$ be a decomposition of $R$
  - $R_1 \cup R_2 = R$
- The decomposition is lossless if, for all legal instances of $r$:
  - $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$
- A simple definition…
Lossless Decompositions (2)

- Can define with functional dependencies:
  - $R_1$ and $R_2$ form a lossless decomposition of $R$ if at least one of these dependencies is in $F^+$:
    \[ R_1 \cap R_2 \rightarrow R_1 \]
    \[ R_1 \cap R_2 \rightarrow R_2 \]
  - $R_1 \cap R_2$ forms a superkey of $R_1$ and/or $R_2$
    - Test for superkeys using attribute-set closure
Decomposition Examples (1)

- The employee example:
  
  ```
  employee(emp_id, emp_name, phone, title, salary, start_date)
  ```

- Decomposed into:
  
  ```
  emp_ids(emp_id, emp_name)
  emp_details(emp_name, phone, title, salary, start_date)
  ```

- `emp_name` is not a superkey of `emp_ids` or `emp_details`, so the decomposition is lossy
Decomposition Examples (2)

- The `bor_loan` example:
  
  \( \text{bor\_loan}(\text{cust\_id}, \text{loan\_id}, \text{amount}) \)

- Decomposed into:
  
  \( \text{borrower}(\text{cust\_id}, \text{loan\_id}) \)
  
  \( \text{loan}(\text{loan\_id}, \text{amount}) \quad (\text{loan\_id} \rightarrow \text{loan\_id}, \text{amount}) \)

- `loan\_id` is a superkey of `loan`, so the decomposition is lossless
BCNF Decompositions

- If $R$ is a schema not in BCNF:
  - There is at least one nontrivial functional dependency $\alpha \rightarrow \beta$ such that $\alpha$ is not a superkey for $R$
  - For simplicity, also require that $\alpha \cap \beta = \emptyset$
    - (if $\alpha \cap \beta \neq \emptyset$ then $(\alpha \cap \beta)$ is extraneous in $\beta$)
- Replace $R$ with two schemas:
  - $R_1 = (\alpha \cup \beta)$
  - $R_2 = (R - \beta)$
    - (was $R - (\beta - \alpha)$, but $\beta - \alpha = \beta$, since $\alpha \cap \beta = \emptyset$)
- BCNF decomposition is lossless
  - $R_1 \cap R_2 = \alpha$
  - $\alpha$ is a superkey of $R_1$
  - $\alpha$ also appears in $R_2$
Dependency Preservation

Some schema decompositions are not dependency-preserving

- Functional dependencies that span multiple relation schemas are hard to enforce
- E.g. BCNF may require decomposition of a schema for one dependency, and make it hard to enforce another dependency

Can test for dependency preservation using functional dependency theory
Dependency Preservation (2)

- **Given:**
  - A set \( F \) of functional dependencies on a schema \( R \)
  - \( R_1, R_2, \ldots, R_n \) are a decomposition of \( R \)

- The **restriction** of \( F \) to \( R_i \) is the set \( F_i \) of functional dependencies in \( F^+ \) that only has attributes in \( R_i \)
  - Each \( F_i \) contains functional dependencies that can be checked efficiently, using only \( R_i \)

- **Find all** functional dependencies that can be checked efficiently
  - \( F' = F_1 \cup F_2 \cup \ldots \cup F_n \)
  - If \( F'^+ = F^+ \) then the decomposition is dependency-preserving
Third Normal Form Schemas

- Can generate a 3NF schema from a set of functional dependencies $F$
- Called the 3NF synthesis algorithm
  - Instead of decomposing an initial schema, generates schemas from a set of dependencies
- Given a set $F$ of functional dependencies
  - Uses the canonical cover $F_c$
  - Ensures that resulting schemas are dependency-preserving
3NF Synthesis Algorithm

- Inputs: set of functional dependences $F$, on a schema $R$

  let $F_c$ be a canonical cover for $F$
  $i := 0$
  for each functional dependency $\alpha \rightarrow \beta$ in $F_c$
    if none of the schemas $R_j$, $j = 1, 2, \ldots, i$ contains $(\alpha \cup \beta)$ then
      $i := i + 1$
      $R_i := (\alpha \cup \beta)$
    end if
  done
  if no schema $R_j$, $j = 1, 2, \ldots, i$ contains a candidate key for $R$ then
    $i := i + 1$
    $R_i :=$ any candidate key for $R$
  end if
  return $(R_1, R_2, \ldots, R_i)$
BCNF vs. 3NF

- **Boyce-Codd Normal Form:**
  - Eliminates more redundant information than 3NF
  - Some functional dependencies become expensive to enforce
    - The conditions to enforce involve multiple relations
  - Overall, a very desirable normal form!

