Relational Database Design Theory
Part I

CPS 196.3
Introduction to Database Systems

Announcement

- Homework #1 assigned today
  - Due on Friday, September 12 in my office (D327)
- Extra handouts available in a handout box outside my office
- Reminder of the new schedule:
  12:50pm-2:05pm Mondays and Wednesdays

Motivation

- How do we tell if a design is bad, e.g.,
  StudentEnroll (SID, name, CID)?
- How about 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 $Y$.

```
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>
```

Must be $b$ Could be anything

FD examples

Address ($street\_address, city, state, zip$)

_____ redefined using FD’s
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 functionally determined by $Z$
- 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 more attributes can be added

A more complex example

$StudentGrade (SID, name, email, CID, grade)$

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

- $\mathcal{F}$ includes:

- $\{ \text{CID, email} \}^+ = ?$

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 $X \rightarrow Y$ follow 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 rules of FD’s

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

- Does another FD $X \rightarrow Y$ follow from $\mathcal{F}$?
  - Use the rules to come up with a proof
  - Example:
    - $\mathcal{F}$ includes:
      - $SID \rightarrow name, email, email \rightarrow SID, SID, CID \rightarrow grade$
      - $CID, email \rightarrow grade$
        - $email \rightarrow SID$ (given in $\mathcal{F}$)
      - $CID, email \rightarrow CID, SID$ (augmentation)
      - $SID, CID \rightarrow grade$ (given in $\mathcal{F}$)
      - $CID, email \rightarrow grade$ (transitivity)

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 $X$

Example of redundancy

- $StudentGrade (SID, name, email, CID, grade)$
- $SID \rightarrow name, email$

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>email</th>
<th>CID</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS114</td>
<td>B</td>
</tr>
<tr>
<td>142</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>CPS114</td>
<td>B</td>
</tr>
<tr>
<td>113</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>CPS114</td>
<td>B</td>
</tr>
<tr>
<td>112</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS114</td>
<td>B</td>
</tr>
<tr>
<td>114</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>CPS114</td>
<td>B</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:

Unnecessary decomposition

- Fine: join returns the original relation
- Unnecessary: no redundancy is removed, and now SID is stored twice!

Bad decomposition
Lossless join decomposition

- Decompose relation \( R \) into relations \( S \) and \( T \)
  - \( \text{attr}(R) = \text{attr}(S) \cup \text{attr}(T) \)
  - \( S = \pi_{\text{attr}(S)}(R) \)
  - \( T = \pi_{\text{attr}(T)}(R) \)
- The decomposition is a lossless join decomposition if, given constraints such as FD’s, we can guarantee that \( R = S \Join T \)
- Any decomposition has \( R \subseteq S \Join T \) (why?)
  - A lossy decomposition is one with \( R \subset S \Join T \)

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

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
  - 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

**StudentGrade (SID, name, email, CID, grade)**

BCNF violation: \( SID \rightarrow name, email \)
Another example

StudentGrade (SID, name, email, CID, grade)
BCNF violation:

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 makes 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