Normalization

What and Why Normalization?
- To remove potential redundancy in design
- Redundancy causes several anomalies: insert, delete and update
- Normalization uses concept of dependencies
- Functional Dependencies
- Idea used: Decomposition
- Break R (A, B, C, D) into R1 (A, B) and R2 (B, C, D)
- Use decomposition judiciously.

Insert Anomaly

<table>
<thead>
<tr>
<th>sNumber</th>
<th>sName</th>
<th>pNumber</th>
<th>pName</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Dave</td>
<td>p1</td>
<td>MM</td>
</tr>
<tr>
<td>s2</td>
<td>Greg</td>
<td>p2</td>
<td>ER</td>
</tr>
</tbody>
</table>

Note: We cannot insert a professor who has no students.

Insert Anomaly: We are not able to insert "valid" value(s)

Delete Anomaly

<table>
<thead>
<tr>
<th>sNumber</th>
<th>sName</th>
<th>pNumber</th>
<th>pName</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Dave</td>
<td>p1</td>
<td>MM</td>
</tr>
<tr>
<td>s2</td>
<td>Greg</td>
<td>p2</td>
<td>ER</td>
</tr>
</tbody>
</table>

Note: We cannot delete a student that is the only student of a professor.

Delete Anomaly: We are not able to perform a delete without losing some "valid" information.

Update Anomaly

<table>
<thead>
<tr>
<th>sNumber</th>
<th>sName</th>
<th>pNumber</th>
<th>pName</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Dave</td>
<td>p1</td>
<td>MM</td>
</tr>
<tr>
<td>s2</td>
<td>Greg</td>
<td>p2</td>
<td>ER</td>
</tr>
</tbody>
</table>

Note: To update the name of a professor, we have to update in multiple tuples.

Update Anomaly: To update a value, we have to update multiple rows.

Keys: Revisited
- A key for a relation R (a1, a2, ..., an) is a set of attributes, K that uniquely determine the values for all attributes of R.
- A key K is minimal: no proper subset of K is a key.
- A superkey need not be minimal
- Prime Attribute: An attribute of a key
Keys: Example

<table>
<thead>
<tr>
<th>Student</th>
<th>sNumber</th>
<th>sName</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dave</td>
<td>144FL</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Greg</td>
<td>320FL</td>
<td></td>
</tr>
</tbody>
</table>

Primary Key: <sNumber>
Candidate key: <sName>
Some superkeys: {<sNumber, address>, <sName>, <sNumber>, <sNumber, sName>, <sNumber, sName, address>}
Prime Attribute: (sNumber, sName)

Functional Dependencies (FDs)

<table>
<thead>
<tr>
<th>Student</th>
<th>sNumber</th>
<th>sName</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dave</td>
<td>144FL</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Greg</td>
<td>320FL</td>
<td></td>
</tr>
</tbody>
</table>

Suppose we have the FD sName → address
- for any two rows in the Student relation with the same value for sName, the value for address must be the same
- i.e., there is a function from sName to address

Note:
- We will assume no null values.
- Any key (primary or candidate) or superkey of a relation R functionally determines all attributes of R

Properties of FDs

- Consider A, B, C, Z are sets of attributes
- Reflexive (also called trivial FD): if A ⊇ B, then A → B
- Transitive: if A → B, and B → C, then A → C
- Augmentation: if A → B, then AZ → BZ
- Union: if A → B, A → C, then A → BC
- Decomposition: if A → BC, then A → B, A → C

Inferring FDs

- Why?
  - Suppose we have a relation R (A, B, C) and we have functional dependencies A → B, B → C, C → A
  - what is a key for R?
  - Should we split R into multiple relations?
  - We can infer A → ABC, B → ABC, C → ABC. Hence A, B, C are all keys.

Algorithm for inference of FDs

- Computing the closure of set of attributes {A1, A2, ..., An}, denoted {A1, A2, ..., An}^*
  1. Let X = (A1, A2, ..., An)
  2. If there exists a FD B1, B2, ..., Bm → C, such that every Bi ∈ X, then X = X ∪ C
  3. Repeat step 2 till no more attributes can be added.
  4. (A1, A2, ..., An)^* = X

Inferring FDs: Example 1

- Given R (A, B, C), and FDs A → B, B → C, C → A, what are possible keys for R
- Compute the closure of attributes:
  - (A)^* = (A, B, C)
  - (B)^* = (A, B, C)
  - (C)^* = (A, B, C)
- So keys for R are <A>, <B>, <C>
Inferring FDs: Example 2

