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

C. Faloutsos  
Relational model

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

- 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>Name</th>
<th>Address</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>B</td>
</tr>
</tbody>
</table>

Example: cont’d

Database:

- k-th attribute
- (Ek domain)
- rel. schema (attr./domains)
- tuple
Example: cont’d

- **rel. schema**: (attr+domains)

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

Example: cont’d

- **Dk**: the domain of the k-th attribute (eg., char(10))
- **Formally**: an instance is a subset of \((D_1 \times D_2 \times \ldots \times D_n)\)

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

Example: cont’d

- superkey (eg., ‘ssn’, ‘name’): determines record
- candidate key (eg., ‘ssn’, or ‘st#’): minimal superkey
- primary key: one of the candidate keys

Overview

- history
- concepts
- **Formal query languages**
  - relational algebra
  - relational calculus
  - relational 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, i.e., 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>PT-STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Student</td>
</tr>
<tr>
<td>123 peters</td>
<td>123 smith</td>
</tr>
<tr>
<td>main str.</td>
<td>main str.</td>
</tr>
<tr>
<td>239 lee</td>
<td>234 jones</td>
</tr>
<tr>
<td>5th ave</td>
<td>forbes ave</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:
- $\text{STUDENT} \cap \text{STAFF} = \text{STUDENT} - (\text{STUDENT} - \text{STAFF})$

Relational operators

- $\cup$
- $\cap$
- $-$
- set union $\cup$
- set difference $-$

Other operators?

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

Other operators?

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

$$\sigma_{\text{condition}}(\text{RELATION})$$
Selection - examples

- Find all ‘Smiths’ on ‘Forbes Ave’

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

‘condition’ can be any boolean combination of ‘=’, ‘>’, ‘<’, ‘…’

Relational operators

- selection $$\sigma_{condition} (R)$$
- .
- .
- set union $$R \cup S$$
- set difference $$R - S$$

Relational operators

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

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 St 123</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’

$$\pi_{ssn} (\sigma_{address=Forbes\ ave} (STUDENT))$$

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Smith</td>
<td>Main St 123</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
<td>Forbes Ave</td>
</tr>
</tbody>
</table>

Relational operators

- selection $$\sigma_{condition} (R)$$
- projection $$\pi_{attr} (R)$$
- .
- set union $$R \cup S$$
- set difference $$R - S$$

Relational operators

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!

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>Name</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
</tr>
</tbody>
</table>

Cartesian product

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

<table>
<thead>
<tr>
<th>MALE</th>
<th>FEMALE</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>spike</td>
<td>lassle</td>
</tr>
<tr>
<td>spot</td>
<td>shiba</td>
</tr>
</tbody>
</table>

so what?

• Eg., how do we find names of students taking 415?

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>Name</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
</tr>
</tbody>
</table>

Cartesian product

• A: \( \pi_{\text{name}}(\sigma_{\text{cid}=15-415}(\text{STUDENT \times TAKES})) \)

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>Name</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
</tr>
</tbody>
</table>

Cartesian product

• A: \( \pi_{\text{name}}(\sigma_{\text{cid}=15-415}(\text{STUDENT \times TAKES})) \)

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>Name</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
</tr>
</tbody>
</table>
**FUNDAMENTAL**

Relational operators

- selection: $\sigma_{\text{condition}}(R)$
- projection: $\pi_{\text{male}, \text{female}}(R)$
- 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 operators, for convenience
  - set intersection
  - join (theta join, equi-join, natural join) $\bowtie$
  - 'rename' operator $P_{a}(R)$
  - division $R \div S$

---

**Joins**

- Equijoin: $R \bowtie_{R_{a}=S_{b}} S = \sigma_{R_{a}=S_{b}}(R \times S)$

---

**Cartesian product**

- $A: \cdots \sigma_{\text{student}.\text{ssn}=\text{takes}.\text{ssn}}(\text{STUDENT} \times \text{TAKES})$

<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 st</td>
<td>123</td>
<td>15-113 A</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>jones</td>
<td>forbes ave</td>
<td>123</td>
<td>15-113 A</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>234</td>
<td>15-143 B</td>
<td></td>
</tr>
</tbody>
</table>

---

**Joins**

- Equijoin: $R \bowtie_{R_{a}=S_{b}} S = \sigma_{R_{a}=S_{b}}(R \times S)$
- 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 \( STUDENT \bowtie TAKES \)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>main st.</td>
<td>123</td>
<td>123</td>
<td>A</td>
</tr>
<tr>
<td>Jones</td>
<td>forbes ave</td>
<td>234</td>
<td>234</td>
<td>B</td>
</tr>
</tbody>
</table>

equi-join: 6 \( STUDENT \bowtie_{\text{sid}=\text{cid}} TAKES \)

Natural Joins - nit-picking

- if no attributes in common between \( R, S \):
- nat. join \( \rightarrow \) cartesian product:

Overview - rel. algebra

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

rename op.