- **Third Normal Form:**
  - All [more] dependencies are [probably] easy to enforce...
  - Allows more redundant information, which must be kept synchronized by the database application!
  - Personal banker example:
    ```sql
    works_in(emp_id, branch_name)
    cust_banker_branch(cust_id, branch_name, emp_id, type)
    ```
    - Branch names must be kept synchronized between these relations!
BCNF and 3NF vs. SQL

- SQL constraints:
  - Only key constraints are fast and easy to enforce!
  - Only easy to enforce functional dependencies $\alpha \rightarrow \beta$ if $\alpha$ is a key on some table!
  - Other functional dependencies (even “easy” ones in 3NF) may require more expensive constraints, e.g. CHECK

- For SQL databases with materialized views:
  - Can decompose a schema into BCNF
  - For dependencies $\alpha \rightarrow \beta$ not preserved in decomposition, create materialized view joining all relations in dependency
  - Enforce unique($\alpha$) constraint on materialized view

- Impacts both space and performance, but it works...
Multivalued Attributes

- E-R schemas can have multivalued attributes
- 1NF requires only atomic attributes
  - Not a problem; translating to relational model leaves everything atomic
- Employee example:
  - \texttt{employee(\texttt{emp_id}, \texttt{emp_name})}
  - \texttt{emp_deps(\texttt{emp_id}, \texttt{dependent})}
  - \texttt{emp_nums(\texttt{emp_id}, \texttt{phone_num})}
- What are the requirements on these schemas for what tuples must appear?

<table>
<thead>
<tr>
<th>employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{emp_id}</td>
</tr>
<tr>
<td>\texttt{emp_name}</td>
</tr>
<tr>
<td>{ \texttt{phone_num} }</td>
</tr>
<tr>
<td>{ \texttt{dependent} }</td>
</tr>
</tbody>
</table>
Multivalued Attributes (2)

- Example data:

<table>
<thead>
<tr>
<th>emp_id</th>
<th>emp_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>125623</td>
<td>Rick</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>emp_id</th>
<th>dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>125623</td>
<td>Jeff</td>
</tr>
<tr>
<td>125623</td>
<td>Alice</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>emp_id</th>
<th>phone_num</th>
</tr>
</thead>
<tbody>
<tr>
<td>125623</td>
<td>555-8888</td>
</tr>
<tr>
<td>125623</td>
<td>555-2222</td>
</tr>
</tbody>
</table>

- Every distinct value of multivalued attribute requires a separate tuple, including associated value of emp_id.

- A consequence of 1NF, in fact!
  - If attributes could be nonatomic, could just store list of values in the appropriate column!
  - 1NF requires extra tuples to represent multivalues.
Independent Multivalued Attributes

- Question is trickier when a schema stores several independent multivalued attributes
- Proposed combined schema:
  employee(emp_id, emp_name)
  emp_info(emp_id, dependent, phone_num)
- What tuples must appear in emp_info?
  - emp_info is a relation
  - If an employee has M dependents and N phone numbers, emp_info must contain M × N tuples
    - Exactly what we get if we natural-join emp_depts and emp_nums
  - Every combination of the employee’s dependents and their phone numbers
Independent Multivalued Attributes

- Example data:

<table>
<thead>
<tr>
<th>emp_id</th>
<th>emp_name</th>
<th>dependent</th>
<th>phone_num</th>
</tr>
</thead>
<tbody>
<tr>
<td>125623</td>
<td>Rick</td>
<td>Jeff</td>
<td>555-8888</td>
</tr>
<tr>
<td>125623</td>
<td>Jeff</td>
<td>555-2222</td>
<td></td>
</tr>
<tr>
<td>125623</td>
<td>Alice</td>
<td>555-8888</td>
<td></td>
</tr>
<tr>
<td>125623</td>
<td>Alice</td>
<td>555-2222</td>
<td></td>
</tr>
</tbody>
</table>

