Announcements (Thurs, 1/27)

- HW1 due next week 2/1 (Tues)
  - Please check out posts on Ed with updates and instructions
- Project proposal due next week 2/3 (Thurs)
  - 13 standard, 6 semi-standard, 2 open
  - consider semi-standard and open projects!
- Quiz-1 posted on Gradescope – due 2/8 (Tues)
  - Two problems only, autograded, submit as many times as you want
  - Will demonstrate during lectures
- More quizzes will be posted soon – instructions to be posted to use “Gradiance” for online quiz
  - Will help prepare for the exam - do them early!
- If there are in-class quiz/labs, will be announced in the previous class
- No late days for quiz (gradiance closes automatically)

Where are we now?

We learnt
- Relational Model and Query Languages
- SQL, RA, RC
- Postgres (DBMS)
- XML (overview)
- HW1

Next
- Database Normalization
  - (for good schema design)

Reading Material

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

Acknowledgement:
- The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.
- Some slides have been adapted from slides by Profs. Magda Balazinska, Dan Suciu, and Jun Yang

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

Example

The list of hourly employees in an organization

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

- key = SSN
Redundancy in the table
Suppose for a given rating, there is only one key = SSN

Deletion anomalies:
- The list of hourly employees in an organization

Update anomalies
- If one copy of data is updated, an inconsistency is created unless all copies are similarly updated
- Suppose you update the hourly_wage value in the first tuple using UPDATE statement in SQL -- inconsistency

Why is redundancy bad? 2/4

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn (S)</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>

2. Update anomalies
- If one copy of data is updated, an inconsistency is created unless all copies are similarly updated
- Suppose you update the hourly_wage value in the first tuple using UPDATE statement in SQL -- inconsistency

Why is redundancy bad? 3/4

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn (S)</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>

3. Insertion anomalies:
- It may not be possible to store certain information unless some other, unrelated info is stored as well
- We cannot insert a tuple for an employee unless we know the hourly wage for the employee's rating value

Why is redundancy bad? 4/4

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn (S)</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>

4. Deletion anomalies:
- It may not be possible to delete certain information without losing some other information as well
- If we delete all tuples with a given rating value (Attishoo, Smiley, Madayan), we lose the association between that rating value and its hourly_wage value

Nulls may or may not help

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn (S)</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>

- Does not help redundant storage or update anomalies
- May help insertion and deletion anomalies
- can insert a tuple with null value in the hourly_wage field
- but cannot record hourly_wage for a rating unless there is such an employee (SSN cannot be null) – same for deletion
Summary: Redundancy

Therefore,
- Redundancy arises when the schema forces an association between attributes that is “not natural”
- We want schemas that do not permit redundancy
  - at least identify schemas that allow redundancy to make an informed decision (e.g. for performance reasons)

Solution?

More Decomposition

Unnecessary decomposition

Bad decomposition

Lossless join decomposition

Join returns more rows than the original relation

Association between gid and fromDate is lost

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 \( R \cong S \bowtie T \)

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

Join returns more rows than the original relation
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>Key</th>
<th>Name</th>
<th>Ssn</th>
<th>Lot</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>abc</td>
<td>1987-09-19</td>
<td>01</td>
<td>19</td>
</tr>
<tr>
<td>132</td>
<td>def</td>
<td>1991-12-17</td>
<td>01</td>
<td>19</td>
</tr>
<tr>
<td>123</td>
<td>ghi</td>
<td>1992-09-09</td>
<td>01</td>
<td>19</td>
</tr>
<tr>
<td>114</td>
<td>jkl</td>
<td>1992-09-09</td>
<td>01</td>
<td>19</td>
</tr>
<tr>
<td>105</td>
<td>mno</td>
<td>1987-09-09</td>
<td>01</td>
<td>19</td>
</tr>
</tbody>
</table>

No way to tell which is the original relation

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
  - \( t_1 \in r, t_2 \in r, \Pi_X(t_1) = \Pi_X(t_2) \) implies \( \Pi_Y(t_1) = \Pi_Y(t_2) \)

What is a (possible) FD here?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a2</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>

Armstrong’s Axioms

- \( X, Y, Z \) are sets of attributes
  - Reflexivity: If \( X \supseteq Y \), then \( X \rightarrow Y \)
  - Augmentation: If \( X \rightarrow Y \), then \( XZ \rightarrowYZ \) for any \( Z \)
  - Transitivity: If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z \)

Example

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (ssn, name, lot, rating, hourly_wage, hours_worked)
- Use first letter of attributes for simplicity: \( \text{SNLRWH} \)
  - Basically the set of attributes \( \{S,N,L,R,W,H\} \)
  - FDs on Hourly_Emps:
    - \( \text{ssn is the key: } S \rightarrow \text{SNLRWH} \)
    - rating determines hourly_wages: \( R \rightarrow W \)
Additional Rules

• Follow from Armstrong’s Axioms
• Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
• Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Computing Attribute Closure

Algorithm:
• $\text{closure} = X$
• Repeat until no change
  – if there is an FD $U \rightarrow V$ in $F$ such that $U \subseteq \text{closure}$, then $\text{closure} = \text{closure} \cup V$
• Does $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D \rightarrow E\}$ imply $A \rightarrow E$?
  – i.e., is $A \rightarrow E$ in the closure $F^+$? Equivalently, is $E$ in $A^+$?

