Chapter 3 Design Theory for Relational Databases
Contents

- Functional Dependencies
- Decompositions
- Normal Forms (BCNF, 3NF)
- Multivalued Dependencies (and 4NF)
- Reasoning About FD’s + MVD’s
Our example of chapter 2

Beers(name, manf)
Bars(name, addr, license)
Drinkers(name, addr, phone)
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)

Some questions:

1. Why do we design relations like the example?
2. What makes a good relational database schema?
3. What can we do if it has flaws?

A theory: “dependencies” will be talked first
Functional Dependencies

- $X \rightarrow Y$ is an assertion about a relation $R$ that whenever two tuples of $R$ agree on all the attributes of $X$, then they must also agree on all attributes in set $Y$.
  - Say “$X \rightarrow Y$ holds in $R$.”
  - **Convention**: ..., $X$, $Y$, $Z$ represent sets of attributes; $A$, $B$, $C$,... represent single attributes.
  - **Convention**: no set formers in sets of attributes, just $ABC$, rather than $\{A,B,C\}$. 
Functional Dependency (cont.)

- Exist in a relational schema as a constraint.
- Agree for all instances of the schema (t and u are any two tuples)

<table>
<thead>
<tr>
<th>A’s</th>
<th>B’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td></td>
</tr>
</tbody>
</table>

If t and u agree here, then they must agree here.

We have functional dependency like this: $A_1A_2\ldots \rightarrow B_1B_2\ldots$

Why we call “functional” dependency?
Functional Dependency (cont.)

- Some examples

  Beers(name, manf)
  name → manf          manf → name
  
  Sells(bar, beer, price)
  Bar, beer → price
Splitting Right Sides of FD’s

- \( X\rightarrow A_1A_2...A_n \) holds for \( R \) exactly when each of \( X\rightarrow A_1, X\rightarrow A_2,..., X\rightarrow A_n \) hold for \( R \).
- **Example:** \( A\rightarrow BC \) is equivalent to \( A\rightarrow B \) and \( A\rightarrow C \).
- There is no splitting rule for left sides.
- We’ll generally express FD’s with singleton right sides.
Trivial Functional Dependencies

Sells(bar, beer, price)
Bar, Beer $\rightarrow$ bar  (trivial functional dependencies)
Bar, beer $\rightarrow$ price  (nontrivial functional dependencies)

A’s $\rightarrow$ B’s
A’s $\rightarrow$ C’s
Example: FD’s

Drinkers(name, addr, beersLiked, manf, favBeer)

- Reasonable FD’s to assert:
  1. name -> addr favBeer (combining rule)
     - Note this FD is the same as name -> addr and name -> favBeer. (splitting rule)
  2. beersLiked -> manf
### Example: Possible Data

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Bud, WickedAle</td>
<td>A.B.</td>
<td>WickedAle, WickedAle</td>
</tr>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Bud</td>
<td>A.B.</td>
<td>Bud</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Bud</td>
<td>Pete’s A.B.</td>
<td>Bud</td>
</tr>
</tbody>
</table>

**Explanation:**
- Because `name` -> `addr`
- Because `name` -> `favBeer`
- Because `beersLiked` -> `manf`
Keys of Relations

- $K$ is a **superkey** for relation $R$ if $K$ functionally determines all of $R$.
- $K$ is a **key** for $R$ if $K$ is a superkey, but no proper subset of $K$ is a superkey. (minimality)
Example: Superkey

Drinkers(name, addr, beersLiked, manf, favBeer)

- \{name, beersLiked\} is a superkey because together these attributes determine all the other attributes.
  - name -> addr favBeer
  - beersLiked -> manf
Example: Key

- \{\text{name, beersLiked}\} is a \text{key} because neither \{\text{name}\} nor \{\text{beersLiked}\} is a superkey.
  - name doesn’t -> manf; beersLiked doesn’t -> addr.

- There are no other keys, but lots of superkeys.
  - Any superset of \{\text{name, beersLiked}\}.
Where Do Keys Come From?

1. Just assert a key $K$.
   - The only FD’s are $K \rightarrow A$ for all attributes $A$.

2. Assert FD’s and deduce the keys by systematic exploration.

3. More FD’s From “Physics”
   - Example: “no two courses can meet in the same room at the same time” tells us: hour room $\rightarrow$ course.
We are given FD’s $X_1 \rightarrow A_1$, $X_2 \rightarrow A_2$, ..., $X_n \rightarrow A_n$, and we want to know whether an FD $Y \rightarrow B$ must hold in any relation (instance) that satisfies the given FD’s.

- Example: If $A \rightarrow B$ and $B \rightarrow C$ hold, surely $A \rightarrow C$ holds, even if we don’t say so.

