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

15-515 (Fall 2010)
Lecture #6: Relational Algebra

(Slides from Christos Faloutsos)

Overview

• history
• concepts
• Formal query languages
  – relational algebra
  – rel. tuple calculus
  – rel. domain calculus

History

• before: records, pointers, sets etc
• introduced by E.F. Codd in 1970
• revolutionary!
• first systems: 1977-8 (System R; Ingres)
• Turing award in 1981

Concepts - reminder

• Database: a set of relations (= tables)
• rows: tuples
• columns: attributes (or keys)
• superkey, candidate key, primary key

Example

Database:

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ssn</td>
<td>Name</td>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td></td>
</tr>
</tbody>
</table>

Example: cont’d

Database:

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ssn</td>
<td>Name</td>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td></td>
</tr>
</tbody>
</table>

k-th attribute
(Dk domain)

rel. schema (attr+domains)

tuple
Example: cont’d

• Di: the domain of the i-th attribute (eg., char(10))

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td></td>
</tr>
</tbody>
</table>

rel. schema (attr+domains)

instance

Example: cont’d

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td></td>
</tr>
</tbody>
</table>

rel. schema (attr+domains)

instance

Overview

• history
• concepts
• **Formal query languages**
  – relational algebra
  – rel. tuple calculus
  – rel. domain calculus

Formal query languages

• How do we collect information?
• Eg., find ssn’s of people in 415
• (recall: everything is a set!)
• One solution: Rel. algebra, ie., set operators
  • Q1: Which ones??
  • Q2: what is a minimal set of operators?

Relational operators

• .
• .
• .
• set union \( U \)
• set difference \(-\)

Example:

• Q: find all students (part or full time)
• A: PT-STUDENT union FT-STUDENT

<table>
<thead>
<tr>
<th>FT-STUDENT</th>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>129</td>
<td>peters</td>
<td>main str</td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>lee</td>
<td>5th ave</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PT-STUDENT</th>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td></td>
</tr>
</tbody>
</table>
Observations:

- two tables are 'union compatible' if they have the same attributes ('domains')
- Q: how about intersection \( \cap \)

Observations:

- A: redundant:
- \( \text{STUDENT} \cap \text{STAFF} = \text{STUDENT} \)

Observations:

- A: redundant:
- \( \text{STUDENT} \cap \text{STAFF} = \text{STUDENT} - (\text{STUDENT} - \text{STAFF}) \)

Observations:

- A: redundant:
- \( \text{STUDENT} \cap \text{STAFF} = \text{STUDENT} - (\text{STUDENT} - \text{STAFF}) \)

Relational operators

- .
- .
- .
- set union \( U \)
- set difference \( - \)

Double negation:
We'll see it again, later…
Other operators?

- eg, find all students on ‘Main street’
- A: ‘selection’

$$\sigma_{address='main str'} \ (STUDENT)$$

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
</tr>
</tbody>
</table>

Other operators?

- Notice: selection (and rest of operators) expect tables, and produce tables (-> can be cascaded!!)
- For selection, in general:

$$\sigma_{condition} \ (RELATION)$$

Selection - examples

- Find all ‘Smiths’ on ‘Forbes Ave’

$$\sigma_{name='Smith' \land address='Forbes ave'} \ (STUDENT)$$


Relational operators

- selection

$$\sigma_{condition} \ (R)$$
- .
- .
- set union

$$R \cup S$$
- set difference

$$R \setminus S$$

Relational operators

- selection picks rows - how about columns?
- A: ‘projection’ - eg.: $$\pi_{snum} \ (STUDENT)$$

finds all the ‘ssn’ - removing duplicates

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
</tr>
</tbody>
</table>

Relational operators

Cascading: ‘find ssn of students on ‘forbes ave’

$$\pi_{snum} \ (\sigma_{address='forbes ave'} \ (STUDENT))$$
Relational operators

- selection \( \sigma_{\text{condition}} (R) \)
- projection \( \pi_{\text{att-list}} (R) \)
- set union \( R \cup S \)
- set difference \( R - S \)

Are we done yet?

Q: Give a query we can **not** answer yet!

Relational operators

A: any query across **two** or more tables, eg., ‘find names of students in 15-415’

Q: what extra operator do we need??

A: surprisingly, cartesian product is enough!