- Clearly has unnecessary redundancy
- Can’t formulate functional dependencies to represent multivalued attributes
- Can’t use BCNF or 3NF decompositions to eliminate redundancy in these cases
Multivalued Attributes Example

- Two employees: Rick and Bob
  - Both share a phone number at work
  - Both have two kids
  - Both have a kid named Alice

- Can’t use functional dependencies to reason about this situation!
  - \( emp_id \rightarrow phone_num \) doesn’t hold since an employee can have several phone numbers
  - \( phone_num \rightarrow emp_id \) doesn’t hold either, since several employees can have the same phone number
  - Same with \( emp_id \) and dependent…
Functional dependencies rule out what tuples can appear in a relation

- If $A \rightarrow B$ holds, then tuples cannot have the same value for $A$ but different values for $B$
- Also called equality-generating dependencies

Multivalued dependencies specify what tuples must be present

- To represent a multivalued attribute’s values properly, a certain set of tuples must be present
- Also called tuple-generating dependencies
Multivalued Dependencies

- Given a relation schema $R$
  - Attribute-sets $\alpha \in R$, $\beta \in R$
  - $\alpha \rightarrow \beta$ is a multivalued dependency
  - “$\alpha$ multidetermines $\beta$”

- A multivalued dependency $\alpha \rightarrow \beta$ holds on $R$ if, in any legal relation $r(R)$:
  For all pairs of tuples $t_1$ and $t_2$ in $r$ such that $t_1[\alpha] = t_2[\alpha]$, there also exists tuples $t_3$ and $t_4$ in $r$ such that:
    - $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
    - $t_1[\beta] = t_3[\beta]$ and $t_2[\beta] = t_4[\beta]$
    - $t_1[R - \beta] = t_4[R - \beta]$ and $t_2[R - \beta] = t_3[R - \beta]$
Multivalued Dependencies (2)

- Multivalued dependency $\alpha \rightarrow \beta$ holds on $R$ if, in any legal relation $r(R)$:
  For all pairs of tuples $t_1$ and $t_2$ in $r$ such that $t_1[\alpha] = t_2[\alpha]$, There also exists tuples $t_3$ and $t_4$ in $r$ such that:
  - $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
  - $t_1[\beta] = t_3[\beta]$ and $t_2[\beta] = t_4[\beta]$
  - $t_1[R - \beta] = t_4[R - \beta]$ and $t_2[R - \beta] = t_3[R - \beta]$

- Pictorially:

<table>
<thead>
<tr>
<th></th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$R - (\alpha \cup \beta)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>$a_1...a_i$</td>
<td>$a_{i+1}...a_j$</td>
<td>$a_{j+1}...a_n$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$a_1...a_i$</td>
<td>$b_{i+1}...b_j$</td>
<td>$b_{j+1}...b_n$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$a_1...a_i$</td>
<td>$a_{i+1}...a_j$</td>
<td>$b_{j+1}...b_n$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>$a_1...a_i$</td>
<td>$b_{i+1}...b_j$</td>
<td>$a_{j+1}...a_n$</td>
</tr>
</tbody>
</table>
Multivalued Dependencies (3)

- Multivalued dependency:

<table>
<thead>
<tr>
<th></th>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>( R - (\alpha \cup \beta) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( a_1 \ldots a_i )</td>
<td>( a_{i+1} \ldots a_j )</td>
<td>( a_{j+1} \ldots a_n )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( a_1 \ldots a_i )</td>
<td>( b_{i+1} \ldots b_j )</td>
<td>( b_{j+1} \ldots b_n )</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>( a_1 \ldots a_i )</td>
<td>( a_{i+1} \ldots a_j )</td>
<td>( b_{j+1} \ldots b_n )</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>( a_1 \ldots a_i )</td>
<td>( b_{i+1} \ldots b_j )</td>
<td>( a_{j+1} \ldots a_n )</td>
</tr>
</tbody>
</table>