Important for design of good relation schemas.
1: Inference Test

- To test if $Y \rightarrow B$, start by assuming two tuples agree in all attributes of $Y$.

\[
\begin{array}{c}
\text{Y} \\
\rightarrow \\
\text{00000000...0} \\
\text{00000??...?}
\end{array}
\]
2: Inference Test

- Use the given FD’s to infer that these tuples must also agree in certain other attributes.
  - If B is one of these attributes, then $Y \rightarrow B$ is true.
  - Otherwise, the two tuples, with any forced equalities, form a two-tuple relation that proves $Y \rightarrow B$ does not follow from the given FD’s.
2: Inference Test : example

- R(A,B,C) with FD’s: A→B, B→C
- To prove A→C?

**Inference steps:**

1) Assume two tuples that agree on A

2) Because A→B, b1=b2

3) Because B→C, c1=c2
Inference rules

- **Reflexivity:**
  If \( \{B_1 B_2 \ldots B_m\} \subseteq \{A_1, A_2, \ldots A_n\} \) then
  \( A_1, A_2, \ldots A_n \rightarrow B_1 B_2 \ldots B_m \) called trivial FD’s

- **Augmentation:**
  If \( A_1, A_2, \ldots A_n \rightarrow B_1 B_2 \ldots B_m \) then,
  \( A_1, A_2, \ldots A_n C_1, C_2 \ldots C_k \rightarrow B_1 B_2 \ldots B_m C_1, C_2 \ldots C_k \)

- **Transitivity:**
  If \( A_1, A_2, \ldots A_n \rightarrow B_1 B_2 \ldots B_m, \) and \( B_1 B_2 \ldots B_m \rightarrow C_1, C_2 \ldots C_k \) then, \( A_1, A_2, \ldots A_n \rightarrow C_1, C_2 \ldots C_k \)
3: Closure Test

- An easier way to test is to compute the closure of $Y$, denoted $Y^+$. 
- **Basis:** $Y^+ = Y$. 
- **Induction:** Look for an FD’s left side $X$ that is a subset of the current $Y^+$. If the FD is $X -> A$, add $A$ to $Y^+$. 
- **End:** when $Y^+$ can not be changed.
3: Closure Test: example

- R(A,B,C) with FD’s: A → B, B → C
- To prove A → C?

Calculating steps for $A^+$:

1. $A^+ = A$
2. $A^+ = A, B$
3. $A^+ = A, B, C$ → $A \rightarrow C$

Closure and Keys: if the closure of $X$ is all attributes of a relation, then $X$ is a key / superkey.
Computing the closure of a set of attributes

- The closure algorithm 3.7 (pp. 76) can discover all true FD’s.
- We need a FD’s (minimal basis) to represent the full set of FD’s for a relation.
Closing sets of Functional dependencies

- Example: $R(A,B,C)$ with all FD’s:
  $A \rightarrow B, A \rightarrow C, B \rightarrow A, B \rightarrow C, C \rightarrow A, C \rightarrow B, AB \rightarrow C, AC \rightarrow B, BC \rightarrow A, \ldots$

- We are free to choose any **basis** for the FD’s of $R$, a set of FD’s that imply all the FD’s that hold for $R$:
  
  | FD1:   | A→B, B→A, B→C, C→B |
  | FD2:   | A→B, B→C, C→A       |
Given Versus Implied FD’s

- Typically, we state a few FD’s that are known to hold for a relation R.
- Other FD’s may follow logically from the given FD’s; these are implied FD’s.
- Example: R(A,B,C) with FD’s: A → B, B → C, C → A
- A → C is implied FD
Finding All Implied FD’s

- **Motivation:** “normalization,” the process where we break a relation schema into two or more schemas.

- **Example:** $ABCD$ with FD’s $AB \rightarrow C$, $C \rightarrow D$, and $D \rightarrow A$.
  - Decompose into $ABC$, $AD$. What FD’s hold in $ABC$?
  - Not only $AB \rightarrow C$, but also $C \rightarrow A$!
Thus, tuples in the projection with equal C’s have equal A’s; $C \rightarrow A$. 
Basic Idea for projecting functional dependencies

1. Start with given FD’s and find all nontrivial FD’s that follow from the given FD’s.
   - Nontrivial = right side not contained in the left.