Cartesian product

- eg., dog-breeding: MALE x FEMALE
- gives all possible couples

so what?

- Eg., how do we find names of students taking 415?
Cartesian product

• A: $\sigma_{\text{STUDENT.ssn} \neq \text{TAKES.ssn}} (\text{STUDENT} \times \text{TAKES})$

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
<th>ssn</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td>123</td>
<td>15-415 A</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>123</td>
<td>15-415 A</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td>234</td>
<td>15-413 B</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>234</td>
<td>15-413 B</td>
<td></td>
</tr>
</tbody>
</table>

FUNDAMENTAL
Relational operators

• selection $\sigma_{\text{condition}} (R)$
• projection $\pi_{\text{att-list}} (R)$
• cartesian product MALE x FEMALE
• set union $R \cup S$
• set difference $R - S$

Relational ops

• Surprisingly, they are enough, to help us answer almost any query we want!!

• derived/convenience operators:
  – set intersection
  – join (theta join, equi-join, natural join) $\Join$
  – ‘rename’ operator $\rho^R_S (R)$
  – division $R / S$

Joins

• Equijoin: $R \Join_{a=b} S = \sigma_{a=b} (R \times S)$
Cartesian product
• \( A: \ldots \sigma_{\text{STUDENT} \bowtie \text{TAKES} \bowtie \text{STUDENT}} (\text{STUDENT} \times \text{TAKES}) \)

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
<th>ssn</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main st</td>
<td>123</td>
<td>15-415</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>123</td>
<td>15-415</td>
<td>A</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main st</td>
<td>234</td>
<td>15-413</td>
<td>B</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>234</td>
<td>15-413</td>
<td>B</td>
</tr>
</tbody>
</table>

Joins
• Equijoin: \( R \bowtie_{a=S,b} S = \sigma_{R.a=S,b} (R \times S) \)
• \( \theta \)-joins: \( R \bowtie_{\theta} S \)
  generalization of equi-join - any condition \( \theta \)

Joins
• very popular: natural join: \( R \bowtie S \)
• like equi-join, but it drops duplicate columns:
  STUDENT (ssn, name, address)
  TAKES (ssn, cid, grade)

Joins
• nat. join has 5 attributes \( \text{STUDENT} \bowtie \text{TAKES} \)

Natural Joins - nit-picking
• if no attributes in common between \( R, S \):
  nat. join -> cartesian product

Overview - rel. algebra
• fundamental operators
• derived operators
  – joins etc
  – rename
  – division
• examples
Rename op.

- Q: why?
- A: shorthand; self-joins; …
- for example, find the grand-parents of ‘Tom’, given PC (parent-id, child-id)

\[ \rho_{\text{after}}(\text{BEFORE}) \]

Rename op.

- PC (parent-id, child-id)

\[
\begin{array}{c|c}
\text{p-id} & \text{c-id} \\
\hline
\text{Mary} & \text{Tom} \\
\text{Peter} & \text{Mary} \\
\text{John} & \text{Tom} \\
\end{array}
\]

- we clearly need two different names for the same table - hence, the ‘rename’ op.

Overview - rel. algebra

- fundamental operators
- derived operators
  - joins etc
  - rename
  - division
- examples

Division

- Rarely used, but powerful.
- Example: find suspicious suppliers, ie., suppliers that supplied all the parts in A_BOMB
**Division**

- **Observations:** ~reverse of cartesian product
- **It can be derived from the 5 fundamental operators (!!)**
- **How?**

\[
 r \div s = \pi_{(R \leftarrow S)}(r) - \pi_{(R \leftarrow S)}[\pi_{(R \leftarrow S)}(r) \times s] - r
\]

- **Observation:** find 'good' suppliers, and subtract! (double negation)

\[
 r \div s = \pi_{(R \leftarrow S)}(r) - \pi_{(R \leftarrow S)}[\pi_{(R \leftarrow S)}(r) \times s] - r
\]

- **Observation:** find 'good' suppliers, and subtract! (double negation)

**Division**

- **Answer:**

\[
 r \div s = \pi_{(R \leftarrow S)}(r) - \pi_{(R \leftarrow S)}[\pi_{(R \leftarrow S)}(r) \times s] - r
\]

