Design Theory and Normalization

What will we learn?

- What goes wrong if we have redundant info in a database?
- Why and how should you refine a schema?
- Functional Dependencies – a new kind of integrity constraints (IC)
- Normal Forms
- How to obtain those normal forms

Reading Material

- Database normalization
  - [RG] Chapter 19.1 to 19.5, 19.6.1, 19.8 (overview)
  - [GUW] Chapter 3

Example

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

- key = SSN
Why is redundancy bad?

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn ($)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

• key = SSN
• Suppose for a given rating, there is only one hourly_wage value
• Redundancy in the table
• Why is redundancy bad?

Example

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn ($)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

Why is redundancy bad?

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn ($)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

Nulls may or may not help

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn ($)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>
Decompositions should be used judiciously

1. Do we need to decompose a relation?
   - Several normal forms
   - If a relation is not in one of them, may need to decompose further

2. What are the problems with decomposition?
   - Lossless joins (soon)
   - Performance issues -- decomposition may both
     • help performance (for updates, some queries accessing part of data), or
     • hurt performance (new joins may be needed for some queries)

Functional Dependencies (FDs)

- A functional dependency (FD) \( X \rightarrow Y \) holds over relation \( R \) if, for every allowable instance \( r \) of \( R \):
  - i.e., given two tuples in \( r \), if the \( X \) values agree, then the \( Y \) values must also agree
  - \( X \) and \( Y \) are sets of attributes
  - \( t_1 \epsilon r, t_2 \epsilon r, \Pi_X(t_1) = \Pi_X(t_2) \) implies \( \Pi_Y(t_1) = \Pi_Y(t_2) \)

### Example

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
<th>rating</th>
<th>hourly-waged</th>
<th>hours-worked</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

### Lossless joins

- A join is lossless if the \( \Pi_X(t_1) = \Pi_X(t_2) \) implies \( t_1 = t_2 \)

### Normal Forms

- 1NF: Atomic values (e.g., \( \text{ssn}, \text{name}, \text{lot}, \text{rating} \))
- 2NF: \( X \rightarrow Y \), but we can check if it violates some FD, but we cannot tell if \( f \) holds over \( R \)
- 3NF: \( X \rightarrow Y \), but we cannot tell if \( f \) holds over \( R \)
- BCNF: \( X \rightarrow Y \), but we cannot tell if \( f \) holds over \( R \)
- 4NF: \( X \rightarrow Y \), but we cannot tell if \( f \) holds over \( R \)
- 5NF: \( X \rightarrow Y \), but we cannot tell if \( f \) holds over \( R \)

### Summary: Redundancy

- Solution?
  - decomposition of schema

### Functional Dependencies (FDs)

- An FD is a statement about all allowable relations
  - Must be identified based on semantics of application
  - Given some allowable instance \( r1 \) of \( R \), we can check if it violates some FD, but we cannot tell if \( f \) holds over \( R \)
- \( K \) is a candidate key for \( R \) means that \( K \rightarrow R \)
  - denoting \( R \) = all attributes of \( R \) too
  - However, \( S \rightarrow R \) does not require \( S \) to be minimal
  - e.g., \( S \) can be a superkey
Example

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (ssn, name, lot, rating, hourly_wage, hours_worked)

Notation: We will denote a relation schema by listing the attributes: SNLRWH
- Basically the set of attributes {S,N,L,R,W,H}
- here first letter of each attribute

FDs on Hourly_Emps:
- ssn is the key: S → SNLRWH
- rating determines hourly_wages: R → W

Armstrong’s Axioms

- X, Y, Z are sets of attributes

  - Reflexivity: If X ⊆ Y, then X → Y
  - Augmentation: If X → Y, then XZ → YZ for any Z
  - Transitivity: If X → Y and Y → Z, then X → Z

<table>
<thead>
<tr>
<th>X</th>
<th>Y</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>b1</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c3</td>
<td>d1</td>
</tr>
</tbody>
</table>

Apply these rules on AB → C and check

Armstrong’s Axioms

- X, Y, Z are sets of attributes

  - Reflexivity: If X ⊆ Y, then X → Y
  - Augmentation: If X → Y, then XZ → YZ for any Z
  - Transitivity: If X → Y and Y → Z, then X → Z

These are sound and complete inference rules for FDs
- sound: only generate FDs in F* for F
- complete: by repeated application of these rules, all FDs in F* will be generated

Additional Rules

- Follow from Armstrong’s Axioms

  - Union: If X → Y and X → Z, then X → YZ
  - Decomposition: If X → YZ, then X → Y and X → Z

<table>
<thead>
<tr>
<th>X</th>
<th>Y</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>b1</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c3</td>
<td>d1</td>
</tr>
</tbody>
</table>

A → B, A → C
A → BC
A → BC
A → B, A → C

Closure of a set of FDs

- Given some FDs, we can usually infer additional FDs:
  - SSN → DEPT, and DEPT → LOT implies SSN → LOT

- An FD f is implied by a set of FDs F if f holds whenever all FDs in F hold.