- \( Q: \) why? \( \rho_{AFTER}^{BEFORE} \)
- \( A: \) shorthand; self-joins; ...
- for example, find the grand-parents of "Tom", given \( PC(\text{parent-id, child-id}) \)

rename op.

- \( PC(\text{parent-id, child-id}) \) \( PC \bowtie PC \)

<table>
<thead>
<tr>
<th>PC</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>id</td>
</tr>
<tr>
<td>Mary</td>
<td>Mary</td>
</tr>
<tr>
<td>Peter</td>
<td>Tom</td>
</tr>
<tr>
<td>John</td>
<td>Tom</td>
</tr>
</tbody>
</table>

rename op.

- first, WRONG attempt:
  - \( PC \bowtie PC \)
- (why? how many columns?)
- Second WRONG attempt:
  - \( PC \bowtie PC \bowtie PC \)
rename op.

- We clearly need two different names for the same table - hence, the "rename" op.

\[ p_{\text{RC1}}(PC) \bowtie q_{\text{RC2}}(q_{\text{PC2}}) \]

Overview - rel. algebra

- Fundamental operators
- Derived operators
  -Joins etc
  -Rename
  -Division
- Examples

Division

- Rarely used, but powerful.
- Example: Find suspicious suppliers, i.e., suppliers that supplied all the parts in A_BOMB

\[
\begin{array}{c|c}
\text{SHIPMENT} & \text{ABOME} \\
\hline
s1 & p1 \\
s2 & p1 \\
s1 & p2 \\
s3 & p1 \\
s5 & p3 \\
\end{array}
\]

\[
\div = \begin{array}{c|c|c}
\text{BAD S} & \text{BAD} & \text{S} \\
\hline
\text{s1} & \text{pf} & \text{p1} \\
\text{s1} & \text{pf} & \text{p2} \\
\text{s1} & \text{pf} & \text{p3} \\
\hline
\end{array}
\]

Division

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

Division

- Answer:

\[
r \div s = \pi_{(B-S)}(r) - \pi_{(B-S)}[\pi_{(B-S)}(r) \times s] - r
\]
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>

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 names of students that take 15-415

\[ \pi_{name} [\sigma_{c-id=15-415} (STUDENT \bowleft \bowtie TAKES)] \]

Sample schema
find course names of "smith"

<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>

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 course names of "smith"

\[ \pi_{c-name} [\sigma_{name='smith'} (STUDENT \bowleft \bowtie TAKES \bowleft \bowtie CLASS)] \]
Examples

- find ssn of 'overworked' students, ie., that take 412, 413, 415

Examples

- find ssn of 'overworked' students, ie., that take 412, 413, 415: almost correct answer:
  \[ \sigma_{\text{c-name}=412} (TAKES) \cap \sigma_{\text{c-name}=413} (TAKES) \cap \sigma_{\text{c-name}=415} (TAKES) \]

Examples

- find ssn of 'overworked' students, ie., that take 412, 413, 415 - Correct answer:
  \[ \pi_{\text{ssn}} [\sigma_{\text{c-name}=123} (TAKES)] \cap \]
  \[ \pi_{\text{ssn}} [\sigma_{\text{c-name}=122} (TAKES)] \cap \]
  \[ \pi_{\text{ssn}} [\sigma_{\text{c-name}=121} (TAKES)] \]

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</td>
<td>Smith</td>
</tr>
<tr>
<td>234</td>
<td>Jones</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSN</td>
</tr>
<tr>
<td>123</td>
<td>15-113</td>
</tr>
<tr>
<td>234</td>
<td>15-1413</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
  \[ TAKES + \pi_{\text{ssn}} [\sigma_{\text{c-name}=123} (TAKES)] \]
Conclusions

- Relational model: only tables (‘relations’)
- Relational algebra: powerful, minimal: 5 operators can handle almost any query!
- Most non-trivial op.: join

Cartesian product

\[
\sigma_{\text{cid}=123}(\pi_{\text{student}}(\text{STUDENT} \times \text{TAKES}))
\]

<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 str</td>
<td>123</td>
<td>15-113</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>234</td>
<td>15-113</td>
<td>A</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td>234</td>
<td>15-113</td>
<td>B</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>234</td>
<td>15-113</td>
<td>B</td>
</tr>
</tbody>
</table>