2. Restrict to those FD’s that involve only attributes of the projected schema.
Simple, Exponential Algorithm

1. For each set of attributes $X$, compute $X^+$.  
2. Add $X \rightarrow A$ for all $A$ in $X^+ - X$.  
3. However, drop $XY \rightarrow A$ whenever we discover $X \rightarrow A$.  
   Because $XY \rightarrow A$ follows from $X \rightarrow A$ in any projection.  
4. Finally, use only FD’s involving projected attributes.
A Few Tricks

- No need to compute the closure of the empty set or of the set of all attributes.
- If we find $X^+ = \text{all attributes}$, so is the closure of any superset of $X$. 
Example: Projecting FD’s

- $ABC$ with FD’s $A \rightarrow B$ and $B \rightarrow C$. Project onto $AC$.
  - $A^+=ABC$; yields $A \rightarrow B, A \rightarrow C$.
    - We do not need to compute $AB^+$ or $AC^+$.
  - $B^+=BC$; yields $B \rightarrow C$.
  - $C^+=C$; yields nothing.
  - $BC^+=BC$; yields nothing.
Example -- Continued

- Resulting FD’s: \( A \rightarrow B, A \rightarrow C, \) and \( B \rightarrow C. \)
- Projection onto \( AC: A \rightarrow C. \)
  - Only FD that involves a subset of \( \{A, C\} \).
Relational Schema Design

- Goal of relational schema design is to avoid anomalies and redundancy.
  - *Update anomaly*: one occurrence of a fact is changed, but not all occurrences.
  - *Deletion anomaly*: valid fact is lost when a tuple is deleted.
Example of Bad Design

**Drinkers**(*name, addr, beersLiked, manf, favBeer*)

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Bud</td>
<td>A.B.</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Janeway</td>
<td>???</td>
<td>WickedAle</td>
<td>Pete’s</td>
<td>???</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Bud</td>
<td>???</td>
<td>Bud</td>
</tr>
</tbody>
</table>

Data is redundant, because each of the ‘???’s can be figured out by using the FD’s *name -> addr favBeer* and *beersLiked -> manf*. 
This Bad Design Also Exhibits Anomalies

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Bud</td>
<td>A.B.</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>WickedAle</td>
<td>Pete’s A.B.</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Bud</td>
<td>A.B.</td>
<td>Bud</td>
</tr>
</tbody>
</table>

- **Update anomaly**: if Janeway is transferred to *Intrepid*, will we remember to change each of her tuples?
- **Deletion anomaly**: If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.
Solve the problem

Analysis result: Problems caused by FD’s

Drinkers(name, addr, beersLiked, manf, favBeer) →
decompose into smaller relations:

Drinker = projection (name, addr, favBeer) (Drinkers)
Likes = projection (name, beersLiked) (Drinkers)
Beer = projection (beersliked, manf) (Drinkers)

Drinkers = Drinker join Likes join beer not more, not less
Solve the problem (cont.)

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Bud</td>
<td>A.B.</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>WickedAle</td>
<td>Pete's A.B.</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Bud</td>
<td>A.B.</td>
<td>Bud</td>
</tr>
</tbody>
</table>

Any anomalies?

<table>
<thead>
<tr>
<th>Janeway</th>
<th>Bud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Spock</td>
<td>Bud</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bud</th>
<th>A.B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WickedAle</td>
<td>Peter's</td>
</tr>
</tbody>
</table>
Remember our questions:

- Why do we design relations like the example? -- good design
- What makes a good relational database schema? -- no redundancy, no Update/delete anomalies,
- what we can do if it has flaws? -- decomposition

New Question:
- any standards for a good design?
  → Normal forms: a condition on a relation schema that will eliminate problems
- any standards or methods for a decomposition?
We say a relation $R$ is in BCNF if whenever $X \rightarrow Y$ is a nontrivial FD that holds in $R$, $X$ is a superkey.

- Remember: *nontrivial* means $Y$ is not contained in $X$.
- Remember, a *superkey* is any superset of a key (not necessarily a proper superset).
Example

Drinkers(name, addr, beersLiked, manf, favBeer)
FD’s: name->addr favBeer, beersLiked->manf

- Only key is \{name, beersLiked\}.
- In each FD, the left side is \textit{not} a superkey.
- Any one of these FD’s shows \textit{Drinkers} is not in BCNF.
Another Example

Beers(name, manf, manfAddr)

FD’s: name->manf,  manf->manfAddr

- Only key is \{name\}.
- name->manf does not violate BCNF, but manf->manfAddr does.
Decomposition into BCNF

- Given: relation $R$ with FD’s $F$.
- Look among the given FD’s for a BCNF violation $X \rightarrow Y$.
  - If any FD following from $F$ violates BCNF, then there will surely be an FD in $F$ itself that violates BCNF.
- Compute $X^+$.
  - Not all attributes, or else $X$ is a superkey.
Decompose $R$ Using $X \rightarrow Y$

- Replace $R$ by relations with schemas:
  1. $R_1 = X^+$. 
  2. $R_2 = R - (X^+ - X)$.

