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

`takes1 (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>413</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!

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Main</td>
</tr>
<tr>
<td>234</td>
<td>Full</td>
<td>Jones</td>
<td>Forbes</td>
</tr>
</tbody>
</table>

- **deletion anomaly:**
  - delete the last record of ‘smith’ (we lose his address!)

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>123</td>
<td>415</td>
<td>B</td>
<td>Smith</td>
</tr>
<tr>
<td>123</td>
<td>211</td>
<td>A</td>
<td>Smith</td>
</tr>
</tbody>
</table>

### Solution: decomposition

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

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>123</td>
<td>415</td>
<td>B</td>
<td>Smith</td>
</tr>
<tr>
<td>123</td>
<td>211</td>
<td>A</td>
<td>Smith</td>
</tr>
</tbody>
</table>

?     ?
Overview - detailed

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

Decompositions

There are ‘bad’ decompositions. Good ones are:

- lossless and
- dependency preserving

Decompositions - lossy:

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

<table>
<thead>
<tr>
<th>ssn</th>
<th>grade</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>A</td>
<td>Smith</td>
<td>Main</td>
</tr>
<tr>
<td>124</td>
<td>B</td>
<td>Jones</td>
<td>Dorken</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>213</td>
<td>A</td>
</tr>
<tr>
<td>212</td>
<td>A</td>
</tr>
<tr>
<td>211</td>
<td>A</td>
</tr>
</tbody>
</table>

ssn->name, address
ssn, c-id -> grade
Decompositions - lossy:

- cannot recover original table with a join!

<table>
<thead>
<tr>
<th>ssn</th>
<th>grade</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>A</td>
<td>Smith</td>
<td>Main</td>
</tr>
<tr>
<td>123</td>
<td>B</td>
<td>White</td>
<td>Main</td>
</tr>
<tr>
<td>234</td>
<td>A</td>
<td>Jones</td>
<td>Forbes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ssn</th>
<th>c-id</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>1</td>
<td>Smith</td>
<td>Main</td>
</tr>
<tr>
<td>123</td>
<td>2</td>
<td>White</td>
<td>Main</td>
</tr>
<tr>
<td>234</td>
<td>1</td>
<td>Jones</td>
<td>Forbes</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>London</td>
<td>A</td>
</tr>
<tr>
<td>125</td>
<td>Paris</td>
<td>B</td>
</tr>
<tr>
<td>234</td>
<td>Pitta</td>
<td>A</td>
</tr>
</tbody>
</table>

S# -> address, status
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>London</td>
<td>A</td>
</tr>
<tr>
<td>125</td>
<td>Paris</td>
<td>B</td>
</tr>
<tr>
<td>234</td>
<td>Pitta</td>
<td>A</td>
</tr>
</tbody>
</table>

S# -> address, status
address -> status
Decompositions - lossless

Definition:
consider schema R, with FD ‘F’. R1, R2 is a lossless join decomposition of R if we always have: \( r_1 \circ r_2 = r \)

An easier criterion?

Decomposition - lossless

Theorem: lossless join decomposition if the joining attribute is a superkey in at least one of the new tables

Formally:
\[ R_1 \cap R_2 \rightarrow R_1 \text{ or } R_1 \cap R_2 \rightarrow R_2 \]

Decomposition - lossless

example:

<table>
<thead>
<tr>
<th>ssn, c-id -&gt; grade</th>
<th>ssn-&gt;name, address</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>ssn</td>
<td>c-id</td>
</tr>
<tr>
<td>123</td>
<td>413</td>
</tr>
<tr>
<td>123</td>
<td>415</td>
</tr>
<tr>
<td>234</td>
<td>211</td>
</tr>
<tr>
<td>ssn</td>
<td>c-id</td>
</tr>
<tr>
<td>123</td>
<td>413</td>
</tr>
<tr>
<td>123</td>
<td>415</td>
</tr>
<tr>
<td>234</td>
<td>211</td>
</tr>
</tbody>
</table>
Overview - detailed

- DB design and normalization
  - pitfalls of bad design
  - decomposition
  - lossless join decomp.
  - 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>S#</th>
<th>address</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>London</td>
<td>6</td>
</tr>
<tr>
<td>234</td>
<td>Pitts.</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S#</th>
<th>address</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>London</td>
<td>6</td>
</tr>
<tr>
<td>124</td>
<td>Pitts.</td>
<td>A</td>
</tr>
</tbody>
</table>

S# -> address, status
address -> status

Decomposition - depend. pres.
derpendency preserving decomposition:

<table>
<thead>
<tr>
<th>S#</th>
<th>address</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>London</td>
<td>6</td>
</tr>
<tr>
<td>234</td>
<td>Pitts.</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S#</th>
<th>address</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>London</td>
<td>6</td>
</tr>
<tr>
<td>234</td>
<td>Pitts.</td>
<td>A</td>
</tr>
</tbody>
</table>

S# -> address, status
address -> status
(but: S#->status ?)
Decomposition - depend. pres.

informally: we don’t want the original FDs to span two tables.
More specifically: … the FDs of the canonical cover.

why is dependency preservation good?