Computing FD Closure

• 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 a given FD $X \rightarrow Y$ is in the closure of a set of FDs $F$
  – No need to compute $F^+$
  – Compute attribute closure of $X$ (denoted $X^+$) wrt $F$
  – Check if $Y$ is in $X^+$

Announcements (Thurs, 1/27)

• HW1 due next week 2/1 (Tues)
  – Please check out posts on Ed with updates and instructions
• Project proposal due next week 2/3 (Thurs)
  – 13 standard, 6 semi-standard, 2 open
  – consider semi-standard and open projects!
• Quiz-1 posted on Gradescope – due 2/8 (Tues)
  – Two problems only, autograded, submit as many times as you want
  – Use Ratest https://ratest.cs.duke.edu/ratest/ to debug (last two problems)
  – Will demonstrate during lectures
• More quizzes will be posted soon – instructions to be posted to use “Gradiance” for online quiz
  – Will help prepare for the exam - do them early!
• If there are in-class quiz/labs, will be announced in the previous class
• No late days for quiz (gradiance closes automatically)

Detour - Ratest

• https://ratest.cs.duke.edu/ratest/
• Requires net-id
• Quiz problems (i) & (j)

Normal Forms

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

R is in 4NF
⇒ R is in BCNF
⇒ R is in 3NF
⇒ R is in 2NF (a historical one)
⇒ R is in 1NF (every field has atomic values)

Only BCNF and 4NF are covered in the class

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

Intuitive idea:

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 if there is any non-key dependency, e.g. A → B, decompose!

BCNF decomposition algorithm

- Find a BCNF violation
  - That is, a non-trivial FD X → Y in R where X is not a super key of R
- Decompose R into R₁ and R₂, where
  - R₁ has attributes X ∪ Y
  - R₂ has attributes X ∪ 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

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

BCNF violation: uid → uname, twitterid

User (uid, uname, twitterid)

Member (uid, gid, fromDate)

BCNF

BCNF decomposition example - 2

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

BCNF violation: twitterid → uid

UserName (twitterid, uname)

Check yourself!

BCNF decomposition example - 3

CSJDQPV, key C, F = {JP → C, SD → P, J → S}
- To deal with SD → P, decompose into SDP, CSJDQV
- To deal with J → S, decompose CSJDQV into JS and CIDQV

- Is JP → C a violation of BCNF?
  - No

- 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 → S first)
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>FD</th>
<th>gid</th>
<th>place</th>
</tr>
</thead>
<tbody>
<tr>
<td>dps</td>
<td>143</td>
<td>Springfield</td>
</tr>
<tr>
<td>dps</td>
<td>142</td>
<td>Australia</td>
</tr>
<tr>
<td>abc</td>
<td>456</td>
<td>Morocco</td>
</tr>
<tr>
<td>abc</td>
<td>456</td>
<td>Morocco</td>
</tr>
<tr>
<td>gov</td>
<td>456</td>
<td>Springfield</td>
</tr>
<tr>
<td>gov</td>
<td>456</td>
<td>Morocco</td>
</tr>
</tbody>
</table>

Procedure dependency

Given a set of FD’s and MVD’s

User

- 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

User (uid, gid, place)

- uid → gid
- uid → place
- uid → gid → place
  - Trivial: LHS ∪ RHS = all attributes of R
- uid, gid → uid
  - Trivial: LHS ⊇ RHS

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

Multivalued dependencies

- A multivalued dependency (MVD) has the form
  - X → Y, where X and Y are sets of attributes in a relation R

- X → 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

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

Complete MVD + FD rules

- FD reflexivity, augmentation, and transitivity
- MVD complementation:
  - If X → Y, then X → attrs(R) → X → Y
- MVD augmentation:
  - If X → Y and Y ⊆ W, then XW → YV
- MVD transitivity:
  - If X → Y and Y → Z, then X → Z → Y
- Replication (FD is MVD):
  - If X → Y, then X → Y
- Coalescence:
  - If X → Y and Y → Z and there is some W disjoint from Y such that W → Z, then X → Z

Proof by chase

- In R(A, B, C, D), does A → B and B → C imply that A → C?

<table>
<thead>
<tr>
<th>Have:</th>
<th>Need:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a b_1 c_1 d_1</td>
</tr>
<tr>
<td></td>
<td>a b_2 c_2 d_2</td>
</tr>
<tr>
<td>A → B</td>
<td>a b_1 c_1 d_1</td>
</tr>
<tr>
<td></td>
<td>a b_1 c_2 d_2</td>
</tr>
<tr>
<td>B → C</td>
<td>a b_2 c_1 d_2</td>
</tr>
<tr>
<td></td>
<td>a b_2 c_2 d_1</td>
</tr>
<tr>
<td>A → C</td>
<td>a b_1 c_1 d_2</td>
</tr>
<tr>
<td></td>
<td>a b_1 c_2 d_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>A → B</th>
<th>b₁ = b₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>B → C</td>
<td>c₁ = c₂</td>
</tr>
</tbody>
</table>

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 $A \rightarrow B$?

<table>
<thead>
<tr>
<th>A ↠ BC</th>
<th>b₁ = b₂</th>
</tr>
</thead>
</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>A → B</th>
<th>b₁ = b₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>B → C</td>
<td>c₁ = c₂</td>
</tr>
</tbody>
</table>

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

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 $X \cup Y$ (where $X$ contains $R$ attributes not in $X$ or $Y$)
    – $R_2$ 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>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>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>
</tbody>
</table>

User (uid, gid, place)

4NF violation: uid → gid

Member (uid, gid)

4NF

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>dps</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
</tr>
</tbody>
</table>

Visited (uid, place)

4NF

<table>
<thead>
<tr>
<th>uid</th>
<th>place</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Springfield</td>
</tr>
<tr>
<td>152</td>
<td>Australia</td>
</tr>
<tr>
<td>456</td>
<td>Morocco</td>
</tr>
</tbody>
</table>

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