- Project given FD’s $F$ onto the two new relations.
Decomposition Picture

\[ R - X^+ + X + - X \]

\[ R - X^+ \]

\[ X \]

\[ X^+ - X \]

\[ R_1 \]

\[ R_2 \]

\[ R \]
Example: BCNF Decomposition

Drinkers(name, addr, beersLiked, manf, favBeer)

\[ F = \text{name->addr, name -> favBeer, beersLiked->manf} \]

- Pick BCNF violation name->addr.
- Close the left side: \( \{\text{name}\}^+ = \{\text{name, addr, favBeer}\} \).
- Decomposed relations:
  1. Drinkers1(name, addr, favBeer)
  2. Drinkers2(name, beersLiked, manf)
We are not done; we need to check Drinkers1 and Drinkers2 for BCNF.

Projecting FD’s is easy here.

For Drinkers1(name, addr, favBeer), relevant FD’s are name->addr and name->favBeer.

- Thus, {name} is the only key and Drinkers1 is in BCNF.
For Drinkers2(name, beersLiked, manf), the only FD is beersLiked -> manf, and the only key is \{name, beersLiked\}.

- Violation of BCNF.

beersLiked\(^+\) = \{beersLiked, manf\}, so we decompose Drinkers2 into:

1. Drinkers3(beersLiked, manf)
2. Drinkers4(name, beersLiked)
Example -- Concluded

- The resulting decomposition of *Drinkers*:
  1. Drinkers1(*name*, *addr*, *favBeer*)
  2. Drinkers3(*beersLiked*, *manf*)
  3. Drinkers4(*name*, *beersLiked*)

- Notice: *Drinkers1* tells us about drinkers, *Drinkers3* tells us about beers, and *Drinkers4* tells us the relationship between drinkers and the beers they like.
Classroom Exercise

- Any two-attribute relation $R(A,B)$ is in BCNF

Several Cases:
1) No nontrivial FD’s
2) $A \rightarrow B$
3) $A \rightarrow B, B \rightarrow A$
Third Normal Form -- Motivation

- There is one structure of FD’s that causes trouble when we decompose.
- \( AB \rightarrow C \) and \( C \rightarrow B \).
  - Example: \( A = \) street address, \( B = \) city, \( C = \) zip code.
- There are two keys, \{A,B\} and \{A,C\}.
- \( C \rightarrow B \) is a BCNF violation, so we must decompose into \( AC, BC \).
We Cannot Enforce FD’s

- The problem is that if we use $AC$ and $BC$ as our database schema, we cannot enforce the FD $AB \rightarrow C$ by checking FD’s in these decomposed relations.
- Example with $A = $ street, $B = $ city, and $C = $ zip on the next slide.
An Unenforceable FD

Join tuples with equal zip codes.

Although no FD’s were violated in the decomposed relations, FD street city -> zip is violated by the database as a whole.
3rd Normal Form (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation.

An attribute is *prime* if it is a member of any key.

\[ X \rightarrow A \] violates 3NF if and only if \( X \) is not a superkey, and also \( A \) is not prime.
Example: 3NF

- In our problem situation with FD’s $AB \rightarrow C$ and $C \rightarrow B$, we have keys $AB$ and $AC$.
- Thus $A$, $B$, and $C$ are each prime.
- Although $C \rightarrow B$ violates BCNF, it does not violate 3NF.
## BCNF vs. 3NF

<table>
<thead>
<tr>
<th>conditions</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BCNF</strong></td>
<td>If $X \rightarrow Y$ is a nontrivial FD that holds in $R$, $X$ is a superkey.</td>
</tr>
<tr>
<td><strong>3NF</strong></td>
<td>If $X \rightarrow Y$ is a nontrivial FD that holds in $R$, $X$ is a superkey, or $Y$ is a prime</td>
</tr>
</tbody>
</table>

2 NF: no nonkey attribute is dependent on only a portion of the primary key.

1 NF: every component of every tuple is an atomic value.
Elimination of anomalies by a decomposition, it needs other two properties:

1. *Lossless Join*: it should be possible to project the original relations onto the decomposed schema, and then reconstruct the original.

2. *Dependency Preservation*: it should be possible to check in the projected relations whether all the given FD’s are satisfied.
What 3NF and BCNF Give You

- We can get (1: *Lossless Join*) with a BCNF decomposition.
- We can get both (1) and (2: *Dependency Preservation*) with a 3NF decomposition.
- But we can’t always get (1) and (2) with a BCNF decomposition.
  - street-city-zip is an example.
Testing for a Lossless Join

- If we project \( R \) onto \( R_1, R_2, \ldots, R_k \), can we recover \( R \) by rejoining?
- Any tuple in \( R \) can be recovered from its projected fragments.
- So the only question is: when we rejoin, do we ever get back something we didn’t have originally?
Example

- Any tuple in $R$ can be recovered from its projected fragments.
- Not less, not more.

