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>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>

Example: cont’d

Database:

\[ \text{SSN} c\text{-id} \text{ grade} \]

<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>

Example: cont’d

rel. schema (attr+domains)

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

• two tables are ‘union compatible’ if they have the same attributes (‘domains’)
• Q: how about intersection $\cap$

Observations:

• A: redundant:
  • STUDENT intersection STAFF =

Observations:

• A: redundant:
  • STUDENT intersection STAFF = STUDENT - (STUDENT - STAFF)
Observations:

• A: redundant:
• STUDENT intersection STAFF = STUDENT - (STUDENT - STAFF)

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

Relational operators

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

Other operators?

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

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

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>Address</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>

Notice: selection (and rest of operators) expect tables, and produce tables (\( \rightarrow \) can be cascaded!!)

For selection, in general:

\[ \sigma_{condition} (RELATION) \]
Selection - examples

• Find all ‘Smiths’ on ‘Forbes Ave’

\( \sigma_{\text{name}=\text{Smith}\land \text{address}=\text{Forbes Ave}}(STUDENT) \)

‘condition’ can be any boolean combination of ‘\(=\)', ‘\(\geq\)', ‘\(>\)', ‘\(<=\)', ‘\(<\)', ...

Relational operators

• selection \( \sigma_{\text{condition}}(R) \)
• set union \( R \cup S \)
• set difference \( R - S \)

Relational operators

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

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 str</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
</tr>
</tbody>
</table>

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

\( \pi_{\text{ssn}}(\sigma_{\text{address}=\text{forbes ave}}(STUDENT)) \)
Relational operators

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

Are we done yet?
Q: Give a query we can not answer yet!

A: any query across two or more tables, eg., ‘find names of students in 15-415’
Q: what extra operator do we need??
Cartesian product

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

\[
\text{MALE} \times \text{FEMALE} = \begin{array}{c|c|c}
\text{name} & \text{M.name} & \text{F.name} \\
\text{spike} & \text{spike} & \text{lassie} \\
\text{spot} & \text{spot} & \text{shiba} \\
\end{array}
\]

so what?

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

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-415</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-415</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>B</td>
</tr>
</tbody>
</table>

Cartesian product

- A: \[ \sigma_{\text{STUDENT}.\text{ssn}=\text{TAKES}.\text{ssn}} (\text{STUDENT} \times \text{TAKES}) \]

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

\[ \sigma_{\text{cid}=15-415} (\sigma_{\text{STUDENT}.\text{ssn}=\text{TAKES}.\text{ssn}} (\text{STUDENT} \times \text{TAKES})) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td>15-415</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>15-415</td>
<td>A</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td>15-413</td>
<td>B</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>15-413</td>
<td>B</td>
</tr>
</tbody>
</table>
\[\pi_{\text{name}} \left( \sigma_{\text{cid}=15-415} \left( \sigma_{\text{student} \bowtie \text{takes}} \left( \text{student} \times \text{takes} \right) \right) \right)\]

<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</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>234</td>
<td>15-415</td>
<td>A</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</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>

FUNDAMENTAL

Relational operators

- selection \( \sigma_{\text{condition}} (R) \)
- projection \( \pi_{\text{attribute}} (R) \)
- cartesian product MALE \( \times \) 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)
  - 'rename' operator \( \rho_R (R) \)
  - division \( R + S \)

Joins

- Equijoin: \( R \bowtie_{R.a=S.b} S = \sigma_{R.a=S.b} (R \times S) \)
Cartesian product

- A: \( \sigma_{\text{STUDENT} \times \text{TAKES}} \text{ (STUDENT } \times \text{TAKES)} \)

<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</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>234</td>
<td>15-415</td>
<td>A</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</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: \( R \bowtie_{R.a=S.b} S = \sigma_{R.a=S.b} (R \times S) \)
- 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

- Natural 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? $\rho_{\text{AFTER}}(\text{BEFORE})$
- A: shorthand; self-joins; …
- for example, find the grand-parents of ‘Tom’, given PC (parent-id, child-id)

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

Rename op.

