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 of $Y$
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:
  - \( \{ CID, email \}^+ = ? \)
Using attribute closure

Given a relation $R$ and set of FD's $F$

- Does another FD $X \rightarrow Y$ follow from $F$?
  - Compute $X^+$ with respect to $F$
  - If $Y \subseteq X^+$, then $X \rightarrow Y$ follow from $F$

- Is $K$ a key of $R$?
  - Compute $K^+$ with respect to $F$
  - 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 $F$

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

Decomposition

- Eliminates redundancy
- To get back to the original relation:
Unnecessary decomposition

![Table showing SID, name, email for students]

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

Bad decomposition

![Table showing SID, CID, grade for courses]

- Association between CID and grade is lost
- Join returns more rows than the original relation

Lossless join decomposition

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

- Any decomposition has \( R \subseteq S \bowtie T \) (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

<table>
<thead>
<tr>
<th>SID</th>
<th>CID</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>CPS196</td>
<td>B-</td>
</tr>
<tr>
<td>142</td>
<td>CPS196</td>
<td>B</td>
</tr>
<tr>
<td>123</td>
<td>CPS196</td>
<td>B+</td>
</tr>
<tr>
<td>857</td>
<td>CPS196</td>
<td>A+</td>
</tr>
<tr>
<td>857</td>
<td>CPS130</td>
<td>A+</td>
</tr>
<tr>
<td>456</td>
<td>CPS114</td>
<td>C</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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 → 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:

\( \checkmark \) 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

\( \checkmark \) 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

\( \checkmark \) Functional dependencies: a generalization of the key concept

\( \checkmark \) Non-key functional dependencies: a source of redundancy

\( \checkmark \) BCNF decomposition: a method for removing redundancies
  - BCNF decomposition is a lossless join decomposition

\( \checkmark \) BCNF: schema in this normal form has no redundancy due to FD’s