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>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
  - ??
  - ??

Pitfalls

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

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

• deletion anomaly:
  – delete the last record of ‘smith’ (we lose his 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>
<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>

Solution: decomposition

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

<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>

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 MUST HAVE
• dependency preserving Nice to have
Decompositions - lossy:

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

\[
\begin{array}{ccc}
\text{ssn} & \text{name} & \text{address} \\
123 & A & \text{Smith Main} \\
123 & B & \text{Smith Main} \\
234 & A & \text{Jones Forbes} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{c-id} & \text{Grade} \\
413 & A \\
415 & B \\
211 & A \\
\end{array}
\]

ssn->name, address

ssn, c-id -> grade

can not recover original table with a join!

Decompositions - overview

There are ‘bad’ decompositions. Good ones are:

• lossless and MUST HAVE
• dependency preserving Nice to have

Decompositions

example of non-dependency preserving

\[
\begin{array}{ccc}
\text{S} & \text{address} & \text{status} \\
123 & \text{London} & E \\
125 & \text{Paris} & E \\
234 & \text{Pitts.} & A \\
\end{array}
\]

S# -> address, status

S# -> status

address -> status
Decompositions

(drink: 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>E</td>
</tr>
<tr>
<td>125</td>
<td>Paris</td>
<td>E</td>
</tr>
<tr>
<td>234</td>
<td>Pitts.</td>
<td>A</td>
</tr>
</tbody>
</table>

S# -> address, status
address -> status
S# -> address
S# -> status

Decompositions - overview

There are ‘bad’ decompositions. Good ones are:
- #1) lossless and • MUST HAVE
- #2) dependency preserving • Nice to have

How to automatically determine #1 and #2?

Decompositions - lossless

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

R1 ∩ R2 → R\ or
R1 ∩ R2 → R2
Decomposition - lossless

example:

R1

<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>123</td>
<td>415</td>
<td>B</td>
</tr>
<tr>
<td>234</td>
<td>211</td>
<td>A</td>
</tr>
</tbody>
</table>

ssn, c-id -> grade

R2

<table>
<thead>
<tr>
<th>ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>Main</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
<td>Forbes</td>
</tr>
</tbody>
</table>

ssn->name, address

Decomposition - depend. pres.

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

S# -> address, status

<table>
<thead>
<tr>
<th>S#</th>
<th>address</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>London</td>
<td>E</td>
</tr>
<tr>
<td>125</td>
<td>Paris</td>
<td>E</td>
</tr>
<tr>
<td>234</td>
<td>Pitts.</td>
<td>A</td>
</tr>
</tbody>
</table>

S# -> address

Q: Why is it an issue?

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>E</td>
</tr>
<tr>
<td>125</td>
<td>Paris</td>
<td>E</td>
</tr>
<tr>
<td>234</td>
<td>Pitts.</td>
<td>A</td>
</tr>
</tbody>
</table>

S# -> address, status
address -> status

(Q: Why is it an issue?)
(A: insert [999, Pitts., E])

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>E</td>
</tr>
<tr>
<td>125</td>
<td>Paris</td>
<td>E</td>
</tr>
<tr>
<td>234</td>
<td>Pitts.</td>
<td>A</td>
</tr>
</tbody>
</table>

999 Pitts.

S# -> address, status
address -> status

Decomposition - depend. pres.

• A subtle point
• To avoid it, use the ‘canonical cover’ of the FDs

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>E</td>
</tr>
<tr>
<td>125</td>
<td>Paris</td>
<td>E</td>
</tr>
<tr>
<td>234</td>
<td>Pitts.</td>
<td>A</td>
</tr>
</tbody>
</table>

S# -> address, status
address -> status
Decomposition - depend. pres.

Dependency preserving decomposition:

\[
\begin{align*}
S\# & \rightarrow \text{address, status} \\
\text{address} & \rightarrow \text{status} \\
\text{(but: } S\# \rightarrow \text{status ?)}
\end{align*}
\]

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

More specifically: … the FDs of the canonical cover.

Decomposition - depend. pres.