As long as FD $b \rightarrow c$ holds, the joining of two projected tuples cannot produce a bogus tuple.
The Chase Test:

To prove that a tuple \( t \) in the join, using FD’s in \( F \), also to be a tuple in \( R \).

- Suppose tuple \( t \) comes back in the join.
- Then \( t \) is the join of projections of some tuples of \( R \), one for each \( R_i \) of the decomposition.
- Can we use the given FD’s to show that one of these tuples must be \( t \)?

Decomposed into:

\[
R(A,B,C) = R_1(A,B) \times R_2(B,C)
\]

Belong to:

Because \( B \rightarrow C \), \( c_1 = c \), therefore \( a,b,c \) is in \( R \)
The Chase – (cont.)

- Start by assuming $t = abc...$.
- For each $i$, there is a tuple $s_i$ of $R$ that has $a$, $b$, $c$, ... in the attributes of $R_i$.
- $s_i$ can have any values in other attributes.
- We’ll use the same letter as in $t$, but with a subscript, for these components.
Example: The Chase

- Let $R = ABCD$, and the decomposition be $AB$, $BC$, and $CD$.
- Let the given FD’s be $C \rightarrow D$ and $B \rightarrow A$.
- Suppose the tuple $t = abcd$ is the join of tuples projected onto $AB$, $BC$, $CD$. 
The tuples of $R$ projected onto $AB$, $BC$, $CD$.

The **Tableau**

$$
\begin{array}{cccc}
A & B & C & D \\
a & b & c_1 & d_1 \\
a_2 & a & b & c_1 \\
a_3 & b_3 & c & d_2 \\
\end{array}
$$

Use $B \rightarrow A$

We’ve proved the second tuple must be $t$.

Use $C \rightarrow D$
Summary of the Chase

1. If two rows agree in the left side of a FD, make their right sides agree too.
2. Always replace a subscripted symbol by the corresponding unsubscripted one, if possible.
3. If we ever get an unsubscripted row, we know any tuple in the project-join is in the original (the join is lossless).
4. Otherwise, the final tableau is a counterexample.
Example: Lossy Join (more tuples)

- Same relation $R = ABCD$ and same decomposition: $AB$, $BC$, and $CD$.

- But with only the FD $C \rightarrow D$. 
These three tuples are an example $R$ that shows the join lossy. $abcd$ is not in $R$. but we can project and rejoin to get $abcd$. More tuples
Some results (until now)

- Some decompositions can not keep lossless join (lossy join).
- Use chase method to find out whether the decomposition is lossy join.
- BCNF decomposition is lossless join, sometimes it can not keep functional dependencies.
- Relations with 3NF keep lossless join and also functional dependencies.
- How to decompose relations to reach 3NF?
3NF Synthesis Algorithm

- We can always construct a decomposition into 3NF relations with a lossless join and dependency preservation.

- Need *minimal basis* for the FD’s:
  1. Right sides are single attributes.
  2. No FD can be removed.
  3. No attribute can be removed from a left side.
Constructing a Minimal Basis

1. Split right sides.
2. Repeatedly try to remove an FD and see if the remaining FD’s are equivalent to the original.
3. Repeatedly try to remove an attribute from a left side and see if the resulting FD’s are equivalent to the original.
3NF Synthesis – method

1. Find a minimal basis for F
2. One relation for each FD in the minimal basis.
   1. Schema is the union of the left and right sides.
   2. X→A then (XA) is a schema.
3. If no key is contained in an FD, then add one relation whose schema is some key.

Algorithm 3.26 is on pp.103
Example: 3NF Synthesis

- Relation $R = ABCD$.
- FD’s $A \rightarrow B$ and $A \rightarrow C$. Key is AD

**Decomposition**: AB and AC from the FD’s, plus AD for a key.
Another example

- Relation $R(A,B,C,D,E)$
- FD’s $AB \rightarrow C$, $C \rightarrow B, A \rightarrow D$  key?

3NF synthesis:
1) A minimal basis
2) $R_1(ABC)$ $R_2(CB)$ $R_3(AD)$
3) $R_2$ is a part of $R_1$, delete $R_2$
4) No key is in $R_1$, $R_3$, add a key $R_4(ABE)$
Why It Works

- **Preserves dependencies**: each FD from a minimal basis is contained in a relation, thus preserved.
- **Lossless Join**: use the chase to show that the row for the relation that contains a key can be made all-unsubscripted variables.
- **3NF**: hard part – a property of minimal bases.