- PC (parent-id, child-id) $PC \bowtie PC$

<table>
<thead>
<tr>
<th>PC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p-id</td>
<td>c-id</td>
</tr>
<tr>
<td>Mary</td>
<td>Tom</td>
</tr>
<tr>
<td>Peter</td>
<td>Mary</td>
</tr>
<tr>
<td>John</td>
<td>Tom</td>
</tr>
</tbody>
</table>
Rename op.

- first, WRONG attempt:
  \[ PC \times PC \]
- (why? how many columns?)
- Second WRONG attempt:
  \[ PC \times PC_{c=PC.p=id} \]

Overview - rel. algebra

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

Rename op.

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

\[ \rho_{PC_1}(PC) \times PC_{c=PC.p=id} \]
Division

\[ \text{SHIPMENT} \]

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

\[ \text{ABOMB} \]

\[
\begin{array}{c|c}
\text{p#} & \text{p} \\
\hline
\text{p1} & \text{p1} \\
\text{p2} & \text{p2} \\
\end{array}
\]

\[ = \text{BAD_S} \]

\[
\begin{array}{c|c}
\text{s#} & \text{p} \\
\hline
\text{s1} & \text{p1} \\
\end{array}
\]

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

- Answer:

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

- Observation: find ‘good’ suppliers, and subtract! (double negation)

---

Division

\[ \text{SHIPMENT} \]

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

\[ \text{ABOMB} \]

\[
\begin{array}{c|c}
\text{p#} & \text{p} \\
\hline
\text{p1} & \text{p1} \\
\text{p2} & \text{p2} \\
\end{array}
\]

\[ = \text{BAD_S} \]

\[
\begin{array}{c|c}
\text{s#} & \text{p} \\
\hline
\text{s1} & \text{p1} \\
\end{array}
\]

- Answer:

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

- Observation: find ‘good’ suppliers, and subtract! (double negation)
\[ r + s = \mathcal{P}(R-S)(r) - \mathcal{P}(R-S)[(\mathcal{P}(R-S)(r) \times s) - r] \]

- **Answer:**
  - All suppliers
  - All bad parts

\[ r + s = \mathcal{P}(R-S)(r) - \mathcal{P}(R-S)[(\mathcal{P}(R-S)(r) \times s) - r] \]

- **Answer:**
  - All possible suspicious shipments

\[ r + s = \mathcal{P}(R-S)(r) - \mathcal{P}(R-S)[(\mathcal{P}(R-S)(r) \times s) - r] \]

- **Answer:**
  - All suppliers who missed at least one suspicious shipment, i.e.: ‘good’ suppliers

\[ r + s = \mathcal{P}(R-S)(r) - \mathcal{P}(R-S)[(\mathcal{P}(R-S)(r) \times s) - r] \]

- **Answer:**
  - All possible suspicious shipments
  - all possible suspicious shipments

- **Answer:**
  - all suppliers who missed at least one suspicious shipment, i.e.: ‘good’ suppliers
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>
<tr>
<td>Name</td>
<td>c-name</td>
</tr>
<tr>
<td>smith</td>
<td>s.e.</td>
</tr>
<tr>
<td>jones</td>
<td>o.s.</td>
</tr>
<tr>
<td>Address</td>
<td>units</td>
</tr>
<tr>
<td>main str</td>
<td></td>
</tr>
<tr>
<td>forbes ave</td>
<td></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>

Examples

- 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></th>
<th></th>
<th></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>

<table>
<thead>
<tr>
<th>CLASS</th>
<th>c-id</th>
<th>c-name</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-413</td>
<td>s.e.</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15-412</td>
<td>o.s.</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

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

Examples

• find course names of ‘smith’

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

Examples

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

Almost correct answer:

$\sigma_{c-name=412}(\text{TAKES}) \cap \sigma_{c-name=413}(\text{TAKES}) \cap \sigma_{c-name=415}(\text{TAKES})$
Examples

- find ssn of ‘overworked’ students, i.e., that take 412, 413, 415 - Correct answer:

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

Sample schema

<table>
<thead>
<tr>
<th>STUDENT</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ssn</td>
<td>Name</td>
<td>Address</td>
</tr>
<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>

<table>
<thead>
<tr>
<th>CLASS</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c-id</td>
<td>c-name</td>
<td>units</td>
<td></td>
</tr>
<tr>
<td>15-413</td>
<td>s.e.</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15-412</td>
<td>o.s.</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Examples

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

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

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