Overview

- Relational model
  - formal query languages
  - commercial query languages (SQL)
- Integrity constraints
  - domain LC, foreign keys
  - functional dependencies
- DB design and normalization

Overview - detailed

- DB design and normalization
  - pitfalls of bad design
  - decomposition
  - normal forms

Goal

- Design ‘good’ tables
  - sub-goal#1: define what ‘good’ means
  - sub-goal#2: fix ‘bad’ tables
- in short: ‘we want tables where the attributes depend on the primary key, on the whole key, and nothing but the key’
- Let’s see why, and how:

Pitfalls

takes l (ssn, c-id, grade, name, address)

<table>
<thead>
<tr>
<th>Ssn</th>
<th>c-id</th>
<th>Grade</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>413</td>
<td>A</td>
<td>smith</td>
<td>Main</td>
</tr>
</tbody>
</table>

Pitfalls

‘Bad’ - why? because: ssn->address, name

<table>
<thead>
<tr>
<th>Ssn</th>
<th>c-id</th>
<th>Grade</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>413</td>
<td>A</td>
<td>smith</td>
<td>Main</td>
</tr>
<tr>
<td>123</td>
<td>415</td>
<td>B</td>
<td>smith</td>
<td>Main</td>
</tr>
<tr>
<td>123</td>
<td>211</td>
<td>A</td>
<td>smith</td>
<td>Main</td>
</tr>
</tbody>
</table>
Pitfalls

- Redundancy
  - space
  - (inconsistencies)
  - insertion/deletion anomalies:

Pitfalls

- insertion anomaly:
  - “jones” registers, but takes no class - no place to store his address!

Solution: decomposition

- split offending table in two (or more), eg:

Overview - detailed

- DB design and normalization
  - pitfalls of bad design
  - decomposition
    - lossless join decom.
    - dependency preserving
  - normal forms

Decompositions

- there are “bad” decompositions: we want
  - lossless and
  - dependency preserving
Decompositions - lossy:

R1(ssn, grade, name, address)  R2(c-id, grade)

<table>
<thead>
<tr>
<th>ssn</th>
<th>c-id</th>
<th>grade</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>A</td>
<td>45.6</td>
<td>smith</td>
<td>main</td>
</tr>
<tr>
<td>234</td>
<td>B</td>
<td>78.9</td>
<td>john</td>
<td>park</td>
</tr>
</tbody>
</table>

ssn \rightarrow name, address

Decompositions - lossy:
can not recover original table with a join!

<table>
<thead>
<tr>
<th>ssn</th>
<th>c-id</th>
<th>grade</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>A</td>
<td>45.6</td>
<td>smith</td>
<td>main</td>
</tr>
<tr>
<td>234</td>
<td>B</td>
<td>78.9</td>
<td>john</td>
<td>park</td>
</tr>
</tbody>
</table>

Decompositions

example of non-dependency preserving

<table>
<thead>
<tr>
<th>S#</th>
<th>address</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Residence</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>Park</td>
<td>B</td>
</tr>
</tbody>
</table>

S# \rightarrow address, status

Decompositions

(drill: is it lossless?)

<table>
<thead>
<tr>
<th>S#</th>
<th>address</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Residence</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>Park</td>
<td>B</td>
</tr>
</tbody>
</table>

S# \rightarrow address, status

Decompositions - lossless

Definition:
consider schema R, with FD ‘F’. R1, R2 is a lossless join decomposition of R if we always have: r1 \land r2 = r

An easier criterion?
Decomposition - lossless

Example:

<table>
<thead>
<tr>
<th>R1</th>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>113</td>
<td>A. Smith</td>
<td>123 Main St.</td>
</tr>
<tr>
<td>234</td>
<td>515</td>
<td>M. Brown</td>
<td>234 Pioneer Ave.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>113</td>
<td>A. Smith</td>
<td>123 Main St.</td>
</tr>
</tbody>
</table>

\[
\text{ssn} \rightarrow \text{cid} \rightarrow \text{grade} 
\quad \text{ssn} \rightarrow \text{name}, \text{address}
\]

Overview - detailed

- DB design and normalization
  - pitfalls of bad design
  - decomposition
    - lossless join decomps.
    - dependency preserving
      - normal forms

Decomposition - depend. pres.