- F* = closure of F is the set of all FDs that are implied by F

To check if an FD belongs to a closure

- Computing the closure of a set of FDs can be expensive
  - Size of closure can be exponential in #attributes

- Typically, we just want to check if a given FD X → Y is in the closure of a set of FDs F

- No need to compute F*

1. Compute attribute closure of X (denoted X*) wrt F:
   - Set of all attributes A such that X → A is in F*

2. Check if Y is in X*
Computing Attribute Closure

Algorithm:
• closure = X
• Repeat until no change
  – if there is an FD U → V in F such that U ⊆ closure, then closure = closure ∪ V

• Does F = {A → B, B → C, C D → E} imply A → E?
  – i.e., is A → E in the closure F+? Equivalently, is E in A+?

FDs play a role in detecting redundancy

Example
• Consider a relation R with 3 attributes, ABC
  – No FDs hold: There is no redundancy here – no decomposition needed
  – Given A → B: Several tuples could have the same A value, and if so, they’ll all have the same B value – redundancy – decomposition may be needed if A is not a key

• Intuitive idea:
  – if there is any non-key dependency, e.g. A → B, decompose!

Boyce-Codd Normal Form (BCNF)

• Relation R with FDs F is in BCNF if, for all X → A in F
  – A ∈ X (called a trivial FD), or
  – X contains a key for R
  – i.e. X is a superkey

Next lecture: BCNF decomposition algorithm

Normal Forms

• Question: given a schema, how to decide whether any schema refinement is needed at all?
• If a relation is in a certain normal forms, it is known that certain kinds of problems are avoided/minimized
• Helps us decide whether decomposing the relation is something we want to do

FDs play a role in detecting redundancy

Example
• Consider a relation R with 3 attributes, ABC
  – No FDs hold: There is no redundancy here – no decomposition needed
  – Given A → B: Several tuples could have the same A value, and if so, they’ll all have the same B value – redundancy – decomposition may be needed if A is not a key

• Intuitive idea:
  – if there is any non-key dependency, e.g. A → B, decompose!

Boyce-Codd Normal Form (BCNF)

• Relation R with FDs F is in BCNF if, for all X → A in F
  – A ∈ X (called a trivial FD), or
  – X contains a key for R
  – i.e. X is a superkey

Next lecture: BCNF decomposition algorithm

Normal Forms

• Question: given a schema, how to decide whether any schema refinement is needed at all?
• If a relation is in a certain normal forms, it is known that certain kinds of problems are avoided/minimized
• Helps us decide whether decomposing the relation is something we want to do
Unnecessary decomposition

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ralph</td>
<td><a href="mailto:alphwiggum@gmail.com">alphwiggum@gmail.com</a></td>
<td>1987-01-01</td>
</tr>
<tr>
<td>Lisa</td>
<td><a href="mailto:lisasimpson@gmail.com">lisasimpson@gmail.com</a></td>
<td>1988-02-02</td>
</tr>
<tr>
<td>Bart</td>
<td><a href="mailto:bartjsimpson@gmail.com">bartjsimpson@gmail.com</a></td>
<td>1989-03-03</td>
</tr>
</tbody>
</table>

- Fine: join returns the original relation
- Unnecessary: no redundancy is removed; schema is more complicated (and uid is stored twice!)

CompSci 516:

Lossless join decomposition

- Decompose relation $R$ into relations $S$ and $T$
  - $\text{attrs}(R) = \text{attrs}(S) \cup \text{attrs}(T)$
  - $S = \pi_{\text{attrs}(S)}(R)$
  - $T = \pi_{\text{attrs}(T)}(R)$
- The decomposition is a lossless join decomposition if, given known constraints such as FD’s, we can guarantee that $R = S \bowtie T$

  - $R \subseteq S \bowtie T$ or $R \subseteq S \bowtie T$?
  - Any decomposition gives $R \subseteq S \bowtie T$ (why?)
    - A lossy decomposition is one with $R \not\subseteq S \bowtie T$

Loss? But I got more rows!