Question:
Why we say “BCNF decomposition”, “3NF synthesis”??
A New Form of Redundancy

- A relation is trying to represent more than one many-many relationship.

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>phones</th>
<th>beersLiked</th>
</tr>
</thead>
<tbody>
<tr>
<td>sue</td>
<td>a</td>
<td>p1</td>
<td>b1</td>
</tr>
<tr>
<td>sue</td>
<td>a</td>
<td>p2</td>
<td>b2</td>
</tr>
<tr>
<td>sue</td>
<td>a</td>
<td>p2</td>
<td>b1</td>
</tr>
<tr>
<td>sue</td>
<td>a</td>
<td>p1</td>
<td>b2</td>
</tr>
</tbody>
</table>

Then these tuples must also be in the relation.

- A drinker’s phones are independent of the beers they like.
- Thus, each of a drinker’s phones appears with each of the beers they like in all combinations.
Definition of MVD

- A *multivalued dependency* (MVD) on $R$, $X \rightarrow\rightarrow Y$, says that if two tuples of $R$ agree on all the attributes of $X$, then their components in $Y$ may be swapped, and the result will be two tuples that are also in the relation.

- i.e., for each value of $X$, the values of $Y$ are independent of the values of $R$-$X$-$Y$. 
Example: MVD

**Drinkers(name, addr, phones, beersLiked)**

- A drinker’s phones are independent of the beers they like.
  - `name->phones` and `name->beersLiked`.
- Thus, each of a drinker’s phones appears with each of the beers they like in all combinations.
- This repetition is unlike FD redundancy.
  - `name->addr` is the only FD.
Tuples Implied by \textit{name}-->\textit{->}phones

If we have tuples:

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>phones</th>
<th>beersLiked</th>
</tr>
</thead>
<tbody>
<tr>
<td>sue</td>
<td>a</td>
<td>p1</td>
<td>b1</td>
</tr>
<tr>
<td>sue</td>
<td>a</td>
<td>p2</td>
<td>b2</td>
</tr>
<tr>
<td>sue</td>
<td>a</td>
<td>p2</td>
<td>b1</td>
</tr>
<tr>
<td>sue</td>
<td>a</td>
<td>p1</td>
<td>b2</td>
</tr>
</tbody>
</table>

Then these tuples must also be in the relation.
Picture of MVD $X \rightarrow\rightarrow Y$

\[
\begin{array}{ccc}
X & Y & \text{others} \\
\text{equal} & \text{exchange} & \\
\end{array}
\]
MVD Rules

- Every FD is an MVD (*promotion*).
  - If $X \rightarrow Y$, then swapping $Y$’s between two tuples that agree on $X$ doesn’t change the tuples.
  - Therefore, the “new” tuples are surely in the relation, and we know $X \rightarrow\rightarrow Y$.

- **Complementation**: If $X \rightarrow\rightarrow Y$, and $Z$ is all the other attributes, then $X \rightarrow\rightarrow Z$. 
Splitting Doesn’t Hold

- Like FD’s, we cannot generally split the left side of an MVD.
- But unlike FD’s, we cannot split the right side either --- sometimes you have to leave several attributes on the right side.
Drinkers(name, areaCode, phone, beersLiked, manf)

- A drinker can have several phones, with the number divided between areaCode and phone (last 7 digits).
- A drinker can like several beers, each with its own manufacturer.
Example Continued

- Since the areaCode-phone combinations for a drinker are independent of the beersLiked-manf combinations, we expect that the following MVD’s hold:

  name $\rightarrow\rightarrow$ areaCode phone
  name $\rightarrow\rightarrow$ beersLiked manf
Example Data

Here is possible data satisfying these MVD’s:

<table>
<thead>
<tr>
<th>name</th>
<th>areaCode</th>
<th>phone</th>
<th>beersLiked</th>
<th>manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>650</td>
<td>555-1111</td>
<td>Bud</td>
<td>A.B.</td>
</tr>
<tr>
<td>Sue</td>
<td>650</td>
<td>555-1111</td>
<td>WickedAle</td>
<td>Pete’s</td>
</tr>
<tr>
<td>Sue</td>
<td>415</td>
<td>555-9999</td>
<td>Bud</td>
<td>A.B.</td>
</tr>
<tr>
<td>Sue</td>
<td>415</td>
<td>555-9999</td>
<td>WickedAle</td>
<td>Pete’s</td>
</tr>
</tbody>
</table>

But we cannot swap area codes or phones by themselves. That is, neither name->areaCode nor name->phone holds for this relation.
Fourth Normal Form

- The redundancy that comes from MVD’s is not removable by putting the database schema in BCNF.

