Relational Database Design Theory
Introduction to Databases
CompSci 316 Fall 2018
Announcements (Tue. Sep. 11)

• Homework #1 due next Tuesday (11:59pm)
• Gradiance autograder to be deployed by tonight
  • Uses a (hidden) test db different from the sample
• An experimental RA debugger available for a subset of the problems
  • https://ratest.cs.duke.edu/
  • You are not required to use it, but the bonus is that
    • It uses the same (hidden) test db as the autograder
    • If your query is wrong, it will “explain” how with a very simple example db (with tuples drawn from the hidden test db)
• Course project description posted
  • Read it!
  • “Mixer” in a week and a half
  • Milestone #1 right after fall break
  • Teamwork required: 5 people per team on average
Motivation

<table>
<thead>
<tr>
<th>uid</th>
<th>uname</th>
<th>gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>dps</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>gov</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>abc</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>gov</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>abc</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>gov</td>
</tr>
</tbody>
</table>

• Why is UserGroup \((uid, uname, gid)\) a bad design?
  • It has redundancy—user name is recorded multiple times, once for each group that a user belongs to
    • Leads to update, insertion, deletion anomalies

• Wouldn’t it be nice to have a systematic approach to detecting and removing redundancy in designs?
  • Dependencies, decompositions, and normal forms
Functional dependencies

• A functional dependency (FD) 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 tuples in $R$ agree on all the attributes in $X$, they must also agree on all attributes in $Y$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>$Y$</td>
<td>$Z$</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>?</td>
</tr>
</tbody>
</table>

Must be $b$  
Could be anything
FD examples

Address (street_address, city, state, zip)

• street_address, city, state → zip
• zip → city, state
• zip, state → zip?
  • This is a trivial FD
  • Trivial FD: LHS ⊇ RHS
• zip → state, zip?
  • This is non-trivial, but not completely non-trivial
  • Completely non-trivial FD: LHS ∩ RHS = ∅
A set of attributes $K$ is a key for a relation $R$ if

- $K \rightarrow$ all (other) attributes of $R$
  - That is, $K$ is a "super key"
- No proper subset of $K$ satisfies the above condition
  - That is, $K$ is minimal
Reasoning with FD’s

Given a relation $R$ and a set of FD’s $\mathcal{F}$

• Does another FD follow from $\mathcal{F}$?
  • Are some of the FD’s in $\mathcal{F}$ redundant (i.e., they follow from the others)?

• Is $K$ a key of $R$?
  • What are all the keys of $R$?
Attribute closure

• Given $R$, a set of FD’s $\mathcal{F}$ that hold in $R$, and a set of attributes $Z$ in $R$:
  The closure of $Z$ (denoted $Z^+$) with respect to $\mathcal{F}$ is the set of all attributes $\{A_1, A_2, \ldots\}$ functionally determined by $Z$ (that is, $Z \rightarrow A_1A_2 \ldots$)

• Algorithm for computing the closure
  • Start with closure $= Z$
  • If $X \rightarrow Y$ is in $\mathcal{F}$ and $X$ is already in the closure, then also add $Y$ to the closure
  • Repeat until no new attributes can be added
A more complex example

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

Assume that there is a 1-1 correspondence between our users and Twitter accounts

• uid → uname, twitterid
• twitterid → uid
• uid, gid → fromDate

Not a good design, and we will see why shortly
Example of computing closure

• \( \{ \text{gid}, \text{twitterid} \}^+ = ? \)

• \( \text{twitterid} \rightarrow \text{uid} \)
  • Add \( \text{uid} \)
  • Closure grows to \( \{ \text{gid}, \text{twitterid}, \text{uid} \} \)

• \( \text{uid} \rightarrow \text{uname}, \text{twitterid} \)
  • Add \( \text{uname}, \text{twitterid} \)
  • Closure grows to \( \{ \text{gid}, \text{twitterid}, \text{uid}, \text{uname} \} \)

• \( \text{uid}, \text{gid} \rightarrow \text{fromDate} \)
  • Add \( \text{fromDate} \)
  • Closure is now all attributes in UserJoinsGroup

\( F \) includes:
  \( \text{uid} \rightarrow \text{uname}, \text{twitterid} \)
  \( \text{twitterid} \rightarrow \text{uid} \)
  \( \text{uid}, \text{gid} \rightarrow \text{fromDate} \)
Using attribute closure

Given a relation $R$ and set of FD’s $\mathcal{F}$

• Does another FD $X \rightarrow Y$ follow from $\mathcal{F}$?
  • Compute $X^+$ with respect to $\mathcal{F}$
  • If $Y \subseteq X^+$, then $X \rightarrow Y$ follows from $\mathcal{F}$