- “Loss” refers not to the loss of tuples, but to the loss of information
  - Or, the ability to distinguish different original relations

BCNF decomposition algorithm

- Find a BCNF violation
  - That is, a non-trivial FD $X \rightarrow Y$ in $R$ where $X$ is not a super key of $R$
- Decompose $R$ into $R_1$ and $R_2$, where
  - $R_1$ has attributes $X \cup Y$
  - $R_2$ has attributes $X \cup Z$, where $Z$ contains all attributes of $R$ that are in neither $X$ nor $Y$
- Repeat until all relations are in BCNF

- Also gives a lossless decomposition!

BCNF decomposition example - 1

- $\text{CSJDPQV}$, key $C$, $F = \{ J \rightarrow C, SD \rightarrow P, J \rightarrow S \}$
  - To deal with $SD \rightarrow P$, decompose into $\text{SDP}$, $\text{CSJDPQV}$.
  - To deal with $J \rightarrow S$, decompose $\text{CSJDPQV}$ into $\text{JS}$ and $\text{CJDPQV}$

- Is $JP \rightarrow C$ a violation of BCNF?

  - Note:
    - several dependencies may cause violation of BCNF
    - The order in which we pick them may lead to very different sets of relations
      - there may be multiple correct decompositions (can pick $J \rightarrow S$ first)
Recap

• Functional dependencies: a generalization of the key concept
• Non-key functional dependencies: a source of redundancy
• BCNF decomposition: a method for removing redundancies
  — BCNF decomposition is a lossless join decomposition
• BCNF: schema in this normal form has no redundancy due to FD’s

Multivalued dependencies

• A multivalued dependency (MVD) has the form $X \rightarrow Y$, where $X$ and $Y$ are sets of attributes in a relation $R$
• $X \rightarrow Y$ means that whenever two rows in $R$ agree on all the attributes of $X$, then we can swap their $Y$ components and get two rows that are also in $R$

BCNF decomposition example - 2

UserJoinsGroup (uid, uname, twitterid, gid, fromDate)

BCNF violation: uid → uname, twitterid

User (uid, uname, twitterid)

BCNF

Member (uid, gid, fromDate)

BCNF

BCNF decomposition example - 3

UserJoinsGroup (uid, uname, twitterid, gid, fromDate)

BCNF violation: twitterid → uid

Userid (twitterid, uid)

BCNF

UserJoinsGroup' (twitterid, uname, gid, fromDate)

BCNF

UserName (twitterid, uname)

BCNF

Member (twitterid, gid, fromDate)

BCNF

BCNF = no redundancy?

• User (uid, gid, place)
  — A user can belong to multiple groups
  — A user can register places she’s visited
  — Groups and places have nothing to do with other
  — FD’s?
    • None
    • BCNF?
      • Yes
      • Redundancies?
        • Tons!

User (uid, gid, place)

• uid → gid
• uid → place
  — Intuition: given uid, attributes gid and place are “independent”
• uid, gid → place
  — Trivial: LHS ∪ RHS = all attributes of $R$
• uid, gid → uid
  — Trivial: LHS ⊄ RHS

MVD examples
Complete MVD + FD rules

- FD reflexivity, augmentation, and transitivity
- MVD complementation: If $X \rightarrow Y$, then $X \rightarrow attr(R) - X - Y$
- MVD augmentation: If $X \rightarrow Y$ and $V \subseteq W$, then $XW \rightarrow YV$
- MVD transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z - Y$
- Replication (FD is MVD): If $X \rightarrow Y$, then $X \rightarrow Y$
- Coalescence: Try proving things using these?

If $X \rightarrow Y$ and $Z \subseteq Y$ and there is some $W$ disjoint from $Y$ such that $W \rightarrow Z$, then $X \rightarrow Z$

An elegant solution: “chase”

- Given a set of FD’s and MVD’s $D$, does another dependency $d$ (FD or MVD) follow from $D$?
- Procedure
  - Start with the premise of $d$, and treat them as “seed” tuples in a relation
  - Apply the given dependencies in $D$ repeatedly
    - If we apply an FD, we infer equality of two symbols
    - If we apply an MVD, we infer more tuples
  - If we infer the conclusion of $d$, we have a proof
  - Otherwise, if nothing more can be inferred, we have a counterexample

Proof by chase

- In $R(A, B, C, D)$, does $A \rightarrow B$ and $B \rightarrow C$ imply that $A \rightarrow C$?