- There is a stronger normal form, called 4NF, that (intuitively) treats MVD’s as FD’s when it comes to decomposition, but not when determining keys of the relation.
4NF Definition

- A relation $R$ is in 4NF if: whenever $X \rightarrow\rightarrow Y$ is a nontrivial MVD, then $X$ is a superkey.
  - *Nontrivial MVD* means that:
    1. $Y$ is not a subset of $X$, and
    2. $X$ and $Y$ are not, together, all the attributes.
  - Note that the definition of “superkey” still depends on FD’s only.
BCNF Versus 4NF

- Remember that every FD $X \rightarrow Y$ is also an MVD, $X \rightarrow\rightarrow Y$.
- Thus, if $R$ is in 4NF, it is certainly in BCNF. Because any BCNF violation is a 4NF violation (after conversion to an MVD).
- But $R$ could be in BCNF and not 4NF, because MVD’s are “invisible” to BCNF.
Decomposition and 4NF

- If $X \rightarrow\rightarrow Y$ is a 4NF violation for relation $R$, we can decompose $R$ using the same technique as for BCNF.
  1. $XY$ is one of the decomposed relations.
  2. All but $Y - X$ is the other.
Example: 4NF Decomposition

Drinkers(name, addr, phones, beersLiked)
FD: name -> addr
MVD’s: name --> phones
      name --> beersLiked

- Key is \{name, phones, beersLiked\}.
- All dependencies violate 4NF.
Example Continued

- Decompose using `name -> addr`:
  1. `Drinkers1(name, addr)`
     - In 4NF; only dependency is `name -> addr`.
  2. `Drinkers2(name, phones, beersLiked)`
     - Not in 4NF. MVD’s `name ->> phones` and `name ->> beersLiked` apply. No FD’s, so all three attributes form the key.
Example: Decompose Drinkers2

- Either MVD name $\rightarrow$ phones or name $\rightarrow$ beersLiked tells us to decompose to:
  - Drinkers3(name, phones)
  - Drinkers4(name, beersLiked)
Reasoning About MVD’s + FD’s

- **Problem**: given a set of MVD’s and/or FD’s that hold for a relation $R$, does a certain FD or MVD also hold in $R$?

- **Solution**: Use a tableau to explore all inferences from the given set, to see if you can prove the target dependency.
Why Do We Care?

1. 4NF technically requires an MVD violation.
   - Need to infer MVD’s from given FD’s and MVD’s that may not be violations themselves.

2. When we decompose, we need to project FD’s + MVD’s.
Example: Chasing a Tableau With MVD’s and FD’s

- To apply a FD, equate symbols, as before.
- To apply an MVD, generate one or both of the tuples we know must also be in the relation represented by the tableau.
- We’ll prove: if $A \rightarrow BC$ and $D \rightarrow C$, then $A \rightarrow C$. 
The Tableau for A→C

Goal: prove that $c_1 = c_2$.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b₀</td>
<td>c₁</td>
<td>d₁</td>
</tr>
<tr>
<td>a</td>
<td>b₁</td>
<td>c₂</td>
<td>d₂</td>
</tr>
<tr>
<td>a</td>
<td>b₂</td>
<td>c₂</td>
<td>d₁</td>
</tr>
</tbody>
</table>

Use $A\rightarrow\rightarrow BC$ (first row’s $D$ with second row’s $BC$).

Use $D\rightarrow C$ (first and third row agree on $D$, therefore agree on $C$).
Example: Transitive Law for MVD’s

- If $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow C$, then $A \rightarrow\rightarrow C$.
  - Obvious from the complementation rule if the Schema is $ABC$.
  - But it holds no matter what the schema; we’ll assume $ABCD$. 
The Tableau for A-->->C

**Goal:** derive tuple \((a, b_1, c_2, d_1)\).

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>b_1</td>
<td>c_1</td>
<td>d_1</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b_2</td>
<td>c_2</td>
<td>d_2</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b_2</td>
<td>c_2</td>
<td>d_2</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b_1</td>
<td>c_2</td>
<td>d_2</td>
</tr>
</tbody>
</table>

Use **A-->->B** to swap \(B\) from the first row into the second.