Q: why is dependency preservation good?

\[
\begin{align*}
S\# & \rightarrow \text{address} \\
\text{S\#} & \rightarrow \text{status} \\
\text{(address->status: ‘lost’)}
\end{align*}
\]

A1: insert [999, Pitts., E] -> REJECT

\[
\begin{align*}
\text{S\#} & \rightarrow \text{address} \\
\text{address} & \rightarrow \text{status} \\
\text{S\#} & \rightarrow \text{status} \\
\text{(address->status: ‘lost’)}
\end{align*}
\]
Decomposition - depend. pres.
A2: eg., record that ‘Philly’ has status ‘A’

<table>
<thead>
<tr>
<th>S#</th>
<th>address</th>
<th>S#</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>London</td>
<td>123</td>
<td>E</td>
</tr>
<tr>
<td>125</td>
<td>Paris</td>
<td>125</td>
<td>A</td>
</tr>
<tr>
<td>134</td>
<td>Villa</td>
<td>134</td>
<td>null</td>
</tr>
</tbody>
</table>

ADDRESS -> status

(address->status: ‘lost’)

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

Or

• semi-formally: every determinant (of the cover) is a candidate key

Formally: for every FD $a \rightarrow b$ in $F$

– $a \rightarrow b$ is trivial ($a$ superset of $b$) or

– $a$ is a superkey

Example and counter-example:

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>Main</td>
</tr>
<tr>
<td>999</td>
<td>smith</td>
<td>Shady</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
<td>Forbes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>A</td>
<td>Main</td>
</tr>
<tr>
<td>123</td>
<td>B</td>
<td>Main</td>
</tr>
<tr>
<td>234</td>
<td>A</td>
<td>Forbes</td>
</tr>
</tbody>
</table>

Drill: Check formal dfn:

• $a \rightarrow b$ trivial, or

• $a$ is superkey
Normal forms

Example and counter-example:

<table>
<thead>
<tr>
<th>ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>Main</td>
</tr>
<tr>
<td>999</td>
<td>smith</td>
<td>Shady</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
<td>Forbes</td>
</tr>
</tbody>
</table>

<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>222</td>
<td>425</td>
<td>B</td>
<td>smith</td>
<td>Main</td>
</tr>
<tr>
<td>234</td>
<td>221</td>
<td>A</td>
<td>Jones</td>
<td>Forbes</td>
</tr>
</tbody>
</table>

Normal forms - BCNF

Theorem: given a schema R and a set of FDs ‘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)

Drill: Check formal dfn:
- a->b trivial, or
- a is superkey

Normal forms - BCNF

- 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

<table>
<thead>
<tr>
<th>grade</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>c-id</td>
<td></td>
</tr>
</tbody>
</table>

not in book
Normal forms - BCNF

ssn, c-id -> grade

ssn->name, address

<table>
<thead>
<tr>
<th>Ssn</th>
<th>c-id</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>113</td>
<td>A</td>
</tr>
<tr>
<td>123</td>
<td>915</td>
<td>B</td>
</tr>
<tr>
<td>234</td>
<td>211</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>Main</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>Main</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>Forbes</td>
</tr>
</tbody>
</table>

pictorially: we want a ‘star’ shape

Normal forms - BCNF

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

STUDENT(ssn, st#, name, address)
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)

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)

How is this possible?
Subtle answer

In some rare cases, like the (Student, Teacher, subject) setting:

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

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)

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 )

in this case: impossible to have both
- BCNF
- dependency preservation

Welcome 3NF!

Informally, 3NF ‘forgives’ the red arrow in the canonical 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? What is the cand. key?
Q2: what is the decomposition to 3NF?
Normal forms - 3NF

Example:
R: ABC
F: A->B, C->B
Q1: what is the cover? What is the cand. key?
A1: ‘F’ is the cover; ‘AC’ is the cand. key
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</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>Michael</td>
</tr>
</tbody>
</table>

not 1NF
Normal forms - more details

2NF: 1NF and non-key attr. fully depend on the key
counter-example: TAKES1(ssn, c-id, grade, name, address)

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

not 2NF

Normal forms - more details

\begin{itemize}
\item 3NF: 2NF and no transitive dependencies
\item counter-example:
\end{itemize}

in 2NF, but not in 3NF

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”