Motivation

How do we tell if a design is bad, e.g., StudentEnroll (SID, name, CID)?
- This design has redundancy, because the name of a student is recorded multiple times, once for each course the student is taking.

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 \( Y \).

<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>
</tbody>
</table>

Must be | Could be anything

FD examples

Address (street_address, city, state, zip)
- street_address, city, state \( \rightarrow \) zip
- zip \( \rightarrow \) city, state
- zip, state \( \rightarrow \) zip?
  - This is a trivial FD
  - Trivial FD: LHS \( \supseteq \) RHS
- zip \( \rightarrow \) state, z \( \neq \) zip?
  - This is non-trivial, but not completely non-trivial
  - Completely non-trivial FD: LHS \( \cap \) RHS = \( \emptyset \)

Keys redefined using FD’s

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 \( F \)
- Does another FD follow from \( F \)?
  - Are some of the FD’s in \( F \) 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 \( F \) that hold in \( R \), and a set of attributes \( Z \) in \( R \):

- The closure of \( Z \) (denoted \( Z^+ \)) with respect to \( F \) 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 \( F \) 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

- \( \text{StudentGrade} \) (\( \text{SID} \), \( \text{name} \), \( \text{email} \), \( \text{CID} \), \( \text{grade} \))
  - \( \text{SID} \rightarrow \text{name} \), \( \text{email} \)
  - \( \text{email} \rightarrow \text{SID} \)
  - \( \text{SID}, \text{CID} \rightarrow \text{grade} \)

- Not a good design, and we will see why later

Example of computing closure

- \( F \) includes:
  - \( \text{SID} \rightarrow \text{name}, \text{email} \)
  - \( \text{email} \rightarrow \text{SID} \)
  - \( \text{SID}, \text{CID} \rightarrow \text{grade} \)

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

- \( \text{email} \rightarrow \text{SID} \)
  - Add \( \text{SID} \); closure is now \( \{ \text{CID}, \text{email}, \text{SID} \} \)

- \( \text{SID} \rightarrow \text{name}, \text{email} \)
  - Add \( \text{name}, \text{email} \); closure is now \( \{ \text{CID}, \text{email}, \text{SID}, \text{name} \} \)

- \( \text{SID}, \text{CID} \rightarrow \text{grade} \)
  - Add \( \text{grade} \); closure is now all the attributes in \( \text{StudentGrade} \)

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:
    - \( \text{SID} \rightarrow \text{name}, \text{email} \), \( \text{email} \rightarrow \text{SID} \), \( \text{SID}, \text{CID} \rightarrow \text{grade} \)
    - \( \text{CID}, \text{email} \rightarrow \text{grade} \)
      - \( \text{email} \rightarrow \text{SID} \) (given in \( F \))
      - \( \text{CID}, \text{email} \rightarrow \text{SID} \) (augmentation)
      - \( \text{SID}, \text{CID} \rightarrow \text{grade} \) (given in \( F \))
      - \( \text{CID}, \text{email} \rightarrow \text{grade} \) (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 $X$

$$
\begin{array}{ccc}
   A & B & C \\
   1 & 2 & 3 \\
   4 & 5 & 6 \\
   \vdots & \vdots & \vdots \\
\end{array}
$$

The fact that $a$ is always associated with $b$ is recorded in multiple rows: redundancy!

Example of redundancy

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

```
<table>
<thead>
<tr>
<th>SID</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>CPS196</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>123</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>CPS196</td>
<td>B+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS196</td>
<td>A+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS130</td>
<td>A+</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>CPS114</td>
<td>C</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
</tbody>
</table>
```

Decomposition

- Eliminates redundancy
- To get back to the original relation: $\Join$

```
<table>
<thead>
<tr>
<th>StudentID</th>
<th>name</th>
<th>email</th>
<th>CourseID</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>CPS196</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>123</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>CPS196</td>
<td>B+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS196</td>
<td>A+</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>CPS130</td>
<td>A+</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>CPS114</td>
<td>C</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
</tbody>
</table>
```

Unnecessary decomposition

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

BAD decomposition

- 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 $T$
  - $attrs(R) = attrs(S) \cup attrs(T)$
  - $S = \pi_{attr(S)}(R)$
  - $T = \pi_{attr(T)}(R)$
- 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

```
SID CID grade
142 CPS196 B-
142 CPS114 B
123 CPS196 B+
857 CPS196 A+
857 CPS130 A+
456 CPS114 C
... ...
```

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
  - 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

Student (SID, name, email)
  BCNF

Grade (SID, CID, grade)
  BCNF
```

Another example

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

StudentID (email, SID)
  BCNF

StudentGrade' (email, name, CID, grade)
  BCNF violation: email \rightarrow name

StudentName (email, name)
  BCNF

Grade (email, CID, grade)
  BCNF
```
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:
  \[ R \subseteq \pi_{XY}(R) \triangleright \pi_{XZ}(R) \]
  - Sure; and it doesn’t depend on the FD
- Anything that comes back in the join must be in the original relation:
  \[ R \supseteq \pi_{XY}(R) \triangleright \pi_{XZ}(R) \]
  - 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
  - BCNF decomposition is a lossless join decomposition
- BCNF: schema in this normal form has no redundancy due to FD’s