<table>
<thead>
<tr>
<th>Have: $A \rightarrow B$</th>
<th>Need: $A \rightarrow C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow B$</td>
<td>$A \rightarrow C$</td>
</tr>
<tr>
<td>$a \rightarrow b$</td>
<td>$a \rightarrow b$</td>
</tr>
<tr>
<td>$b \rightarrow c_1$</td>
<td>$b \rightarrow c_1$</td>
</tr>
<tr>
<td>$b \rightarrow c_2$</td>
<td>$b \rightarrow c_2$</td>
</tr>
</tbody>
</table>

Another proof by chase

- In $R(A, B, C, D)$, does $A \rightarrow B$ and $B \rightarrow C$ imply that $A \rightarrow C$?

<table>
<thead>
<tr>
<th>Have: $A \rightarrow B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow B$</td>
</tr>
<tr>
<td>$a \rightarrow b$</td>
</tr>
<tr>
<td>$b \rightarrow c_1$</td>
</tr>
<tr>
<td>$b \rightarrow c_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Need: $A \rightarrow C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow C$</td>
</tr>
<tr>
<td>$a \rightarrow b$</td>
</tr>
<tr>
<td>$b \rightarrow c_1$</td>
</tr>
<tr>
<td>$b \rightarrow c_2$</td>
</tr>
</tbody>
</table>

Counterexample by chase

- In $R(A, B, C, D)$, does $A \rightarrow BC$ and $CD \rightarrow B$ imply that $A \rightarrow B$?

<table>
<thead>
<tr>
<th>Have: $A \rightarrow BC$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow BC$</td>
</tr>
<tr>
<td>$a \rightarrow b$</td>
</tr>
<tr>
<td>$b \rightarrow c_1$</td>
</tr>
<tr>
<td>$b \rightarrow c_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Need: $b_1 = b_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_1 = b_2$</td>
</tr>
</tbody>
</table>

4NF

- A relation $R$ is in Fourth Normal Form (4NF) if
  - For every non-trivial MVD $X \rightarrow Y$ in $R$, $X$ is a superkey
  - That is, all FD’s and MVD’s follow from “key→ other attributes” (i.e., no MVD’s and no FD’s besides key functional dependencies)

- 4NF is stronger than BCNF
  - Because every FD is also a MVD
4NF decomposition algorithm

- Find a 4NF violation
  - A non-trivial MVD $X \rightarrow Y$ in $R$ where $X$ is not a superkey
- Decompose $R$ into $R_1$ and $R_2$, where
  - $R_1$ has attributes $X \cup Y$
  - $R_2$ has attributes $X \cup Z$ (where $Z$ contains $R$ attributes not in $X$ or $Y$)
- Repeat until all relations are in 4NF
- Almost identical to BCNF decomposition algorithm
- Any decomposition on a 4NF violation is lossless

4NF decomposition example

<table>
<thead>
<tr>
<th>UID</th>
<th>Place</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Springfield</td>
<td>dps</td>
</tr>
<tr>
<td>142</td>
<td>Australia</td>
<td>dps</td>
</tr>
<tr>
<td>456</td>
<td>Springfield</td>
<td>abc</td>
</tr>
<tr>
<td>456</td>
<td>Morocco</td>
<td>abc</td>
</tr>
<tr>
<td>456</td>
<td>Morocco</td>
<td>gov</td>
</tr>
<tr>
<td>456</td>
<td>Morocco</td>
<td>go</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Member (uid, gid)**

**Visited (uid, place)**

4NF violation: $uid \rightarrow gid$

Other kinds of dependencies and normal forms

- Dependency preserving decompositions
- Join dependencies
- Inclusion dependencies
- 5NF, 3NF, 2NF
- See book if interested (not covered in class)

Summary

- Philosophy behind BCNF, 4NF:
  - Data should depend on the key, the whole key, and nothing but the key!
  - You could have multiple keys though
- Redundancy is not desired typically
  - not always, mainly due to performance reasons
- Functional/multivalued dependencies – capture redundancy
- Decompositions – eliminate dependencies
- Normal forms
  - Guarantees certain non-redundancy
  - BCNF, and 4NF
- Lossless join
- How to decompose into BCNF, 4NF
- Chase