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></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>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>

Example: cont’d

Database:

k-th attribute (Dk domain)

<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>

rel. schema (attr+domains)

tuple

Example: cont’d

rel. schema (attr+domains)

instance

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

Example: cont’d

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></td>
<td>129</td>
<td>peters</td>
<td>main str</td>
</tr>
<tr>
<td></td>
<td>239</td>
<td>lee</td>
<td>5th ave</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></td>
<td>123</td>
<td>smith</td>
<td>main str</td>
</tr>
<tr>
<td></td>
<td>234</td>
<td>jones</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:
- \( \text{STUDENT} \cap \text{STAFF} = \text{STUDENT} \)

Observations:

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

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

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’

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

<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:

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

• Find all ‘Smiths’ on ‘Forbes Ave’

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

‘condition’ can be any boolean combination of *=*, >=*, >*, *=*, ...
Relational operators

- selection  \( \sigma_{\text{condition}} (R) \)
- projection  \( \pi_{\text{attr-list}} (R) \)
- 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??

A: surprisingly, cartesian product is enough!
Cartesian product

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

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

so what?

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

\[
\begin{array}{|c|c|}
\hline
\text{STUDENT} & \text{SSN} \\
\hline
\text{Ssn name Address} & \text{c-id grade} \\
\hline
123 smith main str & 123 15-415 A \\
234 jones forbes ave & 234 15-413 B \\
\hline
\end{array}
\]

Cartesian product

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

\[
\begin{array}{|c|c|c|}
\hline
\text{Ssn name Address} & \text{ssn c-id grade} \\
\hline
123 smith main str & 123 15-415 A \\
234 jones forbes ave & 234 15-413 B \\
\hline
\end{array}
\]

Cartesian product

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

\[
\begin{array}{|c|c|c|}
\hline
\text{Ssn name Address} & \text{ssn c-id grade} \\
\hline
123 smith main str & 123 15-415 A \\
234 jones forbes ave & 234 15-413 B \\
\hline
\end{array}
\]
\[ \pi_{\text{name}}( \sigma_{\text{cid}=15\text{-}415}(\sigma_{\text{STUDENT,x=TAKESS,sm}}(\text{STUDENT} \times \text{TAKESS}))) ) \]

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

**Relational operators**

- Selection: \( \sigma_{\text{condition}}(R) \)
- Projection: \( \pi_{\text{att-list}}(R) \)
- 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:** \( \sigma_{STUDENT.ssn=TAKES.ssn} (STUDENT \times TAKES) \)

<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>15-415</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: \( R \bowtie_{a=S.b} S = \sigma_{a=S.b} (R \times S) \)
- 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)

<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>15-415</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>
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? \( \rho_{\text{AFTER}}(B\text{EFORE}) \)
• A: shorthand; self-joins; …
• for example, find the grand-parents of ‘Tom’, given PC (parent-id, child-id)

Rename op.

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

<table>
<thead>
<tr>
<th>PC</th>
<th>PC</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:
  \[ PC \bowtie PC \]

- (why? how many columns?)

- Second WRONG attempt:
  \[ PC \bowtie_{PC.c\text{-id}=PC.p\text{-id}} PC \]

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.

\[ \rho_{PC} (PC) \bowtie_{PC.c\text{-id}=PC.p\text{-id}} PC \]

Division

- Rarely used, but powerful.
- Example: find suspicious suppliers, i.e., suppliers that supplied all the parts in A\_BOMB
**Division**

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

**Division**

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

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

**Division**

- Answer:

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

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

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

- All suppliers
- All bad parts

Division

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

- All possible suspicious shipments

Division

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

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

Division

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

- All possible suspicious shipments
- 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>Name</td>
<td>c-name</td>
</tr>
<tr>
<td>Address</td>
<td>units</td>
</tr>
<tr>
<td>123</td>
<td>15-413</td>
</tr>
<tr>
<td>smith</td>
<td>s.e.</td>
</tr>
<tr>
<td>234</td>
<td>15-412</td>
</tr>
<tr>
<td>jones</td>
<td>o.s.</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>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>
</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 course names of ‘smith’

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

Examples

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

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

Examples

• find ssn of ‘overworked’ students, ie., that take 412, 413, 415: almost correct answer:
Examples

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

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

Sample schema

<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>

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

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

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