Use **B-->->C** to swap \(C\) from the third row into the first.
Rules for Inferring MVD’s + FD’s

- Start with a tableau of two rows.
  - These rows agree on the attributes of the left side of the dependency to be inferred.
  - And they disagree on all other attributes.
  - Use unsubscripted variables where they agree, subscripts where they disagree.
Inference: Applying a FD

- Apply a FD $X \rightarrow Y$ by finding rows that agree on all attributes of $X$. Force the rows to agree on all attributes of $Y$.
  - Replace one variable by the other.
  - If the replaced variable is part of the goal tuple, replace it there too.
Inference: Applying a MVD

- Apply a MVD $X \rightarrow \rightarrow Y$ by finding two rows that agree in $X$.
  - Add to the tableau one or both rows that are formed by swapping the $Y$-components of these two rows.
Inference: Goals

- To test whether $U \rightarrow V$ holds, we succeed by inferring that the two variables in each column of $V$ are actually the same.
- If we are testing $U \rightarrow \rightarrow V$, we succeed if we infer in the tableau a row that is the original two rows with the components of $V$ swapped.
Inference: Endgame

- Apply all the given FD’s and MVD’s until we cannot change the tableau.
- If we meet the goal, then the dependency is inferred.
- If not, then the final tableau is a counterexample relation.
  - Satisfies all given dependencies.
  - Original two rows violate target dependency.
# Relationships Among Normal Forms

<table>
<thead>
<tr>
<th>Property</th>
<th>3NF</th>
<th>BCNF</th>
<th>4NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliminates Redundancy due to FDs</td>
<td>Most</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Eliminates redundancy Due to MVDs</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Preserves FDs</td>
<td>Yes</td>
<td>maybe</td>
<td>maybe</td>
</tr>
<tr>
<td>Preserves MVD</td>
<td>maybe</td>
<td>maybe</td>
<td>maybe</td>
</tr>
<tr>
<td>Equal to original relation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Summary of the chapter

- Functional dependencies
- Keys of a relation
- Minimal basis for a set of FD’s
- BCNF and 3NF
- BCNF decomposition and 3NF synthesis (with lossless-join and dependency-preserving)
- Multivalued dependencies
- 4NF
- Reasoning about MVD and FD
Classroom exercise 3.7.1

Use the chase test to tell whether each of the following dependencies hold in a relation R(A,B,C,D,E) with the dependencies A→→BC, B→D, and C→→E

a) A→D

b) A→→E
Classroom Exercise 3.3.1 a) c) and 3.5.1

R(A,B,C,D) with
a) FD’s AB→C, C→D, D→A

c) AB→C, BC→D, CD→A, AD→B

1. Indicate all the BCNF violations.
2. Decompose the relations, as necessary into collections of relations that are in BCNF
3. Indicate all the 3NF violations
4. Decompose the relations into 3NF.
Solution (a)

1) C->A, C->D, D->A, AC->D, CD->A
2) Key are AB, BC, and BD
BCNF: R1(AC), R2(BC), R3(CD)
Or R1(CD), R2(BC), R3(AD) or…
3) No 3NF violations and why?
4) R(A,B,C,D) is already in 3NF
Solution c

i) indicate all the BCNF violations
- Consider the closures of all 15 nonempty subsets of \{A,B,C,D\}.
- \(A+=A\), \(B+=B\), \(C+=C\), and \(D+=D\). Thus we get no new FD’s.
- \(AB+=BC+=CD+=AD+=ABCD\), \(AC+=AC\), and \(BD+=BD\). Thus we get new nontrivial FD’s: \(AB \rightarrow D\), \(BC \rightarrow A\), \(CD \rightarrow B\), \(AD \rightarrow C\).
- \(ABC+=ABD+=ACD+=BCD+=ABCD\). Thus we get new nontrivial FD’s: \(ABC \rightarrow D\), \(ABD \rightarrow C\), \(ACD \rightarrow B\), \(BCE \rightarrow A\).
- \(ABCD+=ABCD\), so we get no new FD’s.
- To sum up, we can deduce 8 new nontrivial FD’s from the given 4 FD’s.
- From above, we find that \(AB\), \(BC\), \(CD\) and \(AD\) are keys, and that all the nontrivial FD’s for \(R\) contain a key on the left side. Thus there’re no BCNF violations.

ii) \(R(A,B,C,D)\) is already in BCNF.

iii) No 3NF violations.

iv) \(R(A,B,C,D)\) is already in 3NF.
Homework

- Exercise 3.2.1
- Exercise 3.5.2
- Exercise 3.6.3 a), c)

作业批改助教（TA）：章建
联系地址(contact)：zj26@yahoo.cn
Exercise 3.2.1
Consider a relation with schema R(A,B,C,D) and FD’s AB→C, C→D and D→A

a) What are all the nontrieval FD’s that follow from the given FD’s?
b) What are all the keys of R?
c) What are all the superkeys for R that are not keys?
Solution
Exercise 3.6.3 a), c)

a) R(A, B, C, D) with MVD’s A →→ B, A →→ C

c) A relation R(A, B, C, D) with MVD AB →→ C, and FD B → D

- Find all the 4NF violations
- Decompose the relations into a collection of relation schemas in 4NF
Solution (a)
Solution (c)