A: eg., 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’, ie., a set of rules.

Typically: Boyce-Codd Normal form
Normal forms - BCNF

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

Example and counter-example:

| Name | Address | | Name | Grade |
tag{ssn} Smith Shady | 883 Main | | 123 456 A | 883 Main |
| 234 Jones Forbes | ssn -> name, address |

Formally: for every FD $a \rightarrow b$ in F
  – $a \rightarrow b$ is trivial ($a$ superset of $b$) or
  – $a$ is a superkey
Normal forms - BCNF

Theorem: given a schema R and a set of FD ‘F’, we can always decompose it to schemas R1, … Rn, so that
– R1, … Rn are in BCNF and
– the decompositions are lossless.
(but, some decomp. might lose dependencies)

- for every FD X->A that violates BCNF,
  decompose to tables (X,A) and (R-A)
- repeat recursively
eg. TAKES1(ssn, c-id, grade, name, address)
  ssn -> name, address
  ssn, c-id -> grade

eg. TAKES1(ssn, c-id, grade, name, address)
  ssn -> name, address
  ssn, c-id -> grade
Normal forms - BCNF

<table>
<thead>
<tr>
<th>ssn</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>413</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>415</td>
<td>B</td>
</tr>
<tr>
<td>345</td>
<td>211</td>
<td>A</td>
</tr>
</tbody>
</table>

Normal forms - BCNF

pictorially: we want a 'star' shape

grade

name

address
c-id

not in BCNF

Normal forms - BCNF

pictorially: we want a 'star' shape

A

B

C

D

E

F

G

H

or
Normal forms - BCNF

or a star-like: (eg., 2 cand. keys):
STUDENT(ssn, st#, name, address)

name

address

ssn

st#

Normal forms - BCNF

but not:

A

B

D

C

or

F

D

E

G

H

Normal forms - 3NF

consider the ‘classic’ case:
STJ( Student, Teacher, subj ect)

T -> J

S -> T

is it BCNF?
Normal forms - 3NF

STJ( Student, Teacher, Subject)
T -> J  S,J -> T
How to decompose it to BCNF?

1) R1(T,J)  R2(S,J)
   (BCNF?   - lossless? - dep. pres.? )
2) R1(T,J)  R2(S,T)
   (BCNF?   - lossless? - dep. pres.? )
Normal forms - 3NF

STJ( Student, Teacher, subJect)

T -> J  S,J -> T

in this case: impossible to have both
• BCNF and
• dependency preservation

Welcome 3NF!

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

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
• b: part of a candidate key
Normal forms - 3NF

how to bring a schema to 3NF?
two algo’s in book: First one:
• start from ER diagram and turn to tables
• then we have a set of tables R1, ... Rn which are in 3NF
• for each FD (X->A) in the cover that is not preserved, create a table (X,A)

Normal forms - 3NF

how to bring a schema to 3NF?
two algo’s in book: Second one (‘synthesis’)  
• take all attributes of R
• for each FD (X->A) in the cover, add a table (X,A)
• if not lossless, add a table with appropriate key

Example:
R: ABC
F: A->B, C->B
Q1: what is the cover?

Q2: what is the decomposition to 3NF?
Normal forms - 3NF

Example:
R: ABC
F: A->B, C->B
Q1: what is the cover?
A1: ‘F’ is the cover
Q2: what is the decomposition to 3NF?
A2: R1(A,B), R2(C,B), ...

[Is it lossless??]
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

Normal forms - more details

- why ‘3’NF? what is 2NF? 1NF?
- 1NF: attributes are atomic (ie., no set-valued attr., a.k.a. ‘repeating groups’)

<table>
<thead>
<tr>
<th>ssn</th>
<th>Name</th>
<th>Dependents</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
<td>Peter, Mary, John</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
<td>Ann, Michael</td>
</tr>
</tbody>
</table>

Not 1NF

2NF: 1NF and non-key attr. fully depend on the key

counter-example: TAKES1(ssn, c-id, grade, name, address)

<table>
<thead>
<tr>
<th>ssn</th>
<th>c-id</th>
<th>name</th>
<th>address</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>101</td>
<td>John</td>
<td>Smith</td>
<td>85</td>
</tr>
<tr>
<td>234</td>
<td>202</td>
<td>Mary</td>
<td>Jones</td>
<td>90</td>
</tr>
</tbody>
</table>
Normal forms - more details

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

```
A -> B
B -> C
D
```

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”