- Consider \( R (A, B, C, D, E) \) with FDs \( A \rightarrow B, B \rightarrow C, CD \rightarrow E \), does \( A \rightarrow E \)?
- Let us compute \( (A)^+ \)
  \( (A)^+ = \{A, B, C\} \)
- Therefore \( A \rightarrow E \) is false

Decomposing Relations

- StudentProf

<table>
<thead>
<tr>
<th>sNumber</th>
<th>sName</th>
<th>pNumber</th>
<th>pName</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Dave</td>
<td>p1</td>
<td>MM</td>
</tr>
<tr>
<td>s2</td>
<td>Greg</td>
<td>p2</td>
<td>MM</td>
</tr>
</tbody>
</table>

FDs: \( p\text{Number} \rightarrow p\text{Name} \)

<table>
<thead>
<tr>
<th>sNumber</th>
<th>sName</th>
<th>pNumber</th>
<th>pName</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Dave</td>
<td>p1</td>
<td>MM</td>
</tr>
<tr>
<td>s2</td>
<td>Greg</td>
<td>p2</td>
<td>MM</td>
</tr>
<tr>
<td>p1</td>
<td>MM</td>
<td>p2</td>
<td>MM</td>
</tr>
</tbody>
</table>

Decomposition: Lossless Join Property

Generating spurious tuples

<table>
<thead>
<tr>
<th>Student</th>
<th>Professor</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNumber</td>
<td>sName</td>
</tr>
<tr>
<td>s1</td>
<td>Dave</td>
</tr>
<tr>
<td>s2</td>
<td>Greg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StudentProf</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNumber</td>
</tr>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
</tbody>
</table>

Normalization Step

- Consider relation \( R \) with set of attributes \( A_R \).
- Consider a FD \( A \rightarrow B \) (such that no other attribute in \( (A_R - A - B) \) is functionally determined by \( A \)).
- If \( A \) is not a superkey for \( R \), we may decompose \( R \) as:
  - Create \( R' (A_R - B) \)
  - Create \( R'' \) with attributes \( A \cup B \)
  - Key for \( R'' = A \)

Normal Forms: BCNF

- Boyce Codd Normal Form (BCNF): For every non-trivial FD \( X \rightarrow a \) in \( R \), \( X \) is a superkey of \( R \).

BCNF example

- SCI (student, course, instructor)
- FDs:
  - student, course \( \rightarrow \) instructor
  - instructor \( \rightarrow \) course
- Decomposition:
  - SI (student, instructor)
  - Instructor (instructor, course)
Dependency Preservation

We might want to ensure that all specified FDs are captured. BCNF does not necessarily preserve FDs. 2NF, 3NF preserve FDs.

<table>
<thead>
<tr>
<th>Student</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dave</td>
<td>ER</td>
</tr>
</tbody>
</table>

SCI (from SI and Instructor)

<table>
<thead>
<tr>
<th>Student</th>
<th>Instructor</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dave</td>
<td>MM</td>
<td>DB 1</td>
</tr>
<tr>
<td>Dave</td>
<td>ER</td>
<td>DB 1</td>
</tr>
</tbody>
</table>

SCI violates the FD: student, course → instructor

Normal Forms: 3NF

- Third Normal Form (3NF): For every non-trivial FD X → a in R, either a is a prime attribute or X is a superkey of R.

3NF - example

Lot (propNo, county, lotNum, area, price, taxRate)

- Candidate key: <county, lotNum>
- FDs:
  - county → taxRate
  - area → price

Decomposition:

Lot (propNo, county, lotNum, area, price)
County (county, taxRate)
Area (area, price)

3NF - example

Lot (propNo, county, lotNum, area, price)

- Candidate key: <county, lotNum>
- FDs:
  - county → taxRate
  - area → price

Decomposition:

Lot (propNo, county, lotNum, area, price)
County (county, taxRate)
Area (area, price)

Define Foreign Keys?

- Consider the normalization step: A relation R with set of attributes A_R, and a FD A → B, and A is not a key for R, we decompose R as:
  - Create R' (A_R – B)
  - Create R'' with attributes A ∪ B
  - Key for R' = A
- We can also define foreign key
  - R' (A) references R'' (A)
- Question: What is key for R'?
How does Normalization Help?

Employees (ssn, name, lot, dept)
Dept (did, dName)
PK: Employees (dept)
REFERENCES Dept (did)

Suppose: employees of a dept
are in the same lot
FD: dept → lot

Decomposing:
Employees (ssn, name, dept)
Dept (did, dName)
DeptLot (dept, lot)

(or)
Employees (ssn, name, dept)
Dept (did, dName, lot)