- If \( \alpha \rightarrow \beta \) then \( R - (\alpha \cup \beta) \) is independent of this fact
  - Every distinct value of \( \beta \) must be associated once with every distinct value of \( R - (\alpha \cup \beta) \)

- Let \( \gamma = R - (\alpha \cup \beta) \)
  - If \( \alpha \rightarrow \beta \) then also \( \alpha \rightarrow \gamma \)
  - \( \alpha \rightarrow \beta \) implies \( \alpha \rightarrow \gamma \)
  - Sometimes written \( \alpha \rightarrow \beta \ | \ \gamma \)
Trivial Multivalued Dependencies

- $\alpha \rightarrow\rightarrow \beta$ is a trivial multivalued dependency on $R$ if all relations $r(R)$ satisfy the dependency.
- Specifically, $\alpha \rightarrow\rightarrow \beta$ is trivial if $\beta \subseteq \alpha$, or if $\alpha \cup \beta = R$.
- Employee examples:
  - For schema $emp\_deps(emp\_id, dependent)$, $emp\_id \rightarrow dependent$ is trivial.
  - For $emp\_info(emp\_id, dependent, phone\_num)$, $emp\_id \rightarrow dependent$ is not trivial.
Inference Rules

- Can reason about multivalued dependencies, just like functional dependencies
- There is a set of complete, sound inference rules for MVDs

Example inference rules:

- Complementation rule:
  - If $\alpha \rightarrow \beta$ holds on $R$, then $\alpha \rightarrow R - (\alpha \cup \beta)$ holds

- Multivalued augmentation rule:
  - If $\alpha \rightarrow \beta$ holds, and $\gamma \subseteq R$, and $\delta \subseteq \gamma$, then $\gamma \alpha \rightarrow \delta \beta$ holds

- Multivalued transitivity rule:
  - If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma - \beta$ holds

- Coalescence rule:
  - If $\alpha \rightarrow \beta$ holds, and $\gamma \subseteq \beta$, and there is a $\delta$ such that $\delta \subseteq R$, and $\delta \cap \beta = \emptyset$, and $\delta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ holds
Functional Dependencies

- Functional dependencies are also multivalued dependencies

- Replication rule:
  - If $α → β$, then $α →→ β$ too
  - Note there is an additional constraint from $α → β$: each value of $α$ has at most one associated value for $β$

- Usually, functional dependencies are not stated as multivalued dependencies
  - The extra caveat is *important*, but not obvious in notation
  - Also, functional dependencies are easier to reason about!
Closures and Restrictions

- For a set $D$ of functional and multivalued dependencies, can compute closure $D^+$
  - Use inference rules for both functional and multivalued dependencies to compute closure
- Sometimes need the restriction of $D^+$ to a relation schema $R$, too
- The restriction of $D$ to a schema $R_i$ includes:
  - All functional dependencies in $D^+$ that include only attributes in $R_i$
  - All multivalued dependencies of the form $\alpha \rightarrow \beta \cap R_i$, where $\alpha \subseteq R_i$, and $\alpha \rightarrow \beta$ is in $D^+$
Fourth Normal Form

- **Given:**
  - Relation schema $R$
  - Set of functional and multivalued dependencies $D$

- $R$ is in 4NF with respect to $D$ if:
  - For all multivalued dependencies $\alpha \rightarrow \beta$ in $D^+$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
    - $\alpha \rightarrow \beta$ is a trivial multivalued dependency
    - $\alpha$ is a superkey for $R$
  - **Note:** If $\alpha \rightarrow \beta$ then $\alpha \rightarrow \beta$

- A database design is in 4NF if all schemas in the design are in 4NF
4NF and BCNF

- Main difference between 4NF and BCNF is use of multivalued dependencies instead of functional dependencies
- Every schema in 4NF is also in BCNF
  - If a schema is not in BCNF then there is a nontrivial functional dependency \( \alpha \rightarrow \beta \) such that \( \alpha \) is not a superkey for \( R \)
  - If \( \alpha \rightarrow \beta \) then \( \alpha \rightarrow \rightarrow \beta \)
4NF Decompositions

