Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

C. Faloutsos
Integrity Constraints

Constraints:
ge., E-R Model:
• Key
• cardinalities of a Relationship

Overview
• Domain; Ref. Integrity constraints
• Assertions and Triggers
• Functional dependencies

Domain Constraints
• Domain Types, eg, SQL
  – Fixed Length characters
  – Int; float; (date)
• null values eg.,
  create table student( ssn char(9) not null, ...)

Referential Integrity constraints
‘foreign keys’ - eg:
create table takes(
  ssn char(9) not null,
c-id char(5) not null,
grade integer,
primary key(ssn, c-id),
foreign key ssn references student,
foreign key c-id references class)

Referential Integrity constraints
...
foreign key ssn references student,
foreign key c-id references class)
Effect:
– expects that ssn to exist in ‘student’ table
– blocks ops that violate that - how??
• insertion?
• deletion/updated?
Referential Integrity constraints

... foreign key ssn references student
    on delete cascade
    on update cascade,
...
• -> eliminate all student enrollments
• other options (set to null, to default etc)

Weapons for IC:

• assertions
  – create assertion <assertion-name> check
    <predicate>
• triggers (~ assertions with ‘teeth’)  
  – on operation, if condition, then action

Triggers - example

define trigger zero grade on update takes
  (if new takes.grade < 0
    then takes.grade = 0)

Triggers - discussion

• more complicated: “managers have higher
  salaries than their subordinates”  - a trigger
  can automatically boost mgrs salaries
• triggers tricky (infinite loops…)

Overview

• Domain: Ref. Integrity constraints
• Assertions and Triggers
• Functional dependencies
  – why
  – definition
  – Armstrong’s “axioms”
  – closure and cover

Functional dependencies

• motivation: ‘good’ tables
  takes1 (ssn, c-id, grade, name, address)
  ‘good’ or ‘bad’?
Functional dependencies

takes 1 (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>
<tr>
<td>123</td>
<td>415</td>
<td>B</td>
<td>smith</td>
<td>Main</td>
</tr>
<tr>
<td>123</td>
<td>411</td>
<td>A</td>
<td>smith</td>
<td>Main</td>
</tr>
</tbody>
</table>

Functional dependencies

‘Bad’ - why?

<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>411</td>
<td>A</td>
<td>smith</td>
<td>Main</td>
</tr>
</tbody>
</table>

Functional Dependencies

- Redundancy
  - space
  - inconsistencies
  - insertion/deletion anomalies (later...)
- What caused the problem?

Functional dependencies

- ‘name’ depends on the ‘ssn’
- define ‘depends’

<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>411</td>
<td>A</td>
<td>smith</td>
<td>Main</td>
</tr>
</tbody>
</table>

Overview

- Domain: Ref, Integrity constraints
- Assertions and Triggers
- Functional dependencies
  - why
  - definition
  - Armstrong’s “axioms”
  - closure and cover

Functional dependencies

Definition: \( a \rightarrow b \)
‘a’ functionally determines ‘b’

<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>411</td>
<td>A</td>
<td>smith</td>
<td>Main</td>
</tr>
</tbody>
</table>
Functional dependencies

Informally: ‘if you know ‘a’, there is only one ‘b’ to match’

<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>411</td>
<td>A</td>
<td>smith</td>
<td>Main</td>
</tr>
</tbody>
</table>

Formally:

\[ X \rightarrow Y \quad \Rightarrow \quad \forall x_1 = r(x_1) \forall y_1 = r(y_1) \]

If two tuples agree on the ‘X’ attribute, the *must* agree on the ‘Y’ attribute, too (e.g., if ssn is the same, so should address)

Functional dependencies

- ‘X’, ‘Y’ can be sets of attributes
- other examples??

<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>411</td>
<td>A</td>
<td>smith</td>
<td>Main</td>
</tr>
</tbody>
</table>

Functional dependencies

- ssn -> name, address
- ssn, 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>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>411</td>
<td>A</td>
<td>smith</td>
<td>Main</td>
</tr>
</tbody>
</table>

Overview

- Domain: Ref, Integrity constraints
- Assertions and Triggers
- Functional dependencies
  - why
  - definition
  - Armstrong’s “axioms”
  - closure and cover

Functional dependencies

Closure of a set of FD: all implied FDs - eg:

- ssn -> name, address
- ssn, c-id -> grade

Imply

- ssn, c-id -> grade, name, address
- ssn, c-id -> ssn
FDs - Armstrong’s axioms

Closure of a set of FD: all implied FDs - eg.: 
- ssn -> name, address
- ssn, e-id -> grade

how to find all the implied ones, systematically?