Informally: we don’t want the original FDs to span two tables - counter-example:

<table>
<thead>
<tr>
<th>R1</th>
<th>S#</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123</td>
<td>A. Smith</td>
<td>123 Main St.</td>
</tr>
<tr>
<td>234</td>
<td>234</td>
<td>M. Brown</td>
<td>234 Pioneer Ave.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>S#</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123</td>
<td>A. Smith</td>
<td>123 Main St.</td>
</tr>
</tbody>
</table>

\[
\text{S#} \rightarrow \text{address}, \text{status} 
\quad \text{address} \rightarrow \text{status}
\]

Decomposition - depend. pres.

dependency preserving decomposition:

<table>
<thead>
<tr>
<th>R1</th>
<th>S#</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123</td>
<td>A. Smith</td>
<td>123 Main St.</td>
</tr>
<tr>
<td>234</td>
<td>234</td>
<td>M. Brown</td>
<td>234 Pioneer Ave.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>S#</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123</td>
<td>A. Smith</td>
<td>123 Main St.</td>
</tr>
</tbody>
</table>

\[
\text{S#} \rightarrow \text{address} \quad \text{S#} \rightarrow \text{status} 
\quad \text{address} \rightarrow \text{status} 
\quad \text{(but: S# \rightarrow \text{status}?)}
\]

Decomposition - depend. pres.

Informally: we don’t want the original FDs to span two tables.

More specifically: … the FDs of the canonical cover.

<table>
<thead>
<tr>
<th>R1</th>
<th>S#</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123</td>
<td>A. Smith</td>
<td>123 Main St.</td>
</tr>
<tr>
<td>234</td>
<td>234</td>
<td>M. Brown</td>
<td>234 Pioneer Ave.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>S#</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123</td>
<td>A. Smith</td>
<td>123 Main St.</td>
</tr>
</tbody>
</table>

\[
\text{S#} \rightarrow \text{address} 
\quad \text{S#} \rightarrow \text{status} 
\quad \text{address} \rightarrow \text{status} 
\quad \text{(address \rightarrow \text{status}; lost?)}
\]

Decomposition - depend. pres.

Why is dependency preservation good?

<table>
<thead>
<tr>
<th>R1</th>
<th>S#</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123</td>
<td>A. Smith</td>
<td>123 Main St.</td>
</tr>
<tr>
<td>234</td>
<td>234</td>
<td>M. Brown</td>
<td>234 Pioneer Ave.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>S#</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123</td>
<td>A. Smith</td>
<td>123 Main St.</td>
</tr>
</tbody>
</table>

\[
\text{S#} \rightarrow \text{address} 
\quad \text{S#} \rightarrow \text{status} 
\quad \text{(address \rightarrow \text{status}; lost?)}
\]
Decomposition - depend. pres.
A: e.g., record that ‘Philly’ has status ‘A’

Decomposition - conclusions
• decompositions should always be lossless
  – joining attribute -> superkey
• whenever possible, we want them to be
dependency preserving (occasionally,
  impossible - see ‘STJ’ example later…)

Overview - detailed
• DB design and normalization
  – pitfalls of bad design
  – decomposition (-> how to fix the problem)
  – normal forms (-> how to detect the problem)
    • BCNF,
    • 3NF
    • (1NF, 2NF)

Normal forms - BCNF
We saw how to fix ‘bad’ schemas -
but what is a ‘good’ schema?

Answer: ‘good’, if it obeys a ‘normal form’,
i.e., a set of rules.

Typically: Boyce-Codd Normal form

Normal forms - BCNF
Defn.: Rel. R is in BCNF wrt F, if
• informally: everything depends the full key,
  and nothing but the key
• semi-formally: every determinant (of the
  cover) is a candidate key

Normal forms - BCNF
Example and counter-example:
Normal forms - BCNF

Formally: for every FD $a \rightarrow b$ in $F^+$
- $a \rightarrow b$ is trivial ($a$ superset of $b$) or
- $a$ is a superkey (or both)

