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>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>5sn</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>

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>rel. schema (attr + domains)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5sn</td>
<td>tuple</td>
</tr>
</tbody>
</table>

Example: cont’d
Example: cont’d

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>rel. schema (attr+domains)</th>
<th>instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>San</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>

Example: cont’d

• 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 =
Observations:

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

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

Relational operators

- .
- .
- .
- set union \( \cup \)
- set difference \( - \)
Other operators?

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

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

<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 (\(\Rightarrow\) 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 \( \sigma_{\text{condition}} (R) \)
- set union \( R \cup S \)
- set difference \( R - S \)

Relational operators

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

finds all the ‘ssn’ - removing duplicates

<table>
<thead>
<tr>
<th>Student</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’

\[ \pi_{\text{ssn}}(\sigma_{\text{address='forbes ave'}}(\text{STUDENT})) \]
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??
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

\[
\begin{array}{cc}
\text{MALE} & \times \\
\text{name} & \text{name} \\
spike & lassie \\
spit & shiba
\end{array}
\]

\[
\begin{array}{cc}
\text{FEMALE} & = \\
\text{name} & \\
lassie & spike \\
shiba & spit
\end{array}
\]

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

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

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

### Cartesian product

- \( \sigma_{\text{cid = 15-415}} (\sigma_{\text{STUDENT in \text{STUDENT}}} \times \text{ Takes in \text{STUDENT}} \times \text{STUDENT} \times \text{TAKES}) \)

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

### \( \pi_{\text{name}} (\sigma_{\text{cid = 15-415}} (\sigma_{\text{STUDENT in \text{STUDENT}}} \times \text{ Takes in \text{STUDENT}} \times \text{STUDENT} \times \text{TAKES})) \)

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

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

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 $\rho_{\sigma}(R)$
  - division $R \div S$

Equijoin:

$$R \bowtie_{R.a=S.b} S = \sigma_{R.a=S.b}(R \times S)$$
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>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>15-416</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)
Joins

- nat. join has 5 attributes $STUDENT \times TAKES$

<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>15-413</td>
<td>B</td>
</tr>
<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>15-413</td>
<td>B</td>
</tr>
</tbody>
</table>

equi-join: 6 $STUDENT \times TAKES \times TAKES$

Natural Joins - nit-picking

- if no attributes in common between R, S:
  - nat. join $\to$ 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></th>
<th>ID</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:
  \( PC \bowtie_{\text{PC.c-id=PC.p-id}} PC \)
Rename op.

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

\[ \rho_{PC}(PC) \bowtie_{PC_{id}=PC_{id}} PC \]

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
### Division

#### Observations: ~reverse of cartesian product

- It can be derived from the 5 fundamental operators (!!!)
- How?

#### Answer:

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

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

- Answer:

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

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

All suppliers

All bad parts

all possible suspicious shipments
**Division**

- **Answer:**

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

---

**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>Name</td>
<td>c-name</td>
</tr>
<tr>
<td>Address</td>
<td>units</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
</tr>
<tr>
<td></td>
<td>main</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
</tr>
<tr>
<td></td>
<td>forbes</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

\[ \pi_{\text{name}} \left[ \sigma_{\text{c-id}=15-415} (\text{STUDENT} \bowtie \text{TAKES}) \right] \]
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>

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’

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

• 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:

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

Examples

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

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

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:

$$\pi_{\text{ssn}}[\sigma_{c\cdot\text{name}=123}(TAKES)] \cap \pi_{\text{ssn}}[\sigma_{c\cdot\text{name}=123}(TAKES)] \cap \pi_{\text{ssn}}[\sigma_{c\cdot\text{name}=123}(TAKES)]$$
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>15-413</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!