FDs - Armstrong’s axioms

“Armstrong’s axioms” guarantee soundness and completeness:
- Reflexivity: \( Y \subseteq X \Rightarrow X \rightarrow Y \)
e.g., ssn, name -> ssn
- Augmentation \( X \rightarrow Y \Rightarrow XW \rightarrow YW \)
e.g., ssn->name then ssn.grade-> ssn.grade

FDs - Armstrong’s axioms

- Transitivity \( X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z \)
  ssn->address
  address-> county-tax-rate
  THEN:
  ssn-> county-tax-rate

FDs - Armstrong’s axioms

Reflexivity: \( Y \subseteq X \Rightarrow X \rightarrow Y \)
Augmentation: \( X \rightarrow Y \Rightarrow XW \rightarrow YW \)
Transitivity: \( X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z \)

“sound” and “complete”

FDs - Armstrong’s axioms

Additional rules:
- Union \( X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ \)
- Decomposition \( X \rightarrow YZ \Rightarrow X \rightarrow Y, Y \rightarrow Z \)
- Pseudo-transitivity \( X \rightarrow Y, \Rightarrow XW \rightarrow Z \)

FDs - Armstrong’s axioms

Prove “Union” from three axioms:
- \( X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ \)
FDs - Armstrong’s axioms

Prove ‘Union’ from three axioms:

1. \( X \rightarrow Y \)
2. \( X \rightarrow Z \)
3. \( X + \text{augm.} w/ Z \Rightarrow XZ \rightarrow YZ \)
4. \( X + \text{augm.} w/ X \Rightarrow XY \rightarrow XZ \)

but \( XX \) is \( X \); thus

2+(4) and transitivity \( \Rightarrow X \rightarrow YZ \)

FDs - Armstrong’s axioms

Prove Pseudo-transitivity:

\[
\begin{align*}
Y \subseteq X & \Rightarrow X \rightarrow Y \\
X \rightarrow Y & \Rightarrow XW \rightarrow YW \\
X \rightarrow Y & \Rightarrow X \rightarrow Z \\
Y \rightarrow Z & \Rightarrow X \rightarrow Z
\end{align*}
\]

\[
\begin{align*}
X \rightarrow Y & \subseteq XW \rightarrow Z \\
YW & \rightarrow Z \\
XW & \rightarrow Z
\end{align*}
\]

FDs - Armstrong’s axioms

Prove Decomposition

\[
\begin{align*}
Y \subseteq X & \Rightarrow X \rightarrow Y \\
X \rightarrow Y & \Rightarrow XW \rightarrow YW \\
X \rightarrow Y & \Rightarrow X \rightarrow Z \\
Y \rightarrow Z & \Rightarrow X \rightarrow Z
\end{align*}
\]

Overview

- Domain; Ref, Integrity constraints
- Assertions and Triggers
- Functional dependencies
  - why
  - definition
  - Armstrong’s “axioms”
  - closure and cover

FDs - Closure F+

Given a set \( F \) of FD (on a schema)

\[
F = \{ \text{ssn, c-id} \rightarrow \text{grade}, \text{ssn} \rightarrow \text{name, address} \}
\]

\[
F^+ = \{ \text{ssn, c-id} \rightarrow \text{name, address}, \text{ssn} \rightarrow \text{ssn}, \text{ssn, c-id} \rightarrow \text{address}, \text{c-id, address} \rightarrow \text{c-id} \}
\]
FDs - Closure A+

Given a set F of FD (on a schema)
A+ is the set of all attributes determined by A:
takes(ssn, c-id, grade, name, address)
ssn, c-id -> grade  } F
ssn -> name, address  

\{ssn\}+ = ??

FDs - Closure A+

takes(ssn, c-id, grade, name, address)
ssn, c-id -> grade  } F
ssn -> name, address  

\{c-id\}+ = ??

FDs - Closure A+

if A+ = \{all attributes of table\}
then ‘A’ is a candidate key

FDs - A+ closure - not in book

Diagrams

AB->C (1)
A->BC (2)
B->C  (3)
A->B  (4)
FDs - ‘canonical cover’ Fe

Given a set $F$ of FD (on a schema)

$F_\epsilon$ is a minimal set of equivalent FD. E.g.,
takes($ssn$, $c-id$, $grade$, $name$, $address$)

$ssn$, $c-id$ $\rightarrow$ $grade$
$ssn$, $name$, $address$
$ssn$, $name$ $\rightarrow$ $name$, $address$
$ssn$, $c-id$ $\rightarrow$ $grade$, $name$

$F$

FDs - ‘canonical cover’ Fe

• why do we need it?
• define it properly
• compute it efficiently

FDs - ‘canonical cover’ Fe

• why do we need it?
  – easier to compute candidate keys
• define it properly
• compute it efficiently

FDs - ‘canonical cover’ Fe

• define it properly - three properties
  – every FD $a \rightarrow b$ has no extraneous attributes on
    the RHS
  – dito for the LHS
  – all LHS parts are unique

FDs - ‘canonical cover’ Fe

‘extraneous’ attribute:
  – if the closure is the same, before and after its elimination
  – or if $F$ before implies $F$ after and vice-versa
FDs - ‘canonical cover’ Fc

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

F

FDs - ‘canonical cover’ Fc

Algorithm:
- examine each FD; drop extraneous LHS or RHS attributes
- merge FDs with same LHS
- repeat until no change

FDs - ‘canonical cover’ Fc

Trace algo for

- A\rightarrow B \quad (4)
- B\rightarrow C \quad (3)
- A\rightarrow B \quad (4)
- (4) and (2) merge:

FDs - ‘canonical cover’ Fc

Trace algo for

- A\rightarrow C \quad (1)
- A\rightarrow B \quad (2')
- B\rightarrow C \quad (3)

in (2): 'C' is extr.

FDs - ‘canonical cover’ Fc

- A\rightarrow C \quad (1)
- A\rightarrow B \quad (2')
- B\rightarrow C \quad (3)

in (1): 'A' is extr.

CMU - 15-415
Overview - conclusions

- Domain: Ref, Integrity constraints
- Assertions and Triggers
- Functional dependencies
  - why
  - definition
  - Armstrong’s “axioms”
  - closure and cover