Normal forms - BCNF

Theorem: given a schema $R$ and a set of FD $F$, we can always decompose it into schemas $R_1, \ldots, R_n$ so that
- $R_1, \ldots, R_n$ are in BCNF and
- the decompositions are lossless,
  (but, some decomp. might lose dependencies)

Normal forms - BCNF

How? algorithm in book - essentially, break off FDs of the cover

e.g. $\text{TAKES1}(\text{ssn}, \text{c-id}, \text{grade}, \text{name}, \text{address})$

\begin{align*}
\text{ssn} & \rightarrow \text{name}, \text{address} \\
\text{ssn}, \text{c-id} & \rightarrow \text{grade}
\end{align*}

Normal forms - BCNF

\begin{align*}
\text{ssn} & \rightarrow \text{name}, \text{address} \\
\text{ssn}, \text{c-id} & \rightarrow \text{grade}
\end{align*}

Normal forms - BCNF

pictorially: we want a ‘star’ shape

\begin{align*}
\text{ssn} & \rightarrow \text{name}, \text{address} \\
\text{ssn}, \text{c-id} & \rightarrow \text{grade}
\end{align*}
Normal forms - BCNF

pictorially: we want a ‘star’ shape

Normal forms - BCNF

or a star-like (e.g., 2 cand. keys):

STUDENT(ssn, std#, name, address)

Normal forms - BCNF

but not:

Normal forms - 3NF

consider the ‘classic’ case:

STJ( Student, Teacher, subject)

T -> J  S J -> T

is it BCNF?

Normal forms - 3NF

STJ( Student, Teacher, subject)

T -> J  S J -> T

1) R1(TJ)  R2(SJ)
   (BCNF?  lossless?  dep. pres.?)

2) R1(TJ)  R2(S,T)
   (BCNF?  lossless?  dep. pres.?)

How to decompose it to BCNF?

CMU - 15-415
Normal forms - 3NF

STJ( Student, Teacher, subject)
T -> J  SJ -> T
1) R1(T|J)  R2(S|J)
   (BCNF? Y+Y - lossless? N - dep. pres.? N )
2) R1(T|J)  R2(S,T)
   (BCNF? Y+Y - lossless? Y - dep. pres.? N )

Normal forms - 3NF

STJ( Student, Teacher, subject)
T -> J  SJ -> T
in this case: impossible to have both
• BCNF
• dependency preservation
Welcome 3NF!

Normal forms - 3NF

STJ( Student, Teacher, subject)
T -> J  SJ -> T

informally, 3NF
‘forgives’ the red arrow
in the can, cover

Normal forms - 3NF

STJ( Student, Teacher, subject)
T -> J  SJ -> T

Formally, a rel. R with
FDs ‘F’ is in 3NF if;
for every a->b in F:
• it is trivial or
• a is a superkey or
• each b-a attr.: part of a
cand. key

Normal forms - 3NF vs BCNF

• If ‘R’ is in BCNF, it is always in 3NF (but
  not the reverse)
• In practice, aim for
  • BCNF; lossless join; and dep. preservation
• if impossible, we accept
  • 3NF, but insist on lossless join and dep.
  preservation

how to bring a schema to 3NF?
also in book:
for each FD in the cover, put it in a table
Normal forms - more details

- why '3'NF? what is 2NF? 1NF?
- 1NF: attributes are atomic (i.e., no set-valued attr., a.k.a. 'repeating groups')
- 2NF: INF and non-key attr. fully depend on the key
  counter-example: TAKES(ssn, c-id, grade, name, address)
  ssn -> name, address  ssn, c-id -> grade

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Dependants</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
<td>Peter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mary</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
<td>Ann</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bill</td>
</tr>
</tbody>
</table>

not INF

Normal forms - more details

• 3NF; 2NF and no transitive dependencies
• counter-example:

```
  D  
   |   
   |   
  B  
   |   
  A  
```

in 2NF, but not in 3NF

Normal forms - more details

• 4NF, multivalued dependencies etc: IGNORE
• in practice, E-R diagrams usually lead to tables in BCNF

Overview - conclusions

DB design and normalization
- pitfalls of bad design
- decompositions (lossless, dep. preserving)
- normal forms (BCNF or 3NF)

"everything should depend on the key, the whole key, and nothing but the key"

CMU - 15-415