- **Observation:** find 'good' suppliers, and subtract! (double negation)

\[
 r \div s = \pi_{(R \leftarrow S)}(r) - \pi_{(R \leftarrow S)}[\pi_{(R \leftarrow S)}(r) \times s] - r
\]

**Division**

- **Answer:**

\[
 r \div s = \pi_{(R \leftarrow S)}(r) - \pi_{(R \leftarrow S)}[\pi_{(R \leftarrow S)}(r) \times s] - r
\]

- **Observation:** find 'good' suppliers, and subtract! (double negation)

\[
 r \div s = \pi_{(R \leftarrow S)}(r) - \pi_{(R \leftarrow S)}[\pi_{(R \leftarrow S)}(r) \times s] - r
\]

**Division**

- **Answer:**

\[
 r \div s = \pi_{(R \leftarrow S)}(r) - \pi_{(R \leftarrow S)}[\pi_{(R \leftarrow S)}(r) \times s] - r
\]

- **Observation:** find 'good' suppliers, and subtract! (double negation)

\[
 r \div s = \pi_{(R \leftarrow S)}(r) - \pi_{(R \leftarrow S)}[\pi_{(R \leftarrow S)}(r) \times s] - r
\]

**Division**

- **Answer:**

\[
 r \div s = \pi_{(R \leftarrow S)}(r) - \pi_{(R \leftarrow S)}[\pi_{(R \leftarrow S)}(r) \times s] - r
\]

- **Observation:** find 'good' suppliers, and subtract! (double negation)

\[
 r \div s = \pi_{(R \leftarrow S)}(r) - \pi_{(R \leftarrow S)}[\pi_{(R \leftarrow S)}(r) \times s] - r
\]
Overview - rel. algebra

- fundamental operators
- derived operators
  - joins etc
  - rename
  - division
- examples

Sample schema

find names of students that take 15-415

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ssn</td>
<td>c-id</td>
</tr>
<tr>
<td>123</td>
<td>15-413</td>
</tr>
<tr>
<td>234</td>
<td>15-412</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
</tr>
<tr>
<td>123</td>
</tr>
<tr>
<td>234</td>
</tr>
</tbody>
</table>

- find names of students that take 15-415

$$\pi_{\text{name}}[\sigma_{\text{c-id}=15-415}(\text{STUDENT} \bowtie \text{TAKES})]$$
Sample schema

find course names of ‘smith’

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>CLASS</th>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ssn</td>
<td>c-id</td>
<td>SSN</td>
</tr>
<tr>
<td>Name</td>
<td>c-name</td>
<td>c-id</td>
</tr>
<tr>
<td>Address</td>
<td>units</td>
<td>grade</td>
</tr>
</tbody>
</table>

- 123 smith main str
- 234 jones forbes ave

Examples

• find course names of ‘smith’

\[ \pi_{\text{name}}[\sigma_{\text{name} = \text{smith}}(\text{STUDENT} \bowtie \text{TAKES} \bowtie \text{CLASS})] \]

Examples

• find ssn of ‘overworked’ students, ie., that take 412, 413, 415

- Correct answer:

\[ \pi_{\text{ssn}}[\sigma_{\text{c-name} = 412}(\text{TAKES})] \cap \\
\pi_{\text{ssn}}[\sigma_{\text{c-name} = 413}(\text{TAKES})] \cap \\
\pi_{\text{ssn}}[\sigma_{\text{c-name} = 415}(\text{TAKES})] \]

Examples

• find ssn of students that work at least as hard as ssn=123, ie., they take all the courses of ssn=123, and maybe more
Sample schema

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>Name</td>
</tr>
<tr>
<td>123 smith</td>
<td>main str</td>
</tr>
<tr>
<td>234 jones</td>
<td>forbes ave</td>
</tr>
</tbody>
</table>

TAKES

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-413</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>B</td>
</tr>
</tbody>
</table>

Examples

- find ssn of students that work at least as hard as ssn=123 (ie., they take all the courses of ssn=123, and maybe more

\[ \pi_{\text{ssn}, \text{c-id}}(TAKES) \vdash \pi_{\text{c-id}}(\sigma_{\text{ssn}=123}(TAKES)) \]

Conclusions

- Relational model: only tables (‘relations’)
- relational algebra: powerful, minimal: 5 operators can handle almost any query!