- Decomposition rule very similar to BCNF
- If schema $R$ is not in 4NF with respect to a set of multivalued dependencies $D$:
  - There is some nontrivial dependency $\alpha \rightarrow \beta$ in $D^+$ where $\alpha \subseteq R$ and $\beta \subseteq R$, and $\alpha$ is not a superkey of $R$
    - Also constrain that $\alpha \cap \beta = \emptyset$
  - Replace $R$ with two new schemas:
    - $R_1 = (\alpha \cup \beta)$
    - $R_2 = (R - \beta)$
Employee Information Example

- **Combined schema:**
  
  \[
  \begin{align*}
  &\text{employee}(\text{emp\_id}, \text{emp\_name}) \\
  &\text{emp\_info}(\text{emp\_id}, \text{dependent}, \text{phone\_num})
  \end{align*}
  \]

  - Also have these dependencies:
    - \( \text{emp\_id} \rightarrow \text{emp\_name} \)
    - \( \text{emp\_id} \rightarrow \text{dependent} \)
    - \( \text{emp\_id} \rightarrow \text{phone\_num} \)

- **emp\_info** is not in 4NF

- **Following the rules for 4NF decomposition produces:**
  
  \[
  \begin{align*}
  &(\text{emp\_id}, \text{dependent}) \\
  &(\text{emp\_id}, \text{phone\_num})
  \end{align*}
  \]

  - **Note:** Each relation’s candidate key is the entire relation. The multivalued dependencies are trivial.
Can also define lossless decomposition with multivalued dependencies

$R_1$ and $R_2$ form a lossless decomposition of $R$ if at least one of these dependencies is in $D^+$:

$R_1 \cap R_2 \rightarrow R_1$

$R_1 \cap R_2 \rightarrow R_2$
Additional normal forms with various constraints

Example: join dependencies

Given $R$, and a decomposition $R_1$ and $R_2$ where $R_1 \cup R_2 = R$:

- The decomposition is lossless if, for all legal instances of $r(R)$,
  \[ \Pi_{R_1}(r) \Join \Pi_{R_2}(r) = r \]

Can state this as a join dependency: \((R_1, R_2)\)

- This is actually identical to a multivalued dependency!
- \((R_1, R_2)\) is equivalent to $R_1 \cap R_2 \rightarrow R_1 \mid R_2$
Join Dependencies and 5NF

- Join dependencies (JD) are a generalization of multivalued dependencies (MVD)
  - Can specify JDs involving N relation schemas, $N \geq 2$
  - JDs are equivalent to MVDs when $N = 2$
  - Can easily construct JDs where $N > 2$, with no equivalent set of MVDs

- Project-Join Normal Form (a.k.a. PJNF or 5NF):
  - A relation schema $R$ is in PJNF with respect to a set of join dependencies $D$ if, for all JDs in $D^+$ of the form
    *
    
    $(R_1, R_2, \ldots, R_n)$ where $R_1 \cup R_2 \cup \ldots \cup R_n = R$, at least one of the following holds:
    - $(R_1, R_2, \ldots, R_n)$ is a trivial join dependency
    - Every $R_i$ is a superkey for $R$
Join Dependencies and 5NF (2)

- If a schema is in Project-Join Normal Form then it is also in 4NF (and thus, in BCNF)
  - Every multivalued dependency is also a join dependency
  - (Every functional dependency is also a multivalued dependency)

- One small problem:
  - There isn’t a complete, sound set of inference rules for join dependencies!
  - Can’t reason about our set of join dependencies $D$...
  - This limits PJNF’s real-world usefulness
Domain-Key Normal Form

- Domain-key normal form (DKNF) is an even more general normal form, based on:
  - **Domain constraints**: what values may be assigned to attribute $A$
    - Usually inexpensive to test, even with **CHECK** constraints
  - **Key constraints**: all attribute-sets $K$ that are a superkey for a schema $R$ (i.e. $K \rightarrow R$)
    - Almost always inexpensive to test
  - **General constraints**: other predicates on valid relations in a schema
    - Could be very expensive to test!

- A schema $R$ is in DKNF if the domain constraints and key constraints logically imply the general constraints
  - An “ideal” normal form difficult to achieve in practice…