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>123</td>
<td>123-413</td>
<td>Smith</td>
<td>Main Str</td>
</tr>
<tr>
<td>234</td>
<td>234-413</td>
<td>Jones</td>
<td>Forbes Ave</td>
</tr>
</tbody>
</table>

Example: cont’d

Database:

<table>
<thead>
<tr>
<th>Student</th>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123-413</td>
<td>Smith</td>
<td>Main Str</td>
</tr>
<tr>
<td>234</td>
<td>234-413</td>
<td>Jones</td>
<td>Forbes Ave</td>
</tr>
</tbody>
</table>
Example: cont’d

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

rel. schema (attr+domains)

Example: cont’d

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

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

rel. schema (attr+domains)

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

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

<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 (\(\rightarrow\) can be cascaded!!)
• For selection, in general:

\[ \sigma_{\text{condition}} \text{(RELATION)} \]

Selection - examples

• Find all ‘Smiths’ on ‘Forbes Ave’

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

‘condition’ can be any boolean combination of ‘\(=\)’, ‘\(>\)’, ‘\(>=\)’, ‘\(<\)’, ‘\(<=\)’, ‘\(!=\)’, ‘\(\in\)’, ‘\(\not\in\)’...
Relational operators

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

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>

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>Ssn</td>
</tr>
<tr>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>234</td>
<td>234</td>
</tr>
</tbody>
</table>

Cartesian product

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

**MALE**
- name
- spike
- spot

**FEMALE**
- name
- lassie
- shiba

**MALE** x **FEMALE** =
- M name
- F name
- spike
- spike
- spot

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>Ssn</td>
</tr>
<tr>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>234</td>
<td>234</td>
</tr>
</tbody>
</table>
Cartesian product

A:

\[
\sigma_{\text{STUDENT}.\text{sid}=\text{TAKES}.\text{cid}}(\text{STUDENT} \times \text{TAKES})
\]

<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>15-415</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>

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

<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>15-415</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>
FUNDAMENTAL
Relational operators

- selection \( \sigma_{\text{condition}} (R) \)
- projection \( \pi_{\text{attr-list}} (R) \)
- 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) \( \bowtie \)
  - ‘rename’ operator \( \rho_{\pi} (R) \)
  - division \( R \div S \)

Joins

- Equijoin: \( R \bowtie_{\text{a}=b} S = \sigma_{\text{a}=b} (R \times S) \)
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>16-415A</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>234</td>
<td>15-413B</td>
<td>B</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td>234</td>
<td>15-413B</td>
<td>B</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>234</td>
<td>15-413B</td>
<td>B</td>
</tr>
</tbody>
</table>

Joins

- Equijoin: \( R \bowtie_{a,b} S = \sigma_{a,b}(R \times S) \)
- \( \theta \)-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 $STUDENT \bowtie TAKE$

<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>15-416</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
<td>123</td>
<td>15-416</td>
<td>A</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
<td>234</td>
<td>15-413</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>

equi-join: 6 $STUDENT \bowtie TAKE$

**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)

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>
<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: $PC \bowtie_{PC.p-id=PC.c-id} PC$
Rename op.

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

\[ \rho_{PC}(PC) \bowtie_{PC_{\text{old}}=PC_{\text{new}}} PC \]

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:

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

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

- Answer:

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

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

- Answer:

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

all possible suspicious shipments that didn’t happen

- 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>Address</td>
</tr>
<tr>
<td>smith</td>
<td>main str</td>
</tr>
<tr>
<td>jones</td>
<td>forbes ave</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

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

find course names of ‘smith’

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>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>

Examples

• find course names of ‘smith’

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

Examples

• find ssn of ‘overworked’ students, i.e., that take 412, 413, 415
Examples

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

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

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>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)

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

Conclusions

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