• Is $K$ a key of $R$?
  • Compute $K^+$ with respect to $\mathcal{F}$
  • If $K^+$ contains all the attributes of $R$, $K$ is a super key
  • Still need to verify that $K$ is minimal (how?)
Rules of FD’s

• Armstrong’s axioms
  • Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$
  • Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any $Z$
  • Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

• Rules derived from axioms
  • Splitting: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
  • Combining: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

Using these rules, you can prove or disprove an FD given a set of FDs
Non-key FD’s

• Consider a non-trivial FD $X \rightarrow Y$ where $X$ is not a super key
  • Since $X$ is not a super key, there are some attributes (say $Z$) that are not functionally determined by $X$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>$Y$</td>
<td>$Z$</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

That $b$ is associated with $a$ is recorded multiple times: 
**redundancy, update/insertion/deletion anomaly**
Example of redundancy

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

• uid $\rightarrow$ uname, twitterid

(... plus other FD’s)

<table>
<thead>
<tr>
<th>uid</th>
<th>uname</th>
<th>twitterid</th>
<th>gid</th>
<th>fromDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>@BartJSimpson</td>
<td>dps</td>
<td>1987-04-19</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>@MilhouseVan_</td>
<td>gov</td>
<td>1989-12-17</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>@lisasimpson</td>
<td>abc</td>
<td>1987-04-19</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>@lisasimpson</td>
<td>gov</td>
<td>1988-09-01</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>@ralphwiggum</td>
<td>abc</td>
<td>1991-04-25</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>@ralphwiggum</td>
<td>gov</td>
<td>1992-09-01</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

...
Decomposition

- Eliminates redundancy
- To get back to the original relation: Π

<table>
<thead>
<tr>
<th>uid</th>
<th>uname</th>
<th>twitterid</th>
<th>gid</th>
<th>fromDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>@BartJSimpson</td>
<td>dps</td>
<td>1987-04-19</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>@MilhouseVan_</td>
<td>gov</td>
<td>1989-12-17</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>@lisasimpson</td>
<td>abc</td>
<td>1987-04-19</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>@lisasimpson</td>
<td>gov</td>
<td>1988-09-01</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>@ralphwiggum</td>
<td>abc</td>
<td>1991-04-25</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>@ralphwiggum</td>
<td>gov</td>
<td>1992-09-01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
<th>fromDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>dps</td>
<td>1987-04-19</td>
</tr>
<tr>
<td>123</td>
<td>gov</td>
<td>1989-12-17</td>
</tr>
<tr>
<td>857</td>
<td>abc</td>
<td>1987-04-19</td>
</tr>
<tr>
<td>857</td>
<td>gov</td>
<td>1988-09-01</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>1991-04-25</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>1992-09-01</td>
</tr>
</tbody>
</table>

...
**Unnecessary decomposition**

<table>
<thead>
<tr>
<th>uid</th>
<th>uname</th>
<th>twitterid</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>@BartJSimpson</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>@MilhouseVan_</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>@lisasimpson</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>@ralphwiggum</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

- Association between gid and fromDate is lost
- Join returns more rows than the original relation
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 $R = S \bowtie T$

• Any decomposition gives $R \subseteq S \bowtie T$ (why?)
  • A lossy decomposition is one with $R \subset S \bowtie T$
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

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
<th>fromDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>dps</td>
<td>1987-04-19</td>
</tr>
<tr>
<td>123</td>
<td>gov</td>
<td>1989-12-17</td>
</tr>
<tr>
<td>857</td>
<td>abc</td>
<td>1988-09-01</td>
</tr>
<tr>
<td>857</td>
<td>gov</td>
<td>1987-04-19</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>1991-04-25</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>1992-09-01</td>
</tr>
</tbody>
</table>

No way to tell which is the original relation
Questions about decomposition

• When to decompose

• How to come up with a correct decomposition (i.e., lossless join decomposition)
An answer: BCNF

• A relation $R$ is in **Boyce-Codd Normal Form** if
  • For every non-trivial FD $X \rightarrow Y$ in $R$, $X$ is a super key
  • That is, all FDs follow from “key $\rightarrow$ other attributes”

• When to decompose
  • As long as some relation is not in BCNF

• How to come up with a correct decomposition
  • Always decompose on a BCNF violation (details next)
    ❁ Then it is guaranteed to be a lossless join decomposition!
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
BCNF decomposition example

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

BCNF violation: $uid \rightarrow uname, twitterid$

User (uid, uname, twitterid)

uid $\rightarrow$ uname, twitterid

twitterid $\rightarrow$ uid

BCNF

Member (uid, gid, fromDate)

uid, gid $\rightarrow$ fromDate

BCNF
Another example

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

BCNF violation: twitterid → uid

UserId (twitterid, uid)

UserJoinsGroup’ (twitterid, uname, gid, fromDate)

BCNF violation: twitterid → uname

twitterid → uname

twitterid, gid → fromDate

UserName (twitterid, uname)

Member (twitterid, gid, fromDate)

BCNF

BCNF

uid → uname, twitterid

twitterid → uid

uid, gid → fromDate
Why is BCNF decomposition lossless

Given non-trivial $X \rightarrow Y$ in $R$ where $X$ is not a super key of $R$, need to prove:

• Anything we project always comes back in the join: 
  
  $R \subseteq \pi_{XY}(R) \bowtie \pi_{XZ}(R)$
  
  • Sure; and it doesn’t depend on the FD

• Anything that comes back in the join must be in the original relation:
  
  $R \supseteq \pi_{XY}(R) \bowtie \pi_{XZ}(R)$
  
  • Proof will make use of the fact that $X \rightarrow Y$
Recap

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

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
<th>place</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>dps</td>
<td>Springfield</td>
</tr>
<tr>
<td>142</td>
<td>dps</td>
<td>Australia</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>Springfield</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>Morocco</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>Springfield</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>Morocco</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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 $R$
MVD examples

User \((uid, gid, place)\)

• \(uid \rightarrow gid\)

• \(uid \rightarrow place\)
  • Intuition: given \(uid\), \(gid\) and \(place\) are \text{“independent”}\)

• \(uid, gid \rightarrow place\)
  • Trivial: \(LHS \cup RHS = \text{all attributes of } R\)

• \(uid, gid \rightarrow uid\)
  • Trivial: \(LHS \supseteq RHS\)
Complete MVD + FD rules

• FD reflexivity, augmentation, and transitivity
• MVD complementation:
  If $X \rightarrow Y$, then $X \rightarrow \text{attrs}(R) - X - Y$
• 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 $X \rightarrow Y$
  
  Try proving things using these!?
• Coalescence:
  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 $\mathcal{D}$, does another dependency $d$ (FD or MVD) follow from $\mathcal{D}$?

• Procedure
  • Start with the premise of $d$, and treat them as “seed” tuples in a relation
  • Apply the given dependencies in $\mathcal{D}$ 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:</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$d_1$</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Need:</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>
| $\leftarrow$ | a | $b_1$ | $c_2$ | $d_1$ | ✅
| $\leftarrow$ | a | $b_2$ | $c_1$ | $d_2$ | ✅

$A \rightarrow B$

| a | $b_2$ | $c_1$ | $d_1$ |
| a | $b_2$ | $c_2$ | $d_2$ |

$B \rightarrow C$

| a | $b_2$ | $c_1$ | $d_2$ |
| a | $b_2$ | $c_2$ | $d_1$ |

$B \rightarrow C$

| a | $b_1$ | $c_2$ | $d_1$ |
| a | $b_1$ | $c_1$ | $d_2$ |
Another proof by chase

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$d_1$</td>
</tr>
<tr>
<td>$a$</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_2$</td>
</tr>
</tbody>
</table>

Have: $A \to B \quad b_1 = b_2$

$B \to C \quad c_1 = c_2$

Need: $c_1 = c_2$

In general, with both MVD’s and FD’s, chase can generate both new tuples and new equalities.
Counterexample by chase

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

Have:

<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>$b_1$</td>
<td>$c_1$</td>
<td>$d_1$</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_2$</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_1$</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$d_2$</td>
<td></td>
</tr>
</tbody>
</table>

Need:

$b_1 = b_2 \Downarrow$

Counterexample!
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 $\rightarrow$ 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

User (uid, gid, place)
4NF violation: \(uid \rightarrow gid\)

Member (uid, gid)
4NF

Visited (uid, place)
4NF

---

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
<th>place</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>dps</td>
<td>Springfield</td>
</tr>
<tr>
<td>142</td>
<td>dps</td>
<td>Australia</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>Springfield</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>Morocco</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>Springfield</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>Morocco</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td></td>
</tr>
<tr>
<td>456</td>
<td></td>
</tr>
<tr>
<td>456</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
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

• Other normal forms
  • 3NF: More relaxed than BCNF; will not remove redundancy if doing so makes FDs harder to enforce
  • 2NF: Slightly more relaxed than 3NF
  • 1NF: All column values must be atomic